UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

Fluxos de Trabalho Interorganizacionais Descritos de Forma Global e Comum: A linguagem CBPEL

António Gelásio Frazão Isidro Teófilo

Licenciado

Dissertação para a obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador: Doutor Alberto Manuel Rodrigues da Silva

Júri

Presidente: Doutor Luís Manuel Marques Custódio

Vogal: Doutor Pedro Alexandre de Mourão Antunes

Vogal: Doutor Fernando Henrique Corte Real Mira da Silva

Vogal: Doutor Alberto Manuel Rodrigues da Silva

Setembro de 2009

Agradecimentos

Gostaria de agradecer ao meu orientador, o Professor Doutor Alberto Silva, a persistência mostrada ao longo deste processo.

Gostaria de agradecer a todos os meus colegas de trabalho no ISEL, pelo seu forte apoio.

Gostaria de agradecer a todos os meus amigos pelo seu constante incentivo.

Gostaria de agradecer a toda a minha família pela paciência e compreensão que demonstraram durante a execução deste processo.

Por fim, gostaria de agradecer o inesgotável incentivo, apoio, abnegação e muito mais que não consigo expressar por palavras, à minha querida noiva Maria do Céu Araújo.

Resumo

A tecnologia convencional de fluxos de trabalho visa melhorar a operacionalidade dos processos dentro de uma organização. Contudo, cada vez mais, as organizações ao intervirem no mercado global, suportado pela Internet, participam em processos que as envolvem com outras organizações. Estes processos são designados de processos interorganizacionais.

Dentro da tecnologia de processos interorganizacionais salienta-se os cenários que envolvem várias organizações, mas em que nenhuma delas detém o controle total do cenário. Essas situações são denominadas de coreografia de processos.

Esta dissertação aborda a concepção de coreografias de processos descritas de um ponto de vista global e comum, pois afirma-se que essa forma permite uma descrição mais simplificada e menos propensa à ocorrência de erros. Assim propõe uma linguagem, denominada de CBPEL, para descrição de coreografias, descritas de um ponto de vista global e comum, baseada nos conceitos da linguagem BPEL, com suporte a tratamento de falhas e compensação ao nível do bloco. Além da linguagem, é apresentado a extracção dos processos das entidades participantes a partir das coreografias, o protocolo de coordenação e as nuances da interligação dos processos dos participantes para com a figura do coordenador.

Esta dissertação apresenta então uma perspectiva do ciclo completo de uma coreografia, pois parte da própria coreografia, gera os processos participantes, define a parte de coordenação e termina no ponto de permitir a simulação, ou execução, de todo o cenário.

A linguagem proposta também tem como objectivo contribuir para uma melhor compreensão da linguagem WSCDL, que é uma linguagem com os mesmos objectivos mas com um âmbito mais abrangente e que apresenta várias dificuldades de instanciação.

Palavras-chave: fluxos de trabalho interorganizacionais, coreografia de processos, coreografias coordenadas, WSCDL, CBPEL, BPEL.

Abstract

The conventional technology of workflows aims to improve the functionality of processes within an organization. However, by intervening in the global market, supported by Internet, the organizations participate more and more in processes which involve them with other organizations. These processes are called inter-organizational processes.

In the inter-organizational processes technology, the scenarios which involve several organizations are stressed when none of them has the total control of the scenario. These situations are designated as processes' choreography.

This dissertation approaches the conception of processes' choreography described from a global and common point of view, since this way it is possible a more simplified description and less prone to mistakes. Thus, it proposes a language called CBPEL to describe the choreographies, from a global and common point of view, based on the concepts of BPEL language, with support of exception handling and compensation at block level. Besides the language, it is also presented the extraction of the participants' processes from the choreographies, the coordination protocol and the inter-connection between the participants' processes and the coordinator.

So this dissertation presents a perspective of the entire cycle of a choreography, since it starts from the choreography itself, conceives the participant processes, defines the coordination and ends at the point of allowing the simulation, or execution, of the all scenario.

The suggested language has also the objective of contributing for a better understanding of WSCDL language, which is a language with the same aims but with a wider scope and which presents several difficulties of use

Keywords: inter-organizational workflows, choreography, coordinated choreographies, WSCDL, CBPEL, BPEL.

Convenções Tipográficas

Apresentam-se as convenções tipográficas utilizadas neste documento:

– O texto normal é escrito em fonte Century Schoolbook e em estilo normal.

Exemplo: texto normal.

- Os termos em língua inglesa são escritos com a mesma fonte que o texto normal mas em estilo itálico.

Exemplo: um *while* é uma actividade de execução cíclica.

- Os nomes mais relevantes em língua inglesa poderão ser escritos com o mesmo estilo que o texto normal.

Exemplo: Extensible Markup Language.

- Os termos em língua inglesa, que correspondem a tradução directa e que visam clarificar a sua tradução para língua portuguesa, são também colocados entre aspas e parênteses curvos.

Exemplo: fluxo de trabalho ("*workflow*").

- As abreviaturas são escritas com a mesma fonte que o texto normal mas com letras maiúsculas. O estilo será sempre normal independentemente da língua utilizada.

Exemplo: fluxo de trabalho interorganizacional (FTIO)

Índice Geral

Índice de figuras

Índice de tabelas

Capítulo 1

Introdução

Este capítulo visa apresentar uma introdução a esta dissertação. Começa por apresentar o contexto de trabalho, estabelecendo o seu enquadramento. Prossegue com a identificação dos aspectos a explorar e definição dos objectivos a perseguir. Por fim descreve a estrutura da dissertação, dando uma perspectiva da sequência e localização dos assuntos que nela serão abordados.

1.1 Apresentação do contexto

A tecnologia de fluxos de trabalho permite modelar e automatizar os processos dentro de uma organização e já é uma tecnologia madura [Kappel00] [Leymann00]. Com ela, as organizações podem implementar de forma flexível e rápida os seus processos, e assim aumentarem a sua capacidade de resposta, quer em contextos de mercado que são cada vez mais competitivos, quer em contextos institucionais onde a sociedade actual exige cada vez mais eficiência. Tal aumento da facilidade de concepção e alteração dos processos proporciona uma maior competitividade, pois o tempo é hoje em dia um factor essencial. Este aumento de competitividade deve-se também ao facto das tecnologias relacionadas com a Internet proporcionarem uma infra-estrutura de comunicação, disponibilizando às organizações, os meios necessários para estas colocarem os seus serviços *on-line,* e portanto disponíveis a nível mundial, de forma instantânea [Kreger01].

Contudo, nos últimos anos, as organizações estão, de forma geral, inseridas em teias de interligações e dependências, devido à necessidade de se especializarem e assim maximizarem a sua competitividade. Desta forma, necessitam cada vez mais de efectuar

processos que saem fora dos seus limites, de que as situações de *outsourcing* [Benjamin95] e de empresas virtuais [Lima01] são exemplos disso. Passa-se de contextos intra-organizacionais, com processos denominados de intra-organizacionais, em que a empresa detém todo o controlo do processo, para contextos interorganizacionais, com processos denominados também de interorganizacionais, onde o controlo dos processos é repartido por várias organizações. Esses processos efectuados entre organizações vêm portanto colocar novos desafios face ao paradigma dos processos intra-organizacionais. Como exemplo desses desafios temos a heterogeneidade dos sistemas a interligar, a segurança das comunicações, as implicações legais ou a preservação da autonomia das organizações [Bernauer02]. Mas a natureza das interacções também se altera substancialmente, pois o controlo deixa de ser centrado numa organização para ser partilhado entre várias organizações.

Segundo Bernauer [Bernauer02] existem várias linguagens vocacionadas para descreverem processos que envolvem várias organizações. Contudo a maioria dessas linguagens estão vocacionadas para descrever processos que solicitam a execução de serviços noutras organizações. Resultando numa descrição de processos centrada no ponto de vista de um processo.

Ao passarmos para processos que envolvem várias organizações onde o controlo não é exclusivo de nenhuma organização mas sim partilhado entre as organizações participantes, a descrição do cenário interorganizacional não pode ser efectivamente capturada por essas linguagens. É necessário uma descrição conjunta dos processos envolvidos, de um ponto de vista global às várias organizações. A metodologia, de concepção de processos interorganizacionais, consiste então: primeiro, na concepção do processo global; e por segundo, na extracção dos processos dos vários participantes.

Uma abordagem para a modelação de processos de fluxos de trabalho interorganizacionais, consiste na descrição dos fluxos dos vários intervenientes e das suas interligações como um só processo. A concepção destes processos globais torna-se complexa quando os fluxos envolvidos comportam um razoável número de tarefas. Dessa complexidade resulta uma forte propensão à ocorrência de erros na concepção dos fluxos envolvidos, requerendo de forma obrigatória a utilização de uma ferramenta de validação.

Uma outra abordagem, para a modelação de processos de fluxos de trabalho interorganizacionais, consiste em utilizar uma descrição unificadora das perspectivas dos

vários participantes, resultando num fluxo único e comum a todos eles. Esse fluxo único descreverá todo o processo interorganizacional, e terá como principal característica apresentar as interacções também de uma perspectiva comum, ou seja, em vez de conter uma emissão e uma emissão, contém uma actividade com a indicação da emissão conjunta com a de recepção. Da descrição global, comum e única extraí-se os fluxos dos vários participantes, primeiro por replicação do fluxo global, depois por eliminação das actividades (interacções) em que o participante não participa, e depois por adequação e simplificação do restante fluxo. A existência de um único fluxo torna a concepção mais fácil e menos propensa a erros, uma vez que o mesmo fluxo será replicado para os vários participantes, e se a adequação e simplificação não alterarem as características do fluxo extraído, então conserva-se a compatibilidade para com o fluxo global, comum e único. A descrição de fluxo único, pelo facto de unificar as várias perspectivas e por representar as interacções como uma única actividade é também uma descrição mais concisa do que a descrição de um processo interorganizacional por conjunção dos processos das várias organizações envolvidas.

1.2 Objectivos e motivação

Em termos cronológicos o primeiro objectivo desta dissertação consistia em identificar e analisar os desafios criados pelos fluxos de trabalho interorganizacionais. Contudo para isso, identificou-se, logo à partida, a dificuldade de que a descrição de um cenário interorganizacional era extensa e difícil de conceber. O que levou à necessidade de haver uma forma mais expedita de conceber os processos interorganizacionais e também menos propensa a erros de edição. Esse requisito é fundamental para se conseguir lidar com cenários interorganizacionais que facilmente podem possuir mais de quatro organizações e conter várias dezenas de actividades. Como exemplo temos os dois seguintes contextos interorganizacionais, onde se apresenta as várias organizações envolvidas: compra de uma habitação – comprador, vendedor, entidade bancária do comprador, conservatória do registo predial, cartório notarial e repartição de finanças; e escala de um navio num porto – capitão do navio, agente de navegação, armador, capitania do porto, administração do porto, brigada fiscal, alfândega, sanidade marítima, pilotos, operadores portuários e operadores de abastecimentos. De forma a se ter um modo mais expedito de descrever os processos interorganizacionais dever-se-ia adoptar uma linguagem que suportasse a descrição de processos segundo um fluxo único, comum e global para todos os participantes.

Entretanto surgiu a linguagem WSCDL [Kavantzas05], que visa exactamente o mesmo objectivo, pois é uma linguagem dedicada à descrição de processos interorganizacionais onde o cenário é descrito de uma forma global por um fluxo único e comum a todos os participantes. Contudo esta linguagem, pelo facto de ter uns objectivos muito abrangentes, pois tenta modelar vários tipos de comportamento relativos ao tratamento de falhas, e de ter uma especificação algo imprecisa, não tem permitido a sua efectiva utilização. No sétimo capítulo é apresentado uma análise que sustenta esta última afirmação.

Dado esse cenário, surgiu a necessidade de ser criar uma linguagem com uma abrangência mais reduzida, mas que permitisse colocar os cenários interorganizacionais de facto a funcionar. Tal iria, no futuro, permitir fazer um mapeamento da WSCDL para essa linguagem e clarificar se a especificação da WSCDL é ou não realista. Também serviria para fazer um levantamento das dificuldades que seriam encontradas ao longo desse trajecto.

Com essa motivação é, então, aqui apresentada a linguagem CBPEL (COMMON BPEL), que é uma linguagem para descrever processos interorganizacionais de forma comum, ou seja global e única, e baseada na linguagem BPEL (*Business Process Execution Language*) [Andrews03]. A escolha da linguagem BPEL, como linguagem base, deve-se à sua grande adopção pela comunidade científica e empresarial.

A adopção da linguagem BPEL teve como consequência a extrapolação do seu modelo de tratamento de falhas e compensações para a linguagem CBPEL. Deste modo foi incorporado na linguagem CBPEL a noção de coordenação ao nível do bloco e entre blocos. A coordenação entre blocos coloca-se entre blocos encaixados, onde: as falhas podem ser remetidas para o bloco superior; o bloco superior pode compensar um bloco inferior terminado; e etc. A coordenação será implementada recorrendo à figura de um coordenador.

De uma forma sintética os objectivos desta dissertação são: apresentar uma linguagem para descrever fluxos de trabalhos interorganizacionais de forma comum, ou seja, com fluxo global e único para todos os participantes, baseada no comportamento da linguagem BPEL; apresentar os outros aspectos necessários à execução dos processos dos participantes nomeadamente a definição do protocolo de coordenação e a geração dos processos dos participantes; por fim, verificar se a linguagem BPEL conseguiu de facto

suportar os processos gerados da linguagem CBPEL e fazer uma análise verificando em que medida a linguagem WSCDL pode ser equiparada à linguagem CBPEL.

1.3 Estrutura da dissertação

Seguidamente descreve-se o conteúdo dos capítulos que compõem esta dissertação.

O primeiro capítulo, que é o presente capítulo, visa apresentar o contexto da dissertação, os seus objectivos e as motivações que estiveram por detrás deles.

O segundo capítulo visa apresentar os conceitos, características e paradigmas associados, em primeiro aos fluxos de trabalho intra-organizacionais, e em segundo aos fluxos de trabalho interorganizacionais

O terceiro capítulo visa descrever as tecnologias existentes para o suporte a fluxos de trabalho interorganizacionais. Inicia por descrever as linguagens para modelação de fluxos de trabalho, concluindo com uma comparação das certas características relevantes para o suporte aos fluxos de trabalho interorganizacionais, depois apresenta os modelos para suporte transaccional em contextos interorganizacionais; terminando com a apresentação das especificações existentes para suportar os modelos já identificados.

O quarto capítulo inicia a parte de concretização, ou seja, é neste ponto que se começa a descrever a linguagem CBPEL. Este capítulo apresenta primeiro uma perspectiva da representação comum de fluxos de trabalho interorganizacionais, depois contém a descrição dos elementos da parte elementar da linguagem CBPEL, terminando com a transformação dos mesmos para a linguagem BPEL.

O quinto capítulo visa descrever a parte de tratamento de falhas da linguagem CBPEL. Inicia com a descrição do tratamento de falhas na linguagem BPEL, depois apresenta o protocolo de coordenação adoptado para a linguagem CBPEL, seguindo-se a descrição dos elementos CBPEL que suportam o tratamento de falhas, e termina com a transformação dos mesmos para a linguagem BPEL.

O sexto capítulo contém os cenários de aplicação, onde nele são descritos dois cenários, sendo um deles apresentado com e sem tratamento de falhas. Este capítulo visa mostrar a aplicação da linguagem desenvolvida e o resultado da geração dos processos dos participantes.

O sétimo capítulo, visa apresentar primeiro a problemática do suporte da linguagem CBPEL pela linguagem BPEL, por segundo as alterações introduzidas pela versão 2.0 da linguagem BPEL face à versão 1.1, que é a versão utilizada neste trabalho, e por terceiro apresenta uma perspectiva comparativa da linguagem WSCDL face à CBPEL.

O oitavo capítulo termina este trabalho, indicando os aspectos que ficaram em aberto e perspectivam a continuação deste trabalho preconizando possíveis futuras vias de investigação

Capítulo 2

Introdução aos Fluxos de Trabalho Interorganizacionais

Este capítulo visa apresentar a tecnologia de fluxos de trabalho interorganizacionais. Em primeiro lugar será apresentada a tecnologia de fluxos de trabalho intra-organizacionais, de forma a introduzir e clarificar os conceitos e terminologia base. Em segundo lugar será portanto apresentada a tecnologia dos fluxos de trabalho interorganizacionais, com os seus conceitos e diferenças face à tecnologia intra-organizacional.

2.1 Tecnologia de fluxos de trabalho intra-organizacionais

Nesta secção é apresentada a tecnologia de fluxos de trabalho intra-organizacionais, com o objectivo de introduzir e clarificar os conceitos e terminologia base de suporte aos fluxos de trabalho interorganizacionais.

2.1.1 Introdução à tecnologia de fluxos de trabalho

Segundo [RauschScho97], um dos maiores desafios colocados às organizações, por todo o mundo, consiste em adaptarem-se muito rapidamente às alterações exteriores, de modo a serem capazes de competirem efectivamente no então recente mercado global.

Também segundo [RauschScho97] as organizações adquirem uma maior flexibilidade às rápidas alterações do mercado ao tornarem os seus processos explícitos e ao automatizarem o seu controlo.

Como resposta à necessidade enunciada no primeiro parágrafo, e de modo a suportar a solução identificada no segundo parágrafo, surgiu (ou foi evoluindo) a tecnologia de gestão (e execução) assistida por computador de processos organizacionais descritos de forma computorizada [RauschScho97]. Esta tecnologia também é referenciada por tecnologia de gestão de fluxos de trabalho, ou abreviadamente tecnologia de fluxos de trabalho.

Processos organizacionais

Um processo organizacional é um grupo de recursos e trabalho, agregado por um conjunto de tarefas interligadas, iniciadas por um evento, e com um determinado objectivo para a organização [Sharp01].

A modelação do funcionamento das organizações baseada em processos [Weske99], permitiu às organizações possuir um conhecimento mais completo do seu funcionamento, do que quando modelavam o seu funcionamento exclusivamente por pequenas unidades de trabalho, denominadas de funções. Passou-se portanto de uma visão de acontecimentos isolados (as funções), para uma visão de conjuntos de acontecimentos interligados (os processos).

Esta mudança de paradigma traz as seguintes vantagens para as organizações [RauschScho97]:

- Eficácia: uma vez que os processos são representados explicitamente, eles podem ser mais fácil e rapidamente alterados;

- Eficiência: o tempo de execução pode ser reduzido, por identificação de execução paralela, ou por troca directa de informação entre o produtor e o consumidor;

- Transparência: havendo um sistema de gestão de processos, a informação acerca dos processos, assim como a informação que eles processam, fica disponível para toda a organização;

- Consistência: o sistema de gestão de processos é capaz de sistematicamente manter e supervisionar os requisitos de consistência acerca dos processos em execução;

- Inovação: a adopção de um sistema de gestão de processos numa organização, requer uma readaptação dos seus procedimentos, a qual normalmente resulta em melhoramentos da sua qualidade.

Devido às vantagens enumeradas esta tecnologia é praticamente ubíqua [Sheth99]. Vejamos alguns exemplos vindos de diferentes contextos:

- Contexto empresarial: banca – pedido de empréstimo; seguros – tratamento de uma reclamação de prémio; comércio electrónico – realização de uma compra pela Internet; direito – acompanhamento de um caso de uma herança; telecomunicações – contratualização dinâmica de canais de comunicação;

- Contexto de manufactura: produção de um televisor; integração entre sistemas de produção e sistemas de venda;

- Contexto cientifico: experiência com uma enzima, experiência em física de altas energias; aplicações para processamento geográfico de dados;

- Outros contextos organizacionais: hospitalar – atendimento de um paciente nas urgências; educacional – pagamento de propinas de um aluno; marítimo – escala de um navio num porto.

2.1.1.1 Resumo histórico

Segundo [Weske99], os primórdios da tecnologia de fluxos de trabalho, cujo conceito assenta na utilização de ferramentas genéricas, ou de pelo menos métodos genéricos, para suportar os processos organizacionais, remonta à década de 1970, tendo Skip Ellis [Ellis80] e Michael Zisman [Zisman77] sido os seus pioneiros, em Xerox Parc, num trabalho sobre "Sistemas de automação de escritórios". O sistema apresentado em [Zisman77] permite a especificação de procedimentos de escritório, recorrendo a redes de Petri [Petri62] [Peterson81], e a sua execução baseia-se num sistema de correio electrónico.

Esta tecnologia limitou-se à automação de actividades de escritório e processamento de documentos por imagem ("*document imaging*"). Somente na década de 1990 é que

estes sistemas se estabeleceram como parte dos sistemas de informação empresariais. Segundo [Weske99] este atraso foi devido à seguinte série de factores. Primeiro, os sistemas de gestão de fluxos de trabalho necessitam que os utilizadores se liguem a eles. Mas somente na década de 1990 a conectividade por rede de computadores foi generalizada. Segundo, muitos sistemas informáticos que suportavam as operações das organizações não eram orientados aos processos, pelo que a tecnologia de fluxos de trabalho não era vista como uma nova peça de funcionalidade. Por terceiro, a rigidez e o carácter inflexível dos primeiros produtos afastaram muitos dos potenciais clientes.

[RauschScho97] apresenta mais três motivos para justificar o desenvolvimento desta tecnologia somente na década de 90: melhor e mais hardware – hardware com mais funcionalidades e mais acessível para as pequenas e médias empresas; interfaces gráficas para o utilizador – o aparecimento de interfaces gráficas para o utilizador tornaram os sistemas mais amigáveis, e libertou as aplicações dessa tarefa; e bases de dados – os sistemas de gestão de fluxos de trabalho lidam com vários tipos de dados (dados dos processos, e dados sobre os processos), essa informação passou a ser entregue e gerida por sistemas de gestão de bases de dados, ficando garantida a sua consistência, e libertando as aplicações dessa tarefa.

Tal como já referido, hoje em dia, a tecnologia de fluxos de trabalho é uma tecnologia ubíqua aos vários tipos de organizações, existindo um extenso desenvolvimento científico e inúmeras implementações disponíveis [Sheth99].

2.1.1.2 Definições e conceitos básicos

Fluxo de trabalho e sistemas de gestão de fluxos de trabalho

Das várias definições que existem para fluxos de trabalho foi escolhida a seguinte por ser independente do domínio de aplicação.

> Um **fluxo de trabalho** descreve os aspectos operacionais de um procedimento de trabalho: a estrutura das tarefas assim como as aplicações e os humanos que as desempenham; a ordem da execução das tarefas; a sincronização das tarefas; o fluxo de informação de suporte às tarefas; e mecanismos para o rastreio e execução de relatórios que meçam e controlem a execução das tarefas [RauschScho97].

A literatura da tecnologia de fluxos de trabalho refere-se ao sistema informático que permite definir, executar, e supervisionar os fluxos de trabalho como o Sistema de Gestão de Fluxos de Trabalho (SGFT).

> Um **Sistema de Gestão de Fluxos de Trabalho (SGFT)** é um sistema que permite definir, executar e gerir fluxos de trabalho descritos de forma computorizada [Jablonski97].

Um sistema de gestão de fluxos de trabalho automatiza a lógica dos processos organizacionais, sendo estes modelados como processos de fluxos de trabalho que contêm tarefas interligadas. Os humanos e as aplicações externas executam essas tarefas implementando então a lógica das tarefas. Segundo [RauschScho97] a separação da lógica dos processos da lógica das tarefas, permite a alteração de uma parte sem afectar a outra, o que promove a reutilização do software. Pode-se, por exemplo, conceber versões alternativas de um processo, alterando a ordem de execução de certas tarefas, ou por alteração das aplicações externas utilizadas. Tal separação é conseguida pela utilização de uma camada do sistema informático designada de empacotadora ("*wrapper*"), e que visa adaptar as especificidades das aplicações externas às especificidades do gestor de fluxos de trabalho.

Definições de processo e instância de fluxo de trabalho

Um sistema de gestão de fluxos de trabalho executa e gere fluxos de trabalho, segundo a representação computorizada de cada fluxo de trabalho. A representação computorizada do fluxo de trabalho, assente num formalismo, descreve completamente um modelo de um fluxo de trabalho. Essa especificação do fluxo de trabalho é designada normalmente de processo [Jablonski97], esquema [Weske99] ou tipo [RauschScho97] de fluxo de trabalho. Segue-se a definição presente em [Jablonski97]:

> "A definição de um **processo** (de fluxo de trabalho) contém toda a informação necessária, para que possa ser executado pelo *software* de execução de fluxos de trabalho ("*workflow enactment software*"). Isto inclui a informação acerca de que actividades que o constituem, regras de navegação entre elas, tarefas que os utilizadores têm que executar, referências para as aplicações a serem chamadas, e definição de qualquer informação relevante para o fluxo de trabalho que necessite de ser referenciada."

Um processo de um fluxo de trabalho é portanto a definição formal de um fluxo de trabalho, ou seja de um processo organizacional. Nesta dissertação os termos processo e o fluxo de trabalho serão utilizados numa perspectiva equivalente, pois o termo processo refere-se ao fluxo de trabalho subjacente descrito de uma forma precisa e formal.

Quando o sistema de gestão de fluxos de trabalho inicia uma execução interna segundo um determinado processo de fluxo de trabalho, cria uma **instância do processo de fluxo de trabalho** em questão. O caso mais normal será haver dezenas, centenas ou milhares de instâncias de um processo.

> Uma **instância do processo de fluxo de trabalho** é um caso em execução de um processo de um fluxo de trabalho.

Actividades de um fluxo de trabalho

Segundo [RauschScho97] um processo de fluxo de trabalho consiste num conjunto de actividades, interligadas por um fluxo de controlo, e que são executadas por diferentes agentes.

> "Uma **actividade** define uma porção de trabalho a ser realizada por um agente, que pode ser uma aplicação ou uma pessoa." [RauschScho97]

2.1.1.3 Exemplo de um fluxo de trabalho

De forma a proporcionar uma imagem prévia de um fluxo de trabalho, é apresentado um exemplo simples originário da documentação de um sistema de gestão de fluxos de trabalho da IBM [IBM96] e consta na figura 1.

A figura 1 mostra uma versão simplificada do processo descrito, sob a forma de um grafo directo. Neste grafo, as actividades são representadas por nós, e a ordem de execução por arcos directos entre os nós. Uma descrição formal de fluxos de trabalho descritos por grafos directos pode ser analisada em [Weske99].

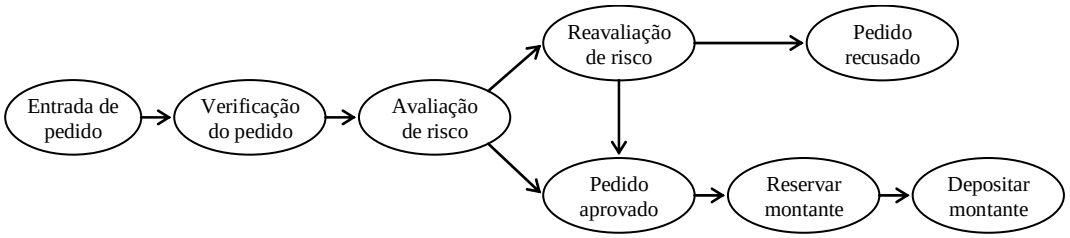


Figura 1 – Exemplo de fluxo de trabalho

O fluxo de trabalho tem a seguinte descrição informal: o fluxo inicia-se quando o cliente solicita um crédito ao banco, preenchendo um pedido de crédito e enviando-o para o banco. A informação do pedido de crédito é recebida pelo sistema de informação do banco, e depois de verificada a validade dos dados, segue para uma avaliação do risco na concessão do crédito. O pedido de crédito é então avaliado por um avaliador de créditos do banco, tendo como base de avaliação o montante pedido e a situação financeira do cliente. Se o avaliador de créditos decide conceder o crédito, é lançada uma série de actividades administrativas para reservar a verba pretendida, e para a depositar na conta do cliente. Se o crédito não for concedido, então um perito em créditos reavalia o caso, depois de obter informação adicional acerca da situação financeira do cliente. Dependendo do seu julgamento o crédito é eventualmente concedido ou definitivamente rejeitado. Se for concedido a mesma sequência de actividades administrativas mencionadas acima é realizada.

Enquanto esta descrição do processo simplifica consideravelmente a situação real, ela proporciona uma base de elucidação que será útil para futuras discussões nesta dissertação.

2.1.1.4 Classificação de fluxos de trabalho

Em [McCready92] classifica-se os fluxos de trabalho segundo a sua complexidade, podendo ser (do mais simples, para o mais complexo): administrativos, *ad hoc*, e de produção.

- Fluxos de trabalho administrativos: são fluxos repetitivos e previsíveis com regras simples de coordenação das actividades. O SGFT controla completamente a sua execução e notifica os utilizadores quando estes têm alguma tarefa para executar. Estes processos não são críticos para as organizações, pelo que a sua falha não é importante.

- Fluxos de trabalho *ad hoc*: estes fluxos tipicamente representam processos de escritório que não têm um padrão definido de resolução. A ordem das actividades não é geralmente controlada pelo SGFT, mas sim definida pelos utilizadores do SGFT. O que implica que são os próprios utilizadores que têm de determinar quando o processo termina e que têm de consultar o SGFT para saber o que existe para ser realizado. São processos que possuem actividades de pequena duração e que requerem uma solução rápida [Silver94]. Estes processos também não são críticos para as organizações.

- Fluxos de trabalho de produção: estes fluxos, tal como os fluxos administrativos, são repetitivos e previsíveis, mas como tipicamente interagem com múltiplos sistemas de informação tornam-se bastante mais complexos. Estes fluxos são críticos para organizações pois são os fluxos que acrescentam mais valias, sendo portanto a razão da existência da organização.

Esta dissertação vai focar essencialmente os processos, entre organizações, de natureza repetitiva, sendo portanto os fluxos de produção o seu foco principal.

Um termo hoje em dia bastante utilizado é o termo processo de negócio. Um processo de negócio é definido em [Hammer94] como "uma colecção de actividades que recebe uma ou mais entradas e cria uma saída que é de valor para o cliente". É portanto um processo que está directamente ligado à cadeia de valor, e logo pertence ao grupo dos processos de produção no contexto de organizações de âmbito empresarial.

2.1.2 Modelação, análise e execução de fluxos de trabalho

De acordo com [Jablonski95] o estudo dos sistemas de gestão de fluxos de trabalho divide-se nas seguintes áreas: modelação de fluxos de trabalho; análise de fluxos de trabalho; e execução de fluxos de trabalho.

2.1.2.1 Modelação de fluxos de trabalho

Esta secção visa apresentar os requisitos para a modelação de fluxos de trabalho. Para tal irá ser utilizado uma decomposição em várias perspectivas, de acordo com o modelo presente em [RauschScho97], o que permite a sua modularização. Pois segundo [Meyer88], a modularidade é um factor essencial para a qualidade de um produto de *software*, especialmente para se obter *software* correcto, eficiente, reutilizável, facilmente expansível, facilmente perceptível, e verificável. Este modelo contém as perspectivas:

funcional, operacional, comportamental, informativa, organizacional, causal, histórica, e transaccional. Sendo para cada perspectiva identificados os requisitos para a sua modelação.

Perspectiva funcional: fluxo de trabalho e actividades

A perspectiva funcional, de um processo de fluxo de trabalho, descreve o que nele tem de ser executado, ou seja, descreve as propriedades do próprio fluxo de trabalho, como um todo, e das actividades nele existentes.

As actividades podem ser directas ou processuais. Se forem processuais, resultam na chamada a outro fluxo (ou processo). Se forem directas, realizam uma operação indivisível, ou seja, atómica e não processual. As actividades directas correspondem à execução de por exemplo: uma aplicação externa; uma tarefa por um utilizador; ou uma transformação simples de dados dentro do processo. Além destes dois tipos de actividades pode ainda haver um outro tipo de actividades que contêm e controlam sub-fluxos de trabalho internos à própria actividade, como por exemplo uma actividade de sequência, que contém três actividades em execução sequencial.

Nesta perspectiva apenas é mencionado o nome do fluxo de trabalho e das actividades que o constituem.

Perspectiva operacional: aplicações

A perspectiva operacional descreve como as actividades são implementadas, ou seja, o que realiza as actividades. A descrição refere-se à concretização de o quê ou quem será o executor das actividades descritas na perspectiva funcional, ou seja, descreve a ligação entre as actividades lógicas descritas na perspectiva funcional e os seus executores reais.

Nesta perspectiva é descrita a ligação entre as actividades da perspectiva funcional e as aplicações externas, os fluxos de trabalho evocados, ou o gestor da execução humana.

Perspectiva comportamental: controlo do fluxo

A perspectiva comportamental descreve quando e quais as actividades são executadas, ou seja, descreve o controlo do fluxo e as condições de início do fluxo de trabalho. Até agora foram descritas as actividades que participam no fluxo, e quem as executa. Agora é definido a ordem pela qual poderão ser executadas.

A definição do controlo do fluxo pode ser concebida recorrendo a construções (actividades) de controlo de fluxo ou baseada em transições.

Na especificação por construções de controlo de fluxo, temos como construções básicas: a sequência (*seq*), a ramificação condicional (*if*), e a ramificação paralela (*par*). A construção de ramificação paralela ainda pode ser de finalização aquando da finalização de todos os seus subfluxos (*and par*), ou de finalização aquando do primeiro subfluxo (*or par*). Além das construções básicas, ainda outras poderão existir, ou serem definidas: execução cíclica (*while do*, *repeat until*, *for*), repetição paralela (execução em paralelo de várias instâncias de um subfluxo), execução opcional, execução em série (execução sequencial mas por uma ordem qualquer), limitação (um subfluxo não pode ser iniciado após o início de outro subfluxo, ou um subfluxo só pode iniciar a sua execução quando outro subfluxo estiver terminado), e execução com dependência existencial (um subfluxo só pode ser iniciado depois de um outro subfluxo ter iniciado a sua execução).

Na especificação de controlo do fluxo baseada em transições, a execução das actividades é activada pelo disparo de transições; em que uma transição pode disparar quando ocorrer um determinado evento e opcionalmente também se verifique uma determinada condição booleana. O disparo de uma transição resulta na transferência da execução de uma ou mais actividades (terminadas) para outra ou outras actividades.

Certos sistemas e suas linguagens permitem especificar condições de início em relação a um fluxo de trabalho através de uma condição booleana, de tal modo que o fluxo só será iniciado se tal condição for verdadeira.

Nesta perspectiva é descrito o controlo do fluxo das actividades que compõem o fluxo de trabalho, assim como as condições de início do fluxo de trabalho.

Perspectiva informativa: estruturas e fluxo de dados

A perspectiva informativa descreve que informação flui entre as actividades, assim como a informação de entrada e saída do fluxo de trabalho.

Assim, a primeira componente desta perspectiva é a descrição de que tipos de dados as actividades recebem como entrada e geram como saída. Outra componente consiste nas transformações ou manipulações de dados, de modo a extrair dados de várias fontes e permitir a realização de operações com eles. À transferência de dados de umas actividades para outras, designa-se de fluxo de dados.

Deverá ser possível suportar tipos de dados básicos e compostos, e ser possível reutilizar tipo compostos na definição de novos tipos compostos.

Nesta perspectiva são descritos, os tipos de dados utilizados, os dados de entrada e saída do fluxo de trabalho e das suas actividades, e as transferências e transformações de dados entre actividades.

Perspectiva organizacional: estrutura e regras

A perspectiva organizacional descreve a ligação entre as actividades, dos fluxos de trabalho, que são de intervenção humana e a organização. Essa ligação descreve quem pode e quem deve executar as actividades.

A afectação do processamento de uma actividade a uma pessoa é uma situação muito rígida, pois caso haja alguma alteração da sua condição normal de trabalho, tal como doença, férias, saída da organização ou mudança de funções, necessita da intervenção de um supervisor para realizar nova afectação a outra pessoa. De forma a flexibilizar o processo de afectação de actividades a pessoas, foi criado o conceito de papel. Este conceito define uma determina competência, o que permite associar pessoas a essa competência, e então estabelecer que uma determinada actividade deverá ser executada por pessoas de uma determinada competência. Assim, cada actividade que requeira a intervenção humana deve indicar que papel de utilizador que necessita para a sua execução. As pessoas por sua vez encontram-se registadas no SGFT, com a indicação de que papéis podem desempenhar. Quando o SGFT necessitar de activar o processamento de uma tarefa de intervenção humana determinará a pessoa a atribuir dentro das pessoas que pertençam ao papel indicado pela actividade a executar.

À tarefa que os SGFTs têm de determinar qual a pessoa a escolher para executar uma determinada actividade designa-se de resolução de papéis. A resolução de papéis pode ser directa, em que a escolha é realizada somente pelo papel em si, ou pode ser dependente dos dados, em que a escolha pode condicionalmente seleccionar o papel em função dos dados. Por exemplo, na figura 1, se o montante for superior a 1.000.000€, a primeira decisão passaria pelo perito que faria a avaliação prévia de risco, e depois passaria ao director do departamento de crédito em caso de reavaliação, que desempenharia o papel de perito neste cenário.

De forma a simplificar a gestão de papéis, existe o conceito de perfil, que representa um conjunto de papéis, ou competências.

Nesta perspectiva são descritos os papéis associados a cada actividade de intervenção humana. Mas para o SGFT conseguir interpretar essa informação necessita de: descrição dos papéis, descrição dos perfis, da associação das pessoas aos perfis ou aos papéis, e das regras de resolução de papéis.

Perspectiva causal: regulações e dependências

A perspectiva causal descreve porque um fluxo de trabalho é especificado de certa maneira e porque deve ser utilizado. Uma primeira categoria de causalidades descreve as razões porque o fluxo de trabalho é modelado. Deverá descrever as políticas da organização, ou as razões legais que levaram à modelação do processo de uma determinada maneira. Uma segunda categoria de causalidades descreve as dependências entre diferentes fluxos de trabalho. Por exemplo, descreve que se uma instância de um fluxo de trabalho for cancelada então uma outra instância de um outro fluxo de trabalho também deverá ser cancelada.

Nesta perspectiva define-se as causas da existência e da execução dos fluxos de trabalhos, assim como as interdependências causais entre fluxos de execução.

Perspectiva histórica: registo

A perspectiva histórica descreve que dados devem ser registados e em que pontos do fluxo de trabalho. O histórico pode conter informação acerca de todas as perspectivas de um fluxo de trabalho. A finalidade deste registo tem várias razões: primeira, de modo semelhante às bases de dados, o registo pode ser utilizado para, após uma falha, repor a execução num ponto consistente; segunda, pode haver necessidade de inferência sobre os fluxos de trabalho passados para os novos fluxos, por exemplo se um cliente voltar a entrar em contacto com a empresa, é-lhe dedicado a mesma pessoa envolvida no contacto anterior; terceira, para efectuar analises e relatórios, contabilísticos ou estatísticos, acerca dos processos, quer do ponto de vista financeiro, quer do ponto de vista de optimização dos processos; quarta, permite fazer auditorias aos registos para verificar a sua correcta execução.

Nesta perspectiva descreve-se o que deve passar para histórico, e quando tal deve acontecer.

Perspectiva transaccional: consistência

Num SGFT as transacções podem ser utilizadas para garantir a consistente execução dos fluxos de trabalho, quer para os fluxos como um todo, quer para as actividades neles envolvidas [RauschScho97].

Além da consistência ao nível dos dados, a consistência ao nível do fluxo de controlo, tem especial relevância nos SGFT. Nestes sistemas são requeridos conceitos transaccionais orientados aos sistemas e aos processos [RauschScho97]. Em que os primeiros tem como principal objectivo a recuperação dos dados operacionais, e os segundos têm como objectivo garantir a correcta execução do fluxo de trabalho dentro da organização, por exemplo: tem de ter em conta que certas actividades não podem ser anuladas, ou repetidas.

Nesta perspectiva define-se o modelo transaccional associado ao fluxo de trabalho.

2.1.2.2 Análise de fluxos de trabalho

A modelação permite especificar os fluxos de trabalho nas várias perspectivas já identificadas. Com base nessa especificação, pode realizar-se análises para verificar ineficiências ou deficiências de concepção. Em [Dinkhoff95] são identificadas três principais áreas de análise de fluxos de trabalho:

- Análise dos fluxos de trabalho – caso a descrição dos fluxos de trabalho seja efectuada segundo um formalismo que o permita, tal como as Redes de Petri [Peterson81], pode-se verificar se o fluxo de trabalho contém anomalias, como situações bloqueantes ("*deadlocks*"), actividades mortas ou terminação incorrecta [Aalst97] [Aalst98] [Adam98];

- Simulação – a simulação permite realizar testes sobre o comportamento do fluxo de trabalho sem consumir recursos reais, e com isso estudar pontos de optimização, e ter uma percepção mais real do fluxo de trabalho dentro da organização; e

- Interpretação dos dados de execução – analisando o histórico, pode-se obter informação que permita optimizar o processo. Por exemplo, verificando o grau de paralelismo assim como o nível de utilização dos recursos [RauschScho97].

À análise e optimização dos fluxos de trabalho, que no contexto empresarial implementam os processos de negócio, designa-se de Reengenharia de Processos de Negócio (*"Business Process Re-engineering"* – BPR).

2.1.2.3 Execução de fluxos de trabalho

Depois das fases de modelação e análise, segue-se a fase de execução. Nesta fase o fluxo de trabalho é colocado como disponível para execução no SGFT. Esta fase tem duas componentes: a execução e a sua supervisão.

A execução de um fluxo de trabalho corresponde a criar uma instância do fluxo de trabalho no SGFT, que pode ser: de forma automática, por recepção de um determinado evento; ou de forma manual, por um agente humano autorizado. Pelo que deverá haver uma descrição para cada processo de quais os eventos ou quais os perfis de agente que devem ou podem criar novas instâncias. Na preparação da instância de um novo fluxo de trabalho, o SGFT deverá passar-lhe a informação inicial e só depois dar início à sua execução. Da fase de execução ainda se salienta os tópicos de listas de trabalho, e tratamento de falhas, que seguidamente serão abordados.

Durante a execução de um processo, um SGFT ao processar uma actividade de execução humana, deverá sinalizar uma pessoa dentro do perfil desejado ou o grupo de pessoas dentro do perfil. Esta ligação entre o SGFT e os agentes (pessoas) geralmente é modelada por listas de trabalhos (*"work lists"*), havendo um componente do SGFT para suporte a essas listas de trabalho (*"work list handler"*). Cada lista de trabalho guardará informação acerca de tarefas a executar, ou em execução, por agente ou por grupo/perfil. A existência de listas por perfil, permite que uma qualquer pessoa com o perfil, possa iniciar o processamento da tarefa.

O SGFT deverá garantir a correcta execução do fluxo de trabalho mesmo em caso de falhas. As falhas geralmente têm duas origens: falhas de sistema ou falhas de aplicações. As falhas de sistema englobam as falhas do sistema operativo assim como as falhas do próprio SGFT, e são geralmente resolvidas pelo comportamento transaccional do SGFT, que normalmente se baseia nas funcionalidades transaccionais de um Sistema de Gestão de Bases de Dados, para repor o devido estado de execução. As falhas das aplicações invocadas pelas actividades, geralmente resultam em notificações, designadas de excepções, para o próprio fluxo de trabalho, e influenciam o resultado do próprio fluxo de trabalho. Assim, o fluxo de trabalho terá que ser reposto em algum ponto anterior de

execução que permita contornar o problema. Os SGFT utilizam mecanismos transaccionais estendidos [Elmagarmid92] [RauschScho97], para permitir o retorno a um ponto que permita a recuperação ("*rollback*") da boa execução do fluxo de trabalho.

A outra componente de execução, que é a supervisão, deverá acompanhar a execução e terá como função detectar e resolver problemas que nela possam ocorrer, como por exemplo: gerir as listas de trabalho e actualizar os perfis.

2.1.3 Arquitectura de um sistema de gestão de fluxos de trabalho

Segundo [Sheth99] em 1999 existiam entre duas a três centenas de SGFT. Estes sistemas não partilhavam uma terminologia comum, e muito menos arquitecturas, ou conceitos arquitecturais. Mas com o crescendo de importância destes sistemas dentro das organizações tornou-se imperativo a questão da interoperabilidade, principalmente a interoperabilidade entre SGFT. Este requisito levou em 1999 um grupo de produtores, utilizadores e investigadores de SGFT, a formar uma coligação denominada de Workflow Management Coalition (WfMC), com o objectivo de "desenvolvimento de terminologia e padrões comuns ("*standards*") para acelerar as oportunidades de exploração da tecnologia de fluxos de trabalho" [WfMC99].

Deste trabalho resultou, entre outras coisas, uma terminologia, uma estrutura genérica para um SGFT, e uma arquitectura de referência. A terminologia visou criar um glossário de termos e suas definições "usados para descrever os conceitos, a estrutura geral, os componentes funcionais e interfaces de um SGFT" [WfMC99].

A estrutura genérica de um SGFT corresponde à apresentada na figura 2 [Hollingswo95], e apresenta os principais elementos de um SGFT e suas relações.

A arquitectura de referência, proposta pelo WfMC, é designada de Modelo de Referência de Fluxos de Trabalho ("Workflow Reference Model") [Hollingswo95], e constitui a especificação de cinco interfaces conforme se pode observar na figura 3. Essas interfaces, designadas de "Workflow APIs and Interchange Formats" (WAPI), definem a interoperabilidade entre um executor de fluxos de trabalho e os módulos que a ele se podem acoplar, e são as seguintes: interface 1 – interface para as ferramentas de modelação de processos de fluxos de trabalho; interface 2 – interface para as aplicações que interagem com os utilizadores dos fluxos de trabalho; interface 3 – interface para o

acesso às aplicações externas; interface 4 – interface para com outros executores de fluxos de trabalho; e interface 5 – interface para as ferramentas de administração e supervisão dos fluxos de trabalho.

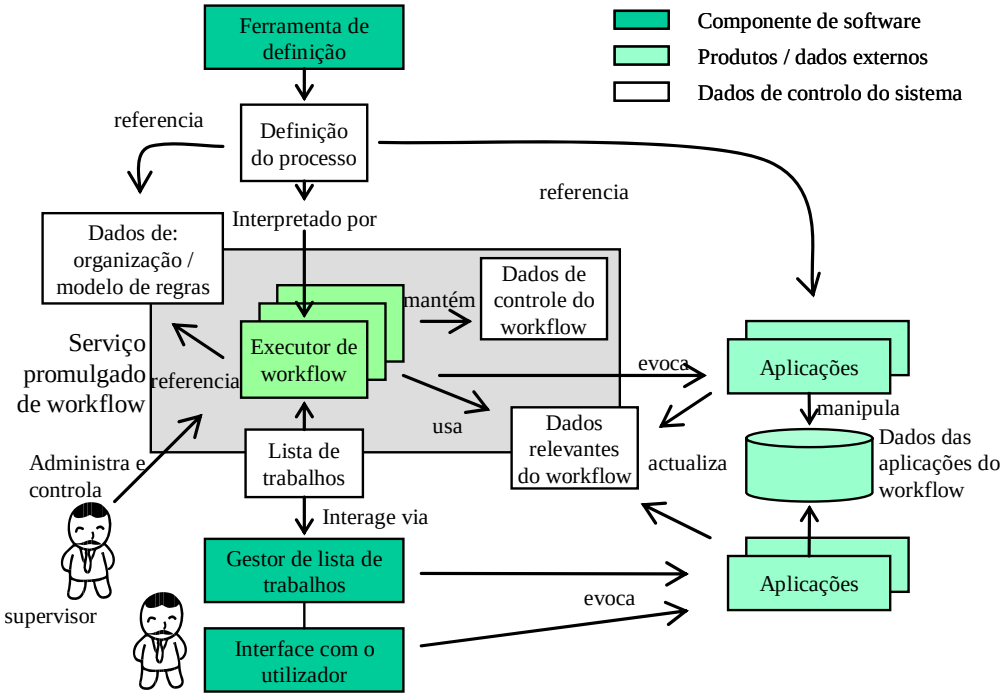


Figura 2 – Estrutura genérica de um SGFT, da WfMC

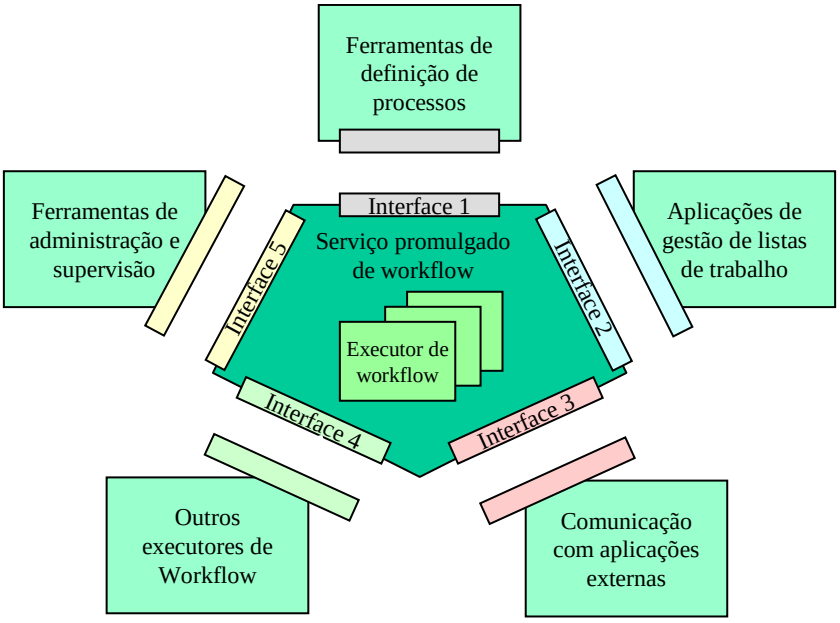


Figura 3 – Modelo de referência de um SGFT, da WfMC

2.2 Tecnologia de fluxos de trabalho interorganizacionais

Nesta secção será apresentada a tecnologia de fluxos de trabalho interorganizacionais. Primeiro será apresentado uma introdução clarificando o seu significado, depois serão apresentadas as suas principais características, seguido dos paradigmas de implementação, dos conceitos elementares, e finalizando com os aspectos relevantes relativos à sua modelação, análise e execução.

2.2.1 Introdução aos fluxos de trabalho interorganizacionais

Como referido na secção 2.1.1, os SGFT tiveram um grande desenvolvimento, na década de 1990, devido ao amadurecimento de várias tecnologias, sendo as principais: a comunicação por rede; as bases de dados; e as interfaces gráficas.

Destas tecnologias salienta-se a comunicação por rede, pois o desenvolvimento dos protocolos que formam a Internet [Cerf74], permitiu e permite, a comunicação a nível global ao planeta. Com base nestes protocolos foi desenvolvida a World Wide Web (Web) [BernersLee02], que possibilita a exposição da informação a nível global mas de uma forma visual e de mais fácil acesso.

Neste cenário de comunicação e exposição a nível global, a flexibilidade das organizações, face às variações do mercado, é um factor crucial para a sua sobrevivência, sobretudo se forem de âmbito empresarial, ou seja que dependam dos resultados da sua actividade. Como já referido, uma forma de aumentar a flexibilidade dentro de uma organização consiste em suportar os seus processos de forma explícita pela tecnologia de fluxos de trabalho.

Contudo, as empresas devido ao aumento de competitividade que esse mesmo mercado global proporciona, cada vez mais têm que se focar num nicho de oportunidade, e portanto acentuarem na especialização em detrimento da generalização. Tal factor torna as empresas muito dependentes do exterior, levando a que o seu negócio seja apenas uma parte de uma cadeia de negócios interligados.

De novo, e à semelhança com a passagem para uma modelação baseada em processos conforme já descrito no âmbito intra-organizacional, surgiu a necessidade de automatizar as interacções entre as organizações de modo a optimizar a utilização de

recursos, a permitir uma maior flexibilização, a facultar uma caracterização mais precisa das interacções, etc., etc.

Deste modo cada organização executa parte de um processo, em que este pode ter várias organizações como participantes. A este tipo de processos designa-se de processos de fluxo de trabalho interorganizacionais.

Um **fluxo de trabalho interorganizacional** (FTIO) é, portanto, um fluxo de trabalho que contém actividades que são executadas por várias organizações [Aalst01].

A tecnologia de fluxos de trabalho interorganizacionais encontra-se correntemente em franca expansão, pois é uma tecnologia que requer uma forte interoperabilidade aplicacional entre organizações, e este requisito só recentemente tem gerado os consensos necessários para uma interoperabilidade generalizada. A comunicação entre aplicações pela Internet, remonta à altura da formação da própria Internet, com a utilização dos seus Sockets. Mas este tipo de comunicação é demasiado livre, pois nem sequer força o tipo de mensagem trocada. Temos então a seguinte evolução, nos paradigmas de comunicação aplicacional:

- Chamada a Procedimentos Remotos ("Remote Procedure Call" – RPC)

  Este paradigma de comunicação estabelece as mensagens a trocar e o seu conteúdo. O modelo de evocação é o de funções com parâmetros de chamada e de retorno. Existe a noção de ligação estabelecida que permite realizar as evocações de funções. Temos como exemplos deste paradigma: RPC DCE, SUN-RPC.

- Chamada a Métodos Remota ("Remote Method Call" – RMC)

  Este paradigma de comunicação acrescenta a noção de objecto ao paradigma anterior. Temos portanto a noção de objecto remoto, sobre o qual são evocados métodos. São exemplos deste paradigma: Corba, DCOM, e RMI.

- Chamada a Serviços Remotos ("Remote Service Call" – RSC)

  Este paradigma de comunicação quebra com os paradigmas anteriores, na medida em que utiliza uma abordagem desligada entre as partes, ou seja, não existe uma ligação estabelecida entre eles. Sendo portanto para cada evocação estabelecida uma ligação própria. Este paradigma surgiu acompanhado de

outras facetas importantes para uma interoperabilidade mais generalizada tais como: utilização da linguagem XML (eXtensible Markup Language) [Bray98] para uma representação de dados independente da arquitectura dos computadores; utilização de espaços de nomes na própria linguagem XML, o que permite a diferenciação clara entre identificadores de proveniências diferentes; o desenvolvimento de uma pilha de protocolos descritos em XML, denominada de Serviços Web (*Web Services*) [Kreger01] de forma a especificar a descrição (WSDL [Chistensen01]), a comunicação (SOAP [Box00]), a publicação e a procura (UDDI [McKee01]) dos próprios serviços; e por fim a utilização generalizada destas tecnologias por parte dos maiores fabricantes de *Software*, que proporciona uma interoperabilidade sem as entropias inerentes ao uso de múltiplas alternativas tecnológicas equivalentes.

2.2.2 Principais características dos fluxos de trabalho interorganizacionais

Em [Bernauer02] são identificadas três características que distinguem os fluxos de trabalho interorganizacionais (FTIO) dos tradicionais fluxos de trabalho, designados agora de fluxos de trabalho intra-organizacionais, e que são: a interoperabilidade; a autonomia das organizações; e a abertura do ambiente.

Interoperabilidade

A interoperabilidade nos FTIO é então a capacidade destes fluxos serem devidamente interpretados e executados pelos seus vários intervenientes. Tal requer que haja um acordo entre todos os intervenientes, de modo a fixar as especificações dos vários componentes envolvidos no ciclo de vida dos FTIO, que vão desde os protocolos de comunicação, até à definição do formato e significado das mensagens trocadas [Wegner96], assim como dos valores dos campos das mesmas.

Assim sendo, a interoperabilidade é um pré-requisito crítico para os FTIO, pois somente com a garantia da sua existência se consegue a devida compreensão, aceitação e execução dos FTIO entre as várias organizações envolvidas.

Autonomia das organizações envolvidas

A autonomia das organizações, nos FTIO, consiste na potencialidade que os vários intervenientes deverão ter de modo a adaptar a sua parte do FTIO aos seus interesses,

mas preservando a interoperabilidade, ou seja, conservando os aspectos acordados com os outros parceiros.

Em [Bernauer02] são identificados vários tipos de autonomia em função dos estágios do ciclo de vida dos fluxos de trabalho:

- Autonomia de associação, em tempo de acordo ("*agreement time*") – a autonomia de associação consiste na potencialidade de negociação aquando da definição do fluxo de trabalho entre os vários intervenientes (organizações);

- Autonomia de desenho, em tempo de desenho ("d*esign time*") – a autonomia de desenho consiste na potencialidade de alterar o fluxo de trabalho, de cada um, contudo preservando a interoperabilidade global;

- Autonomia de comunicação, em tempo de execução ("*run time*") – a autonomia de comunicação consiste na potencialidade de escolha dos meios de comunicação de entre os definidos em tempo de acordo; e

- Autonomia de execução, em tempo de execução ("r*un time*") – a autonomia de execução define que cada organização é livre de suportar a execução da sua parte do FTIO do modo que entender.

Abertura do ambiente

O contexto de uma interacção entre várias organizações, obriga a requisitos que não predominam nos contextos dos fluxos de trabalho intra-organizacionais, tais como: legalidade, confiança, privacidade e segurança.

2.2.3 Paradigmas de implementação de fluxos de trabalho interorganizacionais

Segundo [Shen01] existem dois paradigmas para a implementação de FTIO, um baseado em actividades, e outro baseado em pontos de interacção.

2.2.3.1 Fluxos de trabalho interorganizacionais baseados em actividades

Neste paradigma a execução do fluxo de trabalho global é partilhada directamente pelas organizações participantes, ou seja, quando o processo global necessita que uma

actividade seja executada por uma organização, ele tem de lhe passar o controlo e o estado da execução. Neste paradigma existe portanto uma partilha do estado do processo onde cada interveniente pode monitorizar o progresso do mesmo [Shen01]. Para tal, são utilizadas interfaces dedicadas ao controlo do estado de execução, que descrevem o estado de execução das actividades, e que todos os parceiros devem disponibilizar. Como exemplos do uso deste paradigma temos os trabalhos presente em [Shen01], [Georgakopo99], [Muehlen00], [Schulz00] e [Chiu01]. A autonomia de cada interveniente reside ao nível da actividade, pois a sua implementação não é do domínio público. Este paradigma pode ser implementado recorrendo ao paradigma de execução de fluxos de trabalho distribuídos [Muth98].

2.2.3.2 Fluxos de trabalho interorganizacionais baseados em pontos de interacção

Este paradigma tem como premissa que a interface externa de um fluxo de trabalho participante de um FTIO deve ser somente a descrição dos seus pontos de interacção, ou seja, que mensagens são trocadas e a sua ordem.

Os FTIO são portanto descritos em termos globais, reunindo os aspectos de todos os participantes, numa descrição partilhada. Dessa descrição global, extrai-se os fluxos de trabalho que serão implementados por cada participante, e que são designados de fragmentos ("*workflow fragments*") [Lindert99], ou domínios [Aalst01], ou vistas ("*process view*") [Shen01]. Nesta dissertação será adoptada a designação de vista de um processo.

Como exemplos do trabalho de modelação de FTIO por pontos de interacção temos os trabalhos presente em [Casati01], [Lindert99], [Shen01] e [Aalst99].

A autonomia deste paradigma é maior que a do paradigma anterior, pois além da autonomia da implementação das actividades, também existe autonomia na execução e monitorização, uma vez que não há conhecimento do estado interno da execução do fluxo de trabalho de cada interveniente. Como veremos no capítulo seguinte este é o paradigma adoptado nas recentes linguagens de descrição de fluxos de trabalho para a Web, e é também o paradigma adoptado neste estudo.

2.2.4 Conceitos elementares de fluxos de trabalho interorganizacionais

Esta secção visa clarificar os seguintes conceitos base dos FTIO: conceito de fluxos de trabalho públicos e privados; conceito de fluxos executáveis e abstractos; e conceito de orquestração e coreografia de fluxos de trabalho.

2.2.4.1 Conceito de visibilidade de fluxo de trabalho: fluxo público e fluxo privado

Um **fluxo de trabalho interorganizacional** (FTIO) é um fluxo que contém os fluxos de trabalho dos vários participantes e as suas interdependências. Em termos de visibilidade este fluxo é público, pois o seu conteúdo foi acordado entre os vários participantes. O FTIO também pode ser designado de **fluxo de trabalho global público**, de forma a contrastar com os fluxos locais a cada participante. O FTIO pode ser descrito como a conjunção directa dos fluxos participantes, ou pode ser descrito utilizando apenas um fluxo unificador dos vários fluxos participantes.

Conforme já descrito, do fluxo global público extraí-se as vistas das várias organizações participantes. Assim, cada organização fica com a parte do fluxo interorganizacional que tem que implementar. Essa vista, ou esse fluxo, é designado por **fluxo de trabalho local público**, ou **vista local pública do FTIO**.

Como cada organização é autónoma para implementar a sua vista local pública, podendo recorrer a uma aplicação dedicada, ou de uma forma mais coerente recorrer a um fluxo de trabalho interno. Esse fluxo poderá ainda conter actividades extra desde que não altere a vista pública local do FTIO. A esse fluxo interno à organização, e que verifica as propriedades da sua vista pública, e eventualmente internamente alterado, designa-se de **fluxo de trabalho local privado**, pois a sua visibilidade é restrita à organização.

Assim, em termos genéricos temos:

- **Um fluxo de trabalho público** é, segundo [Bernauer02], "uma definição partilhada entre as organizações colaborantes, que providencia um conhecimento comum do fluxo de trabalho interorganizacional". Como essa definição contém um

conhecimento partilhado do fluxo interorganizacional acordado, ela tem que ser preservada pelos vários participantes.

– **Um fluxo de trabalho privado** é, segundo [Bernauer02], "uma definição do completo fluxo de trabalho interno a uma organização, incluindo tanto as actividades públicas como as actividades privadas". É uma especificação do domínio privado, que implementa um ou mais fluxos públicos, e que pode conter actividades privadas à organização.

Na figura 4 encontra-se uma ilustração de um fluxo de trabalho interorganizacional com dois participantes, modelada com Redes de Petri. Esse fluxo global corresponde à junção dos dois fluxos públicos dos dois participantes. Cada participante tem, portanto, o seu fluxo público, que é apresentado no seu lado interno da figura, e tem o seu fluxo privado, que é apresentado no seu lado externo. Os fluxos privados contêm as actividades existentes nos fluxos públicos e são equivalentes a eles do ponto de vista externo.

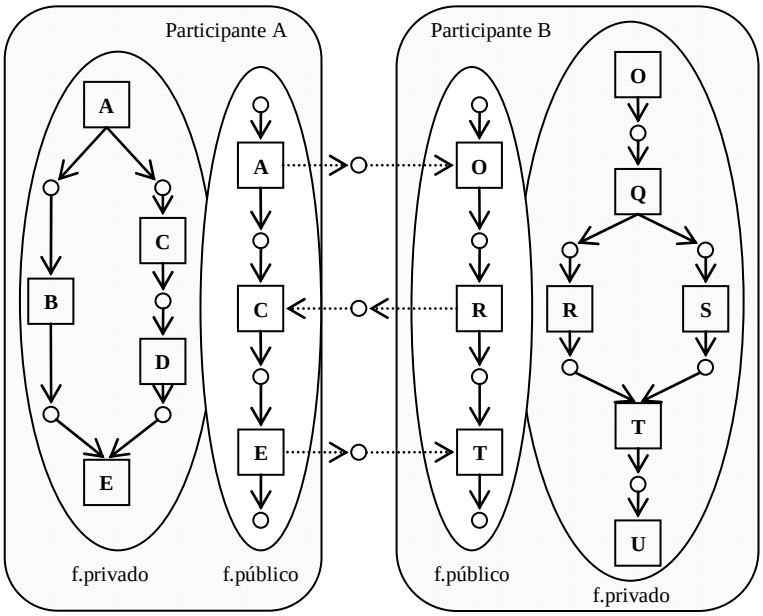


Figura 4 – Fluxo público e fluxo privado

2.2.4.2 Conceito de exequibilidade de um fluxo de trabalho: fluxo de trabalho executável e fluxo de trabalho abstracto

Um **fluxo de trabalho executável** é um fluxo de trabalho que pode ser executado por um Sistema de Gestão de fluxos de Trabalho. É um fluxo de trabalho que possui todos os elementos necessários à sua execução

Um **fluxo de trabalho abstracto** é um fluxo que não é executável, pois não contém toda a informação necessária para tal. Este tipo de fluxo destina-se a descrever as características principais de um fluxo de trabalho, sendo por isso também designado de **protocolo de negócio** [Andrews03]**.** É um fluxo que necessita de ser completado para ser transformado num fluxo executável.

De modo geral os fluxos de trabalho interorganizacionais são abstractos, pois na sua mínima expressão apenas têm de conter a descrição das interacções e o seu fluxo de controlo. Os fluxos de trabalho locais públicos, que resultam da extracção da componente de cada participante do FTIO, também são fluxos abstractos, pois nada acrescentam à definição global. Cada participante deve alterar a sua vista pública e abstracta transformando-a num fluxo executável.

2.2.4.3 Conceito associado à composição de fluxos de trabalho: orquestração de fluxos de trabalho e coreografia de fluxos de trabalho

Relativamente à composição de fluxos de trabalho, existem duas formas de os compor: em orquestração; e em coreografia. Vejamos então a definição destes termos.

Orquestração de fluxos de trabalho – este termo é utilizado quando um fluxo de trabalho interage com outros fluxos de trabalho, mas em que ele detém o controlo do desenrolar do cenário, ou seja, o desenvolvimento global depende fundamentalmente das suas decisões. Do ponto de vista das interacções interorganizacionais, uma orquestração é um cenário reduzido em que um participante assume um controlo exclusivo da evolução do contexto. Na figura 5 temos uma representação visual de uma orquestração de fluxos de trabalho, em que o fluxo central controla o desenrolar do cenário.

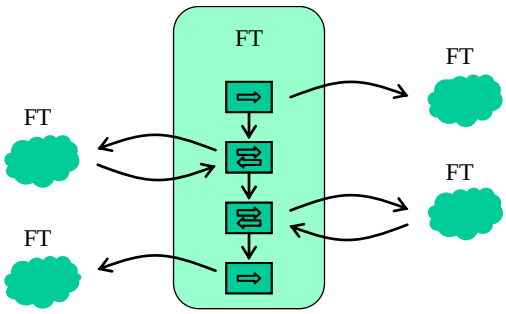


Figura 5 – Orquestração de fluxos de trabalho (FT)

Coreografia de fluxos de trabalho – este termo é utilizado quando vários fluxos de trabalho interagem entre si, formando um fluxo de trabalho global, mas em que o controlo se encontra disperso pelos vários participantes. Uma coreografia de fluxos de trabalho é portanto uma visão conjunta de um grupo de fluxos de trabalho interoperantes. Na figura 6 encontra-se uma coreografia envolvendo cinco fluxos de trabalho.

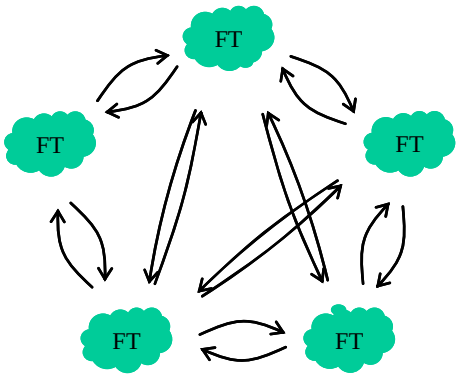


Figura 6 – Coreografia de fluxos de trabalho (FT)

2.2.5 Modelação, análise e execução de fluxos de trabalho interorganizacionais

À semelhança com o estudo realizado para os fluxos de trabalho intra-organizacionais, será agora apresentado os novos requisitos nas áreas da modelação, análise e execução dos fluxos de trabalho interorganizacionais.

2.2.5.1 Modelação de fluxos de trabalho interorganizacionais

Em [Bernauer02] é identificado um conjunto de requisitos gerais que a modelação de fluxos de trabalho interorganizacionais deve contemplar. Esse trabalho, por sua vez, é

baseado no modelo generalista para a especificação de fluxos de trabalho intra-organizacional apresentado em [RauschScho97] e descrito na secção 2.1.2. O estudo presente em [Bernauer02] restringe-se às perspectivas funcional, operacional, comportamental, informativa, interaccional, organizacional, e transaccional, excluindo as perspectivas causal e histórica, e acrescenta a perspectiva interaccional, de modo a lidar com o novo requisito que são as interacções entre organizações. Para cada perspectiva são identificados os novos requisitos necessários para a modelação dos FTIO, focando especialmente os aspectos relacionados com a interoperabilidade e autonomia.

Perspectiva funcional: fluxo de trabalho e actividades

A perspectiva funcional descreve o que tem de ser executado, ou seja, o fluxo de trabalho e suas actividades.

Os novos requisitos identificados em [Bernauer02] são:

- **Especificação do fluxo de trabalho interorganizacional** – deve ficar identificado quais os participantes envolvidos nas interacções e que actividades são desempenhadas por cada participante. Da descrição global deve ser possível extrair as vistas dos vários participantes.

- **Actividades públicas e privadas**[1] – deve ser possível descrever fluxos de trabalho públicos assim como privados. Assim, além da parte pública acordada deve-se permitir às organizações introduzir as suas actividades privadas.

Perspectiva operacional: implementação de actividades

A perspectiva operacional descreve como as actividades são implementadas, ou seja, o que realiza as actividades.

O único requisito identificado em [Bernauer02] é a **implementação de actividades**, que corresponde à especificação de como as actividades são implementadas. Este requisito permite separar a implementação das actividades nos fluxos públicos, salvaguardando a autonomia das organizações. É portanto um requisito do âmbito dos fluxos privados.

[1] No original encontra-se sob a designação de *Information Hiding*.

Perspectiva comportamental: fluxo de controlo

A perspectiva comportamental descreve quando e quais as actividades são executadas, ou seja, descreve o controlo da execução do fluxo. A especificação do controlo do fluxo é fundamental para os FTIO pois visa garantir a coerência entre os vários fluxos de trabalho participantes.

Os novos requisitos identificados em [Bernauer02] são:

- **Primitivas de controlo de fluxo** – tal como nos fluxos de trabalho intra-organizacionais, os FTIO necessitam de suportar as primitivas básicas de controlo de fluxo, por exemplo: execução sequencial, execução condicional, execução cíclica, e execução paralela. No entanto, surge a necessidade de suporte a novas primitivas relativas à decisão tardia. Este tipo de decisão, consiste na possibilidade de recepção de múltiplas mensagens, mas em que a primeira mensagem a ser recebida determina o prosseguimento do fluxo. É uma decisão em função do tipo de mensagem recebida, em vez de ser a tradicional decisão baseada no valor de uma expressão. A decisão tardia também tem a funcionalidade de após receber a primeira mensagem, inibir a possibilidade de recepção das outras mensagens.

- **Encaminhamento para as instâncias ("*instance routing*")** – uma vez que nos FTIOs a interacção ocorre entre instâncias de fluxos de trabalho, tem de haver um mecanismo de encaminhamento para a instância certa no participante certo.

- **Restrições temporais** – Como os FTIO são acordos entre organizações, deve ser permitido especificar limitações temporais, tais como a duração do fluxo total, a duração de grupos de actividades, ou a duração da execução de uma actividade. No entanto a existência destas limitações requer que haja um entendimento global do tempo entre os vários participantes.

- **Tratamento de excepções** – A execução de um fluxo de trabalho interorganizacional implica uma execução conjunta entre vários participantes. Cada participante poderá possuir uma componente privada do fluxo, mas terá obrigatoriamente que possuir a componente pública do fluxo acordado. A ocorrência de excepções pode provir da sua componente privada, podendo ser

excepções de carácter tecnológico (como a falha no acesso a uma aplicação) ou de carácter de negócio (como uma subempreitada que entrou em falha no tempo de execução), ou pode provir da sua componente pública, podendo ser indicações de falhas ocorridas noutros participantes, ou podem provir da própria infra-estrutura de execução do fluxo (por exemplo falhas de comunicação). Todos estes tipos de falhas devem estar contemplados nos fluxos de trabalho, devendo: as falhas da componente pública estarem contempladas no fluxo de trabalho interorganizacional; as falhas da componente privada deverão estar contempladas no fluxo privado; e as falhas da infra-estrutura dependendo da sua natureza devem ser consideradas parte do fluxo público ou do fluxo privado.

Perspectiva informativa: estruturas e fluxos de dados

A perspectiva informativa descreve que informação (dados) flúi entre as actividades, assim como a informação de entrada e saída dos fluxos de trabalho.

Os novos requisitos identificados em [Bernauer02] para os fluxos interorganizacionais são:

- **Tipos de dados** – como os fluxos de trabalho interorganizacionais lidam com conjuntos de dados complexos, torna-se imprescindível a definição de tipos de dados complexos. A semântica de cada campo deve ser clara para todos os participantes.

- **Bibliotecas de tipos de dados** – deve existir bibliotecas de tipo de dados, para que estes possam ser reutilizáveis. Esses tipos de dados poderão servir como blocos elementares para a construção de novos e mais ricos tipos de dados, ou utilizados em diferentes processos de negócio.

- **Fluxo de dados** – deve ser possível especificar que dados são criados ou acedidos por cada actividade (seja ela de uma actividade de interacção entre participantes, ou uma actividade do domínio público mas local a um participante), por cada fluxo de trabalho, ou por cada transformação de dados, quer do domínio público quer do privado.

Perspectiva interaccional: interacções

A perspectiva interaccional descreve como as interacções entre organizações são modeladas. Nos fluxos de trabalho interorganizacionais, as organizações necessitam de interagir directamente com outras organizações, pelo que surge a necessidade de especificar como essas interacções são suportadas.

Os novos requisitos identificados em [Bernauer02] são:

- **Primitivas de interacção** – deve ser possível especificar as primitivas de interacção nos FTIOs. Deve ser suportado as primitivas de envio num só sentido ("*one-way*"), e de pedido e resposta ("*request / response*").

- **Independência face à implementação** – as primitivas de interacção devem ser separadas da ligação a instanciações concretas de dados ou de protocolos, de modo a permitir a reutilização da definição do fluxo de trabalho em diferentes implementações.

- **Implementação das interacções** – a especificação das interacções deve conter a ligação dos elementos abstractos do ponto anterior para representações concretas.

Perspectiva organizacional

A perspectiva organizacional, nos fluxos de trabalho intra-organizacionais descreve a ligação entre as actividades dos fluxos de trabalho e as pessoas da organização. De modo a flexibilizar essa ligação utiliza-se a noção de papel. Nos fluxos de trabalho interorganizacionais esta perspectiva lida com as organizações que participam no fluxo. Novamente, para flexibilizar a relação entre os fluxos e instâncias de organizações deve-se suportar a noção de papel. Assim, esta perspectiva deverá descrever os papéis existentes em cada FTIO. As organizações, por seu lado, deverão poder descrever publicamente que papeis podem desempenhar, para eventualmente poderem ser escolhidas de forma automática, e poderem participar nesses fluxos.

Os novos requisitos identificados em [Bernauer02] são:

- **Papeis interorganizacionais** – De modo aos FTIO poderem ser reutilizáveis, deverão evitar as referências directas para as organizações que nele participam, mas basearem-se em papéis.

- **Perfis interorganizacionais** – As organizações devem ter perfis públicos, que indiquem quais os papéis que podem desempenhar. Tal informação deve ficar registada, e poder ser consultada, num serviço de registo, por exemplo numa implementação do UDDI (*Universal Description, Discovery & Integration registry*) [McKee01]

- **Participação dinâmica** – Alguns FTIOs requerem a selecção flexível, em tempo de execução, de quais as organizações que neles irão participar. Por exemplo: se o fornecedor principal não tiver certos produtos em stock, pode ser necessário seleccionar um outro fornecedor; ou então uma loja que envia para um serviço de entregas os dados do cliente, para que esse serviço interaja directamente com ele. Estas duas situações contêm dois processos diferentes de obter uma referência dinâmica: por consulta a um registo, e por recepção numa interacção.

Perspectiva transaccional

A perspectiva transaccional descreve a modelação do comportamento transaccional dos fluxos de trabalho. Os FTIOs requerem comportamentos transaccionais muito diferentes dos utilizados nos fluxos de trabalho intra-organizacionais. Em [Bernauer02] são identificados os dois tipos de comportamentos transaccionais, pois na modelação dos FTIO serão necessárias transacções entre organizações, e transacções dentro de organizações.

- **Transacções intra-organizacionais** – Na especificação de transacções dentro do fluxo de cada participante, podem ser utilizados modelos transaccionais, a aplicar sobre as actividades, sobre grupos de actividades, ou sobre o fluxo inteiro de cada participante, e que vão deste o modelo ACID, a modelos estendidos de transacções. Estes modelos estendidos relaxam os princípios, do primeiro modelo, tornando-se adequados para fluxos de trabalho de longa duração [Shet93];

- **Transacções interorganizacionais** – Para a especificação de transacções, que cruzam várias organizações, é necessário um modelo que seja centrado na autonomia dos participantes, devendo portanto, ser um modelo com uma semântica de fraco acoplamento [Bernauer02], tal como o modelo de transacções interoperáveis [Weigand98].

2.2.5.2 Análise de fluxos de trabalho interorganizacionais

Na secção análoga nos fluxos de trabalho intra-organizacionais, foram apresentados três aspectos desta área de estudo: análise de anomalias em tempo de desenho, simulação em tempo de pré-execução, e interpretação dos dados de execução, em tempo de pós-execução. Nos FTIOs são aplicáveis as mesmas análises. Contudo vai ser dado algum ênfase à análise de anomalias em tempo de desenho, pois é a análise mais crítica para o sucesso destes fluxos.

Novamente, caso a descrição dos fluxos de trabalho seja efectuada segundo um formalismo, tal como Redes de Petri, pode efectuar-se verificações se o fluxo de trabalho contém anomalias como situações bloqueantes (*"deadlocks"*), actividades mortas ou terminação incorrecta [Aalst97] [Aalst98] [Adam98], tal como foi indicado na secção análoga nos FT intra-organizacionais.

Nos fluxos de trabalho interorganizacionais o mesmo tipo de análise pode ser efectuado, no fluxo global público e nos fluxos locais públicos. Os fluxos locais privados também devem ser analisados para se verificar se eles são concordantes com o fluxo local público que devem desempenhar.

Em [Aalst01] é definida uma metodologia de concepção de FTIOs, designada de "Abordagem de Público para Privado de Fluxos de Trabalho Interorganizacionais" (*"Public to Private Approach to Interorganizational Workflows"*), que visa a extracção de fluxos locais públicos, a partir de um fluxo global público, e a concepção de fluxos privados partindo de fluxos públicos.

Seguidamente descreve-se resumidamente os três passos desta metodologia, tendo em conta que todos os fluxos, quer interorganizacionais, quer intra-organizacionais, são modelados como uma *Workflow Net* (*WF-net*) [Aalst98b]:

1. As organizações envolvidas acordam num fluxo de trabalho global público, ou seja, o fluxo de trabalho interorganizacional, que serve de contracto entre elas. Um FTIO é uma *WF-net* válida (*"sound"*), que não contém anomalias, como por exemplo: bloqueios ao fluxo (*"deadlocks"*); partes não executáveis (transições mortas); ou terminação incorrecta.

2. Do FTIO válido extraí-se as vistas de cada participante. Cada tarefa do fluxo de trabalho global público é mapeada na vista do participante correspondente. Cada

participante é responsável pela execução pela sua vista pública do FTIO. Partindo de um FTIO válido, é garantido que a extracção de vistas, neste trabalho designadas de domínios, gera *WF-nets* também válidas.

3. Cada participante fazendo uso da sua autonomia cria um fluxo de trabalho privado, que é concordante com a sua vista pública. São identificadas várias transformações possíveis de ser aplicadas aos fluxos locais públicos, de modo a que os fluxos resultantes continuem a verificar as propriedades do fluxo local público original, ou seja que preservam o comportamento dinâmico do fluxo original. A preservação do comportamento dinâmico é formalizada pela noção de herança por projecção [Basten98]. As regras de transformação identificadas são: inserção de um tarefa em série com outras; inserção de uma tarefa em paralelo com outras, inserção de tarefa em ciclo.

Com a utilização desta metodologia, e das suas regras, garante-se que o fluxo de trabalho interorganizacional é válido, ou seja, isento de anomalias, e que os fluxos privados resultantes também são igualmente válidos, e sobretudo concordantes com o fluxo interorganizacional global. Da continuação desse trabalho foi desenvolvido um analisador designado de *Woflan* ("*Workflow Analyzer*") [Hauschildt97], o qual permite verificar a validade ("*soundness*") de uma *WF-net* (interorganizacional ou intra-organizacional).

2.2.5.3 Execução de fluxos de trabalho interorganizacionais

A execução de um fluxo de trabalho interorganizacional, corresponde à execução de vários fluxos de trabalho intra-organizacionais interactuantes entre si. Esses fluxos requerem além dos requisitos identificados para os fluxos intra-organizacionais, na secção 2.1.2.3, também os novos requisitos resultantes da interacção com outras organizações.

Em [Pargfriede02] são identificados os seguinte requisitos para a fase de tempo de execução ("*run-time*"): selecção automática do serviço óptimo, assegurar a qualidade do serviço fornecido, segurança, flexibilidade, desempenho, expansibilidade, correcção, fiabilidade, estabilidade, disponibilidade, e persistência.

2.3 Conclusão

Neste capítulo foi, primeiro, apresentada a tecnologia de fluxos de trabalho, na perspectiva tradicional de um controlo centralizado e intra-organizacional. Neste contexto foram apresentados os conceitos base e expostas as fases de modelação, análise e execução. Sobre a modelação foi apresentado um conjunto de perspectivas dedicadas ao estudo dos vários aspectos da descrição dos fluxos de trabalho. O objectivo foi estabelecer os conceitos e requisitos base dos fluxos de trabalho intra-organizacionais, que serão necessários para o estudo dos fluxos de trabalho interorganizacionais.

Na segunda parte deste capítulo foi apresentado o conceito de fluxo de trabalho interorganizacional e analisadas as características destes fluxos. São abordados os conceitos de fluxos privados e públicos, fluxos executáveis e abstractos, e de orquestração e coreografia de fluxos de trabalho. São descritas, para as várias perspectivas de modelação, as novas exigências dos fluxos de trabalho interorganizacionais. São salientadas as diferenças na análise e execução dos fluxos interorganizacionais face aos fluxos intra-organizacionais.

Capítulo 3

Suporte à Descrição de Fluxos de Trabalho Interorganizacionais

Este capítulo tem por objectivo descrever as possibilidades actuais de descrição de fluxos de trabalho interorganizacionais. Dada a relevância da interoperabilidade nos fluxos de trabalho interorganizacionais este estudo vai focar-se nas tecnologias para a Internet, nomeadamente nas tecnologias baseadas nos Serviços Web. Primeiro será analisado as linguagens existentes e depois os protocolos de coordenação.

3.1 Tecnologias base para os fluxos de trabalho baseados na Internet

Segundo [Bray98] um Serviço Web é uma aplicação de software identificada por um URI (Identificador Uniforme de Recursos – Uniform Resource Identifier), cujas interfaces e ligações ("*bindings*") são capazes de serem definidas, descritas e descobertas, por artefactos XML e que suportam interacções directas com outras aplicações de software pela utilização de protocolos baseados na Internet e mensagens XML.

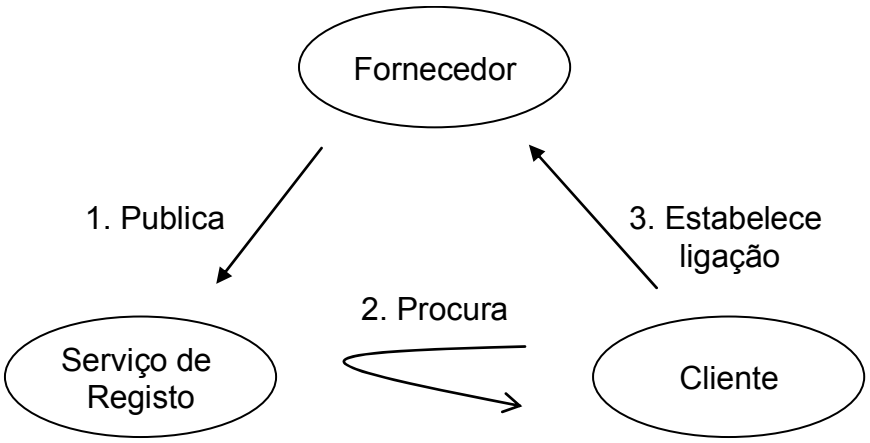


Figura 7 – A arquitectura base dos Serviços Web

A arquitectura dos Serviços Web [Kreger01] é uma arquitectura:

- **Baseada em serviços**: as aplicações disponibilizam as suas funcionalidades sob a forma de serviços acessíveis pela Internet; seguindo o comportamento usual de: publicar; procurar; e ligar. Ou seja: os fornecedores de serviços publicam os serviços num serviço de registos; os clientes procuram pelos serviços desejados nos serviços de registos; os clientes depois de encontrarem os serviços pretendidos ligam-se a eles, efectuando pedidos. Esta visão pode ser observada na figura 7.

- **Baseada em especificações para a Interne**t: as evocações de serviços são baseadas em especificações abertas e normalizadas, o que permite uma interoperabilidade generalizada. Numa perspectiva de baixo para cima temos as seguintes especificações base: XML (Extensible Markup Language) como a linguagem genérica de descrição de conteúdos; o SOAP (Simple Object Access Protocol) [Box00], como o serviço de mensagens; o WSDL (Web Service Description Language) [Chistensen01] como a especificação da descrição do serviço; e o UDDI (Universal Description, Discovery and Integration) [McKee01] que especifica o serviço de registo de serviços, que faculta a publicação e descoberta de serviços. Na figura 8, pode-se observar este conjunto de especificações e suas funcionalidades, onde o XML é a linguagem de descrição utilizada. Nessa figura é referenciado também o serviço de transporte das mensagens pois a especificação SOAP não abrange esse nível. Neste serviço os protocolos utilizados podem ser, por exemplo: HTTP (Hyper Text Transfer Protocol); FTP (File Transfer Protocol); ou MIME (Multipurpose Internet Mail Extensions). Destas especificações, será somente descrito a especificação WSDL

por ser a especificação base, normalmente utilizada, para as linguagens de modelação de fluxos de trabalho baseadas na Internet.

Especificação	Funcionalidade
UDDI	Publicação e descoberta de serviços
WSDL	Descrição de serviços
SOAP	Serviço de mensagens
HTTP, FTP, ...	Serviço de transporte na rede

Figura 8 – A pilha de especificações dos Serviços Web

3.1.1 Web Service Description Language – WSDL

A linguagem de descrição de Serviços Web WSDL (Web Service Description Language) [Chistensen01] é uma gramática XML, para descrever serviços acessíveis pela Internet. Essa linguagem pode descrever para cada serviço: as suas operações; as suas excepções (falhas); o formato das mensagens envolvidas; e a forma de acesso ao serviço. A WSDL é independente do protocolo de troca de mensagens a utilizar, pelo que descreve o serviço em termos abstractos. Contudo, também permite ligar ("*binding*") as declarações abstractas a protocolos e formatos reais. A sua especificação contempla a ligação ("*binding"*) para com os protocolos SOAP, HTTP e MIME Os elementos principais da WSDL são:

Definições abstractas:

- **Message – mensagem**: uma mensagem é um elemento (ou tipo) XML composto por partes (*part*), e constitui o elemento de transporte de dados num Serviço Web. As partes são definidas recorrendo a sistemas de tipos abstractos. Estes tipos são abstractos porque não correspondem a uma representação concreta a ser utilizada nas comunicações. Por esse facto, as mensagens também são designadas de abstractas.

- **Operation – operação:** Uma operação é a definição de uma interacção de um Serviço Web. As interacções podem ser num só sentido, como acontece numa notificação, ou nos dois sentidos, como acontece num pedido com resposta, e

podem devolver opcionalmente uma ou mais mensagens de erro denominadas de *faults*. As operações também são elementos abstractos pois não estão comprometidas com nenhuma especificação concreta de dados nem de protocolo de transporte.

– ***PortType* – tipo porto**: Um *portType* é um conjunto de operações (abstractas) referenciado por um nome.

Ligação a especificações concretas de formatos de mensagens e protocolos de troca de mensagens:

– ***Binding* – ligação:** Uma ligação define para um *portType* o formato das suas mensagens e os detalhes das suas operações para um protocolo de troca de mensagens. Cada *portType* pode ser ligado (*binding*) a vários protocolos.

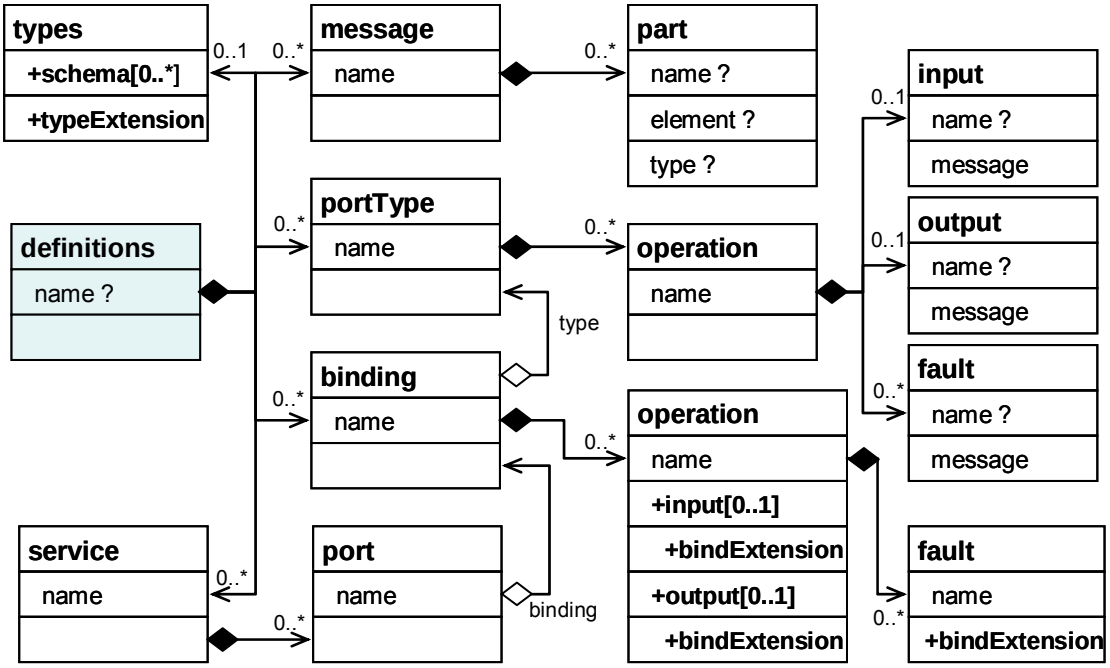


Figura 9 – Meta-modelo da linguagem WSDL

Definição de endereçamento para a Web:

– **Port – *porto*:** Um porto especifica um endereço (nome) para uma ligação (*binding*).

– **Service – serviço:** Um serviço é um grupo de portos (*ports*) relacionados entre si e identificados por um nome.

As relações entre estes elementos podem ser observadas no diagrama do meta-modelo da linguagem que se encontra presente na figura 9. Essa figura utiliza uma adaptação à notação UML, que é descrita no anexo A, de modo a reduzir o espaço ocupado pelos artefactos XML.

3.2 Linguagens de modelação de fluxos de trabalho interorganizacionais

O objectivo desta secção é realizar uma análise comparativa entre as várias linguagens existentes para suportar os fluxos de trabalho interorganizacionais.

As linguagens a comparar foram seleccionadas de acordo com o critério de serem linguagens de modelação de fluxos de trabalho, serem descritas em XML e baseadas na pilha de protocolos dos serviços Web; e de serem normalizadas ou estarem em vias de normalização. Assim, as linguagens seleccionadas são: XLANG, BPEL, WSFL, BPML, WSCL, WSCI, XPDL, WSCDL e BPSS. Estas linguagens serão apresentadas de seguida.

A comparação destas linguagens, ou parte delas, não é um estudo novo como se pode verificar pelos seguintes trabalhos:

- [Aalst02]: Comparação entre as linguagens BPEL, XLANG, WSFL, BPML, e WSCI nos tópicos de: padrões de troca de mensagens, e primitivas de controlo de fluxo.

- [Bernauer02]: Comparação entre as linguagens WSDL, WSFL, XLANG, BPML, WSCL, BPSS, e XPDL (WPDL), na óptica de suporte a fluxos de trabalho interorganizacionais, comparando vários aspectos das perspectivas enunciadas no capítulo anterior.

- [Shapiro02]: Comparação elemento a elemento da linguagem BPML face à linguagem BPEL e XPDL.

- [Wohed02]: Comparação entre as linguagens BPML e BPEL, nos tópicos de padrões de troca de mensagens, e primitivas de controlo de fluxo.

- [Mendling03]: Comparação entre as linguagens BPML e BPEL, na óptica de relacionar os conceitos existentes em cada linguagem de modo estabelecer relações entre conceitos equivalentes.

- [Mendling04]: Comparação entre as linguagens BPEL, BPML, BPSS, WSCDL, WSCI, WSCL, WSFL, XLANG, XPDL e outras, com o objectivo de desenvolver um meta-modelo completo, ou seja, que contemple os principais conceitos identificados entre todas as especificações.

- [Lippe05]: Comparação entre as linguagens BPSS, BPML, XPDL, BPEL, WSCDL e outras, na óptica de suporte aos requisitos gerais dos fluxos de trabalho interorganizacionais.

Contudo, dado que este trabalho se foca na forma como os fluxos interorganizacionais são descritos, será realizada uma comparação entre as linguagens apresentadas de modo a clarificar e proporcionar uma visão conjunta da forma como esses fluxos poderão ser descritos.

O objectivo da comparação das linguagens é, portanto, providenciar uma visão sobre a capacidade de cada linguagem de descrever os FTIO. Assim, cada linguagem será analisada sob o seu modo ou capacidade de descrever os processos interorganizacionais, sendo catalogada segundo três configurações identificadas:

- **Descrição local**: descrição que descreve o fluxo de um participante, sob a sua perspectiva.

- **Descrição de interligação**: descrição que complementa a descrição anterior acrescentando informação acerca da ligação explícita entre as operações dos fluxos dos participantes envolvidos.

- **Descrição comum**: descrição que descreve os FTIO através de um único fluxo onde cada interacção é descrita sob a forma de uma única actividade, indicando quem é o seu emissor e quem é o seu receptor.

Além do tipo de fluxo suportado, também é verificado se a linguagem suporta a noção de **estado global único** entre os vários participantes. A existência de estado único visa assegurar que todos os intervenientes têm a mesma percepção do desenvolvimento do processo global.

A comparação entre as linguagens apresentará primeiro uma breve introdução a cada linguagem, onde será apresentado o seu meta-modelo, de modo a permitir uma melhor visualização das capacidades da linguagem, sendo depois apresentada a análise acerca do suporte da linguagem às três configurações apresentadas.

3.2.1 Web Services Flow Language – WSFL

A linguagem WSFL (Web Services Flow Language) [Leymann01] foi proposta pela IBM, e teve como objectivo contribuir para uma futura norma ("*standard*") nesta área. O seu sucessor foi a linguagem BPEL [Andrews03].

Esta linguagem possui dois tipos de descrições, que são designados de modelos de fluxo e modelos globais.

Os **Modelos de fluxo** (*Flow Models*) visam descrever fluxos de trabalho que interagem com outros fluxos, sendo as interacções modeladas como Serviços Web. Pelo que um Modelo de Fluxo, corresponde então a uma orquestração de Serviços Web. O controle de fluxo é suportado por transições condicionais entre actividades, havendo a primitiva de *join* para unificar fluxos concorrentes. O fluxo de dados entre actividades é suportado por indicações denominadas de *DataLinks*. A implementação das actividades é especificada referindo uma descrição WSDL, que contém a localização da implementação do serviço, que pode ser interna à organização (*internal*), ou externa (*export*). Os serviços externos são localizados recorrendo a descritores denominados de *Locators*, que definem o tipo de localização utilizada, que pode ser: estática (*static*); local (*local*); de acesso a um registo UDDI (*uddi*); de ligação passiva (*any*); e de ligação dinâmica por afectação de dados (*mobility*). É uma linguagem vocacionada para a descrição de processos executáveis. Na figura 10 encontra-se a parte do modelo de fluxo do meta-modelo desta linguagem.

O outro tipo de descrição existente na linguagem WSFL é o designado de **Modelo Global** (*Global Model*). Esta descrição permite indicar ligações entre as operações de processos que sejam definidos por WSDL, como por exemplo os processos WSFL. O modelo global enuncia os participantes, e estabelece as ligações (*PlugLink*), dois a dois, entre as suas operações (*PortType*). Na figura 11 encontra-se descrita a parte do Modelo Global do meta-modelo da linguagem WSFL.

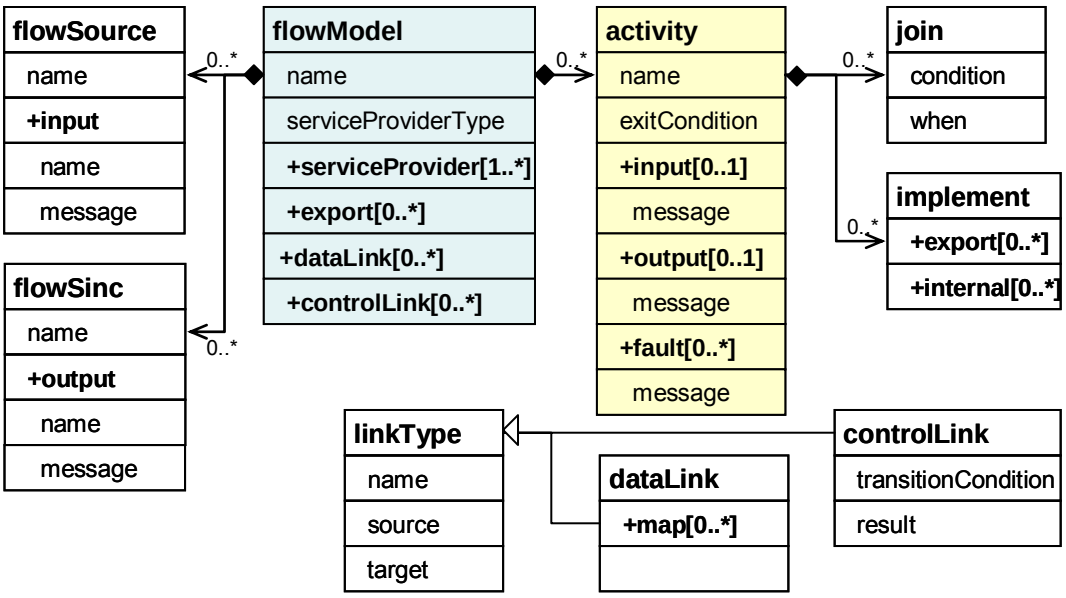


Figura 10 – Meta-modelo da linguagem WSFL, parte relativa ao processo

Como conclusão verifica-se que a linguagem WSFL: permite a Descrição Local através dos modelos de fluxo; permite a Descrição de interligação através do modelo global; não permite a Descrição Comum; e não suporta a noção de estado global único.

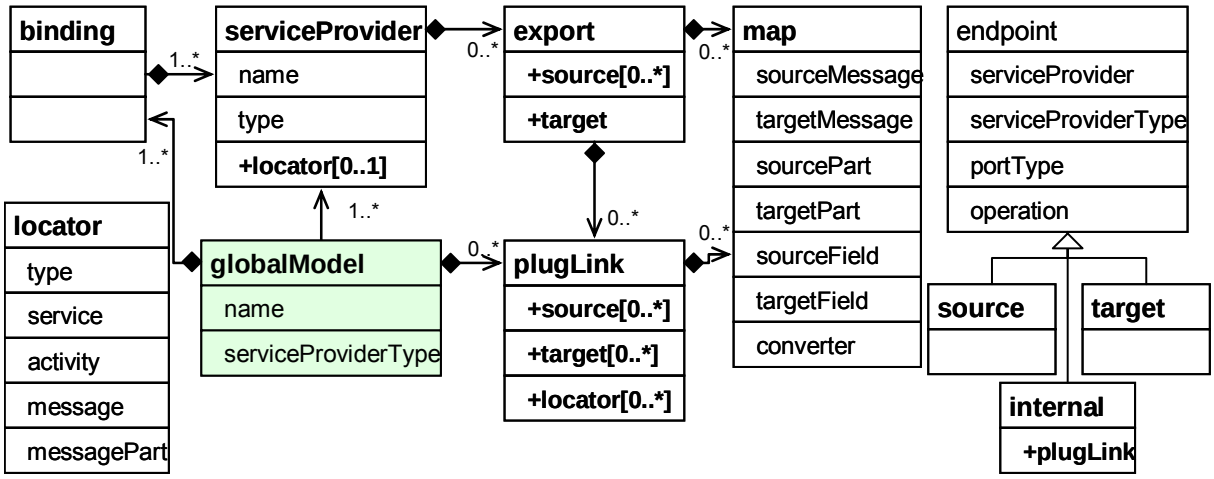


Figura 11 – Meta-modelo da linguagem WSFL, parte com a descrição global

3.2.2 XLang

A linguagem XLang [Thatte02] foi proposta pela Microsoft e tem por objectivo "permitir a especificação da componente pública de um fluxo de trabalho" [Thatte02].

Apesar da especificação ser direccionada para a descrição da componente pública esta linguagem é complementada com directivas proprietárias da Microsoft, no SGFT Biztalk, permitindo a sua utilização na descrição de fluxos privados.

A XLang, tal como o WSFL, estende a WSDL incorporando-lhe o comportamento do serviço, neste caso através do elemento *behavior*, ou seja, um processo XLang é definido como uma extensão (XLang:behavior) de uma especificação *service* da WSDL. Das suas características salienta-se: controle de fluxo estruturado ao bloco; suporte temporal relativo à recepção de mensagens; tratamento de excepções; transacções ACID; transacções abertas e encaixadas ("open nested transactions").

Na figura 12 encontra-se a parte do meta-modelo desta linguagem, descrevendo os elementos de *behavior* e *contract*. Na figura 13 encontra-se a parte do meta-modelo desta linguagem que descreve as várias actividades suportadas.

O elemento *contract* permite descrever de associações entre portos (WSDL) de dois participantes, permitindo clarificar as interligações em cenários com vários participantes.

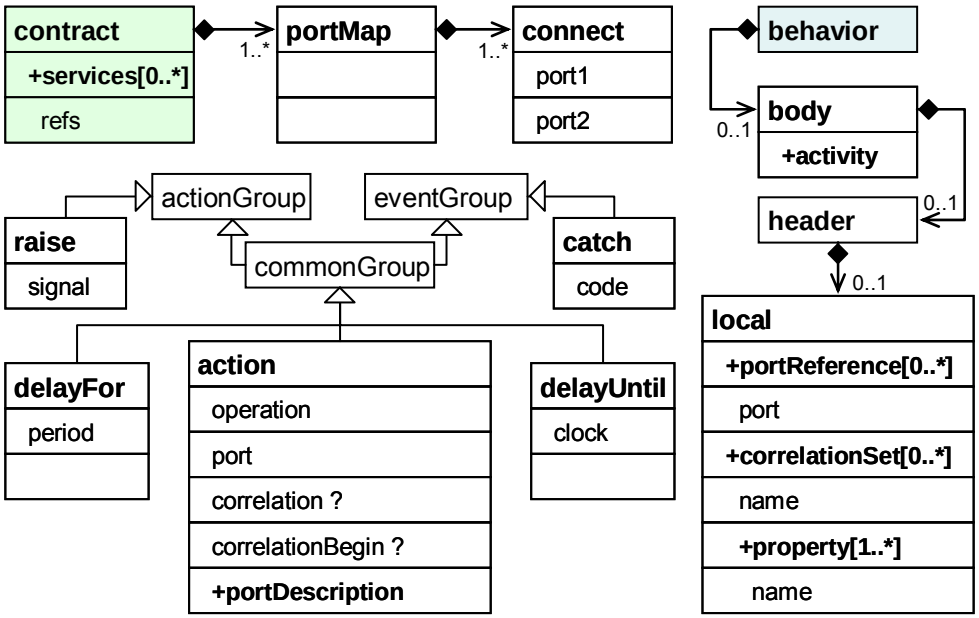


Figura 12 – Meta-modelo da linguagem Xlang, parte relativa ao processo

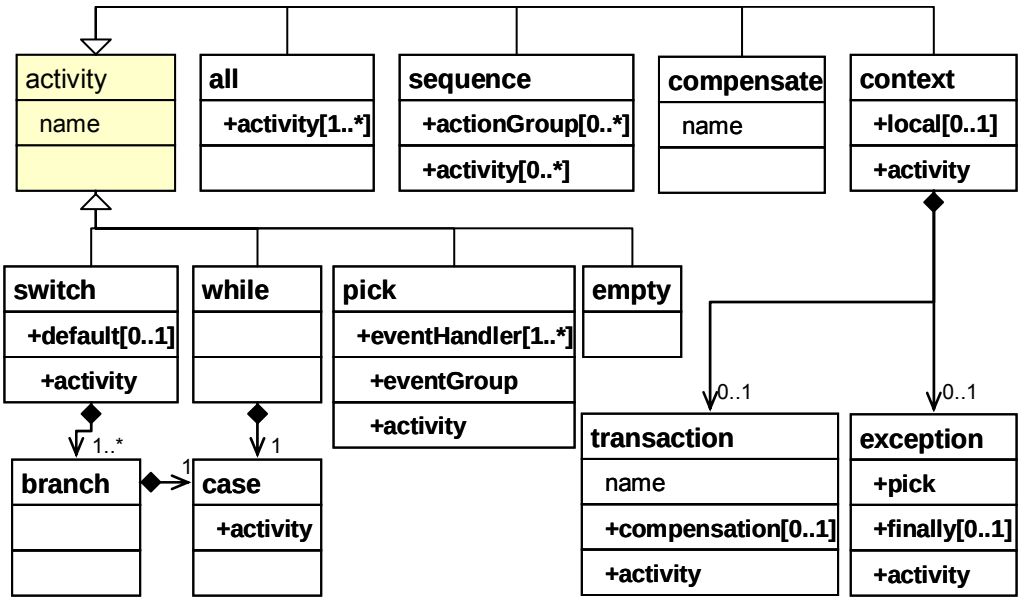


Figura 13 – Meta-modelo da linguagem Xlang, parte relativa às actividades

Assim, verifica-se que a linguagem XLang: permite a Descrição Local através do elemento *behavior*; permite a Descrição de Interligação através do elemento *contract*; não permite a Descrição Comum; e não suporta a noção de estado global único.

3.2.3 Web Services Conversation Language – WSCL

A linguagem WSCL (Web Services Conversation Language) [Banerji02] "permite definir as interfaces abstractas dos Serviços Web" [Banerji02]. Esta linguagem pode portanto descrever a parte pública de um processo.

É uma linguagem que define um processo como sendo apenas actividades (interacções) e transições entre elas, sendo apresentada na perspectiva de um participante. Tem a particularidade de cada interacção poder especificar várias mensagens em alternativa, de entrada assim como de saída, se bem que em tempo de execução somente será considerada a primeira a ocorrer. Não existe qualquer tratamento de temporizações, de excepções nem de falhas, e não há qualquer referência a comportamento transaccional. Em relação aos erros, estes são remetidos para a lógica do processo. As transições podem ter condições associadas, mas estas só podem utilizar o resultado da actividade anterior. Cada descrição só permite interagir com um outro participante. Caso seja pretendido um processo em que a organização interaja com dois parceiros, ter-se-ia que fazer duas conversações, e como não existe modo de definir as suas inter-relações entre essas duas descrições, o processo total não ficaria completamente definido.

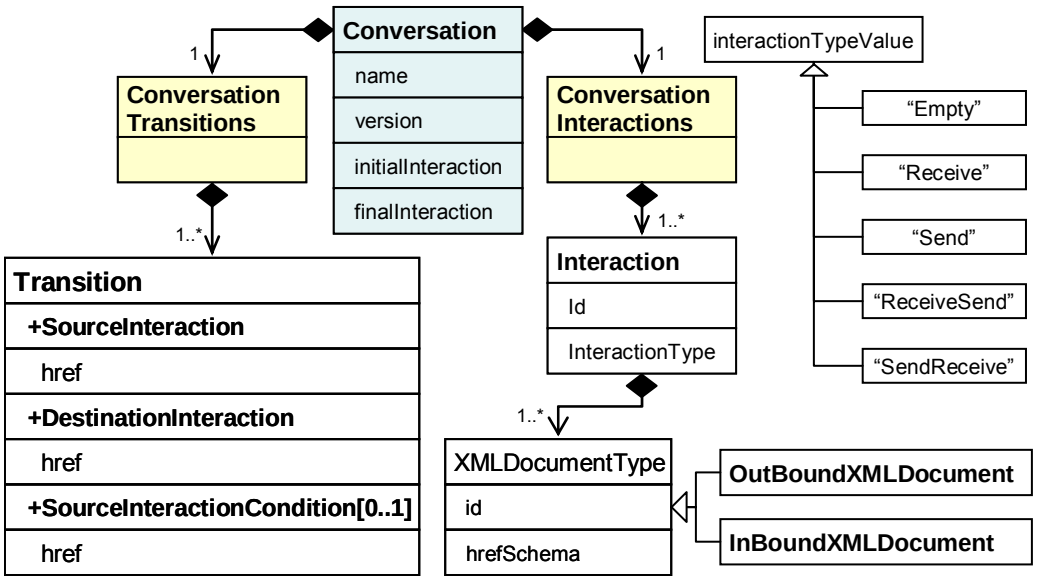


Figura 14 – Meta-modelo da linguagem WSCL

Assim verifica-se que a linguagem WSCL: permite uma versão limitada da Descrição Local; não permite a Descrição de Interligação; não permite a Descrição Comum; e não suporta a noção de estado global único.

3.2.4 Business Process Modeling Language – BPML

A linguagem BPML (Business Process Modeling Language) [Arkin01] "define um modelo formal para expressar processos abstractos e executáveis que contenham todos os aspectos dos processos de negócio empresariais, incluindo actividades de variada complexidade, transacções e sua compensação, manipulação de dados, concorrência, tratamento de excepções e semântica operacional" [Arkin01].

A BPML permite portanto especificar processos públicos, ou seja, abstractos, e processos privados, ou seja, executáveis. Nas especificações executáveis utiliza XPath [Clark99] para manipulação de dados, tais como, condições das transições, restrições temporais, e acesso geral a dados. Permite a composição de processos, ou seja, modelar parte de um processo, recorrendo a um outro processo. Possui tratamento de falhas, excepções, acontecimentos temporais, suporte a ligação dinâmica, e utilização de transacções atómicas, e transacções abertas e encaixadas por meio de blocos de compensação.

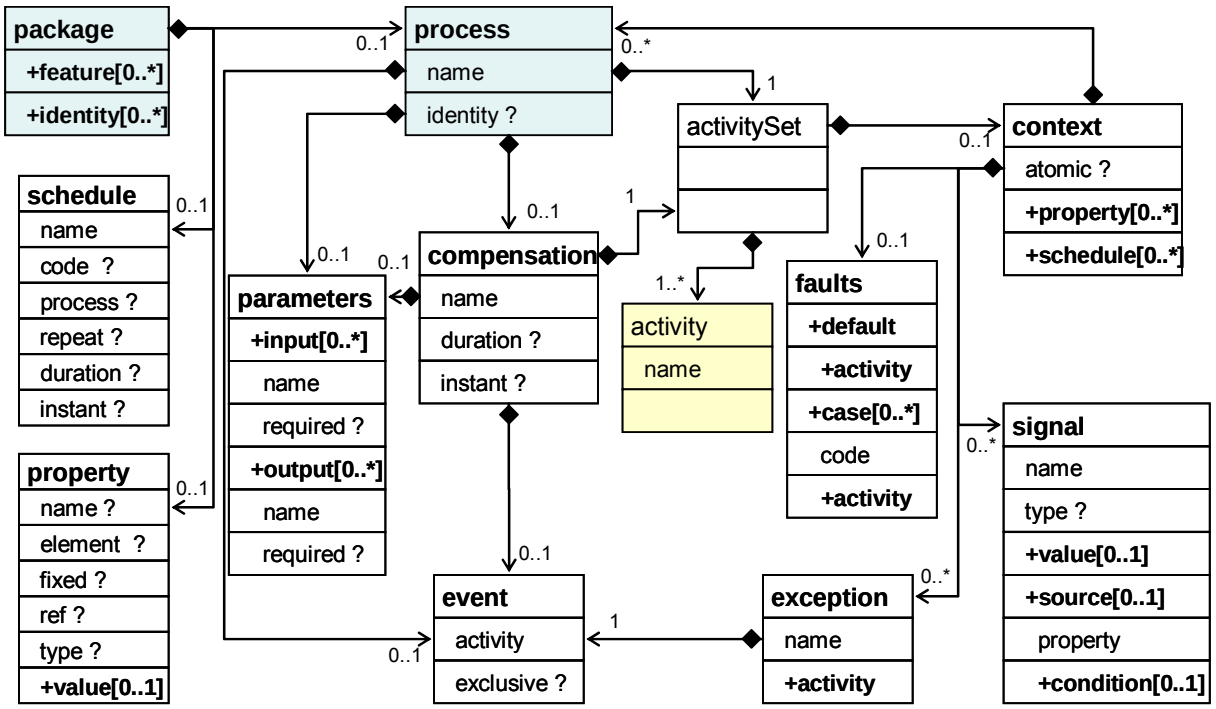


Figura 15 – Meta-modelo da linguagem BPML, parte relativa ao processo

Na figura 15 encontra-se o meta-modelo desta linguagem excluindo a parte que descreve as actividades existentes, a qual se encontra na figura 16.

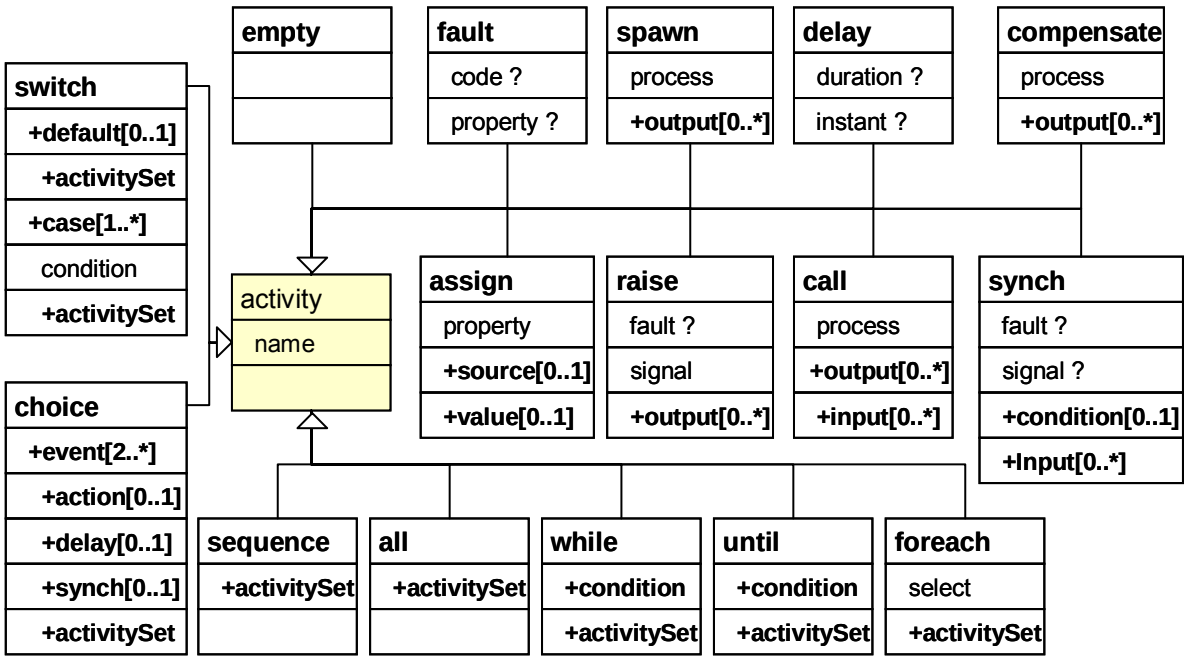


Figura 16 – Meta-modelo da linguagem BPML, parte relativa às actividades

A linguagem BPML: permite a Descrição Local; não permite a Descrição de Interligação; não permite a Descrição Comum; e não suporta a noção de estado global único.

3.2.5 Web Services Choreography Interface – WSCI

A linguagem WSCI (Web Services Choreography Interface) [Arkin02] "descreve o comportamento observável de um Serviço Web" [Arkin02]. A WSCI "também descreve a troca colectiva de mensagens entre Serviços Web, providenciando uma vista global e orientada às mensagens das interacções" [Arkin02].

Esta linguagem permite a descrição dos aspectos públicos de um conjunto de processos, onde cada processo é descrito sob a sua perspectiva. Através do modelo global (*model*) é permitido descrição das interligações entre operações de dois processos.

Na figura 17 encontra-se descrito a parte principal do meta-modelo desta linguagem e na figura 18 está descrito a parte das suas actividades.

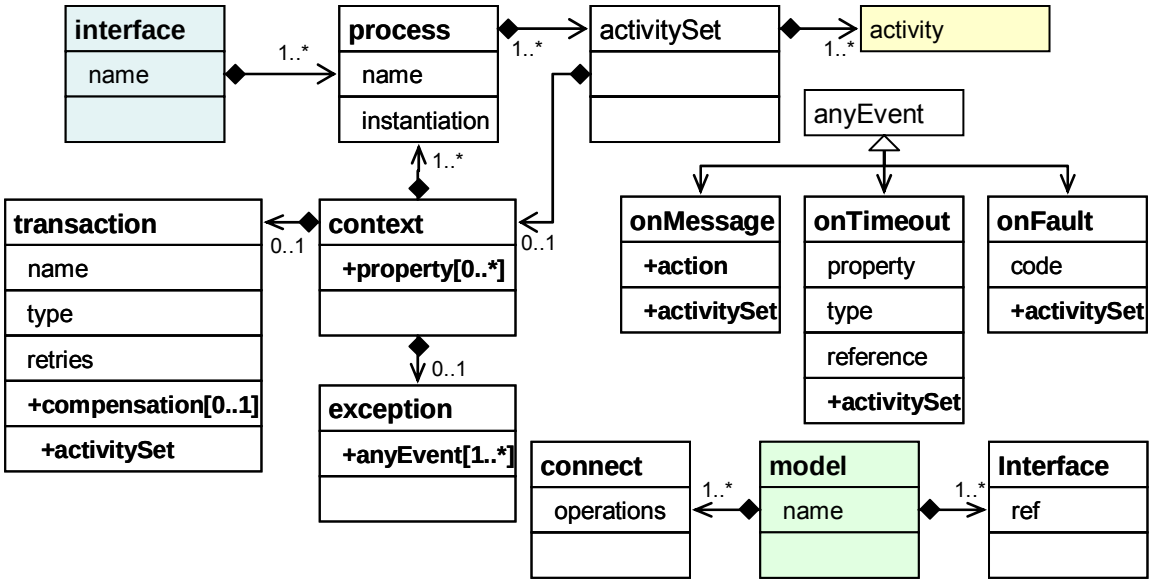


Figura 17 – Meta-modelo da linguagem WSCI, parte principal

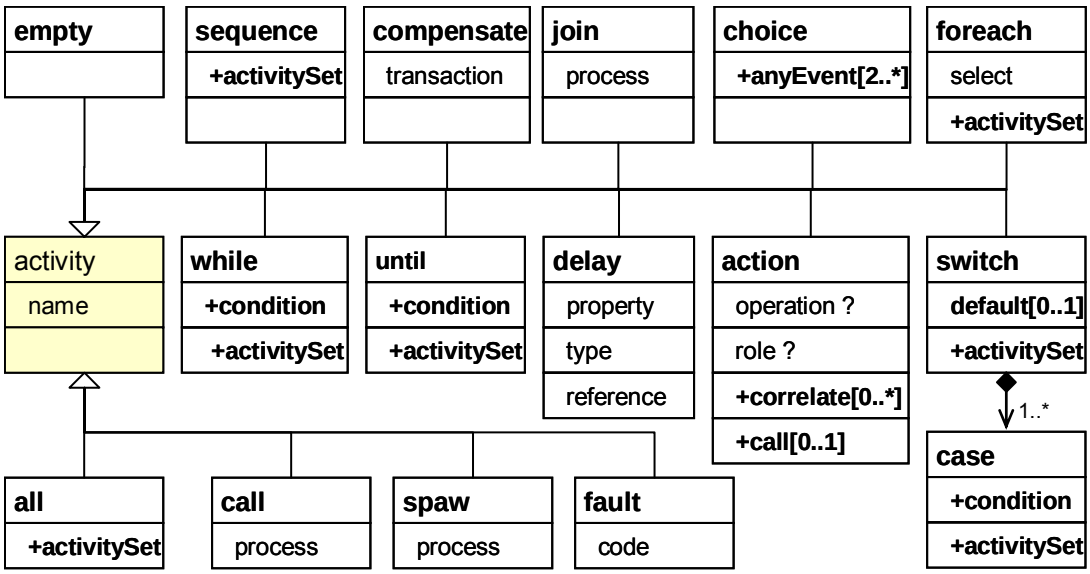


Figura 18 – Meta-modelo da linguagem WSCI, parte relativa às actividades

A linguagem WSCI: permite a Descrição Local; permite a Descrição de Interligação; não permite a Descrição Comum; e não suporta a noção de estado global único.

3.2.6 Business Process Execution Language for Web Services – BPEL

A linguagem BPEL (Business Process Execution Language for Web Services) [Andrews03] resulta da convergência das experiências das linguagens WSFL e XLang, e é uma linguagem dedicada a descrever processos na perspectiva de um participante. Os processos podem ser públicos ou privados, sendo designados respectivamente de protocolos de negócio ("*business protocols*") ou processos executáveis ("*executable processes*"). Cada um destes tipos de processos tem uma extensão dedicada, de modo a lidar com as suas particularidades.

Possui um controle de fluxo principalmente baseado em blocos, mas também permite definir o fluxo por transições, embora com limitações. Para os blocos permite: o lançamento e tratamento de excepções; eventos temporais; e transacções do tipo abertas e encaixadas com compensação.

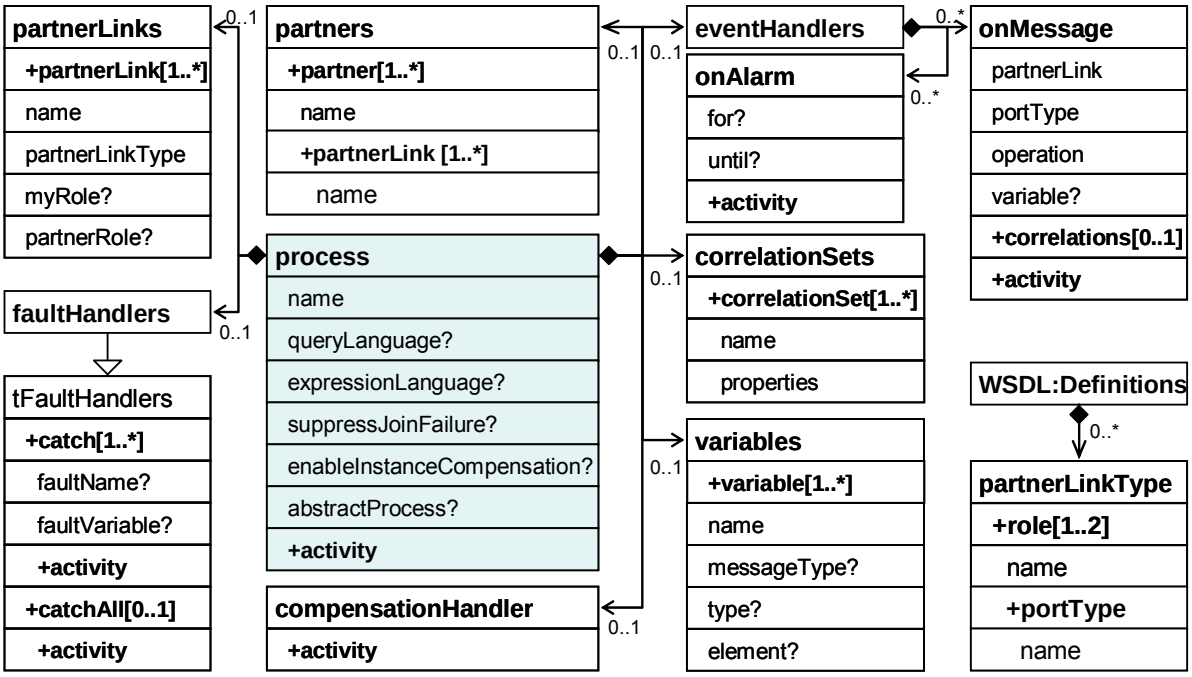


Figura 19 – Meta-modelo da linguagem BPEL, parte relativa ao processo e à WSDL

Na figura 19 encontra-se o diagrama que descreve a parte do meta-modelo da linguagem BPEL centrada no processo e também parte das extensões realizadas à WSDL. Na figura 20 encontra-se o diagrama com a continuação do meta-modelo, contendo a parte relativa às actividades.

A definição de *Definitions* na WSDL é estendida com elementos *partnerLinkType*, que permitem estabelecer uma ligação entre portos WSDL de diferentes participantes, e indicar que papel implementará cada um dos portos. Na definição do processo de cada participante, indicará o seu papel para cada *partnerLinkType* em que participa. Este elemento permite, pois, estabelecer uma ligação entre portos de diferentes participantes, embora colocando a noção intermédia de papel, no *partnerLinkType*.

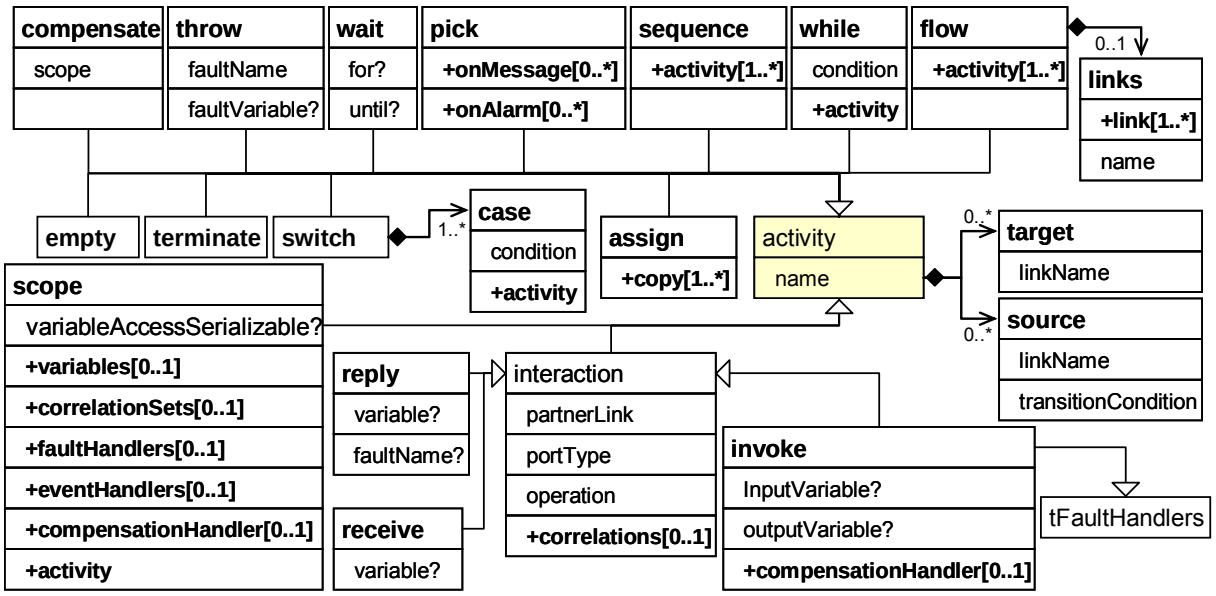


Figura 20 – Meta-modelo da linguagem BPEL, parte relativa às actividades

A linguagem BPEL: permite a Descrição Local; permite a Descrição de Interligação através de *partnerlinkType*; não permite a Descrição Comum; e não suporta a noção de estado global único.

3.2.7 XML Process Definition Language – XPDL

A linguagem XPDL (XML Process Definition Language) [Shapiro08], na sua versão 2.1a tem como objectivo "ser usada como um formato em ficheiro para a especificação BPMN 1.1" [Shapiro08]. A especificação BPMN (Business Process Modeling Notation) [OMG08] por sua vez tem por objectivo "providenciar uma notação gráfica para descrever processos de negócio num diagrama de processos de negócio" [OMG08].

Estas duas especificações permitem a descrição de processos executáveis, processos abstractos e coreografias de processos. As coreografias são descritas pela interligação de vários processos por meio de troca de mensagens.

Na linguagem BPMN é indicado o suporte a transacções entre organizações, no entanto apenas é indicado que tal pode ser representado pelo símbolo de *Group*. Na linguagem XPDL é indicado que uma actividade de sub-processo, marcada como uma transacção, necessita de um protocolo transaccional para garantir que todos os participantes chegam a uma só conclusão acerca do término do sub-processo. Mas indica que a notação e o comportamento exactos para definir este tipo de transacções é ainda

um aspecto em aberto. Pelo que se conclui que não existe um suporte efectivo a um comportamento transaccional entre vários participantes.

Na figura 21 encontra-se descrito a parte principal do meta-modelo desta linguagem e na figura 22 está descrito a parte das suas actividades. Como se pode observar nessas duas figuras a linguagem é bastante extensa, pelo que apenas foi descrito os elementos mais relevantes.

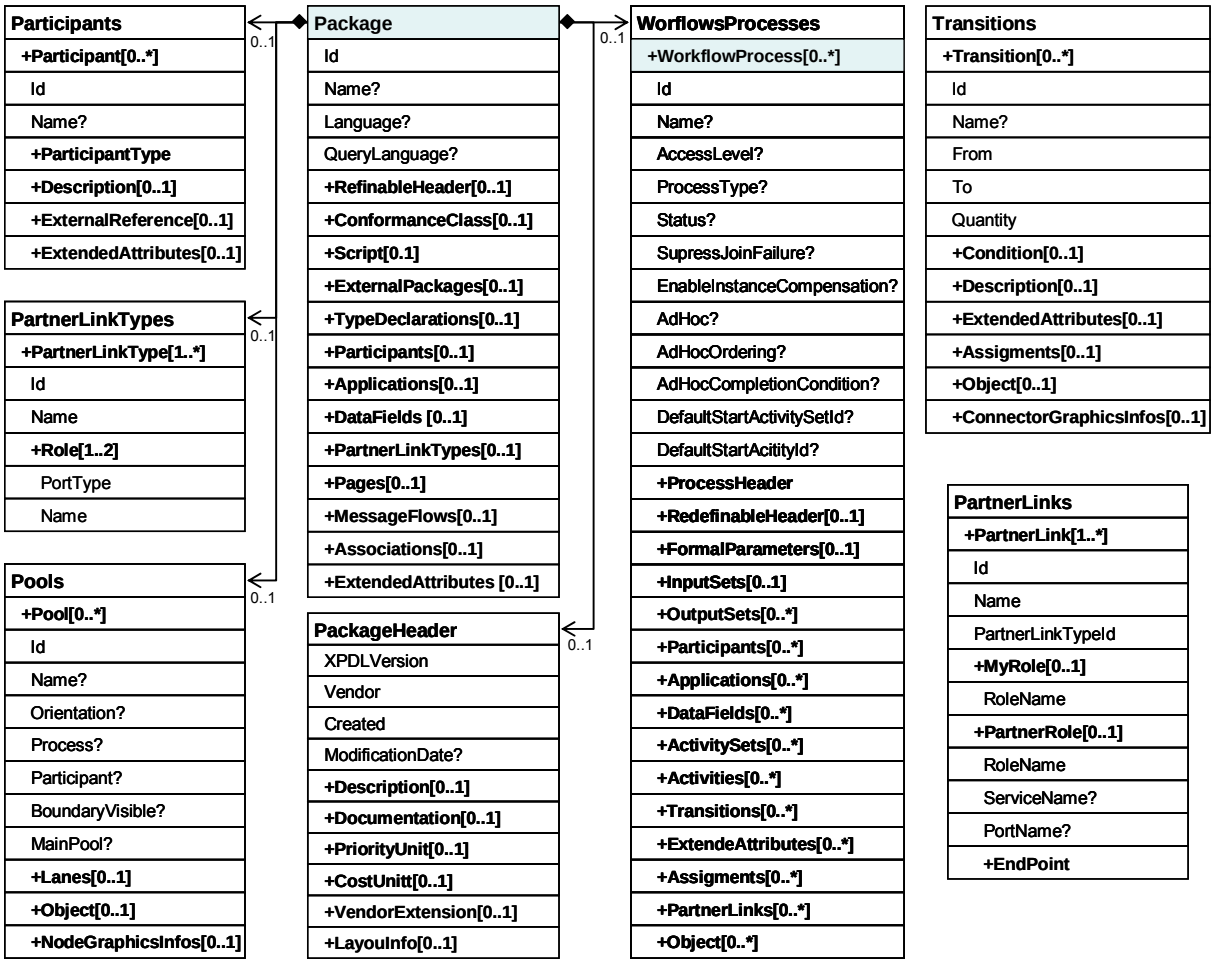


Figura 21 – Meta-modelo da linguagem XPDL, parte relativa ao processo

A linguagem XPDL: permite a Descrição Local; permite a Descrição de Interligação pois encontra-se presente nas trocas de mensagens entre os processos dos participantes; não permite a Descrição Comum, pois utiliza a descrição individualizada dos processos; e não suporta a noção de estado global síncrono entre os vários participantes.

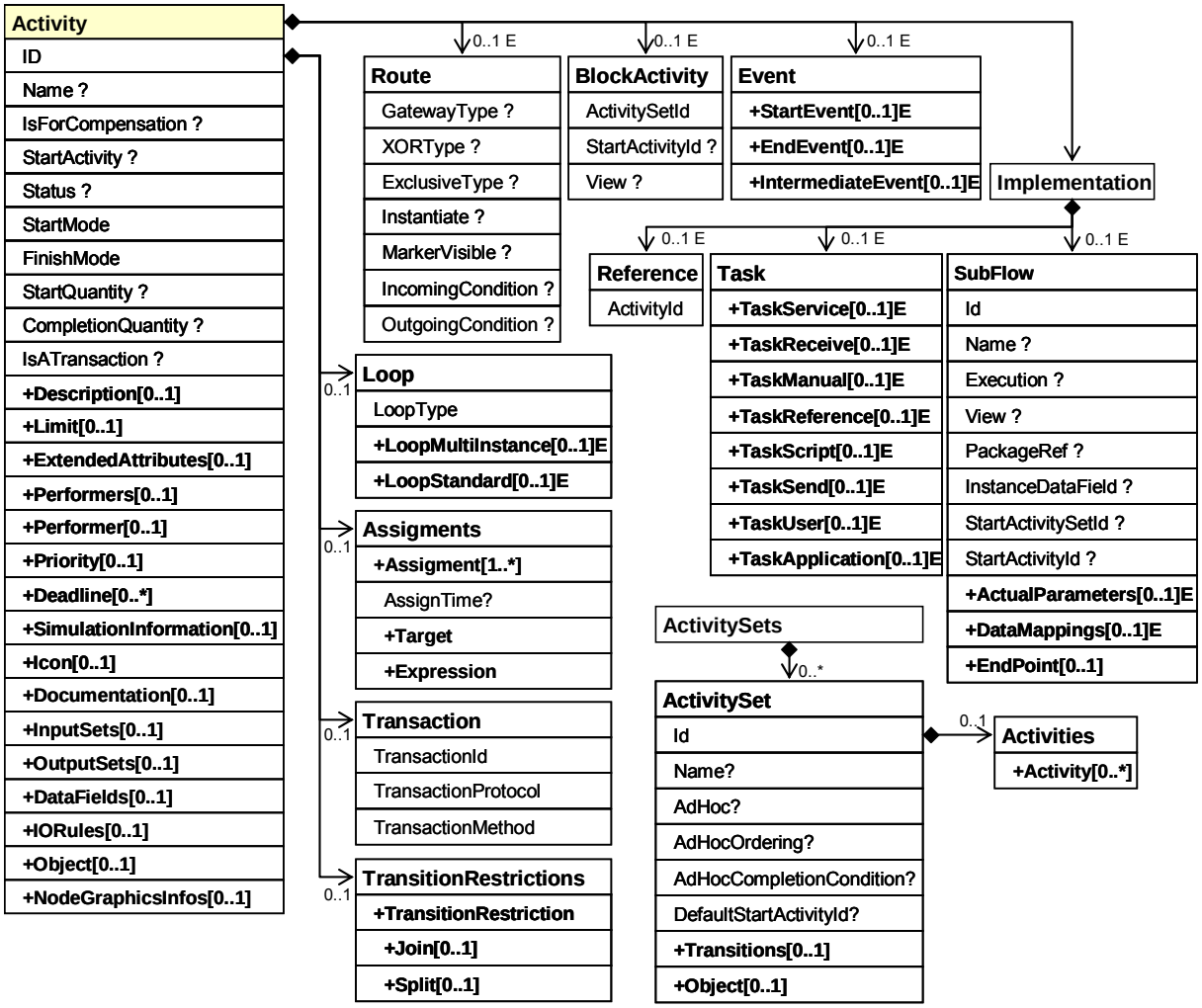


Figura 22 – Meta-modelo da linguagem XPDL, parte relativa às actividades

3.2.8 Web Services Choreography Description Language – WSCDL

A linguagem WSCDL (Web Services Choreography Description Language) [Kavantzas05] "…é uma linguagem baseada em XML, que descreve colaborações ponto-a-ponto modeladas como Serviços Web, definindo de um ponto de vista global o seu comportamento observável e complementar" [Kavantzas05]. Esta linguagem define o processo de um ponto de vista global às várias organizações participantes. Uma vez tendo a definição global do processo, pode extrair-se as definições dos processos das organizações participantes. "A maior vantagem de uma abordagem por definição global, é que ela separa o processo global a ser seguido por uma organização individual, da

definição da sequência segundo a qual ela trocará mensagens com as outras organizações. O que significa que desde que o comportamento observável não varie, a lógica e as regras seguidas num participante podem variar, pois a interoperabilidade está garantida" [Kavantzas05].

Na figura 23 encontra-se o diagrama que descreve a parte do meta-modelo da linguagem WSCDL centrada no seu elemento inicial denominado de *package*. Na figura 24 encontra-se outro diagrama com a continuação do meta-modelo, contendo a parte centrada nas actividades.

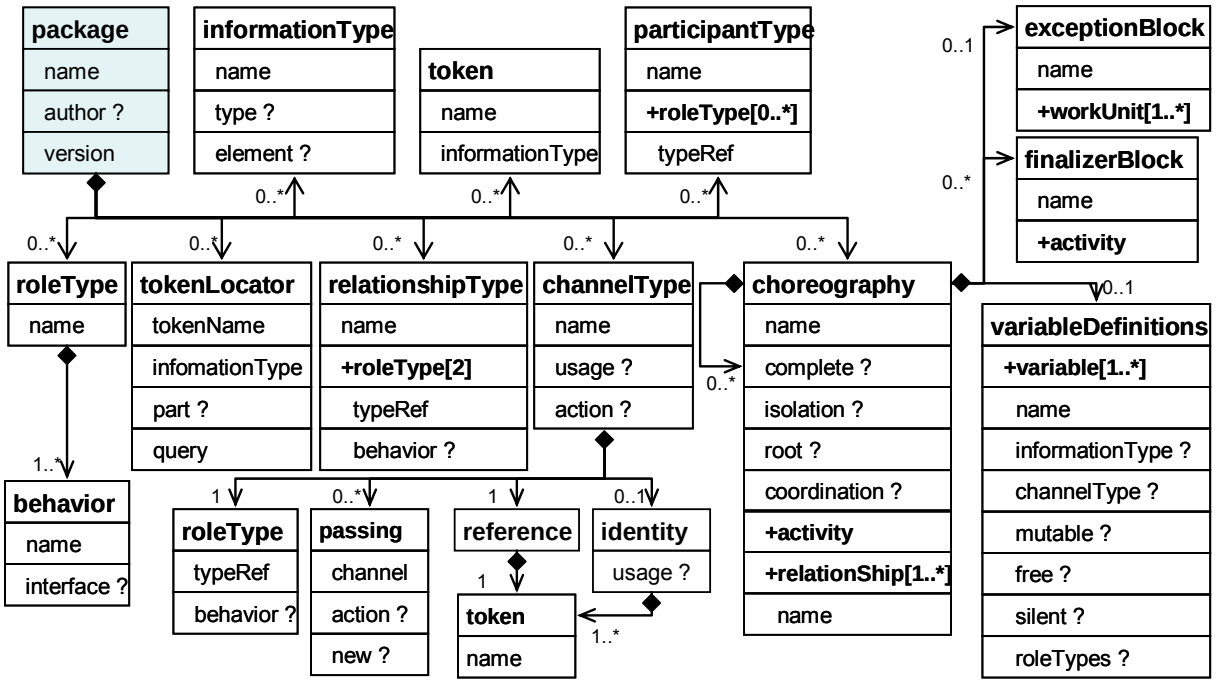


Figura 23 – Meta-modelo da linguagem WSCDL, parte relativa ao processo

Esta linguagem é dedicada a descrever coreografias de processos. Os seus aspectos mais relevantes são:

- Dados: suporta tipos pré-definidos; possui uma abstracção (*token)* de dados relevantes para o protocolo; possui variáveis (*variable*); possui um elemento (*channelType*) que funciona com um canal de interacção para com outros participantes, que define as condições para a comunicação e que pode ser transferido entre participantes.

- Identificação de participantes: possui a noção de papel (*roleType*), a noção de participante (*participantType*) como uma entidade com implementa vários papéis,

e a noção de relação (*relationshipType*) que descreve a associação, e portanto as interacções entre dois papéis.

– Actividades: possui as actividades de: interacção (*interaction*), que modelam as trocas de mensagens assíncronas (*one-way*) ou síncronas (*request-response*); controle de fluxo (*sequence*, *parallel*, *choice)*; blocos com protecção condicional de execução (*workUnit*); execução de coreografias (*perform*); execução de acções de finalização (*finalize*) de uma coreografia e outras actividades complementares (*assign*, *silentAction*, *noAction*).

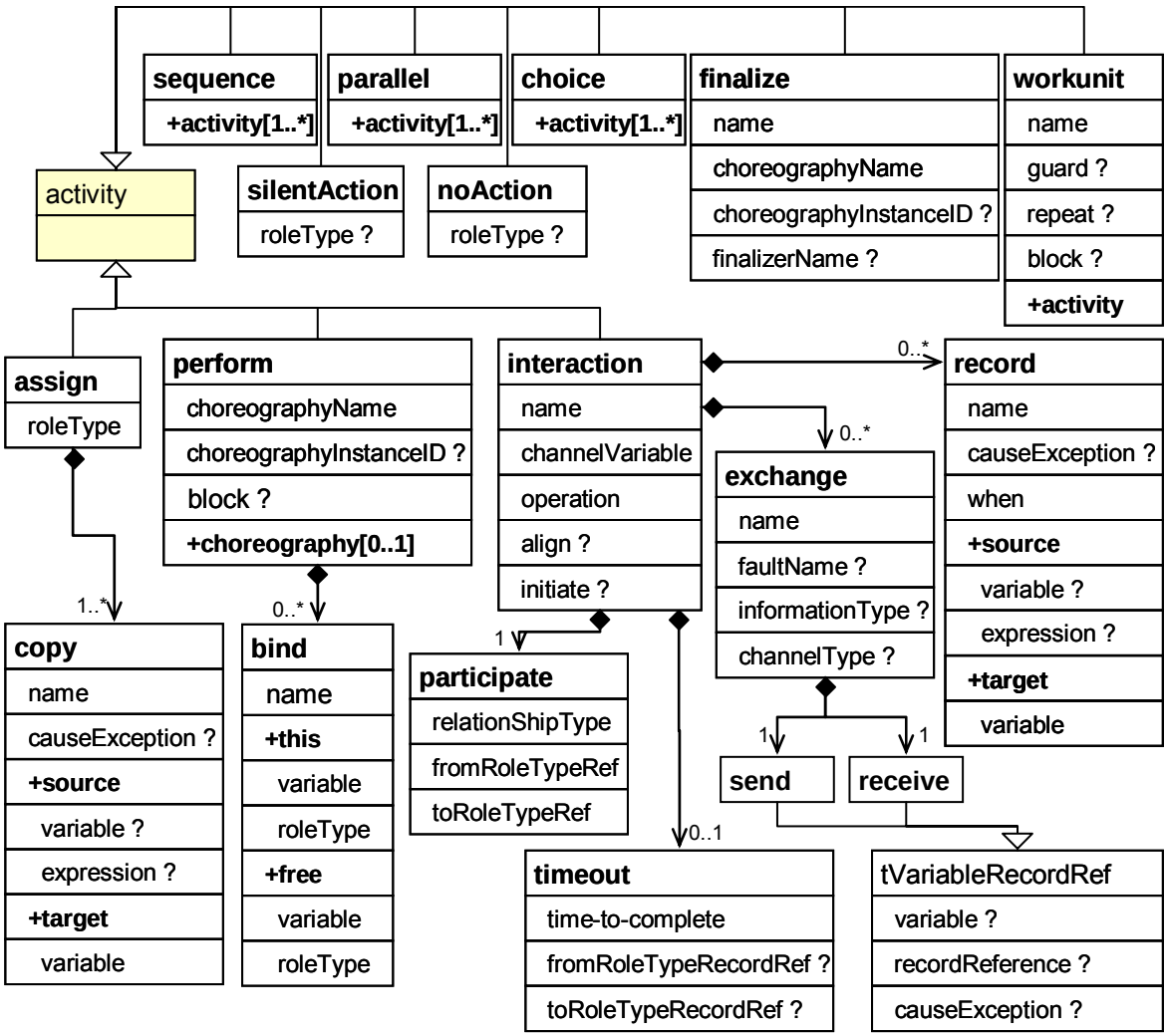


Figura 24 – Meta-modelo da linguagem WSCDL, parte relativa às actividades

– Composição de coreografias: é permitido a concepção de coreografias com base em coreografias já existentes (*perform*).

- Alinhamento de estado: permite que haja coerência de estado entre dois participantes numa interacção (*interaction* com *align* a *true*), contudo tal requer a utilização de um protocolo de coordenação externo.

- Tratamento de excepções: as coreografias são as unidades envolvidas no tratamento de excepções. Cada uma pode ter um bloco *exceptionBlock* que conterá um ou mais blocos *workUnit*, cada um permitindo especificar o comportamento desejado aquando da ocorrência de uma determinada excepção. Pode haver um bloco dedicado a um comportamento por omissão para qualquer excepção que não tenha sido declarada nos outros blocos.

- Acções de finalização: as coreografias podem ter vários blocos *finalizerBlock*, onde cada um deles poderá proceder a um determinado comportamento de finalização da coreografia, que pode ser de confirmação, ou de cancelamento das acções já concretizadas. Cada um desses blocos é identificado por um nome, e será executado quando a actividade de *finalize*, com esse nome, for executada.

- Coreografias coordenadas: são as coreografias que garantem que todos os participantes envolvidos concordam com o seu resultado final, ou seja, que concordam que o bloco terminou com sucesso, ou com uma excepção. Caso tenha terminado com um excepção, todos os participantes deverão saber qual será o bloco de excepção que deverá ser executado, e se terminou com sucesso e se um bloco *finalizerBlock* for executado, então todos os participantes, saberão qual. As coreografias com este requisito devem ser implementadas localmente recorrendo a um protocolo de coordenação.

A linguagem WSCDL: permite indirectamente a Descrição Local, pois permite a sua extracção do processo geral; permite a Descrição de Interligação pois encontra-se presente nas próprias actividades de comunicação; permite a Descrição Comum; e suporta a noção de estado global síncrono nos blocos coordenados, mas remete o seu sincronismo para um protocolo de coordenação externo.

3.2.9 ebXML Business Process Specification Schema – BPSS

A linguagem ebXML BPSS (Business Process Specification Schema) [Levine01] é vocacionada para descrever os aspectos públicos de processos entre dois participantes (binaryCollaborations), de um ponto de vista partilhado e proporcionando um

conhecimento preciso acerca das mensagens trocadas, do seu sequenciamento e do estado final das interacções [Levine01].

Também permite descrever processos com múltiplos participantes (*multiPartyCollaborations*), contudo sendo estes formados por processos binários (*binaryCollaborations*).

Na figura 25 encontra-se o diagrama que contém a parte do meta-modelo desta linguagem relativa ao suporte aos processos, aqui denominados de colaborações. A figura 26 complementa o meta-modelo descrevendo as interacções, nesta linguagem denominadas de transacções de negócio (*BusinessTransitions).*

Os aspectos mais importantes desta linguagem são:

- controlo de fluxo baseado em estados e transições entre estados;
- cada processo, assim como cada interacção, tem uma noção de estado final de sucesso ou de falha;

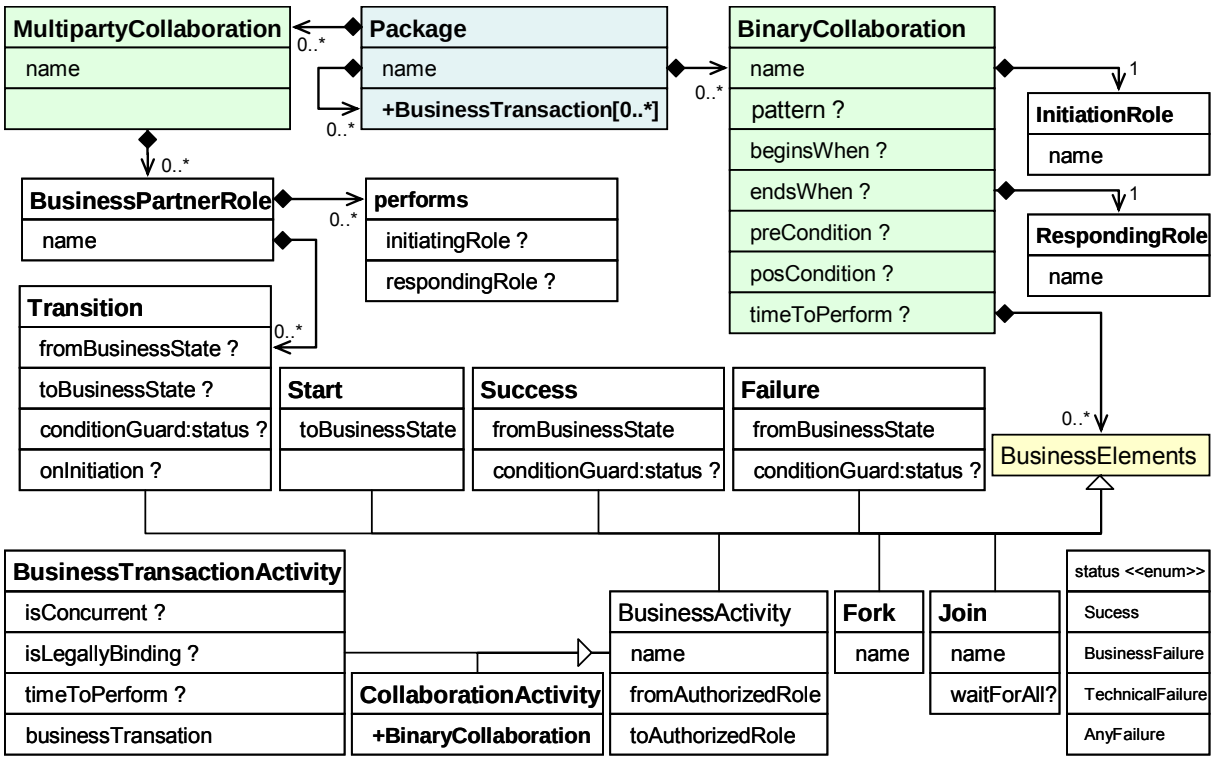


Figura 25 – Meta-modelo da linguagem ebXML BPSS, parte relativa ao processo

- cada processo, assim como cada interacção, pode conter restrições temporais e pré e pós condições;

- cada interacção pode conter exigências, tais como: requerer autenticação, requerer verificação de inteligibilidade, requerer não repúdio, ou requerer garantia de entrega;

- distingue erros de negócio (B*usinessFailure*), de erros técnicos (*TechnicalFailure*) tais como: erros de sintaxe; erros de autenticação, e erros de expiração de tempo de resposta;

- a coordenação entre participantes tem de ser implementada pelo próprio processo, ou seja, não existe um acoplamento a um protocolo de coordenação.

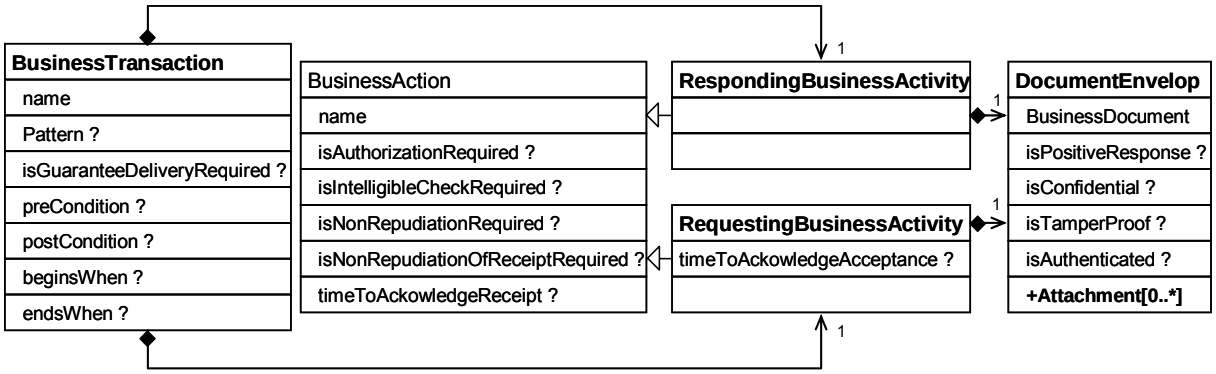


Figura 26 – Meta-modelo da linguagem BPSS, parte das interacções

A linguagem ebXML BPSS: permite indirectamente a Descrição Local; suporta a Descrição de Interligação; e suporta a Descrição Comum entre cada dois participantes por *BinaryCollaboration* e entre mais que dois recorrendo a ligações entre *BinaryCollaborations*; e suporta a noção de estado global único (síncrono), mas entre cada dois participantes.

3.2.10 Comparação entre as várias linguagens

Na tabela 1 é apresentado um sumário das características suportadas pelas várias linguagens analisadas. Nessa tabela o sinal de + representa o suporte directo da característica, o sinal + / – representa o suporte indirecto ou limitado, e o sinal – representa que a característica não é suportada.

Linguagem	Descrição Local	Descrição de Interligação	Descrição Comum	Estado único
WSFL	+	+	–	–
Xlang	+	+	–	–
WSCL	+ / –	–	–	–
BPML	+	–	–	–
WSCI	+	+	–	–
BPEL	+	+	–	–
XPDL	+	+	–	–
WSCDL	+ *	+	+	+
BPSS	+ *	+	+ / –	+ *2

Legenda: * obtida por extracção / *2 entre cada dois participantes

Tabela 1 – Funcionalidades suportadas pelas linguagens analisadas

3.3 Modelos e Protocolos Transaccionais para suporte aos Fluxos de Trabalho Interorganizacionais

Os fluxos de trabalho interorganizacionais, pelo facto de envolverem várias organizações, requerem que haja uma concordância quanto à evolução do fluxo global. Pois, caso esse requisito não seja verificado, pode-se gerar falta de sincronismo entre os vários intervenientes, e o processo global entrar em bloqueio irreversível ou haver disparidade acerca dos factos estabelecidos. Esta secção visa apresentar e comparar os vários modelos e protocolos existentes para providenciar uma consistente evolução dos fluxos de trabalho interorganizacionais.

3.3.1 Transacções ACID e o protocolo 2PC

Uma forma de se obter coerência de estado sobre um conjunto de participantes consiste na utilização do modelo transaccional tradicional que é caracterizado pelas propriedades ACID. Estas propriedades são:

- Atomicidade ("*Atomicity*"): uma transacção executa-se completamente ou não se executa de todo, em caso de falha ou incapacidade de conclusão com sucesso, qualquer acção realizada deve ser desfeita, repondo o sistema no estado inicial;
- Consistência ("*Consistency*"): uma transacção preserva a consistência do sistema, transitando o sistema, em caso de sucesso, de um estado válido para outro estado também válido;

- Isolamento ("*Isolation*"): os estados internos da evolução de uma transacção são isolados das outras transacções;
- Durabilidade ("*Durability*"): os resultados de uma transacção são permanentes.

O que indica que uma transacção ACID, num cenário com vários participantes: é vista como uma operação atómica; preserva a consistência dos vários sistemas; os seus estados internos não são visíveis; e tem efeitos persistentes.

A implementação de transacções ACID envolvendo várias entidades, requer a utilização de um protocolo de coordenação, tal como o protocolo designado de "Confirmação em duas fases" ("*Two Phase Commit*" – 2PC). Este protocolo geralmente possui a figura de coordenador, na qual é centralizada a coordenação do protocolo. Uma vez iniciada uma transacção, existe uma primeira fase designada de preparação, onde cada participante prepara a operação a realizar, indicando ao coordenador o sucesso ou insucesso dessa preparação. O coordenador toma, então, uma decisão final sobre o estado final da transacção, que será de sucesso ou confirmação, se todos os participantes indicaram sucesso, ou de cancelamento, caso pelo menos um participante tenha indicado insucesso. Na segunda fase, o coordenador indica a todos os participantes o resultado da transacção, que em caso de confirmação, os participantes consolidam a operação preparada, e em caso de cancelamento os participantes que indicaram sucesso devem repor o seu estado no estado anterior ao início da transacção em curso. Na figura 27 encontra-se a evolução dos estados do coordenador deste protocolo.

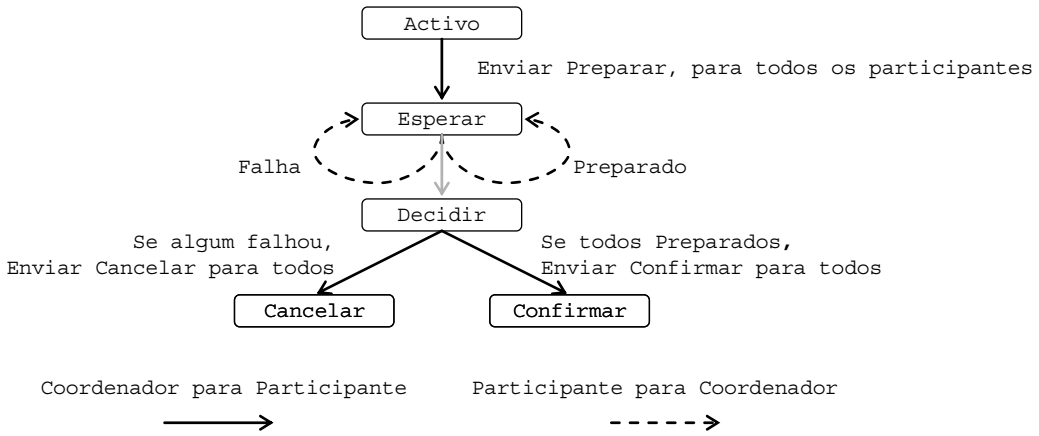


Figura 27 – Protocolo de Confirmação em duas fases (2PC)

3.3.2 Transacções de negócio

Uma transacção de negócio ("*Business Transaction*") é uma consistente mudança, no estado de uma relação de negócio, entre dois ou mais participantes [Ceponlus02]. Como estes processos controlam a execução de actividades de longa duração, pois podem demorar dias, semanas ou meses, não é possível a utilização directa do protocolo 2PC, uma vez que o requisito de isolamento requer o bloqueio ("*locking*") da utilização dos recursos envolvidos na transacção. Tal situação inviabiliza a utilização desses recursos por parte de outras transacções, enquanto a transacção iniciada estiver a decorrer. Não é viável bloquear o acesso a um recurso pelo tempo que uma interacção longa. Por exemplo bloquear novas reservas de um voo, enquanto a reserva corrente não se confirmar.

Para este tipo de processos, o modelo de transacções a utilizar tem que relaxar o isolamento das transacções. Uma forma de obter tal funcionalidade, consiste em executar a operação como se fosse definitiva, e somente em caso de cancelamento aplicar uma operação que anule os efeitos da operação realizada. A esta operação de anulação da operação inicial dá-se o nome de compensação. Em caso de confirmação as acções já se encontram realizadas, pelo que somente é necessário proceder à eliminação da possibilidade de compensação.

Transacções em processos de negócio centralizados

Os processos de negócio centralizados são processos que controlam totalmente a execução de operações noutros sistemas ou aplicações. Toda a tomada de decisões é executada pelo processo, envolvendo o fluxo de controlo normal e de tratamento de erros e sua recuperação. Os outros intervenientes assumem um papel de executantes de pedidos, devolvendo sucesso ou falha do pedido corrente. O processo central decide então continuar o processo ou eventualmente compensar acções que não foram consequentes, desempenhando um papel de coordenador de transacções de longa duração.

O modelo de compensações centrado no processo é o modelo utilizado nas linguagens: BPEL, XLANG, BPML, XPDL e WSCI.

3.3.3 Transacções em processos de negócio interorganizacionais

Os processos de negócio que envolvam várias organizações podem possuir uma estrutura em que um dos processos possua o controlo de todo o processo global. Contudo

tal situação é um caso particular das estruturas possíveis dos processos interorganizacionais. O caso mais normal, será que o processo global tenha o controlo distribuído pelas várias organizações participantes, até porque em certos casos as organizações envolvidas pretendem que o processo não tenha de facto o controlo exclusivo por parte de uma delas. Para este tipo de processos são identificados dois modelos de transacções de negócio: transacções ponto-a-ponto; e transacções distribuídas e orientadas ao bloco, e que seguidamente serão descritos.

3.3.3.1 Transacções ponto-a-ponto

O modelo de transacções ponto-a-ponto centra-se no facto de que as interacções de negócio, entre participantes, serem essencialmente ponto-a-ponto, ou seja, entre duas entidades. Desse facto, resulta que a consistente alteração de estado do processo de negócio, pode ocorrer através dessas interacções binárias, desde que elas tenham comportamento transaccional, ou seja, que a interacção ocorra com sucesso ou com insucesso, mas com garantia de coerência de estado em ambas as partes. Essa coerência de estado pode ser obtida através de um protocolo com entrega fiável.

3.3.3.2 Transacções distribuídas e orientadas ao bloco

O modelo de transacções distribuídas e orientadas ao bloco difere do modelo anterior pelo facto de uma transacção poder envolver vários participantes e englobar um bloco indiscriminado de trabalho. Deste modo o trabalho é estruturado em blocos, onde os vários participantes se registam e depois indicam finalização, com sucesso ou com insucesso, do trabalho dentro desse bloco. A coordenação das transacções pode ser executada por uma entidade coordenadora, que pode ser uma entidade centralizada ou distribuída. Cada bloco pode ter a si associado, e localmente a cada participante, procedimentos de cancelamento e de confirmação, os quais são activados em função da decisão do coordenador, que por sua vez reflecte o estado dos participantes no bloco em questão. Os blocos podem em certos casos ter transacções internas, e portanto serem encaixados uns nos outros, formando uma árvore, e possibilitando a utilização de regras automáticas de confirmação e cancelamento entre si.

A evolução do estado da transacção pode ser suportado por uma variante do protocolo 2PC. O protocolo 2PC utiliza as seguintes fases: fazer / preparar (*start / prepared*); cancelar / cancelado (*cancel / cancelled*); e confirmar / confirmado (*confirm / confirmed*),

que devido ao critério de isolamento, a fase final de cancelar ou confirmar deve desenrolar-se dentro de um espaço de tempo muito curto, na ordem dos milissegundos, face à execução da fase de fazer / preparar. Nas transacções de negócio o tempo entre essas fases pode ser de dias, semanas, meses ou mesmo anos, implicando que o requisito de isolamento deva ser relaxado, de modo de evitar os bloqueios prolongados de acesso aos recursos. A existência de um grande intervalo de tempo entre as fases de fazer / preparar e confirmar / confirmado ou cancelar / cancelado, implica que os recursos tenham que ficar desbloqueados, e portanto visíveis. Deste modo, são identificadas três semânticas de isolamento relaxado:

- Fazer-compensar ("*Do-compensate*"): esta forma, conforme já referido, consiste em realizar o trabalho como se fosse final, na fase de fazer / preparar, e em caso de cancelamento proceder à anulação do trabalho que foi realizado, designando-se esta operação de compensação. Em caso de confirmação não é necessário realizar qualquer tarefa (à excepção de remoção de registos que permitiriam a compensação);

- Provisório-final ("*Provisional-final*"): esta forma consiste em realizar o trabalho, mas marcá-lo como provisório, em caso de confirmação passa a final, e em caso de cancelamento será anulado;

- Validar-fazer ("*Validate-do*"): esta forma consiste em validar a exequibilidade do trabalho, mas só o realizar em caso de confirmação, em caso de cancelamento não é necessário realizar qualquer tarefa.

Estas semânticas variam, portanto, no grau de certeza que oferecem da realização do serviço durante o tempo entre a fase de fazer / preparar e a fase final de confirmação ou cancelamento. A certeza é portanto máxima para a semântica fazer-compensar, mínima para validar-fazer, e intermédia para provisório-final. Esta última propicia-se à gestão probabilística de recursos, de de a situação de excesso de vendas ("*overbooking*") dos voos comerciais é, ou era, um caso paradigmático.

A semântica utilizada neste trabalho será a de fazer-compensar, porque é a semântica adoptada e permitida pela linguagem BPEL, que é a linguagem suporte deste trabalho.

3.3.3.3 Vantagens e desvantagens destes modelos

As transacções ponto-a-ponto têm as seguintes vantagens:

- Uma modelação do processo mais flexível: pois a definição do processo não está condicionada por nenhuma forma estrutural obrigatória.

- Confirmação de estado interacção a interacção: reduzindo ao mínimo o desenvolvimento do processo antes de qualquer confirmação ou negação.

As transacções distribuídas e orientadas ao bloco apresentam as seguintes vantagens:

- Permitem uma modulação do processo mais estruturada: o processo é pois estruturado em blocos, e estes podem ser encaixados ("nested") entre si, o que torna a sua modelação mais prática. A estruturação em blocos permite a existência de procedimentos de cancelamento e confirmação em cada bloco, que cuja execução é desencadeada pelo evolução global do processo, contribuindo para uma melhor estruturação do processo;

- Permitem ter um registo global do processo: a existência de um coordenador que deve ser uma entidade idónea e aceite pelas organizações participantes, permite manter um registo global da evolução do processo, e portanto permite a análise dos processos decorridos e em execução. O que pode funcionar como registo independente para questões legais, relacionadas com algum não cumprimento das obrigações acordadas em relação ao processo.

Como conclusão identifica-se que a utilização das transacções ponto-a-ponto serão mais adequadas para contextos simples, onde o controlo mais preciso poderá fazer sentido em detrimento de uma melhor estruturação. As transacções distribuídas e orientadas ao bloco serão mais adequadas em cenário de maior extensão e de maior partilha do controlo do processo por entre as organizações intervenientes. Proporcionando, este último modelo, um registo importante para a resolução de conflitos entre as organizações.

3.4 Implementações dos modelos transaccionais de negócio

Nesta secção será descrito as implementações existentes, baseadas em XML e na web, dos modelos transaccionais identificados na secção anterior.

Será descrito o ebXML Business Process Specification Schema [Levine01] que é um exemplo do modelo transaccional ponto-a-ponto, e descritas as especificações: OASIS Business Transaction Protocol [Ceponlus02]; Web Services Coordination and Transaction [Cabrera04]; e Web Services Composite Application Framework [Bunting03], que definem protocolos transaccionais distribuídos e orientados ao bloco.

3.4.1 EbXML / BPSS

O protocolo ebXML Business Process Specification Schema (ebXML/BPSS, ou simplesmente BPSS) [Levine01] promovido pelas organizações United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT) e Organization for the Advancement of Structured Information Standards (OASIS), é um exemplo do modelo transaccional ponto-a-ponto.

No protocolo BPSS as interacções são modeladas dentro de uma construção com comportamento transaccional denominada de BusinessTransactions (BT). As BT modelam uma interacção com um pedido, e eventualmente seguido de uma resposta. As BT são construções com estado final atómico e que possuem garantia de entrega, o que permite que ambos os lados da BT tenham a mesma visão do seu estado, que pode ser de sucesso ou de falha. Em caso de falha, a BT não deverá ter qualquer efeito nos dois participantes, sendo cada participante responsável por anular internamente os seus efeitos. No caso da falha poder levar à anulação de trabalho já efectuado, o fluxo do processo deverá contemplar essa situação.

O BPSS não suporta de forma implícita nenhuma semântica transaccional mais flexível em relação ao isolamento. Contudo deixa para o processo a definição explícita de acções de levem à situação pretendida para o caso de confirmação ou de cancelamento.

3.4.2 OASIS Business Transaction Protocol

O protocolo Business Transaction Protocol (OASIS BTP) [Ceponlus02] é também promovido pela organização Organization for the Advancement of Structured Information Standards (OASIS) e insere-se no modelo de protocolo de transacções distribuídas e orientadas ao bloco.

Este protocolo tem por base o funcionamento do protocolo 2PC para garantir a atomicidade das decisões, mas não está comprometido com nenhuma semântica de isolamento. Assegura portanto as fases de preparar / preparado, confirmar / confirmado e

cancelar / cancelado, e remete a semântica das operações para os participantes. Estes podem portanto escolher entre os modelos fazer-compensar, provisório-final, ou validar-fazer, ou o que mais adequado for para o processo. É portanto um protocolo a duas fases, com coordenação de operações abstractas. Apesar de ser orientado ao bloco este protocolo não suporta o conceito de blocos encaixados.

Este protocolo permite a utilização de dois tipos de comportamento transaccional: comportamento atómico ("*atoms*"); e comportamento coesivo ("*cohesions*"), que seguidamente são descritos:

- Comportamento atómico ("a*toms*"): neste tipo de comportamento tem de haver unanimidade nas confirmações, caso contrário a transacção entra na fase de cancelamento.

- Comportamento coesivo ("*Cohesions*"): este tipo de comportamento relaxa a exigência de unanimidade, permitindo que haja uma selecção de participantes que deve confirmar, e que os restantes podem confirmar ou cancelar. Essa selecção é realizada em tempo de execução, e mediante a aplicação de regras de negócio por parte do participante responsável pela transacção.

O BTP utiliza o conceito de conjuntos de confirmação ("*confirmation-sets*") para a implementação dos dois tipos de comportamento. Um conjunto de confirmação corresponde ao conjunto de participantes que têm obrigatoriamente de confirmar (indicar que estão preparados – "*prepared*") para que a transacção suceda. No comportamento atómico, esse conjunto corresponde a todos os participantes registados na transacção. No comportamento coesivo, esse conjunto é definido por um participante, que é designado de terminador da transacção. Neste último caso, quando o conjunto de confirmação estiver definido a transacção passa a ter um comportamento atómico, pois necessita da confirmação de todos os participantes presentes no conjunto de confirmação. É permitido aos participantes abandonar a transacção, mas tal terá que acontecer antes da fixação do conjunto de confirmação.

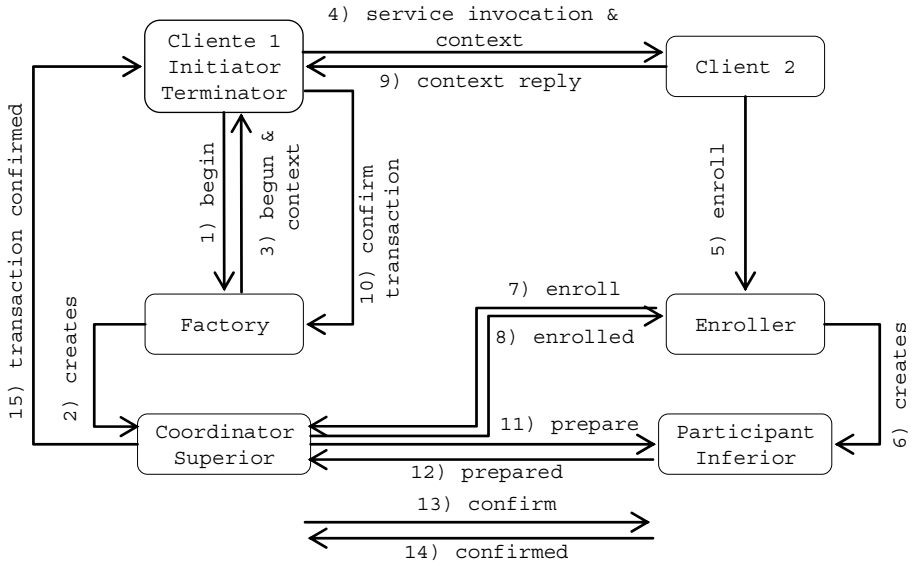


Figura 28 – Cenário entre clientes no protocolo OASIS BTP

Em termos da estruturação funcional do protocolo, cada participante numa transacção tem de obter acesso a um elemento executor do protocolo, que é designado de elemento BTP, e que tem por objectivo proceder à troca de mensagens com outros BTPs e assim controlar e indicar o estado da transacção para a aplicação participante. Os elementos BTP podem ser simples, sendo designados de "*participant*", ou coordenadores sendo então designados de "*composer*" (no caso de comportamento coesivo) ou "*coordinator*" (no caso de comportamento atómico). Os elementos simples, controlam o estado de apenas um participante. Os elementos coordenadores podem controlar um grupo de elementos simples ou coordenadores. A relação entre dois elementos BTP, designa um de inferior e outro de superior consoante o nível ocupado na árvore de elementos BTP. Na figura 28 encontra-se um cenário simples entre um coordenador com um cliente e um elemento simples e seu cliente.

3.4.3 Web Services Coordination and Transaction

A especificação Web Services Coordination and Transaction (WS-CT) [Cabrera04] promovida pela organização OASIS, especifica um ambiente de transacções distribuídas e orientadas ao bloco. Esse ambiente possui uma arquitectura a dois níveis: coordenação geral, em que foi definida a especificação Web Services Coordination; e protocolo específico, em que foram definidas duas concretizações de protocolos transaccionais: Web Services Atomic Transaction; e Web Services Business Activity Framework.

A especificação **Web Services Coordination** (WS-Coordination, ou WS-C), define o nível de suporte à coordenação geral e permite o acoplamento de protocolos específicos de coordenação.

A arquitectura do WS-Coordination é composta pelos seguintes serviços:

- Serviço de activação: que permite a um participante criar uma actividade (transacção), e definir o serviço de coordenação associado.
- Serviço de registo: permite aos restantes participantes registarem-se na actividade (transacção).
- Serviço de coordenação: serviço que implementa o protocolo de coordenação, deverá existir um serviço por cada tipo de protocolo existente.

Suporta interposição, ou seja, não limita a que os clientes tenham que utilizar uma única instância de implementação da WS-CT. Cada cliente pode ter a sua instância, uma vez que elas comunicam entre si, disponibilizando o serviço de forma transparente. Cada instância designa-se de coordenador (*"coordinator"*), em que a instância em que a actividade é criada assume o papel de coordenador raiz (*"root coordinator"*) e as outras assumem o papel de subordinados (*"subordinate cordinator"*).

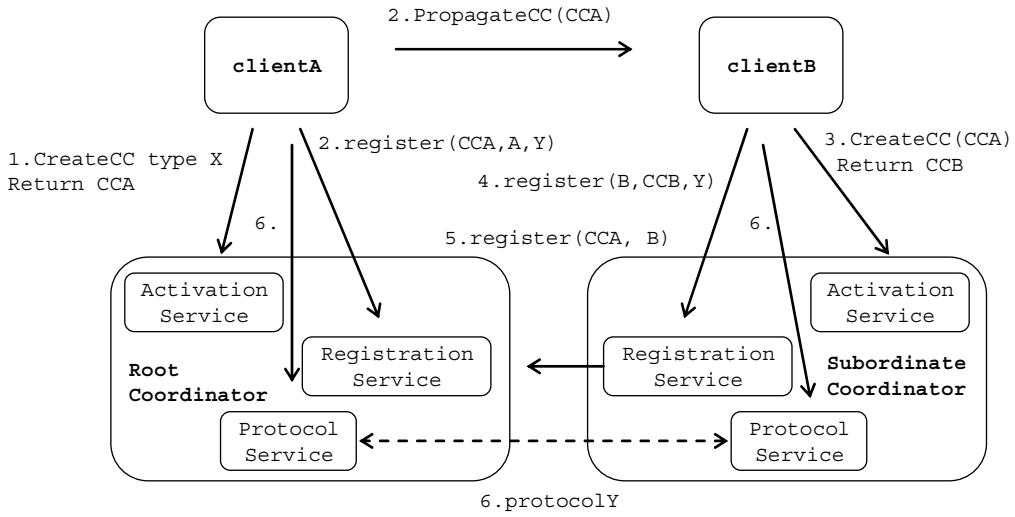


Figura 29 – Interacção entre dois clientes e seus coordenadores no WS-Coordination

Na figura 29 encontra-se uma ilustração acerca da interacção entre dois clientes com os seus coordenadores. O cliente A inicia o cenário criando um contexto (*CoordinatorContext*) do tipo X (1), de seguida passa-o para o cliente B (2), que por sua vez cria um contexto próprio no seu coordenador (3). Entretanto o cliente A regista-se no

seu coordenador (2), no seu serviço de registo na actividade criada e com a indicação do protocolo Y (que é do tipo X). O cliente B, regista-se no seu coordenador (4), passando-lhe o seu contexto já criado, e indicando o protocolo Y. O serviço de registo de B informa o serviço de registo de A, acerca do registo de B (5), indica-lhe o serviço do protocolo Y de B, e recebe a indicação do serviço do protocolo de A. Depois, ambos os clientes interagem com o serviço do protocolo (6) seguindo o protocolo escolhido.

A especificação **Web Services Atomic Transaction** (WS-AtomicTransaction, ou WS-AT) é uma especificação de um protocolo distribuído com coordenador, que é vocacionado para situações com os requisitos do comportamento ACID. A especificação define os subprotocolos de terminação denominados de: Completion, Volatile 2PC e Durable 2PC. O protocolo Completion permite ao participante indicar ao coordenador para fazer *commit* ou *abort*. O protocolo Volatile 2PC permite indicar aos participantes com recursos voláteis, como *cache*, que actualizem os seus recursos. O protocolo Durable 2PC permite preparar os participantes com recursos persistentes, como bases de dados.

A especificação **Web Services Business Activity Framework** (WS-BusinessActivity, ou WS-BA) é vocacionada para a descrição de transacções de negócio, ou seja, transacções de longa duração. Esta especificação utiliza a semântica de fazer-compensar ("*do-compensate*"), como forma de relaxar o critério de isolamento. O modelo utilizado tem as seguintes fases: *start-completed*, *close-closed*, *compensate-compensated*. A especificação define dois tipos de protocolos de coordenação e dois protocolos de coordenação. Os dois tipos de protocolos de coordenação são *AtomicOutcome* e *MixedOutcome*, em que o primeiro requer que o coordenador indique a todos os participantes para fazerem *close* ou *compensate*, enquanto que o segundo pode diferenciar quais dos participantes devem fazer *close* e quais devem fazer *compensate*.

Os protocolos de coordenação são: BusinessAgreementWithParticipantCompletion; e BusinessAgreementWithCoordinatorCompletion. A principal diferença entre estes dois protocolos é que no primeiro é um participante quem dá a indicação de que o trabalho está terminado (*completed*), enquanto que no segundo é o coordenador quem toma essa decisão.

Na figura 30 pode-se observar o diagrama resumido de transição de estados, do protocolo com terminação pelo coordenador. Nessa figura também se pode observar as mensagens que são geradas pelo coordenador e as que são geradas pelos participantes. As mensagens vindas do coordenador são oriundas: do *scope* pai, do *scope* em questão,

pois é ele quem pode indicar para compensar, entrar e falha, encerrar o *scope*, etc; ou de um *scope* filho que tenha entrado em falha.

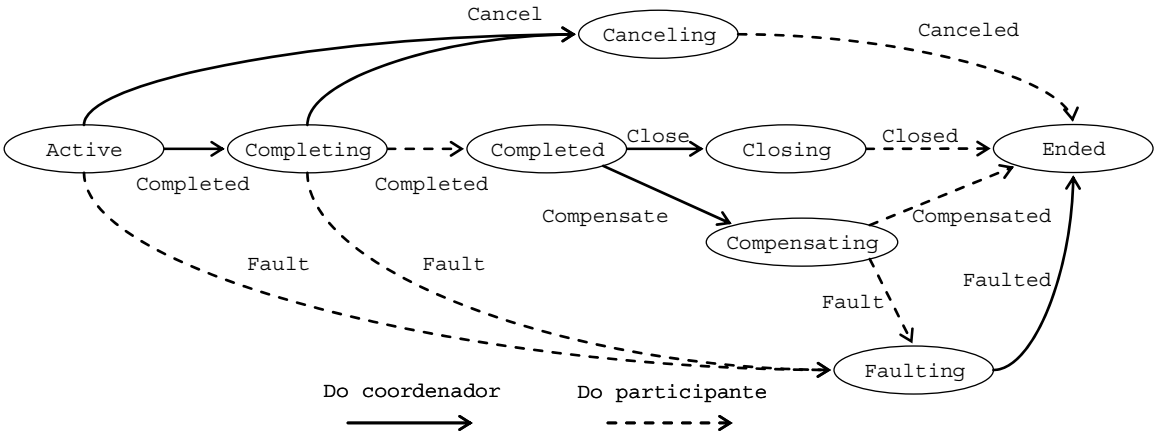


Figura 30 – Estados do protocolo BusinessAgreementWithCoordinatorCompletion

3.4.4 Web Services Composite Application Framework

A especificação Web Services Composite Application Framework (WS-CAF) [Bunting03], também promovida pela organização OASIS, define um ambiente modular de transacções distribuídas com coordenação. Este ambiente tem uma concepção a três níveis, e que são: o nível de suporte ao contexto de actividades, denominado Web Service Context (WS-CTX); o nível de coordenação geral de actividades, denominado Web Services Coordination Framework (WS-CF); e o nível de coordenação específica de actividades, denominado Web Services Transaction Management (WS-TXM). Estas especificações podem ser utilizadas de forma isolada, ou conjuntamente como se pode observar na figura 31.

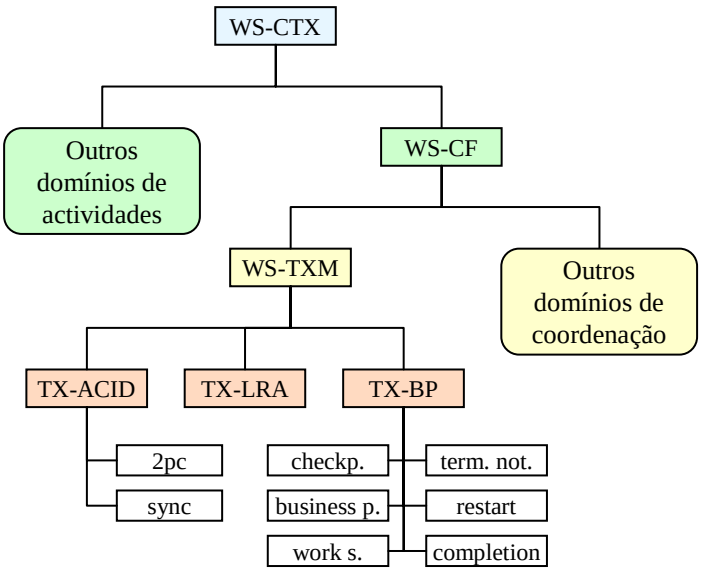


Figura 31 – As várias especificações da WS-CAF

Na figura 32 apresenta-se uma imagem com os principais componentes das três especificações, e serão seguidamente descritos.

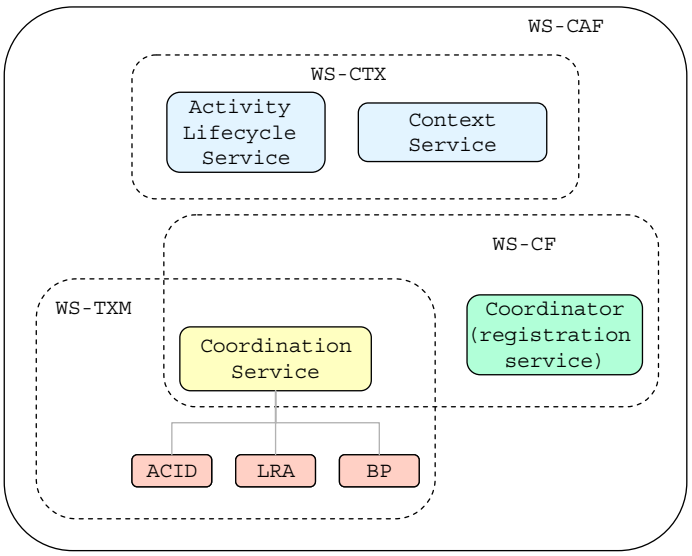


Figura 32 – Os vários componentes da WS-CAF

Nível de suporte ao contexto de actividades: Web Service Context (WS-CTX)

Esta especificação define um mecanismo genérico de gestão de contextos de actividade para a partilha de dados comuns entre Serviços Web. Os seus principais componentes são:

- Componente de contexto ("*Context Component*"): componente que guarda a informação do contexto de cada actividade.

- Serviço de contexto ("*Context Service*"): permite criar actividades e seus contextos, e providencia acesso ao seu conteúdo.

- Serviço de ciclo de vida de actividades ("*Activity Life-cycle Service*" - *ALS*): é um serviço que notifica os participantes acerca das alterações no ciclo de vida das actividades.

Estes serviços são registados nos serviços de contexto de modo a ficarem agregados a uma actividade.

Uma actividade pode ser criada como sendo uma subactividade de outra, podendo-se formar árvores de actividades encaixadas ("*nested*").

Nível de coordenação geral: Web Services Coordination Framework (WS-CF)

O nível de coordenação geral define um mecanismo elementar de coordenação, que pode ser acoplado ao WS-CTX, que se destina a acoplar protocolos específicos de coordenação. Os seus principais componentes são:

- Participante (*Participant*): um participante é uma entidade que se regista num coordenador para participar na actividade disponibilizada pelo serviço de coordenação.

- Coordenador (*Coordinator*): é um serviço onde os participantes se registam de forma a receberem os resultados da coordenação de uma actividade.

- Serviço de Coordenação (*Coordination Service*): é um serviço ALS, que implementa um modelo específico de coordenação de actividades.

Nível de coordenação específica: Web Services Transaction Management (WS-TXM)

O nível de coordenação específica define três modelos de protocolos de coordenação de transacções que podem ser acoplados à WS-CF, e que são: modelo de transacções ACID

(*ACID Transaction*); modelo de acções de longa duração (*Long-Running Action - LRA*); e o modelo de transacções de processos de negócio (*Business Process Transaction - BP*).

Modelo de transacções ACID (*ACID Transaction*):

Este modelo define o protocolo de transacções ACID, com as características das transacções ACID tradicionais (2PC) e que se aplica aos Serviços Web permitindo transacções fortemente acopladas, geralmente utilizadas no contexto de uma organização. Também é definido um subprotocolo denominado de protocolo de sincronização (*Synchronization protocol*), o qual permite informar os participantes com estado volátil ("*caches*"). Estes participantes não integram o protocolo 2PC, mas são consultados antes de fase de *prepared*, e sinalizados depois do protocolo ter terminado. Na fase anterior à fase de *prepared* estes participantes condicionam o sucesso ou insucesso da transacção.

Modelo de acções de longa duração (*Long-Running Action -LRA*):

Este modelo, que também é um protocolo, destina-se a suportar transacções de negócio que apresentam um longo tempo de execução, e utiliza o comportamento da semântica fazer-compensar. Neste protocolo quando uma actividade termina, o protocolo aceita-a ou compensa-a. No entanto, quer a realização da actividade, quer a compensação, são externas ao protocolo. O protocolo limita-se a controlar os eventos e a comunicar que acções devem ser executadas. A compensação é realizada a uma única fase, pelo que não garante a atomicidade da situação final (da compensação). O protocolo permite actividades encaixadas ("*nested*"), possibilitando a uma actividade registar-se, para compensação, na actividade pai. Quando uma actividade termina com insucesso, evocará a compensação, pela ordem inversa, em todas as actividades (filho) registadas para tal. Quando uma actividade termina com sucesso, essa indicação é propagada às actividades registadas, que podem eliminar os registos de compensação, e encerrar a actividade. Aquando do registo para compensação pode ser indicado o tempo máximo para compensação.

Na figura 33 podemos observar a troca de mensagens deste protocolo, do ponto de vista do coordenador.

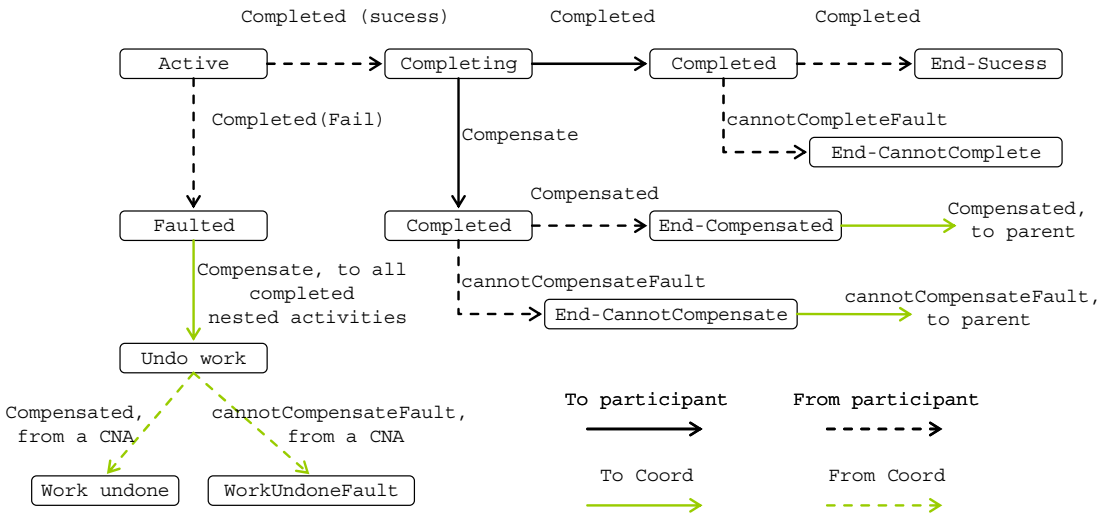


Figura 33 – Protocolo LRA, vista do coordenador

Modelo de transacções de processos de negócio (*Business Process Transaction - BP*):

Este terceiro modelo é destinado a integrar vários participantes com sistemas transaccionais heterogéneos, em transacções de negócio comuns. Neste modelo cada participante reside num domínio de negócio próprio, e é representado por um coordenador denominado de *BusinessTaskCoordinator*. O coordenador raiz da actividade é designado de *RootCoordinator*. Entre o *RootCoordinator* e os *BusinessTaskCoordinators* do domínio dos participantes pode haver interposição por outros *BusinessTaskCoordinators*.

Neste modelo são definidos dois protocolos para lidar com as situações de finalização com sucesso ou insucesso:

- Protocolo *Terminate-notification*: este é o protocolo que um participante que terminou a actividade com sucesso (*TerminatorParticipant*) deve executar. Este protocolo utiliza as seguintes mensagens: *confirmComplete*, *confirmCancel*, *ConfirmCompleted* e *CancelCompleted*.

- Protocolo *BusinessProcess*: Este protocolo deve ser utilizado por um participante (*BusinessProcessParticipant*) que não conseguiu terminar o seu trabalho, e portanto que enviou como resultado uma falha. O coordenador pode marcar que o processo deve falhar de todo, ou que pode anular o participante e continuar. Este protocolo utiliza as seguintes mensagens: *Failure*, *FailureAcknowledge*, *FailureHazard* e *FailureHazardAcknowledge*.

São ainda definidos os seguintes protocolos, entre coordenadores e participantes:

- Protocolo *Checkpoint*: permite criar um ponto de verificação (“*checkpoint*”) em cada domínio. Este protocolo utiliza as seguintes mensagens: *CreateCheckPoint*, *CheckPointingSucceeded*, *CheckPointingFailed*.

- Protocolo *Restart*: permite indicar a cada domínio para voltar a um ponto de verificação. Este protocolo utiliza as seguintes mensagens: *tryRestart*, *restartedSuccessully* e *restartFailed*.

- Protocolo *WorkStatus*: permite inquirir sobre o estado da actividade. Este protocolo utiliza as seguintes mensagens: *getWorkStatus*, *workStatusCompleted*, *workStatusCancelled* e *workStatusProcessing*.

- Protocolo *Completion*: permite pedir ao participante que faça as acções finais e indicar qual o resultado final do protocolo. Este protocolo utiliza as seguintes mensagens: *confirm*, *confirmed*, *confirming*, *cancel*, *cancelled*, *cancelling, unknownResult, processConfirmed, processCancelled, unknowResultOccurred.*, e *mixedResponse*.

3.4.5 Comparação dos protocolos com coordenação

As especificações mencionadas sobrepõem-se na sua generalidade permitindo executar as fases de preparar / preparado, confirmar /confirmado e cancelar / cancelado, conforme pode ser observado na tabela 2. A fase de cancelar / cancelado pode também significar a compensação do trabalho realizado. A tabela contém, para os vários protocolos, as mensagens relativas às referidas fases.

Mensagem Protocolo	Preparar	Preparado	Cancelar	Cancelado	Confirmar	Confirmado
BTP	prepare	prepared	cancel	cancelled	confirm	confirmed
WS-CT AT	prepare	prepared	rollback	aborted	commit	committed
WS-CT BA CC	complete	completed	compensate	compensated	close	closed
WS-CT BA PC	.	completed	compensate	compensated	close	closed
WS-CAF ACID	prepare	vote	rollback	rolledback	commit	committed
WS-CAF LRA	.	addParticipant	compensate	compensated	complete	completed
WS-CAF BP	.	workCompleted	cancel	cancelled	confirm	confirmed

Tabela 2 – Comparação das mensagens dos protocolos de coordenação distribuída

Seguidamente é descrito o suporte a outras características identificadas:

Estrutura do protocolo – qual a estrutura da especificação:

BTP: estrutura a um nível, ou seja, monolítica

WS-CT: estrutura a dois níveis: contexto e coordenação base; e coordenação específica. Com possibilidade de acoplamento de protocolo de coordenação específica.

WS-CAF: estrutura a três níveis: contexto; coordenação base; coordenação específica. Cada um destes níveis pode ser utilizado de forma independente. Cada um destes níveis é aberto a outras especificações.

Protocolo 2PC – suporte ao protocolo 2PC

BTP: o BTP é um protocolo aberto, no sentido, em que são os participantes que definem o tipo de interacção. Suporta a noção de atomicidade da decisão pelo sub-protocolo *atoms*, mas delega nos participantes a responsabilidade de implementarem o protocolo em questão.

WS-CT: suportado directamente pelo subprotocolo *AtomicTransaction*

WS-CAF: suportado directamente pelo subprotocolo ACID *transaction*

Participantes voláteis – possibilidade por parte do protocolo de controlo de participantes voláteis no modelo do protocolo 2PC, na fase anterior à fase de preparar / preparado, em que os participantes são notificados da decisão do coordenador:

BTP: não suportado

WS-CT: suportado pelo subprotocolo *Volatile two-phase Commit Protocol*; os participantes podem não ser notificados da decisão do coordenador

WS-CAF: suportado pelo subprotocolo *Synchronization Protocol*; os participantes recebem a notificação da decisão do coordenador.

Nas seguintes características é utilizado o conceito de conjunto de confirmação ("*confirmation set*"), que reflecte os participantes registados na actividade.

Saída de participantes – que o protocolo permita a saída de participantes, por sua iniciativa, do conjunto de confirmação, sem causar erro:

BTP: é suportado pelo protocolo, pelo envio da mensagem de *resign*.

WS-CT: suportado no protocolo *WS-BusinessActivity*, pelo envio da mensagem de *exit*

WS-CAF: suportado no serviço de coordenação (WS-CF), pelo envio da mensagem de *RemoveParticipant*.

Exclusão de participantes – que o protocolo permita ao coordenador excluir participantes do conjunto de confirmação (será um dos participantes a definir quem deve ser excluído):

BTP: sim, no protocolo *cohesions*

WS-CT: sim, no protocolo *WS-BusinessActivity*

WS-CAF: sim, possível através do serviço de coordenação

Compensação – que o protocolo permita que o processo de reposição de estado (*"rollback"*) seja por compensação:

BTP: sim, contemplado nas possibilidades do protocolo geral

WS-CT: sim, no protocolo *WS-BusinessActivity*

WS-CAF: sim, no protocolo LRA, e no protocolo BP (*BusinessProcess*)

Interposição – que o protocolo permita que seja transparente para um coordenador a interacção directa com um participante, ou indirecta por intermédio de um subcoordenador:

BTP: sim, suportado pela relação de superior – inferior

WS-CT: sim, possível através do serviço de coordenação

WS-CAF: sim, possível através do serviço de coordenação

Actividades encaixadas (*"nested activities"*) – que o protocolo permita actividades encaixadas com noção de pai - filho, tal que a actividade pai quando termina influência o estado das actividades filho:

BTP: não é suportado

WS-CT: suportado no protocolo *WS-BusinessActivity*

WS-CAF: o *context service* suporta o conceito de actividades encaixadas, contudo só é suportado pelos protocolos LRA e BP

3.5 Conclusão

Este capítulo visou, primeiro, apresentar e analisar as linguagens normalizadas de descrição de fluxos de trabalho para a Web. A análise focou-se na forma como as linguagens poderiam descrever os processos interorganizacionais, de modo a clarificar que linguagens melhor os descreveriam. Para tal criou-se três categorias: as linguagens com suporte à descrição local; as linguagens com suporte à descrição de Interligação; e as linguagens com suporte à descrição comum. Dessa análise extraiu-se o seguinte resumo:

- as linguagens WSCL e BPML são linguagens que só suportam a descrição local;
- as linguagens WSFL, XLANG, BPEL, WSCI e XPDL permitem a descrição local e a descrição de interligação; e
- as linguagens WSCDL e BPSS são as únicas linguagens que suportam a descrição comum e a noção de estado global único. Entre estas linguagens destaca-se que a linguagem WSCDL possui uma estrutura mais definida e com construções de controlo de fluxo mais ricas. A linguagem BPSS possui uma capacidade de capturar mais exigências das interacções de negócio.

Neste capítulo foram também apresentados e analisados os modelos e protocolos transaccionais para suporte aos fluxos de trabalho interorganizacionais. Por um lado temos o ebXML\BPSS com modelação ponto-a-ponto, e por outro temos os protocolos distribuídos e orientados ao bloco: BTP, WS-CT, e WS-CAF. Estes protocolos visam cobrir, a coordenação de transacções distribuídas, desde o comportamento ACID até às transacções de longa duração. A especificação BTP utiliza um modelo monolítico geral, deixando para os participantes uma implementação consentânea para a utilização do protocolo. A especificação WS-CT, apresenta dois níveis, permitindo o acoplamento de protocolos específicos de coordenação. A especificação WS-CAF, apresenta uma arquitectura a três níveis, em que separa os aspectos de gestão do contexto da actividade do nível de coordenação geral e de coordenação específica.

Capítulo 4

A Linguagem CBPEL: Parte Elementar

Este capítulo inicia a definição da linguagem proposta para descrever processos interorganizacionais, de forma global e comum, e baseada na linguagem BPEL.

A linguagem proposta é denominada de CBPEL, que significa COMMON-BPEL, ou BPEL-COMUM em português. É uma linguagem baseada na linguagem BPEL, dado que esta tem tido uma forte adesão na comunidade científica e industrial. A linguagem CBPEL é baseada na linguagem BPEL porque extrapola as suas construções e regras de utilização. A sua caracterização será completada no capítulo seguinte, onde será abordada a parte de falhas e seu tratamento.

A linguagem CBPEL além de ser um exercício de extrapolação das construções e regras da linguagem BPEL, visa por uma lado clarificar se esta última se adequa à participar em coreografias descritas de forma global e comum, e por outro lado visa clarificar algumas funcionalidades da linguagem WSCDL.

Este capítulo conterá: uma primeira secção acerca da representação comum; depois, uma segunda secção com a descrição da parte elementar da linguagem CBPEL, que corresponde às primitivas de controlo de fluxo e manipulação de dados; seguido de uma terceira secção com as regras de transformação, dos elementos CBPEL apresentados para elementos da linguagem BPEL; e por fim, haverá uma secção de conclusões acerca destes desenvolvimentos.

4.1 Representação comum de processos interorganizacionais

A designação de "comum" advém do facto de se descrever os processos interorganizacionais de uma forma global e única, em vez do processo global resultar da união dos processos individuais participantes. O processo global é então especificado como um só processo, contendo todas as interacções entre todos os participantes e interligadas num único fluxo de controlo. Nesta forma comum cada interacção é modelada como uma única actividade, onde se representa o envio e a respectiva recepção da mensagem, contendo a identificação do emissor e do receptor, assim como toda a informação necessária às duas partes.

Na figura 34 encontra-se um processo interorganizacional envolvendo quatro participantes (A, B, C e D), em que: do lado direito da figura se encontra o processo CBPEL descrito de forma comum; e do lado esquerdo se encontra os processos dos vários participantes envolvidos.

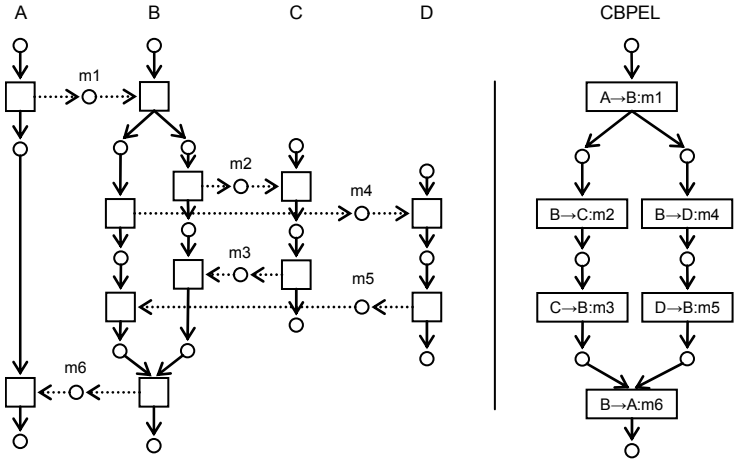


Figura 34 – Exemplo de um processo interorganizacional

Do processo global e comum pode extrair-se o fluxo, com as respectivas actividades, dos vários participantes. A terceira secção deste capítulo abordará as regras para a execução dessa transformação.

Os fluxos então obtidos são os fluxos públicos dos vários participantes, e que contêm o fluxo acordado entre todos eles. Cada participante pode então alterar o seu fluxo, inserindo operações privadas desde que mantenha a estrutura do fluxo público, pois esse é o contracto global acordado, e que garante a exequibilidade do processo global. Em [Aalst01] são definidas três regras de alteração que um participante pode realizar sobre o

seu fluxo público. Essa regras são definidas para as *workflow-nets* consistentes, e preservam as suas características dinâmicas, e que são: 1) inserção de uma operação em série com outras operações, 2) inserção de uma operação em paralelo; e 3) inserção de uma operação em ciclo. Tendo em conta que os processos descritos em linguagem BPEL podem ser descritos como uma *workflow-net* consistente, então essas regras são aplicáveis aos processos desta linguagem.

4.1.1 Vantagens da representação comum

A representação comum apresenta as seguintes vantagens face à representação discreta (individual) dos fluxos participantes:

- **Mais simples**: permite definir o processo global como um único processo;
- **Reduz o número total de acções envolvidas no processo global**: uma vez que uma acção de interacção engloba a componente de envio e recepção, e que cada actividade de controlo de fluxo será eventualmente replicada em vários participantes;
- **Reduz a complexidade do processo global**: é mais simples conceber um fluxo global e comum, do que conceber vários fluxos individuais e coerentes entre si; e
- **Visualização mais evidente**: no caso do fluxo global ser estruturado em blocos permite uma identificação inequívoca das operações pertencentes a cada bloco.

No que resulta numa maior simplicidade de construção e manipulação do processo, uma muito menor propensão à ocorrência de erros por incoerência do processo global, e uma vez construído o processo global, pode gerar-se de forma automática a parte pública dos processos participantes.

4.1.2 Desvantagens da representação comum

A representação comum apresenta as seguintes desvantagens face à representação discreta dos fluxos participantes:

- **Fraca noção visual da responsabilidade individual dos participantes:** pelo facto de haver um fluxo partilhado e as actividades conterem a responsabilidade da emissão e da recepção, fica mais difícil identificar as responsabilidades de cada participante, face à representação discreta de fluxos. Contudo, este factor pode ser atenuado, se o ambiente de edição disponibilizar, em paralelo, com a edição do fluxo global comum, a visualização dos processos

individuais dos vários participantes, ou permita salientar as operações de um (ou cada) participante, por exemplo colorindo as operações em que (cada um) participa.

- **Limitação na definição dos fluxos:** o facto de que o fluxo ser comum e único para os vários os participantes, impede a possibilidade de concepção dos fluxos dos participantes de forma totalmente livre.

Contudo as desvantagens identificadas não se afiguram como significativas, para inviabilizar a utilização da descrição comum.

4.2 Descrição da linguagem CBPEL – Parte elementar

Nesta secção será descrita a linguagem CBPEL, que é um dialecto XML de representação de processos interorganizacionais descritos de forma global e comum. Esta secção só abordará a parte elementar da linguagem, ou seja, só descreverá o controlo normal de fluxo e as suas necessárias declarações. A parte de falhas (excepções), o seu tratamento, e o comportamento transaccional será abordada no capítulo seguinte.

A descrição da linguagem CBPEL é realizada pela extrapolação da funcionalidade dos elementos da linguagem BPEL. O estudo desses elementos está dividido em dois grupos: dados e parceiros; e interacções e controlo do fluxo. Cada elemento conterá a explicação da sua função quer na linguagem BPEL quer na linguagem CBPEL.

Será utilizado uma sintaxe informal para descrever a gramática XML da linguagem CBPEL, de forma a evitar a complexidade da descrição em XML Schema. No entanto é apresentado, no anexo B no final desta dissertação, a descrição completa da linguagem CBPEL em XML Schema. A descrição informal representa os vários elementos como se fossem uma instância dos mesmos, mas onde: os elementos podem ter um atributo de cardinalidade (* para zero ou mais vezes, + para uma ou mais vezes, ? para zero ou uma vez, e n para n vezes) caso seja diferente de 1, e os atributos em vez de valores contêm o seu tipo. O tipo *ncname* refere-se a um identificador simples, e o tipo *qname* refere-se a um identificador qualificado, ou seja, com um identificador de um espaço de nomes (*namespace*).

As descrições em BPEL usam os seguintes *namespaces*:

```
wsdl = "http://schemas.xmlsoap.org/wsdl/"
plnk = "http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
xmlns= "http://schemas.xmlsoap.org/ws/2003/03/business-process/"
```

As descrições em CBPEL usam os seguintes *namespaces*:

```
wsdl  = "http://schemas.xmlsoap.org/wsdl/"
plnk  = "http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
xmlns = "http://schemas.xmlsoap.org/ws/2005/11/common-business-process/"
```

As declarações de tipos (*wsdl:types*), mensagens (*wsdl:message*), propriedades (*plnk:property*), e mapeamentos de propriedades (*plnk:propertyAlias*) são declarados em ficheiro(s) separado(s) do ficheiro que contêm o processo CBPEL. Todas as declarações mantêm as suas formas originais, quer os elementos *types* e *message* herdados do WSDL, quer os elementos *property* e *propertyAlias* vindas da linguagem BPEL.

4.2.1 Dados e parceiros na linguagem CBPEL

Neste grupo irão ser abordados os elementos: *partnerLinkTypes*, *partnerLinks*, *partners*, *variables*, *correlationSets*, *process*, *assign* e *copy*.

4.2.1.1 Elemento partnerLinkTypes

Os *partnerLinkTypes* (PLT) são ligações, ao nível do WSDL, entre dois intervenientes, em cada um assume um determinado papel (*role*). Cada PLT, pode ter dois elementos do tipo *role*, e associa cada *role* a um *portType*. Numa ligação (*partnerLinkType*) onde só um dos intervenientes disponibiliza serviços, só tem um *role*.

Estrutura do elemento *partnerLinkType* na linguagem BPEL:

```
<plnk:partnerLinkType name="ncname">
   <plnk:role name="ncname">
      <plnk:portType name="qname"/>
   </plnk:role>
   <plnk:role name="ncname">?
      <plnk:portType name="qname"/>
   </plnk:role>
</plnk:partnerLinkType>
```

O elemento partnerLinkTypes na linguagem CBPEL

Na linguagem CBPEL, os PLTs continuam a indicar uma ligação entre dois intervenientes, podendo ser explicitamente declarados ou podendo ser inferidos do próprio processo. Pelo que a sua declaração fica idêntica à existente em BPEL, ficando contudo localizada dentro do processo CBPEL e como opcional.

4.2.1.2 Elemento partnerLinks

Cada *partnerLink*, ao qual é associado um nome (*name*), define para um *partnerLinkType*, quem desempenha cada *role*, mais propriamente, se é o próprio processo (*myRole*) ou que outro (*partnerRole*).

Estrutura do elemento *partnerLinks* na linguagem BPEL:

```
<partnerLinks>?
   <partnerLink name="ncname" partnerLinkType="qname"
      myRole="ncname"? partnerRole="ncname"?>+
   </partnerLink>
</partnerLinks>
```

O elemento partnerLinks na linguagem CBPEL

Na definição comum, a vista do processo global é única para todos os participantes, pelo que a noção de *myRole* não faz sentido existir, utilizando-se o elemento *role* em duplicado para indicar a existência de dois papeis (*role*).

Estrutura do elemento *partnerLinks* na linguagem CBPEL:

```
<partnerLinks>?
   <partnerLink name="ncname" partnerLinkType="qname" >+
      <role name="ncname" partnerLinkTypeRoleName=”ncname”/>
      <role name="ncname" partnerLinkTypeRoleName=”ncname”/>?
   </partnerLink>
</partnerLinks>
```

4.2.1.3 Elemento partners

A definição de *partners*, num processo BPEL, identifica quem são os outros parceiros desse processo, e quais os *partnerLinks* em que eles participam.

Estrutura do elemento *partners* na linguagem BPEL:

```
<partners>?
   <partner name="ncname">+
      <partnerLink name="ncname"/>+
   </partner>
</partners>
```

O elemento partners na linguagem CBPEL

A definição de parceiros (*partners*), é fundamental no processo interorganizacional pois estabelece quem são os participantes, e será obrigatória. Vai permitir definir, nos processos CBPEL, quais os papeis globais participantes, e para cada um deles define qual o papel desempenhado em cada *partnerLink*. Como a definição dos participantes é

descrita de forma global, cada *partner* terá que possuir uma associação a cada *partnerlink-role* em que participa.

Estrutura do elemento *partnerLinks* na linguagem CBPEL:

```
<partners>
  <partner name="ncname">+
    <partnerLink name="qname" role="ncname"/>+
  </partner>
</partners>
```

4.2.1.4 Elemento variables

O elemento *variable*, permite definir variáveis na linguagem BPEL. Num processo BPEL todas as variáveis pertencem ao participante executor do processo.

Estrutura do elemento *variables* na linguagem BPEL:

```
<variables>?
   <variable name="ncname" messageType="qname"?
      type="qname"? element="qname"?/>+
</variables>
```

O elemento variables na linguagem CBPEL

A definição de variáveis na linguagem CBPEL, será idêntica à existente na linguagem BPEL, contudo agora cada variável pertence a um dos participantes (*partner*). Pode-se inferir a pertença de cada variável pela simples observação da sua utilização. Mas para facilitar a sua identificação visual introduziu-se de forma opcional o atributo *owner* para clarificar quem é o dono da variável.

Estrutura do elemento *variables* na linguagem CBPEL:

```
<variables>?
   <variable name="ncname" messageType="qname"?
      type="qname"? element="qname"? owner="ncname"?/>+
</variables>
```

4.2.1.5 Elemento correlationSets

Os conjuntos de correlação (*correlationSets*), na linguagem BPEL, permitem definir um conjunto de propriedades como um elemento de correlação. Os conjuntos de correlação são utilizados na identificação de instâncias dos processos individuais. Estes conjuntos são compostos por várias propriedades (*properties*) extraídas das mensagens.

Estrutura do elemento *correlationSets* na linguagem BPEL:

```
<correlationSets>?
   <correlationSet name="ncname" properties="qname-list"/>+
</correlationSets>
```

O elemento correlationSets na linguagem CBPEL

Tendo em conta que os processos individuais utilizam conjuntos de correlação, é fundamental que nos processos globais se defina claramente quais os atributos a utilizar nesses conjuntos. A sua definição será praticamente idêntica à existente na linguagem BPEL, à excepção de por motivos de clareza de leitura é adicionado (como opção) o atributo *owner* que indica qual é o participante dono do conjunto de correlação em questão.

Estrutura do elemento *correlationSets* na linguagem CBPEL:

```
<correlationSets>?
   <correlationSet name="ncname" properties="qname-list"
                   owner="ncname"?/>+
</correlationSets>
```

4.2.1.6 Elemento process

O elemento processo (*process*) é o elemento que contém a definição de um processo individual na linguagem BPEL. Esse elemento contém atributos e definições globais ao processo e uma actividade que conterá todo o fluxo normal de execução.

Os atributos globais ao processo são os seguintes:

- *queryLanguage*: indica qual a linguagem XML de interrogações a utilizar nas selecções de nós em: afectações, definição de propriedades, etc. (XPath por omissão).

- *expressionLanguage*: indica qual a linguagem XML a utilizar nas expressões (XPath por omissão).

- *suppressJoinFailure*: indica se a falha de *joinFailure*, ocorrida dentro do processo, deve ser suprimida ou não. Este estado pode ser redefinido dentro do processo. O estado por omissão é de não suprimir essas falhas.

- *enableInstanceCompensation*: indica se o processo pode, ou não, ser compensado como um todo. Por omissão um processo não pode ser compensado.

– *abstractProcess*: indica se o processo é abstracto ou não (executável). Por omissão um processo é executável.

Uma parte das definições globais é constituída pela definição de: participantes (*partners*), variáveis (*variables*), e conjuntos de correlação (*correlationSets*), que já foram analisados. A outra parte define tratamento de falhas (*faultHandlers*), de compensações (*compensationHandlers*), e eventos assíncronos (*eventHandlers*), contudo estes elementos serão discutidos no capítulo seguinte.

Estrutura do elemento *process* na linguagem BPEL:

```
<process name="ncname" . . .
   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
      <partnerLinks/>?
      <partners/>?
      <variables/>?
      <correlationSets/>?
      <faultHandlers/>?
      <compensationHandler/>?
      <eventHandlers/>?
      activity
</process>
```

O elemento process (commonProcess) na linguagem CBPEL

Na linguagem CBPEL o nome deste elemento será alterado *commonProcess*, para dar relevância à diferença de domínio. Os atributos de *queryLanguage*, *expressionLanguage* e *abstractProcess* mantêm o mesmo significado e utilização. Os atributos *suppressJoinFailure* e *enableInstanceCompensation* serão discutidos no capítulo seguinte.

Estrutura do elemento *commonProcess* na linguagem CBPEL:

```
<commonprocess name="ncname" . . .
   xmlns="http://wwww.temp.org/cbpel/2005/11/common-business-process/">
   <partnerLinksTypes/>?
   <partnerLinks/>?
   <partners/>?
   <variables/>?
   <correlationSets/>?
      ...
   activity
</commonprocess>
```

4.2.1.7 Elementos e atributos standard

Os elementos e os atributos *standard* são um conjunto de características comuns e partilhadas pelas várias actividades. Os atributos definem: o nome da actividade (*name*); uma expressão (*joinCondition*) a aplicar sobre as transições que podem despoletar a

actividade; e a indicação se a condição de *join* gera ou não uma falha (*suppressJoinFailure*) em caso da sua avaliação ser negativa. Os elementos *standard* contêm a declaração das ligações (*links*) de entrada e de saída da actividade.

Estrutura dos atributos *standard* das actividades na linguagem BPEL:

```
<!--standard-attributes" -->
name="ncname"?
joinCondition="bool-expr"?
SuppressJoinFailure="yes|no"?
```

Estrutura dos elementos *standard* das actividades na linguagem BPEL:

```
<!-- standard-elements -->
<target linkName="ncname" />*
<source linkName="ncname" transitionCondition="bool-expr"? />*
```

Atributos e elementos standard na linguagem CBPEL

O atributo *standard* de nome (*name*) existirá em todas as actividades. Os outros atributos e elementos serão discutidos neste capítulo na secção 4.3.5, que é relativa à transformação da actividade de *flow*.

4.2.1.8 Elementos assign e copy

O elemento *assign* é uma actividade BPEL, que pode conter um ou mais elementos *copy*. Cada elemento *copy* permite realizar afectações ou transformações de dados.

Estrutura do elemento *assign* na linguagem BPEL:

```
<assign standard-attributes>
   standard-elements
   <copy>+
      from-spec
      to-spec
   </copy>
</assign>
```

Os elementos assign e copy na linguagem CBPEL

Uma afectação de um campo de uma mensagem é uma acção realizada por um dos participantes. Faz sentido existir na definição comum do processo global pois pode haver afectações que tenham que ser do domínio público. De forma a facilitar a identificação do participante responsável pela afectação cada elemento *copy* poderá indicar quem é o seu executor (*executer*). De resto o elemento *assign* permanece igual à sua definição na linguagem BPEL.

Estrutura dos elementos *assign* e *copy* na linguagem CBPEL:

```
<assign standard-attributes>
   <copy executer="ncname"?>+
      from-spec
      to-spec
   </copy>
</assign>
```

4.2.2 Interacções e controlo do fluxo na linguagem CBPEL

Neste grupo serão abordados os elementos: interacções (*invoke*, *receive*, *reply*), *wait*, *empty*, *sequence*, *switch*, *while*, *pick*, *e flow*. Em todos estes elementos, a parte de tratamento de falhas e compensações será abordada no capítulo seguinte.

4.2.2.1 Interacções

Para modelar interacções, na linguagem BPEL, existem os seguintes elementos: *receive*, *reply* e *invoke*. O elemento *invoke* permite realizar dois tipos de interacções: envio assíncrono; e envio e recepção síncrono, em que síncrono corresponde ao comportamento bloqueante até receber a resposta. O elemento *receive* permite receber uma mensagem (assíncrona ou síncrona) de um parceiro, e o elemento *reply* permite enviar a resposta a uma interacção síncrona.

Estrutura dos elementos *invoke*, *receive* e *reply* da linguagem BPEL:

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
   inputVariable="ncname"? outputVariable="ncname"?
   standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="ncname" initiate="yes|no"?
         pattern="in|out|out-in"/>+
   </correlations>
   <catch faultName="qname" faultVariable="ncname"?>*
      activity
   </catch>
   <catchAll>?
     activity
   </catchAll>
   <compensationHandler>?
      activity
   </compensationHandler>
</invoke>

<receive partnerLink="ncname" portType="qname" operation="ncname"
   variable="ncname"? createInstance="yes|no"?
   standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="ncname" initiate="yes|no"?>+
   </correlations>
</receive>
```

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
   variable="ncname"? faultName="qname"?
   standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="ncname" initiate="yes|no"?>+
   </correlations>
</reply>
```

Interacções na linguagem CBPEL

Como a linguagem CBPEL apresenta uma vista comum sobre o processo, as interacções são modeladas sob a forma de uma perspectiva unificadora, em que um par de envio e recepção (*invoke* e *receive)* é modelado com um único elemento. Esse elemento é designado de "*send*" e terá que possuir toda a informação necessária para se extrair dele as interacções de envio e recepção. De forma a suportar as interacções síncronas, identifica-se as respectivas operações de envio e recepção com o nome da outra operação através dos atributos de *syncReplyName* e *syncInvokeName*, por exemplo no caso de *send(name=SA)* e *send(name=SB)* serem síncronos por parte do emissor de SA, então: *send(name=SA, syncReplyName=SB)*, *send(name=SB, syncInvokeName=SA).*

Estrutura do elemento *send* da linguagem CBPEL:

```
<send partnerLink="ncname"
   fromPartner="ncname"      toPartner="ncname"
   portType="qname"          operation="ncname"
   inputVariable="ncname"    outputVariable="ncname"
   createInstance="yes|no"?  <!—Receiver, default: no -->
   syncReplyName="ncname"?    syncInvokeName="ncname"?
   standard-attributes>
   <correlations>?
      <correlation set="ncname" initiate="yes|no"? pattern="in|out">+
   </correlations>
</send>
```

4.2.2.2 Actividades de wait e empty

A actividade de *wait* realiza uma espera, e a actividade de *empty* faz nada (ou seja, começa e termina).

Estrutura dos elementos *wait e empty* da linguagem BPEL:

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
   standard-elements
</wait>

<empty standard-attributes>
   standard-elements
</empty>
```

Actividades de wait e empty na linguagem CBPEL

As actividades *wait* e *empty* poderão existir num processo global comum por uma questão de informação complementar. A sua descrição será alterada de modo a conter quem é o seu executor (*executer*).

Estrutura dos elementos *wait e empty* na linguagem CBPEL:

```
<wait (for="duration-expr"|until="deadline-expr")
  executer="ncname"  standard-attributes>
</wait>

<empty executer="ncname"? standard-attributes>
</empty>
```

4.2.2.3 Actividade de sequence

A actividade de *sequence* contém uma ou mais actividades, e executa-as sequencialmente

Estrutura do elemento *sequence* da linguagem BPEL:

```
<sequence standard-attributes>
   standard-elements
   activity+
</sequence>
```

Actividade de sequence na linguagem CBPEL

A actividade de *sequence* vai permitir executar actividades em sequência num processo global e comum. Este elemento vai existir na linguagem CBPEL de forma idêntica à que tem na linguagem BPEL.

Estrutura do elemento *sequence n*a linguagem CBPEL:

```
<sequence standard-attributes>
   activity+
</sequence>
```

4.2.2.4 Actividade de flow

A actividade de *flow* permite definir actividades que serão executadas concorrentemente e as dependências entre elas. As actividades concorrentes são definidas como elementos da actividade *flow*. As dependências entre as actividades são definidas por ligações (*link)*, e são declaradas no elemento *links*. Cada actividade, através dos seus elementos *standart,* pode indicar se é a origem ou destino de uma ou mais ligações.

Estrutura do elemento *flow* da linguagem BPEL:

```
<flow standard-attributes>
   standard-elements
   <links>?
      <link name="ncname">+
   </links>
   activity+
</flow>
```

Actividade de flow na linguagem CBPEL

Num processo global e comum é necessário poder definir actividades em paralelo, pelo que esta actividade é imprescindível na linguagem CBPEL. Esta actividade na linguagem CBPEL vai ter a limitação de não permitir a existência de ligações (*links*), pelos motivos que serão apresentados na secção 4.3.5.

Estrutura do elemento *flow* na linguagem CBPEL:

```
<flow standard-attributes>
   activity+
</flow>
```

4.2.2.5 Actividade de switch

Esta actividade permite, na linguagem BPEL, realizar uma decisão de escolha múltipla.

Estrutura do elemento *switch* da linguagem BPEL:

```
<switch standard-attributes>
   standard-elements
   <case condition="bool-expr">+
      activity
   </case>
   <otherwise>?
      activity
    </otherwise>
</switch>
```

Actividade de switch na linguagem CBPEL

Na definição de um processo global e comum a existência de decisões é um requisito fundamental. As decisões podem ser realizadas com critérios privados, mas a existência de prosseguimento alternativo do fluxo tem de estar contemplado nos vários participantes envolvidos. Assim, um participante executará a decisão, condicionando o prosseguimento do fluxo a um dos ramos da decisão. Todos os outros participantes, que tenham actividades dentro da decisão, terão que aguardar pela chegada de uma mensagem que os esclareça de qual o ramo escolhido. Assim esta actividade existirá na

linguagem CBPEL de forma idêntica à existente em BPEL, podendo ter contudo a indicação de qual é o participante que é o seu executor.

Estrutura do elemento *switch* na linguagem CBPEL:

```
<switch standard-attributes executer="ncname"?>
   <case condition="bool-expr">+
      activity
   </case>
   <otherwise>?
      activity
    </otherwise>
</switch>
```

4.2.2.6 Actividade de while

Esta actividade permite executar uma outra actividade de forma cíclica, em que o ciclo é condicionado pelo resultado da avaliação de uma expressão.

Estrutura do elemento *while* da linguagem BPEL:

```
<while condition="bool-expr" standard-attributes>
   standard-elements
   activity
</while>
```

Actividade de while na linguagem CBPEL

À semelhança com a actividade de *switch*, esta actividade permite uma funcionalidade fundamental que deve existir nos processos globais e comuns. Mais uma vez, um só participante executará a decisão de facto de controlo do ciclo. Os restantes participantes envolvidos no ciclo, só saberão se haverá uma iteração, ou se o ciclo terminou, quando receberem uma mensagem que clarifique a situação.

Estrutura do elemento *while* na linguagem CBPEL:

```
<while condition="bool-expr" standard-attributes executer="ncname">
   activity
</while>
```

4.2.2.7 Actividade de pick

Esta actividade permite realizar uma decisão em função do primeiro acontecimento a ocorrer, de entre uma série de mensagens (*onMessage*) e de uma série de temporizações (*onAlarm*).

Estrutura do elemento *pick* da linguagem BPEL:

```
<pick createInstance="yes|no"? standard-attributes>
   standard-elements
   <onMessage partnerLink="ncname" portType="qname"
      operation="ncname" variable="ncname"?>+
      <correlations>?
         <correlation set="ncname" initiate="yes|no"?>+
      </correlations>
      activity
   </onMessage>
   <onAlarm (for="duration-expr" | until="deadline-expr")>*
      activity
   </onAlarm>
</pick>
```

Actividade de pick na linguagem CBPEL

Como esta actividade permite modelar uma recepção de eventos, apresenta uma perspectiva unilateral, pelo que não pode ser utilizada na descrição global e comum. A capacidade de modelar tempos máximos de espera, que despoletam fluxos alternativos de execução, levanta o problema de exigirem sincronismo entre o emissor e o receptor. No capítulo seguinte será abordada a questão de coerência de estado entre participantes, e portanto remete-se esta discussão para esse capítulo.

4.3 Transformação da parte elementar para BPEL

Como já enunciado, uma das grandes vantagens da descrição global e comum, além da edição com menor propensão a erros, é a possibilidade de geração automática dos processos participantes. Esta sessão visa apresentar as regras dessa transformação, que será aplicada aos elementos de um processo descrito na linguagem CBPEL, de modo a gerar os processos locais públicos dos participantes envolvidos em linguagem BPEL.

A linguagem escolhida para a descrição dos processos individuais é a linguagem BPEL, pois é a linguagem que serviu de base à construção da própria linguagem CBPEL, e também porque é uma linguagem com forte aceitação tanto na comunidade cientifica como no mundo empresarial. O facto da linguagem CBPEL basear-se nos conceitos da linguagem BPEL permite uma transformação entre as duas linguagens com uma clara equiparação de conceitos.

Dado que a grande maioria das actividades da linguagem CBPEL tem uma actividade homónima na linguagem BPEL, torna-se, por vezes, necessário clarificar para uma

actividade acerca de qual a linguagem nos estamos a referir. De modo a evitar essas dúvidas serão utilizados, neste capítulo e nos restantes, e sempre que necessário, os prefixos *cbpel:* e *bpel:* de modo a clarificar a identificação da linguagem utilizada, correspondendo à utilização da linguagem CBPEL e BPEL respectivamente.

Em termos práticos, um cenário descrito em CBPEL será constituído por: um ficheiro contendo a descrição do processo CBPEL; um ou mais ficheiros WSDL contendo as definições de propriedades (*property, propertyAlias*), mensagens (*message*), e tipos (*types*); e eventualmente outros ficheiros contendo mais definições de tipos.

A transformação de toda a parte relacionada com o WSDL, incluindo todos os tipos, consiste em colocar num ficheiro WSDL para cada entidade participante, as definições relativas ao seu processo BPEL.

A transformação do processo CBPEL é iniciada com o carregamento do processo para uma árvore, em memória, de elementos XML, em que cada nó descreve um elemento do processo. Por cada participante aplica-se a transformação a uma cópia dessa árvore, sendo gerado uma nova árvore XML com o processo BPEL gerado e que dará origem ao ficheiro ".BPEL" do participante em questão. A transformação inicia-se, para cada participante, com o processamento do elemento *cbpel:commonprocess* o qual: é transformado num elemento *bpel:process*; as suas declarações são filtradas, de modo a cada participante só ficar com declarações estritamente necessárias; e a sua actividade é processada, levando a transformação a todas as suas subactividades de forma recursiva.

O processamento das actividades terá como acção preliminar a eliminação das actividades que não estão relacionadas com o participante em questão (função *exists_partner*), e por fim aplicada a efectiva transformação das actividades na linguagem CBPEL para a(s) correspondente(s) actividade(s) em linguagem BPEL (função *process*). Ainda poderá existir mais uma função denominada de *process_onMessage* que visa obter a primeira recepção do participante em questão. Essa função surge como necessidade da transformação dos elementos *cbpel:switch* e *cbpel:while*, pelo que se remete as explicações para as secções onde estes elementos são abordados.

A transformação utiliza um pseudocódigo baseado em conceitos oriundos do *framework* DOM4J [Rademacher01]. Este *framework* foi o escolhido, e o utilizado, no desenvolvimento da aplicação transformadora, dada a sua simplicidade na manipulação de elementos XML em linguagem Java.

Para cada participante será também gerado mais um ficheiro WSDL, que conterá as definições WSDL do seu processo, nomeadamente os *portType* e *plnk:partnerLinkTypes*. O prefixo *plnk:* refere-se ao *namespace* de *partner-link* da linguagem BPEL. Os *partnerLinkTypes* serão abordados no processamento do elemento *cbpel:commonProcess*. A declaração dos elementos *portType* e das suas operações ocorrerá durante o processamento dos elementos *cbpel:send*, onde por cada operação se verifica se ela já se encontra registada no devido *portType* no ficheiro WSDL do participante em questão, e se caso não esteja, é então acrescentada.

4.3.1 Transformação da construção cbpel:commonprocess

A transformação do elemento *cbpel:commonprocess* resulta na construção de um elemento *bpel:process*, porque é assumido que cada participante apenas contêm um processo. Na situação mais geral, cada participante poderá ter mais que um processo dentro do processo interorganizacional, contudo essa possibilidade não será contemplada. O elemento *cbpel:commonprocess* contém as definições de: parceiros envolvidos (*partners*), ligações entre as operações evocadas e o WSDL (*partnerLinksTypes* e *partnerLinks*), variáveis (*variables*), conjuntos de correlação (*correlationSets*) e a sua actividade inicial.

Seguidamente é apresentada a transformação de cada um dos elementos das definições do processo comum.

Transformação de *cbpel:partnerLinksTypes*

No ficheiro WSDL de cada participante, e dentro do elemento *plnk:partnerLinkTypes*, é inserido um elemento *plnk:partnerLinkType* por cada cbpel:*partnerLinkType* em que ele participe.

Transformação de *cbpel:partnerLinks*

No elemento *bpel:process*, de cada participante, é inserido dentro do elemento *bpel:partnerLinks* um elemento *bpel:partnerLink* por cada *cbpel:partnerLink* em que ele participe.

Transformação de *cbpel:partners*

No elemento *bpel:process*, de cada participante, é inserido dentro do elemento *bpel:partners* um elemento *bpel:partner* por cada *cbpel:partner* com o qual ele se relacione.

Transformação de *cbpel:variables*

No elemento *bpel:process*, de cada participante, é inserido dentro do elemento *bpel:variables* um elemento *bpel:variable* por cada *cbpel:variable* que seja sua.

Transformação de *cbpel:correlationSets*

No elemento *bpel:process*, de cada participante, é inserido dentro do elemento *bpel:correlationSets* um elemento *bpel:correlationSet* por cada *cbpel:correlationSet* que seja seu.

O processamento da sua única actividade, inicia-se com a eliminação das suas subactividades que não estão relacionadas com o participante em questão, e prossegue com a transformação, de forma recursiva, de toda a árvore de actividades, que houver dentro da actividade inicial. Na figura 35 encontra-se o pseudocódigo da transformação deste elemento.

```
[cbpel:commnonProcess] void process (bool instantiated, node bpelNode) {
  node PN = createNode("process", this);
  bpelNode.insertNode(SN);
  -> transformação de partnerLinkTypes
  -> transformação de partnerLinks
  -> transformação de partners
  -> transformação de variables
  -> transformação de correlationSets
  activity.exists_partner(true); // clean
  activity.process(instantiated, PN);
}
```

Figura 35 – Transformação de cbpel:commonProcess

4.3.2 Transformação da actividade cbpel:assign e cbpel:copy

Tendo em conta que numa primeira fase do processamento são removidas as actividades que não estão relacionadas com o participante em questão, então, na fase de transformação só existirão os elementos *cbpel:assign* que contiverem elementos *cbpel:copy* executados pelo participante em questão. Estes elementos (actividades) *cbpel:assign* e *cbpel:copy* são transformados respectivamente nos elementos *bpel:assign* e *bpel:copy*.

4.3.3 Transformação da actividade cbpel:send

A actividade *cbpel:send* é a actividade que modela uma interacção, pelo que o participante pode ser o seu emissor ou o seu receptor, e dará origem respectivamente a um elemento *bpel:invoke* ou *bpel:receive*. O pseudocódigo da transformação encontra-se presente na figura 36. Quando um participante ainda não foi instanciado a sua primeira interacção terá que ser forçosamente uma recepção.

```
[cbpel:send] bool exists_partner(bool clean) {
  return ( toPartner==partner || fromPartner == partner);
}

[cbpel:send] void process (bool instantiated, node bpelNode) {
  node SN;
  if ( toPartner == partner) SN = createNode("receive", this);
  else SN = createNode("invoke", this);
  bpelNode.insertNode(SN);
}

[cbpel:send] node process_onMessage(node bpelNode) {
  if (toPartner != partner) ERRO: emissão no OnMessage
  node OMN = createNode("onMessage", this);
  bpelNode.insert(OMN);
  return OMN;
}
```

Figura 36 – Funções relativas à transformação de cbpel:send

4.3.4 Transformação da actividade cbpel:sequence

A actividade *cbpel:sequence* é a actividade que controla a execução das suas actividades internas executando-as em série. A sua transformação, para a linguagem BPEL, resulta num elemento *bpel:sequence*. Após a remoção da actividades não relacionadas com o participante em questão, pode acontecer que uma sequência fique sem actividades, nesse caso ela não deverá constar do processo BPEL do participante. Um participante não instanciado, ou é instanciado com uma interacção ao nível da sequência, podendo nesse caso possuir várias actividades nesse nível, ou então, é instanciado dentro de uma actividade, dentro da sequência, e nesse caso só pode existir dentro dessa actividade e a actividade de sequência inicial pode ser omitida, pois só contém uma actividade. O pseudocódigo da transformação desta actividade encontra-se presente na figura 37.

```
[cbpel:sequence] bool exists_partner(bool clean) {
  bool exists = false;
  for(int i=0; i<nactivities; ++i)
    if(activity[i].exists_partner(clean)) exists = true;
    else if(clean) deleteNode(activity[i]);
  return exists;
}

[cbpel:sequence] void process (bool instantiated, node bpelNode) {
  if(!instantiated) {
    if (activity[0].type.equals("send") ) {
      if (!activity[0].toPartner.equals(partner))
        ERRO: erro na instanciação
    }
    else {
      if(nactivities > 1) ERRO: actividades acima da instanciação
      activity[0].process(false, bpelNode);  // a sequência é removida
      return;
  } }
  node SN = createNode("sequence", this);  bpelNode.insertNode(SN);
  for(int i=0; i<nactivities; ++i) activity[i].process(true, SN);
}

[cbpel:sequence] node process_onMessage(node bpelNode) {
  node OMN = activity[0].process_onMessage(bpelNode);
  if (activity[0].type.equals("send")) delete(activity[0]);
  return OMN;
}
```

Figura 37 – Funções relativas à transformação de cbpel:sequence

4.3.5 Transformação da actividade cbpel:flow

A actividade *cbpel:flow* é a actividade de controlo de fluxo que executa as suas actividades em paralelo. Antes da apresentação da sua transformação será apresentada uma reflexão acerca da actividade homónima na linguagem BPEL.

A actividade *bpel:flow*

Na linguagem BPEL, a actividade *flow* executa as suas actividades em paralelo, mas a execução de cada actividade pode estar condicionada por dependências de sincronização para com outras actividades. Estas dependências são elementos designados de ligações (*links*). Seguidamente apresenta-se uma reflexão acerca: das ligações (*links*); das condições que se podem associar às ligações (*transitionCondition*); e das expressões de activação (*joinConditions*) que se podem declarar nas actividades.

Apresentação de links

Um *link* corresponde a uma dependência de sincronização entre duas actividades, tal que quando a actividade de origem (*source*) do *link,* termina a sua execução, a actividade destino (*target*), do *link,* pode eventualmente iniciar a sua execução.

Links com condições de transição (*transitionConditions*)

Cada *link* tem uma condição de transição (*transition condition*) que define o seu estado e que é avaliada quando a actividade de origem termina. A condição de transição a existir terá que ser expressa na declaração do *link* na sua actividade de origem, e pode conter uma expressão, de resultado booleano, em função de valores de variáveis. Caso a condição de transição não seja explicitamente expressa, é tido o valor de verdadeiro como o seu valor.

Expressão de activação (*joinConditions*) implícitas

Cada actividade que seja destino de um ou mais *links*, terá uma iniciação dependente desses *links*. A expressão de activação (*joinCondition*) é a expressão que determina a activação dessas actividade mediante o seu valor final, resultando na activação se o valor for verdadeiro, e na sua não activação se o valor for falso. A expressão de activação, por omissão, é a disjunção dos valores de avaliação dos *links* de entrada, ou seja, basta que um deles tenha o estado de verdadeiro para que a actividade se inicie. A expressão de activação de uma actividade só é avaliada quando todos os seus *links* de entrada tenham sido avaliados, que ocorre quando as actividades, de origem dos *links*, terminam.

Caso a expressão de activação resulte no valor falso: se o atributo *supressJoinFailure* estiver a verdadeiro; a actividade é simplesmente tida como executada, mas sem o ser; e se o atributo *supressJoinFailure* estiver com o valor falso, é gerada a excepção (*fault*) *bpws:joinFailure*.

Expressão de activação (*joinConditions*) explícitas

As expressões de activação (*joinConditions*) explícitas, podem utilizar operadores booleanos aplicados ao estado dos seus *links* de entrada. O estado de um *link* de entrada é obtido através da função *getLinkStatus(linkname)*.

Acerca da actividade *CBPEL:flow*

O suporte a *links* na linguagem CBPEL implica que os mesmos tenham que existir após a geração dos processos na linguagem BPEL, porque as dependências teriam que se manter. Contudo na passagem de processos em linguagem CBPEL para os processos em linguagem BPEL as actividades que não estão relacionadas com o participante são removidas, obrigando eventuais *links* a alterarem a sua actividade *source* ou *target*. A

remoção de uma actividade implica a substituição dos seus *links* de entrada (*target*), e de saída (*source*), por outros *links*, tal que cada *link* de entrada teria que se ligar a uma réplica de cada *link* de saída, e cada *link* de entrada conservará a sua entrada tendo como saída a saída do *link* de saída em questão. Deste modo a eliminação de uma actividade com M *links* de entrada e N *links* de saída, gerará um total de M*N *links*. Na figura 38 encontra-se um exemplo com uma actividade com 2 *links* de entrada e 2 de saída. No caso de ser eliminada uma actividade sem *links* de entrada (*target*), então todos os seus *links* de saída podem ser também eliminados, correspondendo à execução imediata das actividades seguintes. No caso de ser removida uma actividade sem *links* de saída, então todos os seus *links* de entrada também podem ser removidos, pois não vão activar nenhuma actividade.

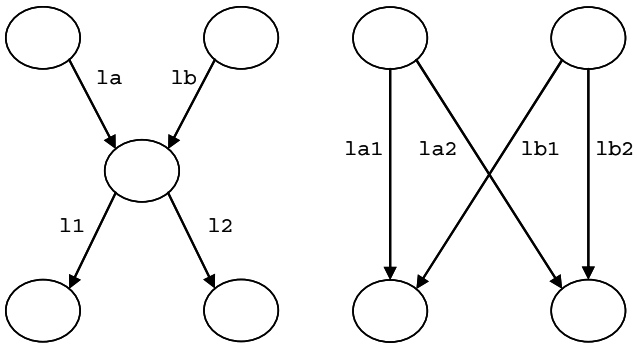


Figura 38 – Supressão de uma actividade com links de entrada e de saída

Os *links* em BPEL podem ter condições de transição, que definem o seu valor de activação face a valores de variáveis locais ao participante. A sua utilização em CBPEL necessita de ser analisada. Também será necessário analisar a existência das expressões de activação das actividades.

Como tudo isto acrescenta mais complexidade ao presente estudo, optou-se por remeter o seu tratamento para trabalho futuro e não incorporar para já, na linguagem CBPEL, os *links* com as suas condições de transição e as expressões de activação das actividades. Como consequência, o atributo *suppressJoinFailure* existente nos atributos *standard* não tem sentido existir, pois não são geradas as falhas de *joinFailure*.

Transformação da actividade CBPEL:flow

A transformação de uma actividade *cbpel:flow,* para a linguagem BPEL, resulta numa actividade *bpel:flow* em cada um dos participantes envolvidos, caso o participante exista em pelo menos duas das actividades internas à actividade de cbpel:*flow*. Como não se

considera a existência de links, essas actividades têm uma execução paralela independente entre si. De modo a evitar a construção de múltiplos processos por cada participante, limitamos que um participante que não esteja instanciado só pode existir numa única actividade interna. O pseudocódigo da transformação a actividade cbpel:*flow* encontra-se presente na figura 39.

```
[cbpel:flow] bool exists_partner(bool clean) {
  bool exists = false;
    for(int i=0; i<nactivities; ++i)
      if(activity[i].exists_partner(clean)) exists = true;
     else if(clean) deleteNode(activity[i]);
  return exists;
}

[cbpel:flow] void process (bool instantiated, node bpelNode) {
  if(!instantiated) {
    if(nactivities>1) Erro: múltiplos processos
    activity[0].process(false, bpelNode);
  }
  else {
    if(nactivities>1) {
      node FN = createNode("flow", this);
      bpelNode.insertNode(FN);
      for(int i=0; i<nactivities; ++i) activity[i].process(true, FN);
   }
   else { // um só ramo para este participante
     activity[0].process(true, bpelNode);
} } }

[cbpel:flow] node process_onMessage(node bpelNode) {
  ERRO: instrução inválida no OnMessage
}
```

Figura 39 – Funções relativas à transformação de cbpel:flow

4.3.6 Transformação da actividade cbpel:while

A actividade *cbpel:while* é a actividade que controla a execução cíclica de uma subactividade interna. O participante que tomará o controlo da execução do ciclo, designa-se de seu executor. Para esse participante o elemento *cbpel:while* é transformado num elemento *bpel:while* e a sua actividade interna transformada normalmente. Para os restantes participantes, que são passivos face à decisão do executor, eles só têm conhecimento que existe uma iteração do ciclo quando receberem uma mensagem que assim o indique. Também, só quando receberem a primeira mensagem possível fora do ciclo é que tomam conhecimento de que o ciclo terminou. Pelo que estes participantes devem esperar ciclicamente tanto a primeira mensagem dentro do ciclo, como a primeira mensagem fora do ciclo. Não é forçoso que tenham de receber apenas uma mensagem, tanto dentro como fora do ciclo, contudo essa é uma simplificação adoptada neste desenvolvimento. De facto, pode acontecer que a primeira mensagem possa ser uma de

entre várias mensagens possíveis. No anexo B apresenta-se uma análise mais detalhada destas possibilidades, mas que por questões de limitação da complexidade se optou por excluir do desenvolvimento.

A transformação do elemento *cbpel:while,* para os participantes passivos (todos excepto o executor), resultará num *bpel:while* e num *bpel:pick*. Em que o *bpel:while* implementa um ciclo, e o *bpel:pick* permite uma execução condicional face à recepção da primeira das duas mensagens possíveis, uma de dentro do ciclo e outra de fora do ciclo. Como a execução do ciclo é automaticamente terminada pela recepção da mensagem de fora do ciclo, pode-se adicionar uma variável de controlo de execução do ciclo. Na figura 40 pode-se observar um exemplo da transformação de um ciclo. Nesse exemplo o participante P3 é um participante passivo no ciclo, e as actividades com '*m?'* assinalam a recepção da mensagem 'm?'. Do lado esquerdo é apresentado o processo comum mas pela vista do participante em questão, e do lado direito é apresentado o resultado da transformação em que o ciclo contém a referida decisão tardia e a variável de controlo do ciclo (S1) que é inicializada antes do ciclo e sinalizada para terminar o ciclo aquando da recepção da primeira mensagem fora do ciclo. A decisão com duplo rebordo representa uma decisão tardia.

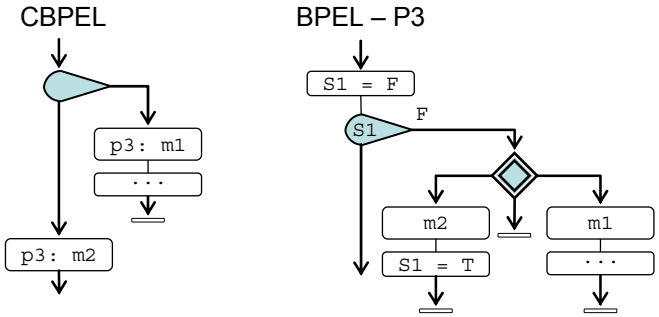


Figura 40 – Exemplo de transformação de um cbpel:while

O pseudocódigo da transformação da actividade *cbpel:while* encontra-se presente na figura 41. De notar, ainda, que: os participantes instanciados dentro do *while*, apenas terão um processo, mas que poderá ser executado múltiplas vezes; e que a primeira mensagem fora do ciclo pelo facto de ser incluída no ciclo terá que ser removida de onde existia.

```
[cbpel:while] bool exists_partner(bool clean) {
  return (activity.exists_partner(clean));
}

[cbpel:while] void process (bool instantiated, node bpelNode) {
  if(!instantiated)
    activity.process(false, bpelNode);
  else {
    if (is_executer()) {
      node WN = createNode("while", this);
      bpelNode.insert(WN);
      activity.process(true, WN);
    }
    else { // not executer
      node SN = createNode("sequence", NULL);  bpelNode.insert(SN);
      SN.insert(createNode("copy-assign", "sair-n=false"));
      node WN = createNode("while", this);     SN.insert(WN);
      node PN = createNode("pick", this);      WN.insert(PN);
      node OMN = activity.process_onMessage(PN);
      activity.process(true, OMN);
      //obter o nextNode a seguir ao while
      node nextNode = bpelNode.nextSibling();
      if(nextNode == NULL) ERRO: While mal terminado
      node OMN = nextNode.process_onMessage(PN);
      OMN.insert(createNode(copy-assign", "sair-n=true");  ++n;
} } }

[cbpel:while] node process_onMessage(node bpelNode) {
  ERRO: instrução inválida no OnMessage
}
```

Figura 41 – Funções relativas à transformação de cbpel:while

4.3.7 Transformação da actividade cbpel:switch

A actividade *cbpel:switch* é a actividade que modela uma decisão múltipla, contendo nós *cbpel:case* para os seus vários ramos. Novamente um dos participantes tomará o controlo da decisão, designando-se de seu executor. Para esse participante o elemento *cbpel:switch* é transformado num elemento *bpel:switch* e os seus nós *cbpel:case* são transformadas em nós *bpel:case*. Para os participantes passivos, face à decisão do executor, eles só sabem que um ramo da decisão foi escolhido quando receberem uma mensagem que assim o indique. Para estes participantes identificam-se duas situações distintas: situação em que o participante existe em todos os ramos da decisão; e situação em que o participante só existe em alguns ramos da decisão. Para a primeira situação, o participante só deverá esperar numa decisão tardia pela primeira mensagem a si endereçada, de cada um dos ramos, e prosseguir com o respectivo ramo. Para a segunda situação, e que o participante não existe em alguns ramos, ele deve esperar pela primeira mensagem nos ramos em que participa, mas também pela primeira mensagem fora da decisão. Essa primeira mensagem fora da decisão deverá ser removida e inserida também no final dos ramos da decisão em que ele existe. Mais uma vez, a referência somente à primeira mensagem, em cada ramo e fora da decisão, surge por uma questão

de simplificação. No anexo B, também se apresenta uma análise mais detalhada das situações envolvendo as decisões, mas que por questões de limitação da complexidade se optou por excluir do desenvolvimento.

A transformação do elemento *cbpel:switch* para os participantes passivos corresponderá a uma decisão tardia *bpel:pick,* e cada ramo *cbpel:case* em que exista corresponderá a uma indicação de recepção da decisão tardia *bpel:onMessage*, assim como a primeira recepção fora da decisão no caso dos participantes que não existam em todos os ramos da decisão.

Na figura 42 encontra-se a transformação de um exemplo de um *cbpel:switch* em que o participante não existe num dos ramos. Do lado esquerdo da figura encontra-se o processo em *CBPEL*, e do lado direito a sua transformação em BPEL.

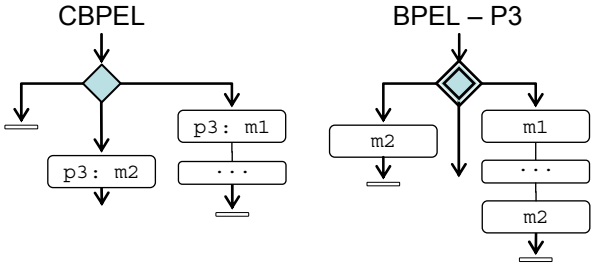


Figura 42 – Transformação de um exemplo de cbpel:switch

O pseudocódigo da transformação de *cbpel:switch* encontra-se presente na figura 43, onde os participantes não instanciados, e de modo a se evitar a existência de múltiplos processos por participante, são limitados a apenas poderem ocorrer dentro de um dos ramos da decisão.

```
[cbpel:switch] bool exists_partner(bool clean) {
  int exists = false;
  for(int i=0; i<ncases; ++i)
    if(case[i].exists_partner(clean)) exists = true;
    else
      { haveCasesWithoutpartner = true; if (clean) deleteNode(case[i]);  }
  return exists;
}

[cbpel:switch] void process (bool instantiated, node bpelNode) {
  if(!instantiated) {
    if(ncases>1) ERRO: múltiplos processos
    case[0].process(false, bpelNode);
  }
  else {
    if (is_executer()) {
      node SN = createNode("switch", this);  bpelNode.insert(SN);
      for(int i=0; i<ncases; ++i) {
        node CN = createNode("case", case[i]);  SN.insert(CN);
        case[i].activity.process(true, CN);
    } }
    else { // not executer
      node PN = createNode("pick", this);  bpelNode.insert(PN);
      for(int i=0; i<ncases; ++i) {
        node OMN = case[i].activity.process_onMessage(PN);
        case[i].activity.process(true, OMN);
      }
      if(haveCasesWithoutpartner) {
        node nextNode = bpelNode.nextSibling();
        if(nextNode == NULL) ERRO: Switch mal terminado
        nextNode.process_onMessage(PN);
} } } }

[cbpel:switch] node process_onMessage(node bpelNode) {
  ERRO: instrução inválida no OnMessage
}
```

Figura 43 – Funções relativas à transformação de cbpel:switch

4.4 Conclusão

Neste capítulo começou por se identificar as vantagens e desvantagens da descrição global e comum.

Depois foi apresentada, a parte elementar da linguagem CBPEL (Common BPEL), que é uma linguagem para descrever processos interorganizacionais de forma global e comum, baseada nos princípios da linguagem BPEL e vocacionada para a geração de processos locais também em linguagem BPEL.

Nesta fase, a linguagem CBPEL ainda não possui o tratamento de excepções, nem compensações, contudo essas funcionalidades serão incorporadas no próximo capítulo.

Seguiu-se depois a transformação de processos CBPEL em processos BPEL. As regras de transformação permitem gerar, a partir de um processo em linguagem CBPEL, os processos dos vários participantes, em linguagem BPEL.

As regras apresentam algumas limitações, de que se destaca as seguintes:

- *flow*: não suporta *links*;
- *while* e *switch:* só é permitida uma recepção em série de uma mensagem, tanto dentro da actividade como imediatamente fora dela;
- um único processo por participante: não é permitido a existência de múltiplos processos para um participante.

A transformação de *switch* e *while*, em situações mais abrangentes, é desenvolvida no anexo B, no qual se pode verificar que pode ocorrer situações de uma alguma complexidade.

Capítulo 5

A Linguagem CBPEL: Parte de Tratamento de Falhas

Este capítulo tem por objectivo apresentar o tratamento de falhas na linguagem CBPEL e a respectiva transformação para a linguagem BPEL. Iniciará com uma descrição acerca de como o tratamento de falhas é efectuado na linguagem BPEL e quais os requisitos para o modelo de tratamento de falhas a ser utilizado na linguagem CBPEL. Por segundo, será desenvolvido, para a linguagem CBPEL, o seu tratamento de falhas e depois o protocolo de coordenação a utilizar. Por terceiro, será apresentado os elementos, da linguagem CBPEL, relativos ao tratamento de falhas. Por quarto, será apresentado a transformação desses elementos, para elementos BPEL, aquando da geração dos processos BPEL das entidades participantes. Por último, serão apresentadas as conclusões deste percurso.

5.1 Transacções e tratamento de falhas

Neste subcapítulo, será primeiramente abordado o suporte a transacções e tratamento de falhas na linguagem BPEL, e por segundo o suporte geral a transacções e falhas em cenários de processos interorganizacionais.

5.1.1 Transacções e tratamento de falhas na linguagem BPEL

A linguagem BPEL permite estruturar os seus processos em blocos de trabalho denominados de *scopes*. Cada *scope* pode conter: variáveis e conjuntos de correlação (*correlationSets*); rotinas de tratamento de eventos assíncronos; rotinas de tratamento de

falhas; uma rotina de compensação; e uma actividade que corresponde ao seu fluxo de execução normal. Todos os tipos de rotinas, assim como a actividade do fluxo normal podem conter outros *scopes*, formando árvores de *scopes* encaixados. As rotinas de tratamento de eventos assíncronos são consideradas parte do fluxo normal, tendo somente a particularidade de que esses eventos são mensagens ou disparos de temporizações, que podem ocorrer em qualquer instante da execução do *scope* e de forma assíncrona, e que podem ser executadas um qualquer número de vezes em paralelo com o fluxo normal. As rotinas de tratamento de falhas são desencadeadas pela ocorrência de uma falha no *scope*, e devem tentar desfazer algum trabalho que já tenha sido realizado no *scope*, como por exemplo compensar os *scopes* filho que tenham terminado com sucesso. A ocorrência de qualquer falha resulta sempre no término do *scope* em questão em estado de falha, e resulta também na terminação forçada dos seus *scopes* filho activos, ou seja, que ainda estejam em execução. A terminação forçada é realizada pelo envio da falha de *ForcedTermination*. Quando um *scope* não possui uma rotina própria para o tratamento de uma falha, a plataforma de execução executa a rotina por omissão, que consiste enviar *ForcedTermination* a todos os *scopes* filho activos, em chamar as rotinas de compensação dos seus *scopes* filho terminados com sucesso, mas pela ordem inversa à da sua terminação, e por fim em lançar a falha para o *scope* acima. A rotina de compensação de um *scope* deverá executar as acções que anulem o resultado da execução normal do *scope*, podendo evocar as rotinas de compensação dos *scopes* filho terminados também com sucesso.

A evolução de um *scope* na linguagem BPEL pode ser observada no diagrama de estados presente na figura 44. Nesse diagrama existem os seguintes estados principais: *Active* – quando o scope se encontra em execução normal; *Faulting* – quando a execução normal entra em falha; *Faulted* – quando finaliza normalmente o tratamento da falha ocorrida durante a execução normal; *Fault-Faulted* – quando no tratamento da falha ocorre um falha; *Completed* – quando o *scope* terminou a execução normal sem falhas; *Compensating* – quando entra em compensação após a sua execução ter sido bem sucedida; *Compensated* – quando a compensação termina normalmente; *Compensate-faulted* – quando a compensação termina pela ocorrência de uma falha; e *Closed* – quando o *scope* encerra definitivamente, eliminando a possibilidade de compensação.

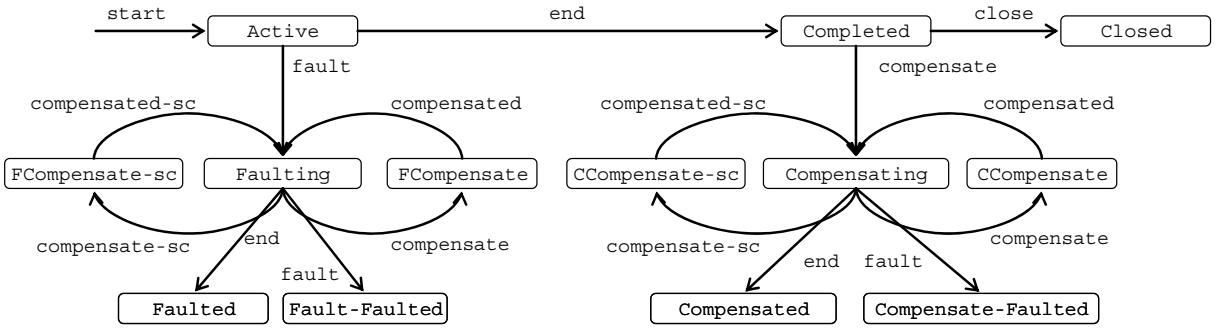


Figura 44 – Estados de um scope na linguagem BPEL

Além dos estados normais ainda existem mais uns outros estados, que podem não ser do *scope* mas sim de um seu fluxo de execução, no caso de haver mais que um fluxo de execução no correspondente estado principal, devido à execução paralela de actividades. Esses estados, existentes dentro do tratamento de falhas ou de compensação, são relativos à compensação de um determinado *scope* (*FCompensate-sc* e *CCompensate-sc*), e à compensação de todos os seus *scopes* bem terminados (*FCompensate* e *CCompensate*).

5.1.2 Transacções e tratamento de falhas em processos interorganizacionais

No contexto de processos interorganizacionais, um bloco de trabalho envolverá vários participantes, pelo que se torna necessário a coerência de estado entre eles, de modo a que o prosseguimento do processo seja devidamente acompanhado por todos os participantes. Deste modo, o estado de cada bloco deverá ser único e partilhado por todos os participantes envolvidos. Qualquer compensação ou falha a ocorrer, também terá que ser do conhecimento de todos os participantes directamente envolvidos.

Uma das formas de haver um consenso sobre o estado de um bloco, que envolve vários participantes, consiste na utilização de um protocolo de coordenação. Seguidamente serão abordados os requisitos para um protocolo coordenação para suporte a processos interorganizacionais baseados na linguagem BPEL.

Requisitos para um protocolo de coordenação para suporte a processos interorganizacionais baseados na linguagem BPEL

Como a linguagem CBPEL utiliza a linguagem BPEL, como linguagem base, é premente a análise dos requisitos provenientes da linguagem BPEL, para a definição de um protocolo de coordenação a utilizar entre os vários participantes. São, então,

identificados os seguintes requisitos, na linguagem BPEL, para um protocolo de coordenação:

- **Estruturação ao bloco**: pois a linguagem BPEL utiliza blocos (*scopes*) para o tratamento de falhas e compensações.

- **Blocos encaixados**: pois num executor de processos BPEL, o estado de um bloco condiciona o estado dos blocos existentes (encaixados) dentro de si. Por exemplo quando um bloco termina com falha, todos os seus blocos encaixados activos são terminados com falha, e por omissão, todos os blocos já terminados com sucesso são compensados. Tal requer um comportamento relativo à gestão de falhas e compensações similar às existentes na linguagem BPEL.

- **Semântica fazer / compensar**: pois é a semântica utilizada na linguagem BPEL.

5.2 Protocolo de coordenação adoptado

Esta secção apresenta o protocolo de coordenação adoptado, e terá as seguintes fases: apresentação do fluxo normal; apresentação do tratamento de falha; apresentação da compensação; apresentação dos problemas relativos à participação em *scopes* encaixados; discussão acerca da utilização dos protocolos de coordenação apresentados no capítulo anterior; e por fim problemas que ficam por resolver.

Optou-se por um protocolo de coordenação com coordenador, porque:

– Simplifica a troca de mensagens entre os vários intervenientes; e

– Permite a existência de um registo global e central a todos os intervenientes.

A inexistência de um coordenador implicaria que a coordenação fosse distribuída entre os vários participantes, e isso iria requerer um maior número de mensagens, o que complicaria o processo geral e também os processos dos participantes. Tal também contrariaria um dos objectivos, da linguagem CBPEL, que é facilitar o desenvolvimento de processo interorganizacionais.

A existência de um registo central e imparcial face aos participantes, relativo aos blocos executados, mas complementado com o processo interorganizacional que é público, permite a realização de auditorias para qualquer esclarecimento relativo ao estado dos

processos, permitindo assim obrigar as partes envolvidas a respeitar o que foi contratado pelo processo. Se bem que não permita uma auditoria total aos processos, dado que não irá conter o conteúdo das mensagens trocadas entre os parceiros, e será nessas mensagens que circularão informações tais como quantidades e preços.

5.2.1 Evolução normal do protocolo

O estudo do fluxo normal, assim como no tratamento de falhas e de compensação será dividido na parte relativa ao coordenador e na parte relativa ao participante.

5.2.1.1 Evolução normal do protocolo no lado do coordenador

O protocolo de coordenação da linguagem CBPEL deverá suportar a existência de estados análogos aos da figura 44, de modo a permitir a boa execução dos participantes, visto que estes são suportados por processos BPEL.

O protocolo de coordenação será descrito com recurso a um diagrama de estados, em que cada transição de estado está associada ao envio ou recepção de uma ou mais mensagens. O diagrama de estados mostrará a evolução do protocolo do lado do coordenador referente à situação de cada participante, resultando na existência de tantas instâncias em execução do diagrama como quantos participantes no *scope* em questão e para cada *scope*. As instâncias da máquina de estados serão criadas aquando da criação do *scope* ou do registo de novos participantes num *scope* já existente.

A interligação entre o estado das várias instâncias de um *scope*, associadas aos vários participantes, é realizada nos pontos com troca de mensagens entre o coordenador.

Na execução concorrente dos vários participantes existirá sempre a possibilidade de uns deles terminarem a execução com sucesso, e outros terminarem com erro. Se todos terminarem com sucesso é disparada a transição assinalada com "*ALL on state*", caso algum gere uma falha, então em todos os outros receberão "*Fault F from scope*" do coordenador, quer tenham concluído com sucesso ou ainda não tenham concluído a sua parte.

Na figura 45 encontra-se o diagrama com os estados do protocolo de coordenação da linguagem CBPEL, referentes à execução normal e à finalização do *scope* sem compensação.

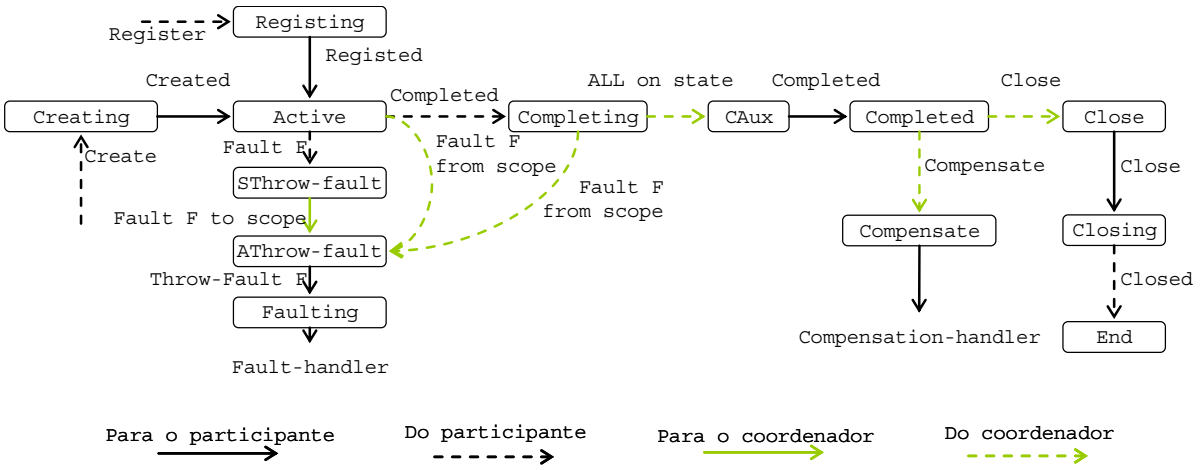


Figura 45 – Estados do protocolo de coordenação: evolução normal de um scope

Seguidamente apresenta-se para cada estado o seu significado e as mensagens que nele podem ocorrer:

- Estado *Creating*: após o participante criador enviar a mensagem de *Create* para o coordenador, o *scope* é criado e passa pelo estado de *Creating*. Neste estado, o coordenador deverá enviar a mensagem de *Created* para o participante criador do *scope*. A mensagem de *Create* deverá indicar qual é o *scope* pai do novo *scope*, se tal existir. A mensagem de *Created* deverá transportar um identificador unívoco do novo *scope*.

- Estado *Registing*: é o estado entre o recebimento de uma mensagem a indicar o registo (*Register*) de um participante no *scope*, e o envio de uma mensagem de confirmação (*Registed*) para o mesmo participante.

- Estado *Active*: é o estado em que se encontra o *scope* depois de ter sido criado e enquanto decorrer a execução normal do *scope*. Neste estado o coordenador pode receber as mensagens: *Completed*, que indica que um participante terminou com sucesso a execução do *scope*, e após a qual transita o participante para o estado *Completing*; *Fault F*, que indica que um participante terminou a execução do *scope* com a ocorrência de uma falha, e após a qual transita o participante para o estado SThrow-fault; e receber "Fault F from scope" proveniente do coordendor, indicando que outro participante entrou em falha, e após a qual transita o participante para o estado *AThrow-fault*.

- Estado *Completing*: este estado conterá os participantes que tenham terminado a execução do *scope* sem falhas. Se ocorrer uma falha num dos participantes que ainda se encontre no estado *Active*, então todos os que se encontrem neste estado (*Completing*) serão notificados dessa falha através da recepção de "*Fault F from scope"* vinda do coordenador, passando o estado para *AThrow-fault*. Se todos os participantes registados no *scope*, chegarem a este estado, então o coordenador envia "*All on state*" assinalando que o *scope* foi bem terminado, e passa-os para o estado *CAux*.

- Estado *CAux*: neste estado, em todos os participantes, é enviado a mensagem de *Completed*, os quais transitam para o estado *Completed*.

- Estado *SThrow-fault*: neste estado o coordenador é notificado (*Fault F to scope*) da ocorrência de uma falha, a qual deve ser entregue a todos os participantes nos *scope*. Após a notificação o participante transita para o estado *AThrow-fault*.

- Estado *AThrow-fault*: neste estado é enviada a mensagem de Throw-fault F para cada participante. Esta mensagem visa efectuar o lançamento local da falha nos vários participantes. Após o envio da referida mensagem cada participante transita para o estado de *Faulting*.

- Estado *Faulting*: este estado corresponde ao início do tratamento da falha e será descrito na secção de tratamento de falhas.

- Estado *Completed*: este estado indica que a execução normal do *scope* foi bem sucedida. Esta conclusão que não era possível obter pela simples presença de um participante no estado de *Completing*, pelo que é necessário esperar pelas mensagens de C*ompleted,* de todos os participantes registados, para que o estado do *scope* transite para este estado. Neste estado, duas evoluções são possíveis: encerrar o *scope*; ou compensá-lo. Após a execução normal do *scope*, a execução passa para outros *scopes*, em que o prosseguimento da execução irá condicionar a situação final do *scope* já terminado. Se ocorrer uma falha, que implique anular o *scope*, será necessário compensá-lo. Nesse caso, como resultado do tratamento da falha associada a um *scope*, será solicitado ao coordenador a compensação do *scope*, pelo que este envia do *scope* pai, do *scope* em questão, a mensagem de *Compensate* para este *scope*, passando-o para o estado de *Compensate*. Caso o *scope* receba a indicação de encerrar definitivamente, pela mensagem de *Close*, o

que pode ocorrer por exemplo pela finalização do processo, ou por fim de um período máximo de compensação, deverá transitar para o estado de *Close*.

- Estado *Compensate*: este estado corresponde ao início da compensação do *scope*, e será descrito na secção de compensações.

- Estado *Close*: este estado serve somente para enviar a mensagem de *Close* a todos os participantes registados no *scope*. Após o envio, o coordenador passa o *scope* para o estado de *Closing*.

- Estado *Closing*: todos os participantes após receberem *Close* devem devolver *Closed*. A linguagem BPEL não permite a recepção de *Close,* nem de acções de encerramento do *scope*, contudo essa funcionalidade é aqui possibilitada de modo a permitir uma melhor gestão do estado dos *scopes*. O estado de *Closing* não permite a devolução de falha, porque os participantes devem apenas registar a notificação de encerramento e anular os registos que permitiriam fazer a compensação. Mesmo que os participantes tenham algum problema, o trabalho já foi realizado, e podem sempre consultar o coordenador de forma a obterem o resultado final do *scope*. Quando todos os participantes enviarem a mensagem de *Closed* ao coordenador, este passa o *scope* para o estado de *End* finalizando o processamento do *scope*.

- Estado *End*: este estado indica que o processamento do *scope* terminou.

5.2.1.2 Evolução normal do protocolo no lado do participante

A descrição das interacções de cada participante para com o coordenador encontra-se descrita no ponto anterior, contudo torna-se necessário esclarecer como o código do participante sinaliza a entrada e a saída de um *scope*.

Para a descrição das acções do lado do participante, em caso da execução sem falhas de um *scope*, utiliza-se o cenário exemplo, em CBPEL, que se encontra presente no lado esquerdo da figura 46. No centro e no lado direito da mesma figura temos os processos dos participantes envolvidos, descritos em linguagem BPEL. O cenário tem o seguinte desenvolvimento: o participante A inicia o cenário com o pedido de criação (*Create*) do *scope* ao coordenador, o qual devolve a confirmação e a identificação do *scope* (*Created*); o participante A entra no *scope* local, correspondente ao *scope* CBPEL; o participante A envia uma notificação ao participante B para que este se registe no *scope* (*Notify*); o

participante B recebe a notificação, regista-se no *scope* (*Register*), recebe a confirmação de registo (*Registed*) no seu *scope* local correspondente ao *scope* CBPEL; segue-se a troca de mensagens dentro do *scope* (m1 e m2); ambos os participantes indicam a finalização do *scope* ao coordenador (*Completed*) e este devolve a confirmação da finalização sem erros por parte de todos os participantes (*Completed*); e por fim ambos os participantes saem do seu *scope* local.

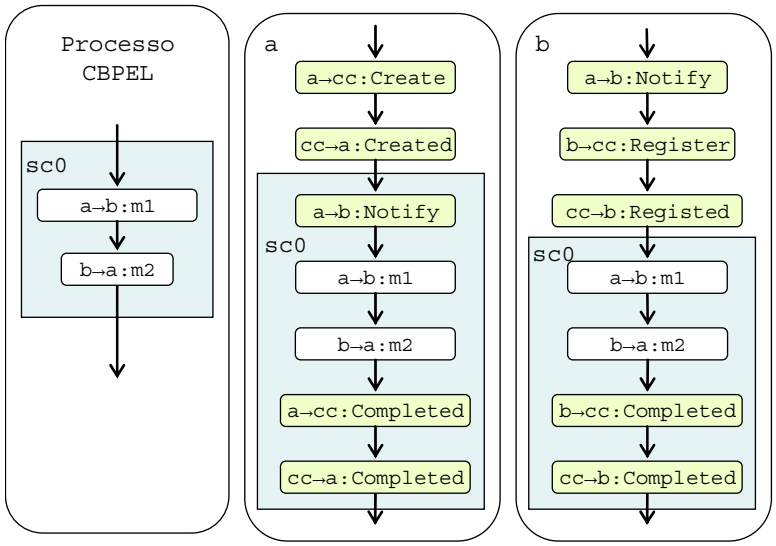


Figura 46 – Exemplo da execução normal de um scope

O cenário descrito é exemplificativo das acções: de criação de um *scope*, de propagação da identificação de um *scope*; de entrada num *scope* existente; e da finalização sem falhas de um *scope*.

Destas acções salienta-se que a opção da propagação da identificação de um *scope* por uma mensagem específica (*notify*), em vez da utilização da primeira mensagem dentro do *scope* para o participante em questão, foi tomada de modo a evitar a alteração das mensagens declaradas no processo CBPEL.

5.2.2 Tratamento de falhas

Nesta secção descreve-se o tratamento de falhas de um *scope*, primeiro na perpectiva do coordenador, e segundo na perspectiva dos participantes.

Uma falha pode ocorrer durante a execução normal do *scope*, ou seja quando o *scope* se encontra no estado *Active*, ou durante a execução de uma rotina de tratamento de falha, ou durante a execução de uma rotina de compensação. Nesta secção serão abordadas as

situações de: falha na execução do código normal; evocação e execução da rotina de tratamento da falha; e falha ocorrida dentro dessa rotina.

5.2.2.1 Tratamento de falhas no lado do coordenador

Uma falha geralmente ocorre somente num dos participantes, o qual deve notificar o coordenador da sua ocorrência. O coordenador deve então suspender a execução normal do *scope*, e notificar todos os participantes da ocorrência da falha, dando início à execução da rotina de tratamento da falha. Na figura 47 pode ser observado o diagrama de estados que descreve o tratamento de falhas do lado do coordenador.

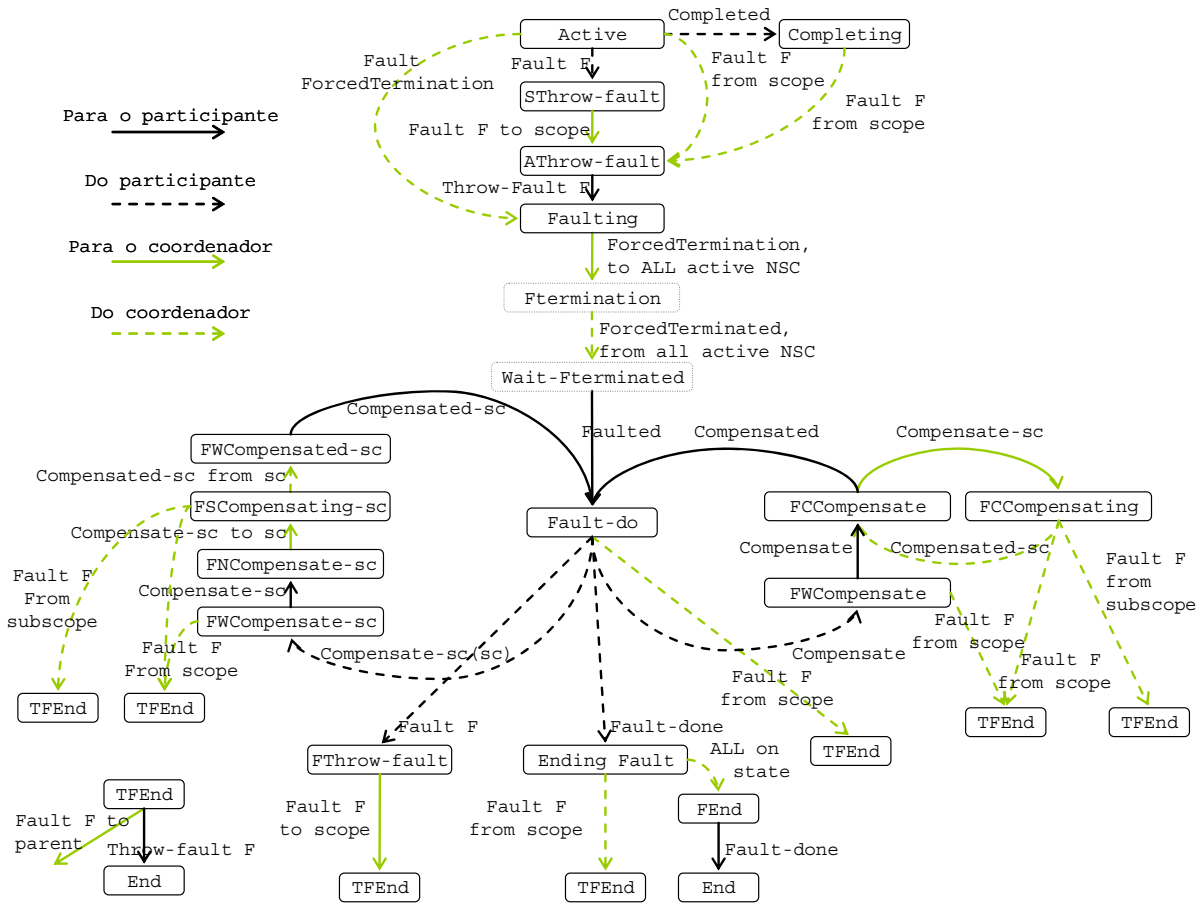


Figura 47 – Estados do coordenador relativos ao tratamento de falhas

A rotina de tratamento da falha, é então executada por todos os participantes, no sentido de desfazer o que foi acordado entre os participantes, assim como as acções privadas em cada participante. Nessa rotina, os participantes podem interagir: trocando mensagens entre si; criando novos *scopes*; gerando mais falhas; ou compensando *scopes* terminados. A troca de mensagens entre os participantes mantém o coordenador no mesmo estado (estado *Fault-do*). A criação de um novo *scope* também não altera o estado

do coordenador no respeitante ao tratamento da falha. A geração de uma falha dentro da rotina de tratamento de falha implica que o coordenador, após receber essa notificação, deva notificar todos os participantes, para eles lançarem a falha localmente, interrompendo a execução da rotina de tratamento de falha. É necessário que todos entrem em estado de falha, pois caso contrário o participante faltoso deixaria de responder normalmente, inviabilizando qualquer troca de mensagens com ele, e também a sua posterior participação em compensações. Durante a execução de uma rotina de tratamento de falha, pode ser evocado a compensação de um *sub-scope* bem terminado, através da actividade de *Compensate scope*. A evocação de uma compensação deve ser feita por todos os participantes do *scope*, que também façam parte do *scope* a ser compensado, pois só assim, se consegue o alinhamento de estado necessário para a execução conjunta. Também é possível evocar a compensação de todos os *sub-scopes* bem terminados, através da actividade de *Compensate*. Os *sub-scopes* são compensados sequencialmente, pela ordem inversa à ordem da sua terminação.

Quando um *scope* entra em falha na linguagem BPEL, a plataforma inicia o tratamento da falha pela terminação de todos os *sub-scopes* activos, pelo lançamento da falha de *ForcedTermination* a todos eles. O que implica que após o coordenador receber a indicação de ocorrência de uma falha, este deva lançar a falha *ForcedTermination* a todos os seus *sub-scopes* activos.

Seguidamente descreve-se os estados presentes na figura 47 e os eventos (mensagens) associados a cada estado:

- Estado *Faulting*: este é o estado inicial, comum a todos os participantes no *scope*, seguido à ocorrência de uma falha. Tal como já foi referido, na linguagem BPEL, quando ocorre uma falha, todos os *scopes* activos, que são filho do *scope* faltoso, devem ser terminados por intermédio da falha *ForcedTermination*. Pelo que essa falha deve ser enviada para todos os seus *scopes* filho activos (*active nested scopes - active* NSC), e o *scope* transita para o estado *FTermination*.

- Estado *FTermination*: neste estado o coordenador espera a recepção da mensagem de *ForcedTerminated* que será enviada pelos *scopes* filho, quando estes tiverem terminado o tratamento da falha. Quando o coordenador tiver recebido a mensagem de todos os *sub-scopes* transita para o estado *Wait-FTerminated*.

- Estado *Wait-FTerminated*: neste estado o coordenador envia a mensagem de *Faulted* para indicar aos participantes que podem começar a executar a rotina de tratamento da falha. Após o seu envio transita para o estado de *Fault-do*.

- Estado *Fault-do*: este é o estado de processamento da falha, aqui podem acontecer as seguintes acções: compensar um *scope* filho terminado, caso receba a mensagem de *Compensate sc*, transitando para o estado *FWCompensate-sc*; compensar todos os *scopes* filho terminados, caso receba a mensagem de *Compensate*, transitando para o estado *FWCompensate*; propagar uma falha, caso receba a mensagem de *Fault F*, transitando para o estado *FThrow-fault*; indicar fim do tratamento da falha, caso receba a mensagem de *Fault-done,* transitando para o estado *Ending-Fault*; e propagar uma falha ocorrida num outro participante, caso receba a mensagem "*Fault F from scope*" do coordenador, transitando neste caso para o estado de *TFEnd*.

- Estado *Ending-Fault*: este estado indica, para os participantes que nele estejam, que terminaram normalmente o processamento da falha. Caso todos os participantes cheguem a este estado, é recebida a mensagem de "*ALL on state*"do coordenador. Neste caso os participantes transitam para o estado *FEnd*. Caso algum participante entretanto tenha gerado uma falha, o coordenador envia a mensagem de "*Fault F from scope"* aos participantes neste estado, os quais transitam para o estado de *TFEnd*.

- Estado *FEnd*: neste estado é enviado a notificação de Fault-done.

- Estado *FThrow-fault*: este estado resulta da ocorrência de uma falha num dos participantes e da recepção do coordenador da mensagem de *Fault*. Neste estado é enviado a notificação de "*Fault F to scope*" para o coordenador, e o estado do participante transita para o estado de TFEnd de modo a propagar essa falha para o processo do participante e para o *scope* pai.

- Estado *FWCompensate-sc*: Neste estado é necessário esperar por todos os participantes no *scope* corrente que também participem no *scope* a ser compensado. Quando tal ocorrer, deverá enviar a mensagem de *Compensate-sc* a todos eles, indicando que todos sinalizaram a operação, e passar à execução para o estado de *FNCompensate-sc*. Na situação de haver participantes neste estado e ter ocorrido uma falha num outro participante, que ainda não chegou a este

estado, é recebida a mensagem de "*Fault F from scope*" e a execução passa para o estado *TFEnd*.

- Estado *FNCompensate-sc*: o coordenador neste estado envia a mensagem de compensação (*Compensate-sc*) ao *scope* em questão, e passa para o estado de *FSCompensating-sc*.

- Estado *FSCompensating-sc*: neste estado o coordenador pode receber: *Compensated-sc*, se a compensação terminou com sucesso; ou *Fault*, se a compensação terminou com erro. Em caso de sucesso a execução transita para o estado de *FWCompensated-sc*. Em caso de falha, ela pode ser proveniente da acção de compensação ou do *scope* corrente. Em qualquer dos casos, a execução passa para o estado *TFEnd*, o qual enviará a falha para os participantes e para o *scope* pai do *scope* corrente.

- Estado *FWCompensated-sc*: é o estado para o qual o coordenador transita depois de receber a mensagem de *Compensated-sc* do *scope* compensado. Neste estado, o coordenador deverá enviar a mensagem de *Compensated-sc,* a todos os participantes envolvidos, de modo a eles prosseguirem de forma síncrona.

- Estado *FWCompensate*: o tratamento de falha transita para este estado quando o participante solicita a compensação de todos os *scopes* terminados normalmente. Contudo é necessário que todos os participantes, que tenham participado em pelo menos um dos *sub-scopes*, cheguem a este estado para se iniciar a compensação. Caso todos os participantes cheguem a este estado, é enviado a mensagem de *Compensate* a cada um deles, e a execução passa para o estado *FCCompensate*. Caso não receba a mensagem *Compensate* por parte de todos os participantes envolvidos, é gerado uma falha e a execução passa para o estado de *TFEnd*. Esta situação só poderá ocorrer se for definido um tempo máximo de espera pela mensagem. Caso tenha ocorrido um falha no *scope* enquanto algum participante já se encontrava em espera neste estado, ele é notificado da falha e a execução transita para o estado *TFEnd*.

- Estado *FCCompensate*: Neste estado o coordenador deve enviar a indicação de compensação (*Compensate-sc*) a todos os *sub-scopes*, em série e pela ordem inversa à da sua terminação. Quando não houver mais *scopes* para compensar o coordenador deverá enviar a mensagem de *Compensated* para os participantes envolvidos.

- Estado *FCCompensating*: por cada *scope* compensado, o coordenador deverá esperar pelo resultado, que pode ser de sucesso (*Compensated-sc*) ou falha (*Fault*). Durante a espera pode acontecer que um fluxo paralelo, num dos participantes no tratamento da falha, gere uma outra falha no *scope* corrente. Então, em caso de sucesso a compensação prossegue, e em caso de falha, o coordenador não permitirá mais compensações, a compensação corrente é terminada, e a execução passa para o estado de TFEnd de modo a propagar a falha ocorrida.

- Estado *TFEnd*: este estado visa comunicar a ocorrência de uma falha ao participante em questão, e propagar essa falha, dentro do coordenador, para o *scope* pai do *scope* corrente.

Quando o processamento da falha chegar ao fim, e no caso da falha que esteve na origem deste processamento ter sido a falha de *ForcedTermination*, então deve ser enviada para o *scope* pai a mensagem com a falha de *ForcedTerminated*, mesmo que tenha ocorrido outra falha durante o processamento.

Neste tratamento releva-se a questão de quem deve notificar o coordenador da compensação de um *scope*. A opção adoptada foi de que a notificação deve ser dada pelos participantes no *scope* que também fazem parte do *scope* a ser compensado. Primeiro, porque será natural que um participante que não esteja envolvido num determinado *scope*, não lhe diga respeito compensar esse *scope*. Segundo, porque o coordenador tem a informação, nos seus registos, de quem participa em cada *scope*.

Um outro caso consiste na ocorrência simultânea de falhas em vários participantes. Nesses casos, haverá um deles que informa primeiro o coordenador, pelo que esse estabelece a falha a ser tratada. Todos os outros, receberão a notificação da ocorrência dessa falha, mesmo tendo gerado outra.

5.2.2.2 Tratamento de falhas no lado do participante

A ocorrência de uma falha do lado do participante, origina uma interrupção da execução do fluxo normal do *scope*, e a consequente passagem da execução para a rotina de tratamento da respectiva falha. No caso de não haver essa rotina, é executada a rotina por omissão, que ao terminar remete a falha para o *scope* acima, ou seja para o *scope* pai do *scope* corrente. As falhas também podem ocorrer dentro das rotinas de tratamento de falhas e dentro das rotinas de compensação.

O despoletar de uma falha, por um participante, ocorre em resposta a uma situação anómala e resulta na evocação da actividade de *throw*. Esta actividade, agora, não poderá proceder ao lançamento unilateral da falha como ocorre normalmente na linguagem BPEL, mas terá que interagir com o coordenador para que a falha possa ser propagada para todos os participantes no *scope*. A actividade CBPEL de *throw* será então decomposta nas acções (ver lado esquerdo da figura 48) de: notificação do participante ao coordenador de falha (p→cc:*Fault f*), e suspensão da execução (*p:Wait*) uma vez que o lançamento da falha ocorrerá num troço de código assíncrono, como se verá de seguida.

Após a ocorrência de uma falha, o coordenador terá que propagar a indicação de falha a todos os participantes registados no *scope*. Como os vários participantes podem estar em plena execução, e por isso não estarem receptivos à recepção desta mensagem, a falha tem de ser enviada de forma assíncrona. A linguagem BPEL permite a execução de rotinas assincronamente (*"event handlers"*) para com o fluxo normal, que são despoletadas pela recepção de eventos assíncronos, tais como uma mensagem. A mensagem assíncrona enviada, pelo coordenador, em caso de falha é a mensagem de *Throw-fault f*. O participante ao receber esta mensagem deverá lançar a respectiva falha no *scope* corrente (ver lado direito da figura 48).

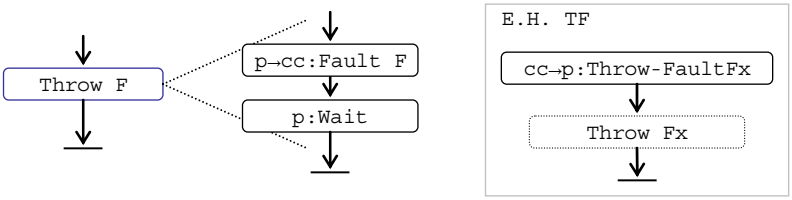


Figura 48 – Emissão e recepção de notificação de falha

Após o lançamento local de uma falha, ela deverá ser atendida pela respectiva rotina de tratamento no *scope* corrente. Caso não seja declarada uma rotina de tratamento de uma falha, devem ser executadas as acções por omissão de acordo com o comportamento da linguagem BPEL, que consiste em: terminar todos os *scopes* filho activos (esta acção é sempre executada, com ou sem rotina de tratamento da falha); compensar os *scopes* filho terminados com sucesso; e lançar a falha para o *scope* pai. A parte de terminar os *scopes* filho activos é implementada pelo envio da falha *forcedTermination*. A parte da compensação corresponde à evocação de *Compensate*. Como pode haver *scopes* em que, quer no tratamento de *ForcedTermination*, quer na compensação, podem pretender trocar mensagens entre os participantes, então é necessário haver coordenação do envio e tratamento da falha de *ForcedTermination* e da compensação. O que implica que todo o

participante tenha que possuir uma rotina de tratamento para todas as falhas possíveis, além da capacidade de receber a sua notificação, e uma rotina de compensação, em todos os *scopes*.

Tal como já referido, o tratamento de uma falha, tem sempre como primeira acção terminar os *sub-scopes* e actividades ainda activas no *scope* corrente. Para tal na linguagem BPEL o *scope* envia a falha *ForcedTermination* a todos eles. Assim, o coordenador deve, como já visto, após receber uma falha, enviar a falha *ForcedTermination* a todos os *sub-scopes* activos (mensagem *Fault ForcedTermination* enviada no estado *Active* da figura 47).

Tratamento por omissão de uma falha

O coordenador, após a fase de terminar os *scopes* activos, dará a indicação aos participantes para prosseguirem com o tratamento da falha. Como o coordenador não terá informação sobre que falhas terão ou não a devida rotina de tratamento, assume-se que todas as falhas terão que possuir uma rotina de tratamento, e que caso o processo não as mencione, o sistema terá que as declarar. Para tal, aquando da geração dos processos dos participantes, identifica-se, por cada *scope*, todas as falhas que nele podem ocorrer, e coloca-se a rotina por omissão de tratamento de falha em cada *scope* e para toda a falha que não tenha uma rotina declarada. Conforme já descrito, a rotina por omissão de tratamento de falhas deve, para além de terminar os *sub-scopes* activos, compensar os *scopes* terminados e lançar a falha para o *scope* pai. No lado esquerdo da figura 49 encontra-se as acções a realizar por esta rotina, em que as actividades a azul correspondem a actividades CBPEL, e que ainda serão decompostas em actividades BPEL. A rotina inicia-se por esperar pela recepção da mensagem de *Faulted* do coordenador. Essa mensagem visa alinhar todos os participantes acerca do início do processamento da rotina de tratamento da falha, de modo a ser possível a troca de mensagens entre eles. Depois da recepção dessa mensagem, resta a indicação de compensação dos *scopes* terminados com sucesso através da acção de *Compensate*, e do lançamento da falha para o *scope* pai através da acção de *Throw f*, em conformidade para com as acções por omissão já descritas.

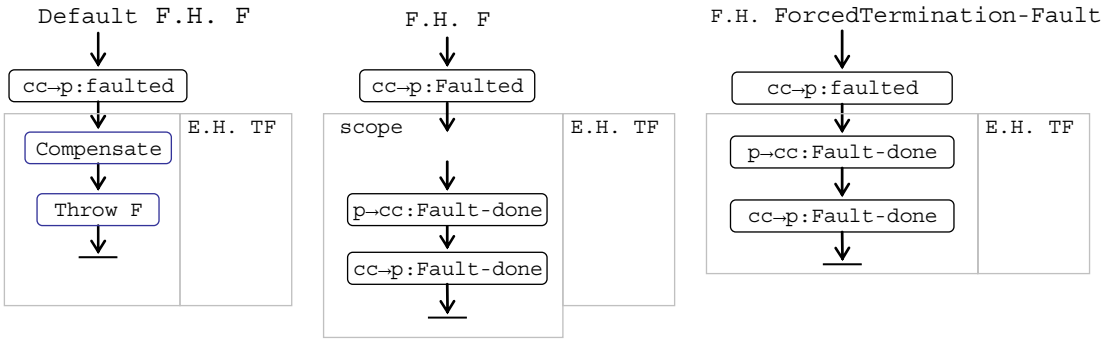


Figura 49 – Rotinas de tratamento de falhas

Tratamento normal de uma falha

Na parte central da figura 49 encontra-se a parte inicial e final de uma rotina de tratamento de falhas. Como já descrito o participante inicia o tratamento da falha esperando pela mensagem de *Faulted*. Pressupondo que o participante chega ao término do tratamento da falha, sem a ocorrência de qualquer falta, ele deve enviar a indicação que terminou sem falhas, através do envio da mensagem de *Fault-done* para o coordenador. Se o coordenador receber as mensagens de finalização sem falhas, de todos os participantes, envia para todos eles a mensagem *Fault-done*, indicando que todos terminaram sem falhas.

Ocorrência de falha no tratamento de uma falha

Caso algum dos participantes entre em falha, então o coordenador após receber essa indicação terá que enviar a indicação de falha para todos os outros participantes através de *Throw-fault F*. A recepção de *Throw-fault* conforme já descrito terá que ser realizada de forma assíncrona (num *Event handler*) na linguagem BPEL, porque os participantes podem encontrar-se em espera por mensagens do participante faltoso ou de outro. Pelo que se torna necessário a instalação de recepção assíncrona dentro do tratamento da falha. A linguagem BPEL requer que se crie um novo *scope* local para tal efeito. Esse é o motivo pelo qual aparece um *scope* com uma rotina assíncrona (Event Handler – E.H.) no tratamento de falha nas várias configurações da figura 49.

Tratamento de ForcedTermination

Um *scope* tem de poder receber a falha de *ForcedTermination*, pois esta é enviada pelo *scope* pai quando este entra em falha. A recepção desta falha resulta na falha do *scope*

corrente, o que por sua vez origina a propagação dessa falha para os *sub-scopes* activos. Essa falha deve ser apanhada por todos os *scopes*, podendo conter algum código no seu tratamento, mas não podendo lançar qualquer falha, uma vez que o *scope* pai já entrou em falha. No lado direito da figura 49 encontra-se as acções por omissão que cada participante deverá executar aquando da recepção da falha de *ForcedTermination*, e que consiste somente em receber a indicação de início de tratamento de falha (*Faulted*) e finalização da mesma (*Faul-done*).

Compensações no tratamento de falhas

Durante a execução de uma rotina de tratamento de falha, assim como numa rotina de compensação, é possível compensar os *sub-scopes* terminados com sucesso. A compensação pode ser efectuada: por evocação da actividade de *Compensate*, a qual compensa todos os *scopes* bem terminados; ou por evocação da actividade de "*Compensate Scope*" (*Compensate sc*), a qual compensa um determinado *scope*.

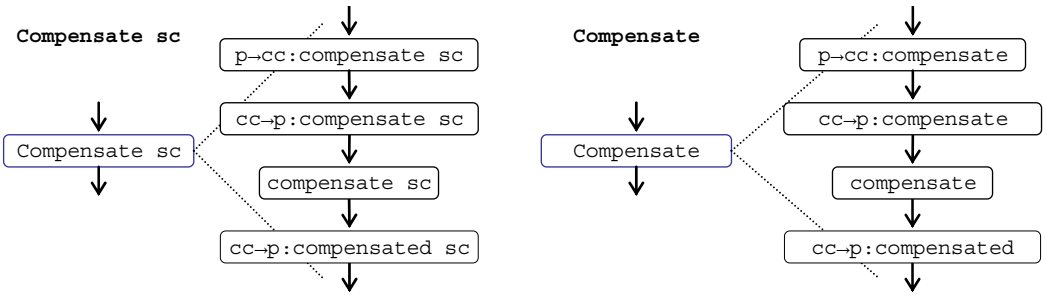


Figura 50 – Chamada a "*Compensate scope*"

Na figura 50 encontra-se o desdobramento das actividades de "*Compensate sc*" e "*Compensate*" da linguagem CBPEL, para as actividades que deverão constar nos processos dos participantes. Assim, no caso de "*Compensate sc*", a indicação de compensação de um *scope* inicia-se com o envio de uma notificação para o coordenador (p→cc: *compensate sc*), este quando receber essa notificação de todos os participantes, notifica-os (cc→p: *compensate sc*), de modo a evocarem a actividade de *compensate* ao *scope* (*compensate sc*) em questão. Caso a compensação termine sem falhas, o coordenador, enviará a mensagem de (cc→p: *compensated sc*). Caso ocorra algum falha na compensação do *scope* e que este a remeta para o *scope* pai, a falha é recebida no *Event Handler* de *Throw-fault f*. Na situação de se compensar todos os *scopes*, através da actividade CBPEL de "*Compensate*", resulta numa sequência idêntica de acções mas agora com "*Compensate*" nas mensagens. O coordenador evocará as compensações de

todos os *scopes* terminados, mas pela ordem inversa à ordem da sua terminação, e depois devolverá o resultado de sucesso ou de falha.

De modo a clarificar o protocolo, apresenta-se de seguida dois cenários com falhas envolvendo dois participantes.

Cenário de falha com throw

Este cenário, que se encontra presente na figura 51, contém os participantes A e B, e nele podem ser geradas as falhas f1, f2 e f3 durante a execução normal do *scope*, e a falha f4 na rotina de tratamento da falha f1. Como não são declaradas explicitamente rotinas de tratamento para as falhas f2 e f3, então será declarada, nos participantes, a rotina por omissão de tratamento de falhas (ver ilustração central da figura 49) para ambas as falhas. A falha f4 sendo gerada numa rotina de tratamento de falhas, é lançada para o *scope* pai do *scope* sc0, que caso não exista é o próprio processo que entra em falha.

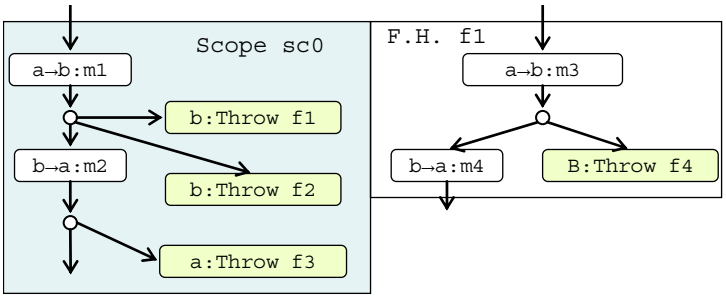


Figura 51 – Cenário de falha com *throw* – CBPEL

A figura 52 contém o fluxo do participante A. Após a criação do *scope* no coordenador (*Create* / *Created*) o participante A cria o *scope* localmente, notifica o participante B, procede à troca de mensagens com B, e depois pode gerar a falha f3 (*Throw f3*), ou terminar indicando sucesso para o coordenador (*Completed*). Considerando que o participante B gera a falha f1, e notifica o coordenador, este envia a mensagem *Throw-fault* de forma assíncrona para o participante A. A mensagem é recebida no "*event handler*" e é lançado a falha localmente por evocação de *Throw f1*, passando a execução para a rotina de tratamento dessa falha.

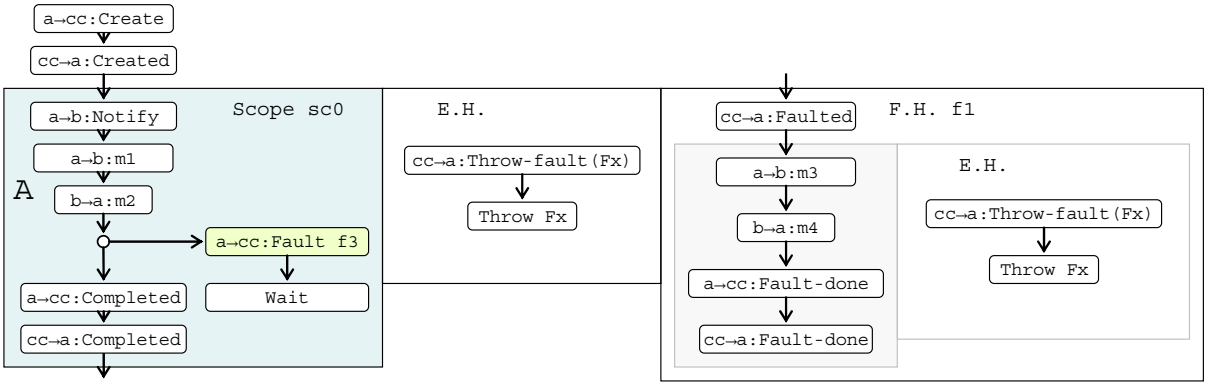


Figura 52 – Cenário de falha com *throw*: participante A

A rotina de tratamento da falha f1 inicia-se com a indicação de sincronismo do coordenador (cc→a: *Faulted*), depois prossegue com a criação de um *scope* local contendo: o código do participante (a→b: m3) e (b→a: m4), a indicação de terminação (a→cc: *Fault-done*); e recepção da notificação de rotina terminada com sucesso por parte do participante (a→cc: *Fault-done*). No caso de o participante B ter gerado a falha f4, o participante A receberia a notificação de falha no "*Event Handler" do scope* local, e lançaria a falha para a rotina de tratamento de f1, que por sua vez lançaria a falha para o *scope* acima.

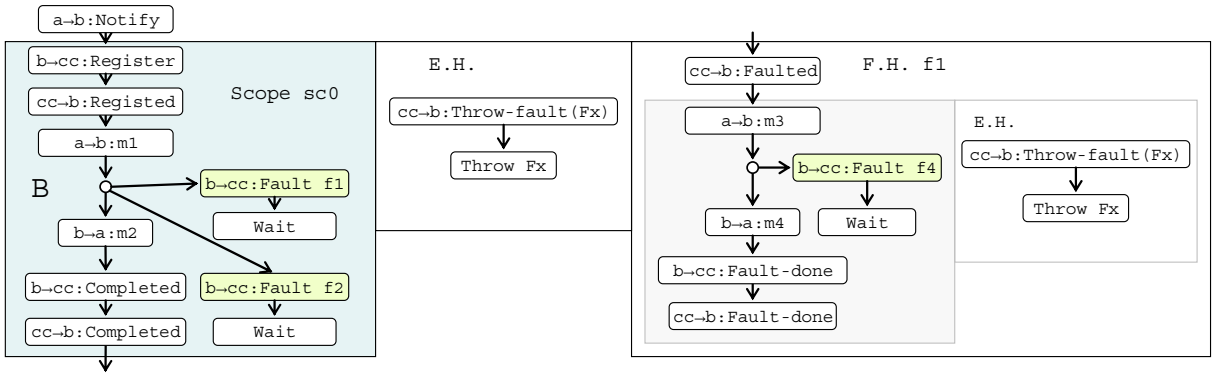


Figura 53 – Cenário de falha com *throw*: participante B

O processo do participante B, que se encontra presente na figura 53, é na sua essência semelhante ao processo do participante A. Como ele é o receptor da mensagem (a→b: m1) então inicia por receber, de A, a notificação de entrada no scope, e depois regista-se no coordenador como participante do *scope*. Depois pode gerar as falhas f1 ou f2 ou enviar a mensagem m2 ao participante A (b→a: m2). A parte final é idêntica à parte final do participante A, pois corresponde à finalização do *scope*. O "*event handler*" permite-lhe

receber as notificações de falha. A rotina de tratamento da falha f1 apenas difere pelo facto de que pode lançar a falha f4.

Cenário de falha com compensação de um *scope*

Este segundo cenário visa mostrar as acções a realizar em caso de haver uma chamada a uma compensação (*compensate*) dentro de uma rotina de tratamento de falha.

O cenário, que se encontra descrito na figura 54, contém os participantes A e B, e tem a seguinte evolução: o participante A envia a mensagem m1 a B; os dois participantes interagem num *sub-scope* sc1; depois o participante B ou gera a falha f1, ou envia a mensagem m2 para A. A rotina de tratamento da falha f1 contém a evocação de compensação do *scope* sc1, por parte dos dois participantes.

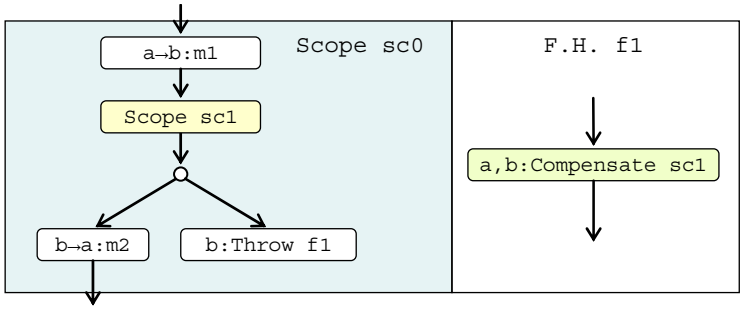


Figura 54 – Cenário de falha com *Compensate sc*: CBPEL

O código do participante A pode ser observado no lado esquerdo da figura 55. As acções do *scope* são semelhantes às do cenário anterior, excluindo o lançamento da falha, e incluindo o *sub-scope* sc1. A rotina de tratamento da falha f1 contém além das acções iniciais e finais da rotina, as acções relativas à actividade *Compensate sc1*, e que são: envio de indicação de compensação do *scope* sc1 ao coordenador (*a→cc: compensate sc1*); recepção da notificação para iniciar a compensação (*cc→a: compensate sc1*); evocação da compensação (*compensate sc1*); e esperar pela notificação do coordenador de compensação bem sucedida (*cc→a: compensated*). O *Event Handler* associado à rotina de compensação permite receber qualquer falha enviada pelo coordenador.

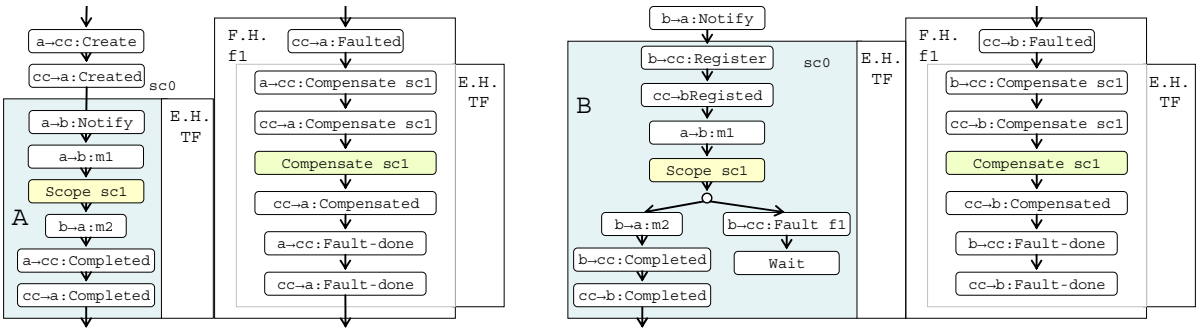


Figura 55 – Cenário de falha com *Compensate sc*: participantes

O código do participante B pode ser observado no lado direito da figura 55. A execução do *scope sc0* será semelhante à do participante B do cenário anterior, não tendo contudo o lançamento da falha f2, e contendo o *sub-scope* sc1 e o lançamento da falha f1 (*Fault f1*). A rotina de tratamento da falha f1 é análoga à rotina de tratamento da mesma falha, neste cenário, no participante A.

5.2.3 Compensações

Nesta secção descreve-se a execução das rotinas de compensação de *scopes*, primeiro na perspectiva do coordenador, e segundo na perspectiva dos participantes.

A compensação de um *scope* visa executar as acções que anulem ou desfaçam as acções efectuadas durante a execução normal do *scope*. A compensação só fica activa para execução após o término, da execução normal do *scope*, sem a ocorrência de falhas.

5.2.3.1 Compensações no lado do coordenador

Na figura 56 encontra-se o diagrama de estados do protocolo, do lado do coordenador, relativo à parte da execução da rotina de compensação de um *scope*.

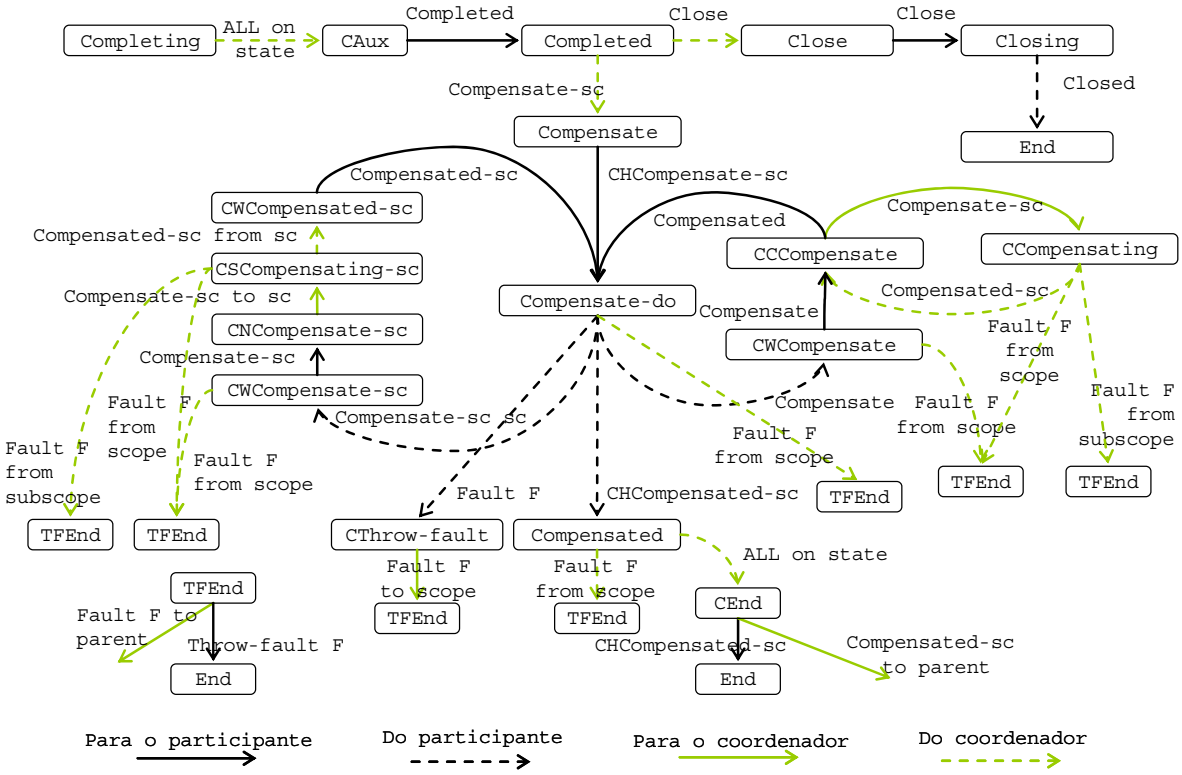


Figura 56 – Estados relativos à compensação de um *scope*

Seguidamente apresenta-se a descrição dos estados presentes na figura 56:

- Estado *Compensate*: este é o estado inicial que sucede à indicação de compensação que é enviada pelo *scope* pai. Neste estado é enviada a mensagem de *CHCompensate-sc* em cada um dos participantes no *scope*.

- Estado *compensate-do*: Este é o principal estado de execução da rotina de compensação. Neste estado podem ocorrer as seguintes situações: fim de compensação para um participante, quando um participante envia a mensagem de CH*Compensated-sc*, transitando ele para o estado *Compensated*; falha na compensação, quando um participante envia uma falha (*Fault F*), transitando para o estado C*Throw-fault*; falha na compensação, vinda de um outro participante, aquando da recepção da notificação de "*Fault F from scope*" do próprio coordenador, prosseguindo a execução no estado *TFEnd*; e entrada em compensação de: um *scope* terminado, por recepção de *Compensate-sc*, e transição para CWCompensate-sc; ou de todos os *scopes* terminados, por recepção de *Compensate*, e transição para o estado de CW*Compensate*

- Estados *CWCompensate-sc*, *CNCompensate-sc*, *CSCompensating-sc*, e *CWCompensated-sc*: estes estados, são relativos à compensação de um *scope*, e são semelhantes aos estados com o mesmo nome, mas iniciados por *F*, existentes no tratamento de falhas.

- Estados *CWCompensate*, *CCCompensate* e *CSCompensating*: estes estados, são relativos à compensação de todos os *scopes* bem terminados, e são semelhantes aos estados com o mesmo nome, mas iniciados por *F*, existentes no tratamento de falhas.

- Estado *CThrow-Fault*: este é o estado para o qual transita o participante que gera uma falha. A falha é enviada ao coordenador para ser entregue a todos os participantes no *scope*. A falha será depois propagada para o *scope* pai.

- Estado *Compensated*: este estado corresponde à indicação de que a compensação terminou correctamente para os participantes que nele se encontrem. O coordenador deve esperar por receber essa indicação de todos os participantes, para poder concluir que a compensação terminou de forma correcta. Quando tal acontecer, o coordenador notifica com a mensagem de "ALL on state" e os participantes transitam para o estado *CEnd*.

- Estado CEnd: neste estado é enviado para cada participante a mensagem de CHCompensated-sc. Também é enviada a notificação de *Compensated-sc* para o *scope* pai do *scope* corrente., indicando que a compensação terminou sem falhas.

- Estado TFEnd: este estado destina-se a enviar a indicação de falha para o participante (*Throw-fault F*), e a notificar o *scope* pai da falha (*Fault F to parent*).

5.2.3.2 Compensações no lado do participante

A compensação de um *scope* é activada sempre após a ocorrência de uma falha. Em que, no tratamento dessa falha foi solicitado a compensação do *scope*, através da actividade de "*Compensate sc*" ou de *Compensate*, quer de forma directa ou indirecta por intermédio de rotinas de tratamento de falhas de *scopes* intermédios. Uma das rotinas de tratamento de falhas também pode evocar a compensação de um *scope* intermédio, que por sua vez pode evocar a compensação de outro *scope*.

No lado esquerdo da figura 57 encontra-se as acções principais referentes à compensação de um *scope* do lado de um participante. A rotina de compensação (em todos os participantes) deve começar por esperar por uma mensagem de sincronismo (cc→p: *CHCompensate-sc*) do coordenador antes de executar qualquer código. Depois de receberem essa mensagem todos os participantes estão síncronos e portanto podem executar o código acordado no processo comum CBPEL. Quando a execução das acções de compensação terminar, os participantes devem notificar o coordenador enviando a mensagem de *CHCompensated-sc* (p→cc: *CHCompensated-sc*). Caso todos os participantes tenham enviado a mensagem *CHCompensated-sc*, o coordenador dá a compensação como bem sucedida e envia a mensagem de *CHCompensated-sc* (cc→p: *CHCompensated-sc*) para todos os participantes. Caso algum dos participantes tenha enviado, ao coordenador, uma indicação da ocorrência de erro na execução da rotina de compensação, então o coordenador deve enviar a mensagem de *Throw-fault F* (*cc→p: Throw-fault F*) para todos os participantes. Os participantes, ao receberem esta mensagem como mensagem assíncrona, num *Event Handler*, devem lançar (*throw*) a falha, a qual suspenderá a execução e será remetida para o *scope* pai. De modo semelhante ao ocorrido no tratamento de falhas, a rotina de compensação no lado dos participantes, tem de criar um *sub-scope* local para poder receber as mensagens assíncronas de falha.

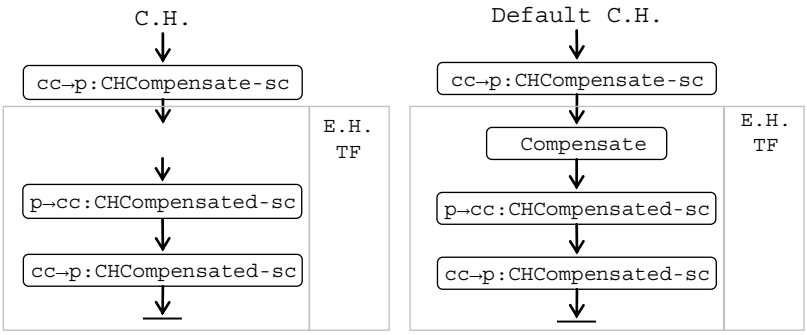


Figura 57 – Rotina de compensação com finalização com sucesso e por omissão

No lado direito da figura 57 encontra-se as acções que devem ser realizadas na rotina de compensação por omissão, ou seja, quando não é declarada nenhuma rotina de compensação. Essas acções são necessárias, porque qualquer *scope* na linguagem BPEL, possui uma rotina de compensação por omissão, que chama a actividade BPEL de *compensate*. A existência da rotina por omissão permite controlar o sincronismo da execução da compensação desse *scope* e dos seus *sub-scopes*. Na rotina de compensação por omissão, não será possível a ocorrência de erro, mas será possível que o *compensate*

evoque a compensação de um *scope* onde ocorra um erro. A recepção de erro termina o *compensate*, e reenvia o erro para o *scope* acima dele.

Seguidamente apresenta-se um cenário de modo a clarificar algumas situações referentes ao protocolo durante a compensação de um *scope*.

Cenário de compensação com evocação de *compensate* e falha

O cenário que se encontra no centro da figura 58 é um cenário CBPEL entre os participantes A e B, contém um *scope* com o *sub-scope* sc1, e a rotina de compensação do primeiro *scope*. Nessa rotina de compensação, ambos os participantes começam por evocar a compensação do *sub-scope* sc1, e depois procedem à troca das mensagens m3 e m4, onde o participante B pode gerar uma falha mediante o conteúdo da mensagem m3.

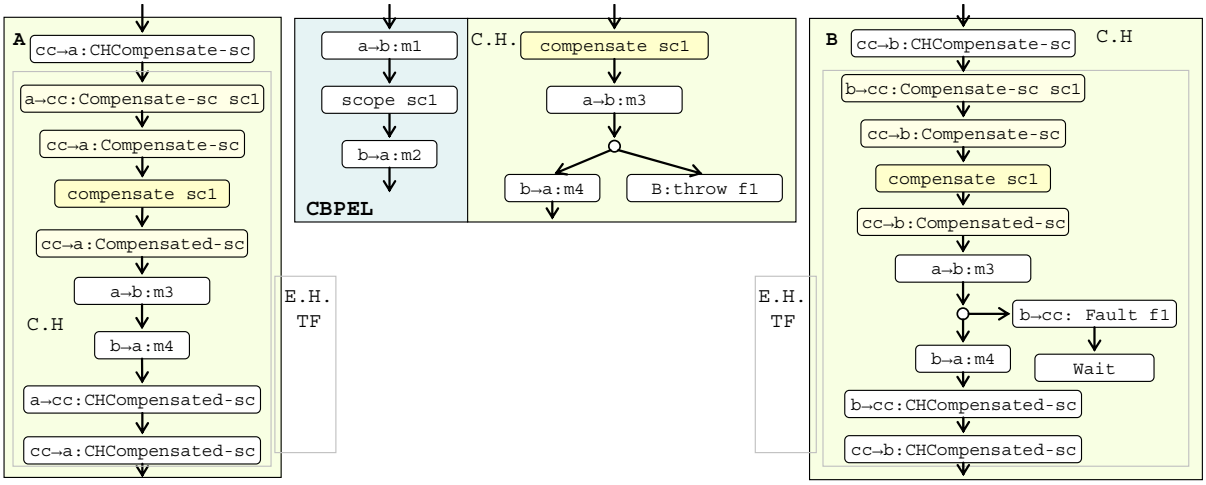


Figura 58 – Cenário de compensação com falha de um dos participantes

Do lado direito da figura 58 encontra-se a rotina de compensação do participante B. Esta rotina tem os seguintes passos: esperar pela indicação de prosseguimento por parte do coordenador (cc→b: *CHCompensate-sc*); início a acção de *compensate sc1*, com envio da mensagem de *Compensate-sc sc1* para o coordenador; esperar pela recepção da indicação de *compensate-sc* vinda do coordenador (cc→b*: Compensate-sc*); evocar a actividade local de *compensate sc1*; esperar pela indicação para prosseguir de parte do coordenador (cc→b*: Compensated-sc*); receber a mensagem m3 de A (a→b*:* m3); e enviar a mensagem m4 para A (b→a: m4); ou lançar a falha f1 (b→cc*: Fault f1,* e *wait*). Caso tenha ocorrido uma falha na compensação do *scope* sc1, então o participante B recebe essa informação através do *event handler* do *scope* local da rotina de compensação. A rotina de compensação do participante A encontra-se no lado esquerdo da figura 58 e é semelhante à do participante B, apenas não contendo o lançamento da falha.

5.2.4 Participação em *scopes* encaixados

Esta secção visa clarificar a problemática associada à participação em *scopes* encaixados (*nested scopes*).

De modo a clarificar a terminologia considera-se que: um *scope* x0 é um *sub-scope* do *scope* y0, se x0 é um *scope* declarado dentro de y0; um *scope* x1 é um *sub-scope* directo do *scope* y1, se x1 for um *sub-scope* de y1 e entre eles não existir nenhum outro *scope*; um *scope* x2 é um *sub-scope* indirecto do *scope* y2, se x2 for um *sub-scope* de y2 e entre eles existir pelo menos um outro *scope*.

Até agora foi implicitamente considerado que todos os participantes de um *scope* também participavam nos seus *sub-scopes*. Contudo pode haver as seguintes situações: participantes que não participem em certos *sub-scopes*, novos participantes nos *sub-scopes*; e participantes que não participem nos *sub-scopes* directos mas que participem em *sub-scopes* indirectos.

A participação em *sub-scopes* requer que os participantes tenham a capacidade de processar as mensagens que podem transitar entre *scopes* e seus *sub-scopes* directos. São identificadas as seguintes comunicações de um *scope* para um seu *sub-scope directo*: iniciação; falha de *ForcedTermination*; e *Compensate scope*, e são identificadas as seguintes comunicações do *sub-scope* directo para o *scope*: fim de execução; lançamento de falhas; *ForcedTerminated*; e *Compensated*.

Nas duas subsecções seguintes são abordados os casos da participação em *sub-scopes* directos e da participação em *sub-scopes* indirectos, sendo a última designada de participação em *scopes* intercalados.

5.2.4.1 Participação em *sub-scopes* directos

Para um *scope* e um seu *sub-scope* directo podem ocorrer as seguintes situações em relação aos participantes envolvidos: existirem participantes que figuram em ambos os *scopes*; existirem participantes que só figuram no *scope*; e existirem participantes que só figuram no *sub-scope*. Seguidamente apresenta-se a problemática que se coloca em cada das referidas situações:

– Participantes que figuram em ambos os *scopes*:

Para os participantes de figuram nos dois *scopes*, o protocolo de coordenação fornece-lhes toda a informação acerca do estado dos mesmos. Contudo, esta situação tem na fase inicial do *sub-scope* um ponto crítico, pois a entrada no *sub-scope* será efectuada em diferentes alturas pelos vários participantes. Nesta fase, se for lançada uma falha, poderão não estar todos os participantes registados no *sub-scope*, o que implica que um *fault handler* associado a essa falha só poderá contar com os participantes até então registados.

A fase final do *sub-scope* não apresenta qualquer problema de pois o protocolo de coordenação indicará o sucesso ou o insucesso no *sub-scope* para todos os participantes, e somente quando todos o tiverem terminado. No entanto há sempre a questão da ausência de sincronismo na saída do *scope* local em cada participante, que no caso de ocorrência de uma falha, pode apanhar alguns participantes ainda dentro do *scope* e outros já fora dele. Essa situação ficaria resolvida caso a saída do *scope* fosse indivisível da recepção da mensagem de finalização vinda do coordenador.

– Participantes que figuram somente no *scope*:

Os participantes que não figurem no *sub-scope* não devem estar relacionados com ele. Pode-se considerar o *sub-scope* como privado dos outros participantes. Deste modo, os participantes que não figurem no *sub-scope* não devem dar a indicação de compensação do mesmo. Somente os participantes que também participem no *sub-scope* é que devem dar a indicação da sua compensação. O coordenador sabe quais os participantes do *scope,* que participaram no *sub-scope*, confrontando a lista de participantes dos dois *scopes*.

Uma outra situação possível consiste na participação opcional num *sub-scope*. Esta situação implica que o participante opcional tenha de possuir um código condicional quer na indicação de compensação, quer na participação na rotina de compensação do *sub-scope*.

– Participantes que figuram somente no *sub-scope*:

Por agora vamos considerar que os participantes que não existem no *scope*, são novos participantes e que portanto iniciam e terminam a sua execução dentro do *sub-scope*. A participação em *scopes* intercalados será abordada da subsecção seguinte. Na presente situação, os novos participantes, uma vez registados no *scope*, receberão toda a informação sobre o progresso do *scope*. Caso haja, a ocorrência de uma falha no *scope* acima, e enviada a falha de *ForcedTermination*, ela será recebida por todos os participantes no *event handler*, e depois gerada a falha. Após a terminação do fluxo normal do *sub-scope*, o *scope* pode pretender compensar o *sub-scope*. Nesta situação os novos participantes no *sub-scope* já terminaram a sua execução, pelo que é necessário correr a rotina de compensação de um processo terminado. A linguagem BPEL permite a compensação de um processo terminado com sucesso, mas remete para as plataformas de suporte à execução a implementação dessa funcionalidade. O lançamento de falhas geradas no *sub-scope,* tem o processamento normal ao de uma qualquer falha.

5.2.4.2 Participação em *scopes* intercalados

Nesta subsecção será abordada a problemática resultante da participação em *scopes* intercalados. Na figura 59: o *scope* sc0 tem a participação dos participantes A, B e C; o *scope* sc1 tem a participação de A e B; e o *scope* sc2 tem a participação de B e C. O participante C participa, portanto, somente nos *scopes* sc0 e sc2.

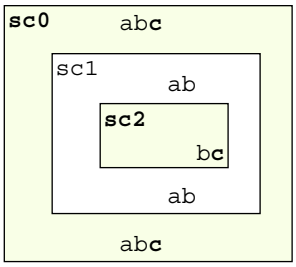


Figura 59 – Participação em *scopes* intercalados

Considerando que cada participante só conterá no seu processo os *scopes* em que participa, então caso fosse gerada uma falha em sc2, e no caso de essa falha não ser tratada nesse *scope*, essa falha seria recebida em sc1 pelos participantes A e B, e em sc0 no participante C. Caso no *scope* sc1 se apanhasse e tratasse a falha, ter-se-ia uma inconsistência no *scope* sc0, pois ele teria recebido a indicação de ocorrência de uma falha

que entretanto já foi tratada. Tal provocaria uma incoerência, uma vez que os vários participantes envolvidos teriam noções diferentes do estado do processo global. Uma forma de evitar essas incoerências consiste em entregar a falha lançada no *scope* sc2 somente no *scope* sc1. Tal implica que o participante intercalado (participante C) tenha de possuir um *scope*, com a função de *scope* tampão, entre sc0 e sc2. Deste modo se a falha for tratada em sc1, o *scope* tampão de C é notificado para terminar o tratamento da falha sem a lançar para o *scope* acima, que é sc0.

No caso de ser gerada uma falha em sc1 e de no seu tratamento ser pedido para compensar sc2, existe a necessidade de haver um receptor do pedido de compensação. Essa recepção pode ser realizada pelo envio de uma falha para o *scope* tampão, e na sua rotina de tratamento solicitar a compensação de sc2.

O *scope* tampão vai portanto permitir que o participante tenha conhecimento dos eventos para si significativos, sem contudo tomar parte activa no *scope*. Desse modo pode acompanhar de forma correcta a evolução do estado do processo global. A esse papel de participação passiva no *scope* é aqui denominado de papel de observador do *scope*.

Seguidamente é apresentada a discussão das implicações da utilização do papel de observador, dividindo as possibilidades em dois cenários: intercalamento de um *scope*; e intercalamento de mais de um *scope*.

Participação em *scopes* intercalados de um só *scope*

Conforme já descrito, de modo a um participante em *scopes* intercalados poder ser receptivo aos acontecimentos no *scope* intermédio, ele terá que ser colocado como observador desse *scope* intermédio. Consistindo o papel de observador em ser-se somente notificado das acções relevantes para o devido acompanhamento do processo global. Assim, um participante com um *scope* intercalado deve poder receber as notificações de: início de *scope* intermédio, falha no *scope* intermédio, compensação do *scope* intermédio e fim de *scope* intermédio. Seguidamente apresenta-se o processamento destas notificações considerando um encadeamento de três *scopes* à semelhança do cenário apresentado na figura 59, mas em que os *scopes* sc0, sc1 e sc2 são genericamente denominados de *scope* superior, *scope* intermédio e *scope* inferior:

- Notificação de início de *scope* intermédio: Como esta notificação será discutida na participação intercalada por mais que um *scope*, remete-se até lá a sua discussão.

- Notificação de falhas no *scope* intermédio: A ocorrência de uma falha no *scope* intermédio enquanto o *scope* inferior se encontra em execução, implica o envio da falha *ForcedTermination* para o *scope* inferior. Do mesmo modo que a ocorrência de uma falha no *scope* superior enquanto o *scope* intermédio estiver activo implica a entrega de *ForcedTermination* no *scope* intermédio. Deste modo o coordenador ao receber a indicação de uma falha no *scope* intermédio, deve notificar a ocorrência da falha a todos os participantes e observadores no referido *scope* e depois enviar a falha *ForcedTermination* para todos os *scopes* inferiores activos. A notificação da ocorrência da falha aos observadores, tendo em conta que estes não conhecem o conteúdo do *scope* intermédio, pode ser realizada pelo envio de uma falha genérica com esse significado. A ocorrência de uma falha no *scope* intermédio pode implicar a compensação do *scope* inferior, ou implicar o encerramento definitivo desse *scope*, pelo que é pertinente a sua recepção.

- Notificação de compensação de *scope* intermédio: esta notificação, a existir, é lançada no *scope* superior, dentro de uma rotina de tratamento de uma falha, ou dentro de uma rotina de compensação. Os observadores têm de poder receber a indicação de compensação, e dentro dela poder receber as notificações significativas, nomeadamente: a notificação de compensação do *scope* inferior; e a indicação de fim de compensação, com ou sem falha.

- Notificação de fim de *scope* intermédio: quando o *scope* intermédio terminar a sua execução, o coordenador deve notificar os seus observadores de modo a que estes prossigam a execução para o *scope* superior.

Deste modo, a declaração de um *scope* no qual se é observador deve possuir as seguintes rotinas: de falha (*fault-handler*), para todas as falhas que o *scope* inferior lhe possa enviar; de falha genérica (*fault-handler*), devido às falhas que podem ocorrer nesse *scope*; de falha *ForcedTermination* (*fault-handler*), devido à possibilidade de envio desta falha por parte do *scope* superior; e de compensação (*compensation-handler*), de modo a permitir a compensação do *scope* inferior aquando da compensação do *scope* observado. Além dessas, ainda deve declarar os *event-handlers* necessários para poder receber as indicações de falha.

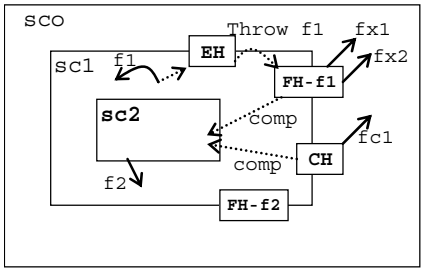
Cada *fault-handler*, após a sua iniciação, deve poder receber as notificações de: *compensate-scope*, *compensate*, e *throw fault*, na medida em que estas existam no *fault-handler* declarado no processo. Por exemplo, se no cenário da figura 59 o *fault-handler* de

uma falha F1 no *scope* sc1, contiver a actividade de "*compensate sc2*" então deve aparecer no *fault-handler* de um observador de sc1 (participante C) a hipótese de receber "*compensate sc2*".

O *fault-handler* de *ForcedTermination* terá o mesmo comportamento que um *fault-handler* normal e como imposto pela linguagem BPEL não poderá lançar falhas, pois o *scope* acima já entrou em falha.

O *compensation-handler* deverá poder receber as mesmas notificações referidas para um *fault-handler*.

Na figura 60 encontra-se do lado esquerdo um exemplo com o *scope* observado sc1, que pode lançar a falha f1, e o *scope* inferior pode lançar a falha f2. No *scope* sc1, o *fault-handler* de f1 (FH-f1) pode lançar as falhas fx1 e fx2, e compensar sc2 quer por *compensate-sc* quer por *compensate*. Em sc1, os observadores também teriam de possuir a rotina de compensação de modo a permitirem a compensação de sc2, aquando desse pedido vindo do *scope* sc0. No lado direito da figura 60 encontra-se uma listagem em pseudocódigo das acções que o *fault-handler* da falha f1 no *scope* sc1 teria que efectuar.




```
sc1:Fault-handler f1

  receive cc:throw-fault-notify
  fim = false
  while(!fim)
     pick
        onMessage cc:throw-fault-fx1
            throw fx1
        onMessage cc:throw-fault-fx2
            throw fx2
        onMessage cc:compensate sc2
            compensate sc2
        onMessage cc:compensate
            compensate
        onMessage cc:faul-done-notify
            fim = true
```

Figura 60 – Falhas e compensações no *scope* observado

Participação em *scopes* intercalados por mais de um *scope*

Extrapolando a metodologia anterior para o caso de haver mais do que um *scope* a intercalar os *scopes* em que um participante intervém, teríamos que colocar esse participante como observador de todos esses *scopes* intermédios. Deste modo, o participante possuiria a noção exacta do processo global, permitindo o acompanhamento de falhas e compensações em todos os *scopes* intermédios. Esta poderá ser uma solução,

mas não será a solução mais desejável, pois o participante intercalado acompanhará o desenrolar de acções que não lhe dizem directamente respeito, e também porque o seu processo poderá ficar desnecessariamente extenso.

Uma melhor solução consiste colocar uma zona tampão composta por dois *scopes* tampão, deste modo evita-se o conhecimento desnecessário de acções e minimiza-se a zona tampão. Esta é a situação apresentada na figura 61, onde do lado esquerdo temos um processo com 6 *scopes* em cadeia, e no lado direito temos o processo de um participante que intervém somente nos *scopes* sc0 e sc5, com somente dois *scopes* tampão.

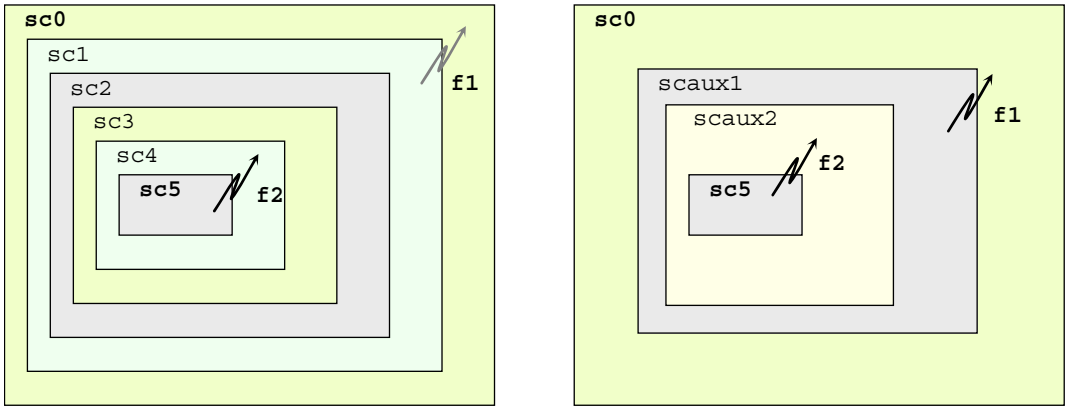


Figura 61 – Processo com scopes encaixados, e participante com dois scopes tampão

A escolha de dois scopes tampão, e não apenas um, foi tomada com base na preferência em evitar alterar o conteúdo dos *scopes* externo sc0 e interno sc5. Assim, o *scope* tampão mais interno (scaux2) destina-se à interacção com o *scope* interno, enquanto que o *scope* tampão mais externo (scaux1) destina-se à interacção com o *scope* externo. Vejamos as acções necessárias para as situações que se podem colocar:

- Falha no *scope* interno: uma falha gerada no *scope* interno sc5 e nele não tratada, será lançada para o *scope* tampão interno, sendo neste apanhada.

- Falha de *ForcedTermination*: quando o *scope* sc4 tiver que entregar a falha de *ForcedTermination* a sc5, então o coordenador deverá enviar a indicação de falha para uma rotina assíncrona no *scope* tampão interno. Na qual será gerada e apanhada a falha de *ForcedTermination*, sendo depois propagada para sc5.

- Falha na zona tampão: Se ocorrer uma falha na zona tampão, que não interaja com o *scope* superior, essa falha não terá qualquer interferência com o participante intercalado. Se a falha for lançada para o *scope* superior, nesse caso a

falha pode ser entregue numa rotina assíncrona no *scope* tampão superior, para que depois seja gerada a falha, que não será apanhada e poder ser entregue no *scope* superior do participante intercalado, ficando a indicação que a zona tampão terminou com falha.

– Compensações: Caso seja solicitado para compensar o *scope* sc5, esse pedido terá que passar pelo *scope* tampão inferior, pelo terá que ser realizado através da recepção de uma notificação assíncrona que desencadeie uma falha, e no tratamento da qual se evoca a compensação do *scope* sc5.

Clarificação da entrada em sub-scopes

A discussão acerca da entrada nos *scopes* e *sub-scopes* foi remetida para esta subsecção dado que agora se pode discutir as situações de entrada directa em *sub-scope*, entrada em *scope* intercalado, e também a situação dos observadores.

Quando um participante normal, ou seja que não é um observador, inicia a execução de um *scope*: primeiro terá que criar um *scope* junto do coordenador, o qual lhe devolve a identificação do novo *scope*; depois pode enviar mensagens para outros participantes se juntarem a ele; e também poderá criar *sub-scopes* dentro do *scope* criado.

Quando o participante, envolvido num *scope*, pretender enviar uma mensagem a outro participante que ainda não tenha intervindo dentro desse *scope*, ele terá que lhe passar a identificação do novo *scope* e terá que haver um registo no coordenador do novo participante.

São identificadas quatro possibilidades de efectuar essas tarefas: enviar directamente ao participante a mensagem com a identificação do *scope* em anexo; envio da mensagem através do coordenador; envio de uma mensagem de entrada por intermédio do coordenador, seguido do envio directo da mensagem; e envio directo de uma mensagem de entrada, seguido do envio directo da mensagem.

Vejamos então as quatro possibilidades.

– Envio directo da mensagem com a identificação do *scope* em anexo:

  Esta solução tem dois problemas: implica alterar a mensagem a enviar; e que o destinatário receba uma mensagem, que deveria receber dentro de um determinado *scope*, mas sem estar registado nesse *scope*.

– Envio da mensagem através do coordenador:

  Esta solução permite o registo prévio no coordenador do destinatário no novo *scope*. Esta solução contudo tem a desvantagem de as mensagens passarem pelo coordenador, o que pode ser visto como uma intromissão nos detalhes das conversações entre participantes e dar possibilidade de acesso não autorizado. Essa situação poderia ser atenuada com a cifra do conteúdo dessas mensagens, mas tal adicionaria complexidade à plataforma de suporte.

– Envio de uma mensagem de entrada via o coordenador, seguido do envio directo da mensagem de dados:

  Esta situação é idêntica à anterior mas separa a mensagem a enviar em duas acções: primeiro, o envio da notificação de entrada no *scope*, e segundo, o envio da devida mensagem. Assim, faz-se o registo dos participantes no coordenador, mas o conteúdo da mensagem não passa pelo mesmo. Esta situação tem o revés de ser outro participante a registá-lo no coordenador, e tal facto ser registado antes de o participante saber que o vai fazer.

– Envio directo de uma mensagem de entrada, seguido do envio também directo da mensagem:

  Esta solução é semelhante à anterior, mas a mensagem de notificação é enviada directamente ao destinatário, ao que ele após receber esta mensagem entra no novo *scope*, regista-se no coordenador, e depois fica receptivo à troca de mensagens no *scope*. Esta situação tem o inconveniente de o participante entrar no *scope* local sem estar registado no *scope* no coordenador. Mas como esta situação poderá ser anulada com a plataforma a disponibilizar a execução indivisa da entrada no *scope* e do registo no coordenador. Esta situação, por apresentar menores problemas, foi a escolhida para este trabalho.

No caso de participantes intercalados, eles só recebem a notificação de entrada no *scope* inferior final, representando para ele a entrada no *scope* tampão como observador, seguido da entrada no *scope* final. Caso a zona tampão tenha mais que um *scope*, então o participante considera a entrada nos dois *scopes* tampão.

5.2.5 Suporte por um protocolo de coordenação existente

Este ponto visa discutir a utilização dos protocolos de coordenação apresentados na secção 3.4, para o suporte ao protocolo de coordenação da linguagem CBPEL.

Protocolo ebXML / BPSS

A utilização do ebXML / BPSS para suporte ao protocolo de coordenação proposto apresenta as seguintes dificuldades: o protocolo não é estruturado ao bloco; e o protocolo é ponto-a-ponto, ou seja, não tem a noção de coordenador. Tais factores implicam que seria necessário implementar todo o protocolo de coordenação com base nas interacções ponto-a-ponto entre o coordenador e cada participante.

Protocolo OASIS BTP

O facto do protocolo de coordenação OASIS / BTP não suportar a noção de blocos encaixados limita a sua adopção para o suporte ao protocolo de coordenação proposto, pois requer que essa funcionalidade seja completamente implementada de forma complementar.

Protocolo WS-CAF

Para se suportar o protocolo de coordenação proposto com base no WS-CAF, ter-se-á que utilizar o seu modelo de acções de longa duração (LRA). Este modelo suporta a semântica fazer-compensar, possui a noção de blocos encaixados, mas não garante a noção de estado global nas compensações e tratamento de falhas, pois são realizadas a uma só fase.

Protocolo WS-CT

A utilização do WS-CT como suporte ao protocolo de coordenação proposto requer a utilização do seu modelo base de coordenação WS-BusinessActivity (WS-BA) e deste modelo utilizar o protocolo BusinessAgreementWithParticipantCompletion, pois é o protocolo em que a finalização da transacção é indicada pelos participantes e não pelo coordenador. Na figura 62 mostra-se um resumido diagrama de estados deste protocolo, contendo as interacções entre um *scope* e o seu *scope* pai, e também com o participante.

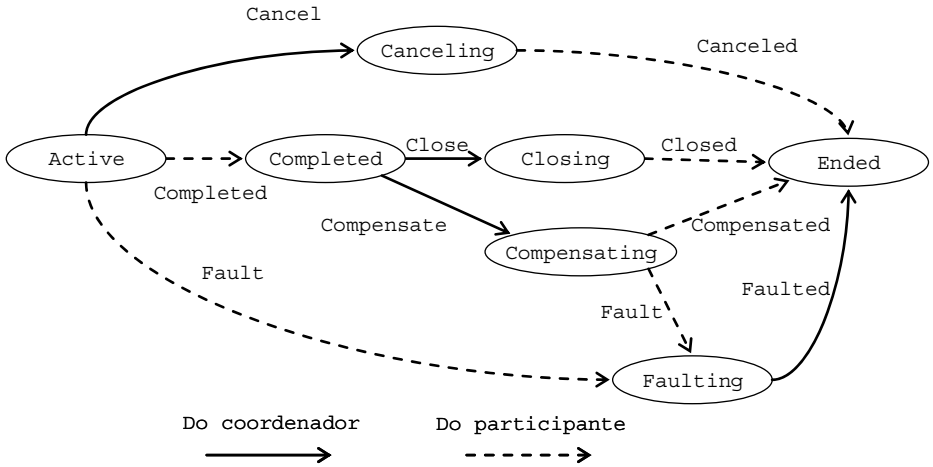


Figura 62 – WS-BusinessActivity com término pelo participante

Este protocolo apesar de não ter um estado análogo ao estado de *Completing* apresenta uma sequência de estados semelhante aos estados principais do coordenador CBPEL, presentes nas figuras 45, 47 e 56.

Conclusão

Dos protocolos existentes, o protocolo WS-CT com o seu modelo base WS-BA e o protocolo BusinessAgreementWithParticipantCompletion apresenta-se como a melhor solução de suporte ao protocolo apresentado neste trabalho.

5.2.6 Problemas na linguagem BPEL no suporte aos processos CBPEL

Uma vez que a linguagem BPEL não foi concebida para descrever processos inseridos numa coordenação, a sua utilização para tal efeito apresenta os seguintes problemas ou limitações:

- Falta de integração entre a coordenação e os processos participantes. O que leva: à recepção da indicação de falhas como eventos assíncronos; à criação de *scopes* internos nas rotinas de tratamento de falhas e compensação; à declaração explicita de rotinas de tratamento por omissão de falhas e compensação; e à utilização explicita de *scope* tampão, aquando da participação intercalada.

- Conflitos na recepção das mensagens de erro. Dado que cada *scope* tem de poder receber todas as mensagens de erro que nele podem ocorrer, então cada *scope* terá que ficar receptivo a essas mensagens. Contudo numa situação de *scopes*

encaixados será muito provável que mais que um *scope* possa receber a mesma mensagem de erro, o que leva a conflito na linguagem BPEL, pois num dado instante não pode haver duas recepções activadas com o mesmo remetente, tipo de mensagem e campo de correlação.

5.3 Descrição da Linguagem CBPEL – Scopes, Falhas e Compensações

Este subcapítulo visa complementar a descrição da linguagem CBPEL, já apresentada no capítulo anterior, acrescentando a parte relativa a *scopes*, lançamento e tratamento de falhas, compensações, e rotinas assíncronas. Serão, portanto, abordados os seguintes elementos: *scope*, *eventHandler*, *faultHandler*, *throw*, *compensationHandler* e *compensate*, sendo para cada um deles apresentado, primeiro, a sua descrição na linguagem BPEL e depois o seu congénere, se o houver, na linguagem CBPEL.

5.3.1 Scope

Um *scope* na linguagem BPEL:

A construção *scope* permite definir um contexto com: declaração de variáveis locais; conjuntos de correlação; rotinas (*handlers*) de tratamento de falhas, de compensação de tratamento de eventos assíncronos e uma actividade que será o fluxo normal do *scope*

Descrição de um *scope* na linguagem BPEL:

```
<scope variableAccessSerializable="yes|no" standard-attributes>
    standard-elements
    <variables/>?
    <correlationSets/>?
    <faultHandlers/>?
    <compensationHandler/>?
    <eventHandlers/>?
    activity
</scope>
```

Um *scope* na linguagem CBPEL:

O elemento de *scope,* na linguagem CBPEL, irá permitir a existência dos mesmos elementos que o seu congénere na linguagem BPEL.

Neste ponto apenas será abordado o *scope* no seu todo, sendo a partes relativas às rotinas de tratamento de falhas, eventos assíncronos e compensação, abordadas em pontos próprios.

A declaração de variáveis e conjuntos de correlação são identificadas como necessários, pois permitem a declaração de novas entidades no processo, evitando a declaração global de todas as entidades necessárias. A declaração de variáveis e conjuntos de correlação será idêntica à declaração destes elementos já enunciada na subsecção 4.2.1.

O atributo *VariableAccessSerializable* quando activo implica que: o acesso a variáveis partilhadas é realizado dentro de controlo de acesso concorrencial face a outro *scope*, com este atributo também activo, resultando na serialização dos acessos entre *scopes; e* que o *scope* não pode conter outros *scopes* com o mesmo atributo activo. Esta funcionalidade é do âmbito de um participante, pelo que num processo multi-entidades não é um aspecto relevante. Cada participante tem de resolver os seus problemas de concorrência, sem interferir com o processo global, pelo que este atributo não deve figurar na descrição CBPEL de *scope*. A concorrência entre participantes, terá que ser resolvida através da adequada troca de mensagens.

Dentro dos *standard-atributes* e *standard-elements* apenas é considerado o atributo *name*, que conterá o nome de *scope*, pois a inexistência de *links* inutiliza os outros atributos e elementos.

Quanto à actividade dentro do *scope*, esta será, agora, uma das actividades da linguagem CBPEL.

Descrição de um *scope* na linguagem CBPEL:

```
<scope name?>
    <variables/>?
    <correlationSets/>?
    <faultHandlers/>?
    <compensationHandler/>?
    <eventHandlers/>?
    activity
</scope>
```

5.3.2 Event handlers de um scope

Event handlers na linguagem BPEL

Os *event handlers* são rotinas despoletadas por eventos assíncronos, e que são executadas em paralelo para com o fluxo normal do *scope*. Estas rotinas são consideradas como parte do processamento normal do *scope*, ao contrário das rotinas de tratamento de falhas ou de compensação. Os eventos despoletadores dessas rotinas podem ser a recepção de uma mensagem (*onMessage*) ou um disparo de uma temporização (*onAlarm*). As rotinas despoletadas por uma mensagem podem ser executadas várias vezes, mesmo concorrencialmente, enquanto que as rotinas despoletadas por uma temporização só podem ser despoletadas no máximo uma vez. Estas rotinas assíncronas estão disponíveis para serem despoletadas enquanto o *scope* estiver na sua execução normal.

As rotina de tratamentos de eventos não podem evocar compensações (*compensate*), mas podem lançar falhas, que são consideradas como falhas ocorridas dentro da execução normal do *scope*.

Estrutura de um *event handler* na linguagem BPEL:

```
<eventHandlers>?
   <onMessage partnerLink="ncname" portType="qname"
       operation="ncname" variable="ncname"?>*
      <correlations/>?
      activity
   </onMessage>
   <onAlarm for="duration-expr"? until="deadline-expr"?>*
      activity
   </onAlarm>
</eventHandlers>
```

Event handlers na linguagem CBPEL:

A execução de rotinas assíncronas, face ao desenvolvimento normal do processo, é uma funcionalidade que necessita de ser modelada na linguagem CBPEL, pois deste modo os processos podem, por exemplo, realizar assincronamente acções, tal como consultas de estado relativas a um pedido realizado.

Já as rotinas associadas ao disparo de temporizações assíncronas têm por objectivo executar acções num determinado instante de tempo. Como os participantes iniciarão um mesmo *scope* em diferentes instantes, e como a noção de tempo envolvendo vários participantes não é um dado directo, optou-se por excluir a possibilidade de existência deste tipo de rotina assíncronas nos processos CBPEL.

Assim sendo, na linguagem CBPEL, a declaração de uma rotina assíncrona, tem a execução despoletada por uma mensagem privada que resultará em que um dos participantes inicie a rotina assíncrona e então dialogue com outros, também de forma assíncrona. Pelo que a rotina CBPEL deverá indicar qual é o participante iniciador da mesma, para que essa indicação fique explicita.

Estrutura de um *event handler* na linguagem CBPEL:

```
<eventHandlers>?
   <messageHandler  //onMessage
       initiator="ncname">*
      activity
   </messageHandler>
</eventHandlers>
```

5.3.3 Throw

Actividade *throw* na linguagem BPEL

A actividade *throw* gera uma falha (*fault*) no *scope* em que é lançada. Uma falha é identificada pelo seu nome, podendo transportar uma variável como contexto de dados para a rotina de tratamento. O lançamento de uma falha termina a execução do *scope* onde é lançada passando a execução para uma rotina de tratamento da falha. Uma falha gerada dentro de uma rotina de tratamento de falha, resulta no término imediato dessa rotina e no envio da falha para o *scope* acima.

Estrutura da actividade de *throw* na linguagem BPEL:

```
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
    standard-elements
</throw>
```

Actividade *throw* na linguagem CBPEL

O lançamento de uma falha, na linguagem CBPEL, será uma acção de um participante, tendo-se portanto de identificar quem é o seu executor num campo (*executer*) criado para tal efeito. O transporte de uma variável aquando do lançamento de uma falha apresenta-se como alguma problemática, porque as falhas não são lançadas de imediato, mas têm o seguinte percurso: a falha é comunicada ao coordenador; o coordenador devolve a indicação de lançamento da falha; essa indicação é recebida num *event handler*; e o *event handler* lança a falha. Nesse trajecto, a falha não poderá transportar a variável. Caso se pretende-se proceder ao lançamento da falha no *event handler* com a variável, por identificação da falha, tal tem o problema de não se

distinguir de um lançamento de falha com variável e sem variável. Também existe o problema de uma falha ser gerada num outro participante e ser confundida com uma falha lançada pelo próprio participante, o que apresenta o problema de nessa altura a variável associada à falha poder ainda não conter os valores correctos para a sua utilização na rotina de tratamento da falha. De modo a evitar essas situações, opta-se por não permitir o transporte de variáveis aquando do lançamento de falhas.

Estrutura da actividade *throw* na linguagem CBPEL:

```
<throw name="ncname" faultName="qname" executer="ncname"/>
```

5.3.4 Fault handlers de um scope

Fault handlers na linguagem BPEL

Um *fault handler* é uma rotina de tratamento (*handler*) de uma falha (*fault*), que é executada aquando da ocorrência de uma falha. Uma falha interrompe sempre a execução normal do *scope,* terminando-o de forma anormal. Uma falha também pode ser lançada dentro de uma rotina de tratamento de falha ou de uma rotina de compensação, mas o seu resultado é a finalização imediata da respectiva rotina. Uma falha pode ser gerada na resposta a uma chamada síncrona, ou pela execução de um *throw*. Numa interacção síncrona, a resposta pode ser uma mensagem *wsdl* de falha (*wsdl fault message*), a qual resulta numa falha (*fault*) BPEL com o respectivo nome definido na mensagem WSDL. Uma falha que não seja apanhada num *scope*, a fim de ser tratada por uma rotina, é lançada para o *scope* imediatamente acima do primeiro. Um *scope* onde ocorreu uma falha nunca é bem terminado, pelo que não pode ser compensado.

Estrutura de um *fault handler* na linguagem BPEL:

```
<faultHandlers>?
    <!— Dentro do elemento faultHandlers deve existir pelo menos um catch
    ou o catchAll. -->
    <catch faultName="qname"? faultVariable="ncname"?>*
          activity
    </catch>
    <catchAll>?
          activity
    </catchAll>
</faultHandlers>
```

Fault handlers na linguagem CBPEL

A análise e descrição de *fault handlers* na linguagem CBPEL será dividida em três partes, sendo primeiro abordado o caso de falhas geradas como mensagens WSDL, por

segundo as falhas geradas pela plataforma de execução, e por terceiro as falhas geradas por *throw*.

Falhas geradas como mensagem WSDL

As falhas geradas pela recepção de mensagens WSDL de falha geram falhas directamente na plataforma BPEL. Logo, no contexto dos processos CBPEL surge o problema de o *throw* da falha não poder ser acompanhado pelas devidas acções da sua propagação para o coordenador. Tal implica que a sua utilização só pode ser considerada quando a plataforma de execução integrar as necessárias acções, caso contrário teríamos uma situação de lançamento unilateral de uma falha, e a consequente falta de sincronismo entre os vários participantes. Como a utilização das mensagens WSDL de falha é uma forma de enviar uma mensagem juntamente com uma semântica implícita de falha, poder-se-á atingir o mesmo objectivo com o envio de uma mensagem normal e remetendo o reconhecimento da falha para a lógica do processo. Desta forma poder-se-á comunicar as falhas mas controlar o seu lançamento através da actividade CBPEL de *throw*. Resultando que este tipo de geração de falhas não será ser utilizado na linguagem CBPEL.

Falhas geradas pela plataforma de execução

Estas falhas têm o mesmo problema que as falhas WSDL, ou seja, geram uma falha com lançamento unilateral. Contudo neste caso, para estas falhas, não há solução, uma vez que somente alterando o comportamento da plataforma de execução se conseguiria a comunicação prévia da falha ao coordenador. Assim sendo, considera-se que nos processos num cenário CBPEL não ocorre falhas da plataforma, remetendo-se a solução para futuras alterações às plataformas de execução de BPEL.

Falhas geradas por *throw*

Todas as falhas serão, portanto, geradas pela actividade de *throw* da linguagem CBPEL. Cada falha será processada por uma rotina de tratamento, ou pela rotina por omissão de tratamento de falhas. Na linguagem CBPEL uma rotina de tratamento de falha é uma rotina com capacidade de troca de mensagens entre os participantes que interagiam no *scope* na altura em que ocorreu em falha, pelo que a utilização da rotina que apanha todas as falhas (*catchAll*) se revela de uma utilização que merece muita atenção, pois terá que se verificar se em todas as falhas possíveis de acontecer todos os participantes presentes na rotina estavam também já presentes dentro do *scope*.

Estrutura de um *fault handler* na linguagem CBPEL:

```
<faultHandlers>?
    <!—Deve existir pelo menos um catch ou o catchAll. -->
    <catch faultName="qname">*
          activity
    </catch>
    <catchAll>?
          activity
    </catchAll>
</faultHandlers>
```

5.3.5 Compensate

Actividade *compensate* na linguagem BPEL

A actividade *compensate* é utilizada para lançar a execução da rotina de compensação de um *scope* que já terminou normalmente. Esta actividade apenas pode ser chamada pelas rotinas de falha ou de compensação do *scope* que imediatamente engloba o *scope* a ser compensado. Esta actividade pode indicar o *scope* a ser compensado, ou pode ser utilizada sem mencionar qualquer *scope* em particular. Neste último caso, a plataforma terá o comportamento de compensar todos os *scopes* bem terminados mas pela ordem inversa à ordem da sua terminação.

Estrutura da actividade de compensate na linguagem BPEL:

```
<compensate scope="ncname"? standard-attributes>
    standard-elements
</compensate>
```

Actividade *compensate* na linguagem CBPEL

Esta actividade na linguagem CBPEL fica com a seguinte estrutura.

```
<compensate scope="ncname"? name="ncname"?/>
```

5.3.6 Compensation handler de um scope

Compensation handlers na linguagem BPEL

Uma rotina (*handler*) de compensação engloba uma actividade que poderá conter outras, e terá a finalidade de proceder à compensação das acções realizadas nas actividades do fluxo normal do *scope*. Uma rotina de compensação não pode afectar o estado do *scope*, ou seja, não pode alterar o conteúdo de variáveis, mas pode aceder ao seu conteúdo. Uma rotina de compensação só fica disponível para execução quando o *scope* termina devidamente, ou seja, termina sem a ocorrência de falhas.

Estrutura de um *compensation handler* na linguagem BPEL:

```
<compensationHandler>?
   activity
</compensationHandler>
```

Compensation handlers na linguagem CBPEL

A compensação na linguagem CBPEL será controlada pelo coordenador, resultando que todos os intervenientes receberão tal indicação. Apesar de que a indicação de compensação de um *scope* terminado já seria suficiente para a anulação das acções realizadas, torna-se pertinente permitir que haja troca de mensagens entre os intervenientes. Teremos portanto uma compensação com execução coordenada entre os vários participantes.

Estrutura de um *compensation handler* na linguagem CBPEL:

Este elemento tem na linguagem CBPEL uma estrutura idêntica à que apresenta na linguagem BPEL.

5.4 Transformação de Scopes, Falhas e Compensações para BPEL

Este subcapítulo visa esclarecer como se poderá realizar a transformação de *scopes* e suas rotinas de tratamento de falhas, de eventos assíncronos e compensação, descritos na linguagem CBPEL, para elementos BPEL, criando processos coerentes entre si e para com um coordenador.

De forma a permitir uma separação da transformação dos vários elementos de um *scope*, será apresentada a transformação de: *scope* considerando somente as declarações; rotinas de eventos assíncronos de um *scope*; actividade de *throw*; rotinas de tratamento de falhas de um *scope*; actividade de *compensate*; e rotina de compensação de um *scope*.

5.4.1 Transformação de scope

A transformação de um *cbpel:scope* resultará num *bpel:scope* para todos os participantes envolvidos no *cbpel:scope*. No entanto haverá um participante que deterá o controlo da execução, que será o primeiro a entrar e que criará o *scope* junto do coordenador. Esse participante será denominado de participante criador do *scope*.

A transformação do elemento *scope* para o participante criador resultará nos seguintes elementos: envio da mensagem de *Create* para o coordenador, com indicação do nome do novo *scope*; recebimento da mensagem de *Created* vinda do coordenador; criação do *scope* local; colocação em último lugar no fluxo dentro do *scope* local do envio da mensagem de *Completed* para o coordenador e do recebimento da mensagem de *Completed* vinda do coordenador.

A transformação do elemento *scope* para um participante que não seja criador resultará nos seguintes elementos: recebimento da mensagem de *Notify* vinda de um participante já dentro do *scope*; envio da mensagem de *Register* para o coordenador; recebimento da mensagem de *Registed* do coordenador; criação do *scope* local; colocação, em último lugar no fluxo dentro do *scope* local, do envio da mensagem de *Completed* para o coordenador, e do recebimento da mensagem de *Completed* vinda do coordenador.

No fluxo dentro do *scope*, o envio de uma mensagem para um participante que ainda não esteja registado no *scope* implica o envio da mensagem de *Notify* para esse participante antes do envio da mensagem original.

A transformação das actividades existentes no fluxo dentro do *scope* segue a transformação já apresentada no capítulo anterior, excepto as actividades de *throw*, e *compensate* que serão descritas nas secções seguintes.

A transformação dos elementos *variables* e *correlationSets* decorrerá, também, de forma idêntica à que foi descrita para os mesmos elementos, no capítulo anterior, na secção 4.3.1.

5.4.2 Transformação das rotinas de eventos assíncronos

A transformação das rotinas de eventos assíncronos, declaradas, num *scope* CBPEL, com o elemento *eventHandlers* resultará na criação de elementos análogos nos *scopes* BPEL dos intervenientes nessas rotinas. Esses intervenientes terão obrigatoriamente de participar no *scope* onde a rotina assíncrona está declarada, e terão de já estar registados no *scope* aquando da execução da rotina assíncrona em que participam.

Para cada rotina assíncrona, identificada com *messageHandler* na linguagem CBPEL, corresponderá uma rotina assíncrona em BPEL. A transformação para o participante iniciador resultará num elemento *bpel:onMessage* com o recebimento de uma mensagem assíncrona de carácter privado ao participante. A transformação, para os outros

participantes, resultará num elemento idêntico ao caso anterior, mas em que a mensagem, que despoleta a rotina, será a primeira mensagem que o participante em questão poderá receber dentro da rotina.

Para a possibilidade de um participante poder ter várias mensagens como a primeira possível, implica ter de se colocar cada mensagem numa rotina assíncrona diferente, com a replicação de código que isso implica. Para simplificar considera-se que cada participante apenas pode ter uma mensagem de início em cada rotina assíncrona CBPEL.

5.4.3 Transformação da actividade de Throw

A transformação da actividade CBPEL *throw* resultará, na colocação no processo do participante em questão, do envio da mensagem de *Fault* para o coordenador, seguido de actividade de *bpel:wait* com espera por tempo indeterminado.

O coordenador depois de receber a mensagem de *Fault* de um participante, enviará a todos os participantes no *scope*, uma mensagem de *Throw-fault*, que indicará a estes para procederem ao lançamento local da falha. Os participantes receberão essa mensagem numa rotina assíncrona, colocada no *scope*, e procederão ao lançamento da falha.

As rotinas assíncronas de recepção dessas indicações de falha, uma vez que terão que existir uma por cada *scope*, levantam o problema de múltiplas recepções de um mesmo participante e com um mesmo tipo de mensagem, para a plataforma de execução, que fica sem saber onde entregar as mensagens que receber. Tal pode ser resolvido utilizando conjuntos de correlação (*correlation sets*) diferentes que referenciem um campo da mensagem com o nome *scope* em questão.

5.4.4 Transformação das rotinas de falhas

Relativamente às rotinas de tratamento de falhas ter-se-à que abordar as situações de: transformação das rotinas declaradas; e de colocação de rotinas por omissão, para as rotinas não declaradas, incluindo para a falha de *ForcedTermination*.

A transformação das rotinas declaradas consiste em colocar, para cada uma, uma rotina de falha no *scope* BPEL, tendo no seu início o recebimento da mensagem de *Faulted* vinda do coordenador, seguido da declaração de um *scope* interno, com as

actividades, declaradas na rotina de falha CBPEL, relativas ao participante em questão, e que terá no final o envio de *FaultDone* para o coordenador e recebimento de uma mensagem vinda do mesmo e com o mesmo identificador. Caso haja lançamento de falhas, dentro de uma rotina de tratamento de falha, elas terão que ser enviadas para o coordenador, e este notificará cada participante para o lançamento da falha. Esta recepção será realizada como uma rotina assíncrona no já referido *scope* interno.

As falhas que não tenham rotina de tratamento declarada deverão ter o comportamento por omissão, que consiste em ter no seu início a recepção da mensagem de *Faulted* vinda do coordenador, seguida da criação do *scope* interno, e dentro dele a transformação das actividades CBPEL de *Compensate* e *Throw*.

A falha de *ForcedTermination* também requer sincronismo, dado que pode haver algum *scope* que tenha declarado algum código para o seu tratamento, pelo que é necessário definir sempre uma rotina de tratamento dessa falha. Esta falha, quando não declarada deve ter seguir os mesmos passos que uma outra qualquer falha que não tenha uma rotina declarada, tal como descrito no parágrafo anterior.

5.4.5 Transformação da actividade de Compensate

A transformação da actividade de *cbpel:Compensate-sc(scope)* resultará na colocação no processo do participante das seguintes actividades BPEL envio de *Compensate-sc(scope)* ao coordenador; recebimento de *Compensate-sc* do coordenador; execução da actividade BPEL de *Compensate(scope)*; e recebimento da mensagem de *Compensated-sc* vinda do coordenador.

A transformação da actividade de *cbpel:Compensate*, que compensa todos os *scopes* bem terminados, resultará na colocação no processo do participante das seguintes actividades BPEL: envio de *Compensate* ao coordenador; recebimento de *Compensate* do coordenador; execução da actividade de *Compensate*; e recebimento da mensagem de *Compensated* vinda do coordenador.

A actividade de *Compensate*, como irá compensar todos os *scopes* bem terminados, e pela ordem inversa à do seu término, levanta um problema de sincronismo, pois pode haver diferenças entre a ordem de término de dois ou mais *scopes* entre os vários participantes. Quando um *scope* é terminado de forma normal, o coordenador envia a mensagem de *Completed* para os vários participantes, e este terminam saindo do *scope* local. Como pode haver um desfasamento entre o envio de *Completed* e a saída do *scope*

local, para se garantir uma sequencialidade dos fechos dos *scopes*, ter-se-á que os encerrar de forma sequencial e com confirmação de fecho da parte dos participantes. Em termos de transformação dos *scopes*, esse aspecto não é contemplado.

5.4.6 Transformação das rotinas de compensação

A transformação de uma rotina de compensação resultará nas seguintes actividades BPEL nos processos dos participantes envolvidos: recebimento da mensagem de *CHCompensate-sc*; criação de um *scope* interno; colocação no seu interior das actividades BPEL resultantes da transformação das actividades CBPEL que se encontram dentro da rotina de compensação; e colocação no final do *scope* interno do envio mensagem de *CHCompensated-sc* para o coordenador e recebimento do mesmo de *CHCompensated-sc*.

Os *scopes* que não tenham uma rotina de compensação declarada, deverão ter o comportamento por omissão, que é compensar todos os *scopes* bem compensados e pela ordem inversa à do seu término. Nestes casos ter-se-à de colocar uma rotina para executar as actividades correspondentes: recebimento da mensagem de *CHCompensate-sc*; criação de um *scope* interno; colocação nesse *scope* da transformação de *cbpel:Compensate*; e nesse *scope* colocar também o envio da mensagem de *CHCompensated-sc* para o coordenador e recebimento do coordenador de *CHCompensated-sc*.

Os *scopes* internos das rotinas de compensação, assim como das rotinas de tratamento de falhas, destinam-se a conter a recepção de mensagens assíncronas de erro vindas do coordenador indicando a ocorrência de uma falha, pelo que necessitam de conter essas rotinas assíncronas e as rotinas de tratamento das respectivas falhas.

5.5 Conclusão

Este capítulo descreve o modelo de falhas e compensações utilizado na linguagem CBPEL e a sua transformação para processos em linguagem BPEL.

O modelo utiliza um protocolo com coordenador de forma a haver uma perspectiva centralizada e independente dos participantes e que determine o resultado final da execução de cada *scope*. O modelo utiliza os principais conceitos existentes na linguagem BPEL, nomeadamente a noção de: *scope*; lançamento de falhas; rotina de tratamento de

falhas; compensação de *scope*; rotina de compensação de *scope*; e a existência de rotinas de execução assíncrona no *scope*.

O capítulo apresenta: o protocolo de coordenação a ser utilizado por um coordenador, e a parte relativa aos participantes; a descrição dos elementos CBPEL relativos ao suporte das características apresentadas; e a transformação desses elementos CBPEL para elementos BPEL, o que permite gerar os processos BPEL partindo de uma descrição CBPEL.

O modelo da linguagem CBPEL irá ser analisado em termos da concepção de processos no capítulo seguinte. Relativamente aos desenvolvimentos apresentados neste capítulo são identificados os seguintes problemas ou imperfeições:

- O protocolo utilizado carece de uma validação formal;

- As falhas geradas pela plataforma de execução BPEL não foram consideradas. Como essas falhas necessitam de ser propagadas para o coordenador, isso implica alterações à plataforma de execução;

- Os processos BPEL ficam muito extensos pela existência: de rotinas assíncronas auxiliares; de *scopes* internos; e de emissão e recepção de mensagens de suporte ao protocolo com o coordenador. Tais elementos são necessários, mas alguns deles poderiam ficar internos à plataforma de execução, o que simplificaria bastante os processos dos participantes;

- A entrada nos *scopes* apresenta uma possibilidade de inconsistência, pois o participante regista-se primeiro no coordenador antes de entrar no *scope* local. Caso, pelo meio dessas duas acções ocorra uma falha, o participante vai registar essa falha como do *scope* pai, enquanto que o coordenador regista a falha como do *scope* filho. A solução para este problema consistiria em executar as acções de entrada no *scope* internamente à plataforma de execução, de modo a serem vistas, da parte dos participantes, como apenas uma instrução;

- Na saída dos *scopes* não se garante um sincronismo temporal entre todos os participantes e o coordenador, o que coloca um problema na ordem de compensação dos *scopes*. Pois estes são compensados pela ordem inversão à ordem da sua terminação. Esta situação poderia ser resolvida por indicação, dos participantes, de fecho do *scope* e por impor ao coordenador uma sequencialidade

no fecho dos *scopes*. Mais uma vez uma implementação ao nível da plataforma de execução permitiria a escrita de processos desnecessariamente complexos.

- Não foi contemplado a existência de eventos temporais. Contudo essa é uma funcionalidade importante para modelar devidamente os processos entre organizações, pois uma resposta em tempo infinito é inaceitável; e

- A participação em *scopes* intercalados foi apresentada, mas não foi completada. Nomeadamente a sua transformação para BPEL não foi concretizada. A existência deste tipo de participação é premente nos processos entre organizações pelo que o aprofundamento do seu estudo é importante.

Foi portanto, indicado o protocolo de coordenação para processos interorganizacionais desenvolvidos, utilizado na linguagem CBPEL, tendo como objectivo a utilização da linguagem BPEL para o suporte à execução dos processos dos participantes. Verificou-se que os desenvolvimentos efectuados indicam que seria conveniente não utilizar directamente a linguagem BPEL, mas sim uma linguagem sua derivada e adaptada aos requisitos que a existência de *scopes* coordenados impõe.

Capítulo 6

Cenários de Aplicação

De forma a se efectuar uma demonstração da aplicabilidade da linguagem CBPEL, são apresentados dois cenários de processos interorganizacionais, um na área da administração electrónica, que consiste na escala de um navio num porto marítimo, e outro na área do negócio electrónico, que consiste no acto de uma compra numa livraria electrónica. Com estes cenários também se pretende demonstrar a concepção de processos interorganizacionais segundo este modelo / linguagem.

O primeiro cenário é um cenário modelado sem *scopes*, ou seja, não utiliza falhas nem compensações e tem o propósito de ilustrar como um processo nessas condições fica modelado na linguagem CBPEL e de como ficam os processos dos vários participantes na linguagem BPEL.

O segundo cenário é primeiramente apresentado sem *scopes*, e numa segunda fase com *scopes*, falhas e compensações. Pretende-se deste modo mostrar a complexidade resultante da introdução do controle de falhas. Como resultado, os processos dos participantes ficam bastante mais complexos na segunda versão. Contudo essa complexidade é necessária para garantir a coerência global do processo.

Os cenários são apresentados, primeiro como um processo global e comum de forma visual e em linguagem CBPEL, e segundo pelos processos dos participantes em linguagem BPEL. Como a geração dos processos dos participantes é feita por uma aplicação que também possui a componente de simulação da execução dos processos BPEL, a sua visualização é aqui apresentada recorrendo a uma estrutura em árvore, pois essa é a estrutura gráfica utilizada para visualizar os processos na referida aplicação. Essa estruturação em árvore tem as vantagens de possibilitar mostrar

somente a informação pertinente de cada elemento e consequentemente ter uma representação bastante compacta em termos de espaço ocupado. Na estrutura em árvore são utilizados os símbolos gráficos da linguagem BPEL, existentes no editor da mesma linguagem da plataforma Eclipse (ver http://www.eclipse.org/bpel), embora com algumas adaptações, conforma se pode observar na tabela 3.

Símbolo gráfico	Elemento CBPEL	Elemento BPEL
	commonProcess	
		process
	partnerLinks, partnerLink	partnerLinks, partnerLink
	partnerLinkType	partnerLinkType
	partners	partners
	partner, initiator	partner
	variables	variables
	variable	variable
	sequence	sequence
	switch	switch
		pick
	case	case
	while	while
	assign	assign
	copy	copy
	empty	empty
	send	
		receive
		invoke
	wait	wait
	scope	scope
	faultHandlers	faultHandlers
	catch	catch
	catchAll	catchAll
	compensationHandler	compensationHandler
	eventHandlers	eventHandlers
	messageHandler	onMessage
	throw	throw
	compensate	compensate
	compensateScope	compensateScope

Tabela 3 – Símbolos utilizados na linguagem CBPEL e BPEL

Por questões de espaço, remete-se para os anexos D e E a listagem completa, em XML, de todos os processos, dos dois cenários.

6.1 Cenário da escala de um navio num porto

Este cenário descreve a escala de um navio num porto marítimo, e é um cenário adaptado de [Velosa00]. Nele irão constar os seguintes participantes: Agente de Navegação (agn), Administração do Porto (adm), Capitania do Porto (cpp), Alfândega (alf), Pilotos (pil), e Operador Portuário (opp). O cenário será composto pelas três seguintes fases: fase de entrada do navio no porto e acostagem; fase das operações de escala; e fase de desembaraço e saída do navio do porto, as quais serão descritas no ponto seguinte.

O cenário não contém todos os participantes descritos na versão original [Velosa00], de modo a não estender demasiado o processo e porque a incorporação das entidades que faltam não iria enriquecer substancialmente o cenário. Essas entidades excluídas são: o Armador, o Capitão do Navio, a Sanidade Marítima, o Veterinário e a Brigada Fiscal.

Primeiro será feita uma descrição textual do cenário, depois uma visualização gráfica do mesmo, depois os resumos do processo em linguagem CBPEL e dos processos gerados em linguagem BPEL.

6.1.1 Descrição do cenário

Conforme já referido este cenário será decomposto nas suas três fases: entrada no porto e acostagem; operações de escala; e desembaraço e saída do porto.

Fase de entrada no porto e acostagem

O cenário inicia-se com o agente de navegação a enviar um pedido de autorização de nova escala à administração do porto. A administração autorizará ou não a nova escala, pois pode detectar que o navio apresenta alguma irregularidade face à documentação apresentada ou histórico existente. Caso não autorize envia uma notificação ao agente de navegação e a escala termina por aqui. Caso autorize, a administração portuária começa por notificar os outros intervenientes portuários da nova escala e depois notifica o agente de navegação da autorização.

O agente de navegação em contacto com o armador e com o capitão do navio vai recebendo informação da localização do mesmo e de possíveis alterações da data de chegada do navio, designada de *Estimated Time of Arrival* (ETA). O agente pode enviar

várias notificações de novas ETA à administração, a qual propaga essa informação para todos os intervenientes portuários.

Quando o navio se encontra à entrada do porto, o agente notifica a administração indicando que o navio se encontra ao largo e pronto a entrar no porto. A administração propaga essa informação para todos os intervenientes portuários, o que indica aos pilotos que devem proceder à acostagem do navio. Quando estes terminarem essa tarefa notificam a administração, fixando a altura em que o navio deu entrada no porto, altura essa que é denominada de *Actual Time of Arrival* (ATA). A administração propaga essa informação a todos os outros intervenientes portuários e ao agente de navegação.

Fase das operações de escala

Nesta fase e segundo o plano da escala, ou segundo novas indicações do armador e do capitão do navio, ocorre as operações de carga e descarga. Para facilitar o processo as operações de abastecimento são englobadas nestas operações. As operações movimentação do navio também fazem parte das actividades que nesta fase podem ocorrer.

As operações de carga e descarga iniciam-se com um pedido do agente de navegação à administração da intenção da sua realização e indicando a sua altura prevista (*Estimated Time of Operation* – ETO). A administração poderá recusar por falta de meios, ou outro motivo operacional ou legal e nesse caso notifica o agente da recusa do pedido. Caso aprove a operação, a administração primeiro notifica as outras autoridades, depois os operadores portuários, e por fim o agente de navegação de que a operação pode prosseguir. Quando os operadores portuários terminarem a operação notificam a administração indicando a altura em que ela terminou (*Actual Time of Operation* – ATO). A administração, então, notifica primeiro as entidades fiscalizadoras e depois o agente de navegação, do final da operação.

Nesta fase das operações de escala, o agente de navegação, também pode solicitar à administração a movimentação do navio para outro cais dentro do porto. Tal pode ocorrer por um silo ou uma grua com determinadas características existir somente noutro cais. Esta operação começa com um pedido de autorização à administração, indicando a altura estimada para a movimentação do navio (*Estimated Time of Move* – ETM), o que pode ser aprovado ou reprovado. Caso seja aprovado, todos os intervenientes portuários são avisados. Quando os pilotos iniciarem a manobra devem notificar a administração indicando a altura de início de movimentação (*Start Time of Move* – STM), e esta deve

propagar a informação para o agente de navegação. Quando os pilotos terminarem a movimentação do navio notificam a administração indicando esse instante (*Final Time of Move* – FTM). Por fim a administração notifica todas as entidades portuárias e depois o agente de navegação do final da operação.

Fase de desembaraço e saída do porto

Quando o agente tiver conhecimento de que as operações de escala estão terminadas, solicita à alfândega a autorização de desembaraço. Esta pode reprovar o pedido, ou aceitá-lo. Neste último caso a alfândega notifica a administração portuária e a capitania e depois o agente de navegação. Em caso de reprovação do pedido, o agente terá que voltar a fazer novo pedido após alterar ou completar a documentação apresentada.

Depois de ter obtido autorização da alfândega, o agente de navegação solicita autorização de desembaraço à capitania do porto, para uma determinada altura (*Estimated Time of Departure* – ETD). Em caso de aprovação, a capitania notifica as restantes entidades portuárias acerca do desembaraço do navio, e depois notifica o agente de navegação. Caso a capitania não aprove o desembaraço do navio, o agente terá que voltar a fazer novo pedido, eventualmente para uma nova data.

Na altura estimada para a partida do navio, os pilotos quando iniciarem a desacostagem do navio, deverão notificar a administração desse facto (*Start Time of Departure* – STD), a qual notifica o agente de navegação. Quando o navio sair do porto, os pilotos notificam a administração com a indicação de navio ao largo e a altura em que tal ocorreu (*Actual Time of Departure* – ATD). A administração notifica, então, todas as entidades portuárias e o agente de navegação da saída do navio, dando por encerrada a escala.

No cenário original apenas estavam previstos os tempos de ETA, ATA, ETD e ATD, mas de modo a salientar a altura dos outros acontecimentos, foram introduzidos os tempos de ETO, ATO, ETM, STM, FTM e STD.

6.1.2 Descrição visual do cenário com vista global e comum

A descrição visual do processo de forma global e comum será dividida também nas três fases de: entrada no porto; operações de escala; e saída do porto.

Nas figuras que serão apresentadas os intervenientes no porto serão identificados pelas suas siglas, já apresentadas, de modo a reduzir a dimensão do texto utilizado. Nos

casos em que uma notificação teria como destinatário vários intervenientes portuários optou-se por colocar as suas iniciais: C – CPP (Capitania); A – ALF (Alfândega); O – OPP (Operadores Portuários); e P – PIL (Pilotos).

Na figura 63 encontra-se o diagrama com o início de escala, desde o pedido inicial à administração até à acostagem do navio.

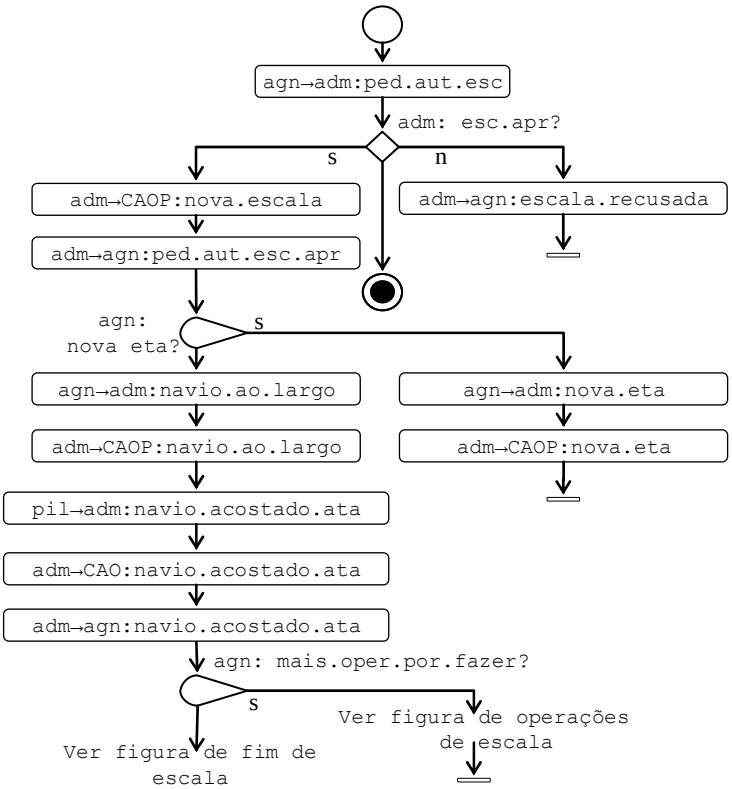


Figura 63 – Cenário marítimo: Início de escala

Na figura 64 encontra-se o diagrama com as operações de escala, ou seja, as operações de carga e/ou descarga e as movimentações do navio. Nessa figura constam algumas actividades que são representadas a tracejado. Essas actividades não são necessárias ao processo e apenas constam por motivos de a transformação de CBPEL para os processos BPEL não estar completa e de requerer que numa decisão só exista um actividade de recepção em cada um dos ramos da decisão quando o participante é passivo na decisão, seja ela, uma simples decisão (*cbpel:switch*) ou uma decisão cíclica (*cbpel:while*). Deste modo em alguns ramos de certas decisões teve-se de introduzir uma notificação para alguns participantes, para que eles só tenham a possibilidade de receber uma única mensagem em cada ramo.

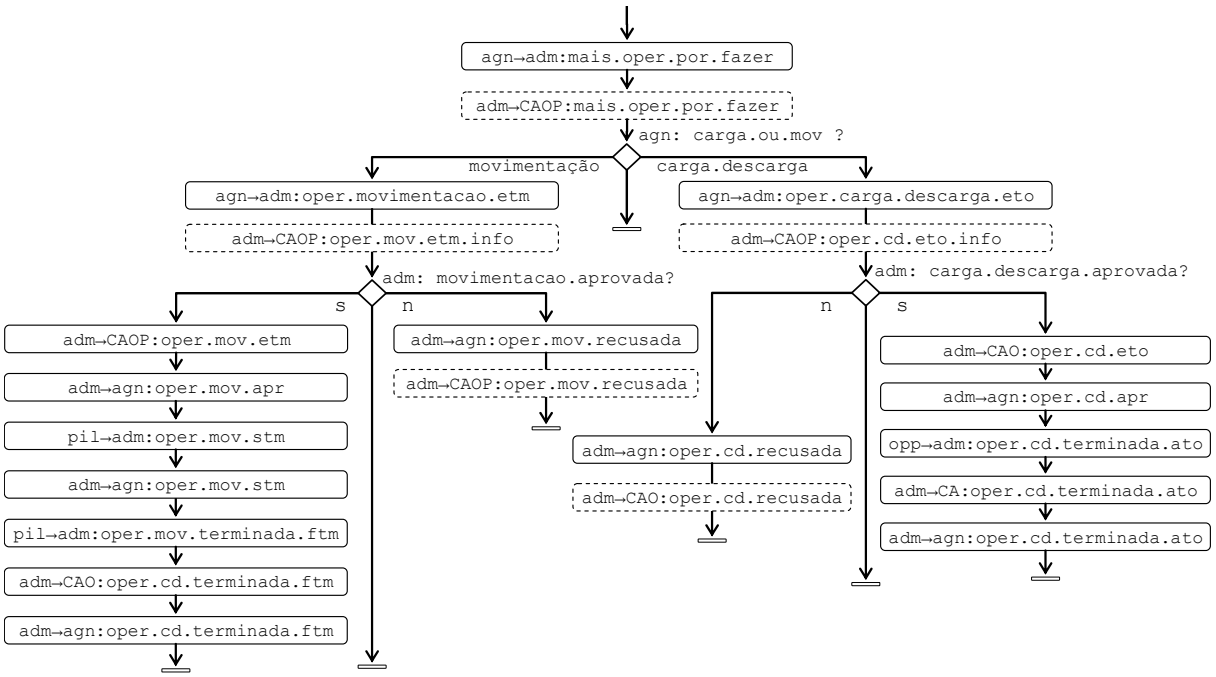


Figura 64 – Cenário marítimo: Operações de escala

Na figura 65 mostra-se o diagrama com a parte do processo relativo à saída do navio do porto, o que corresponde ao final da escala.

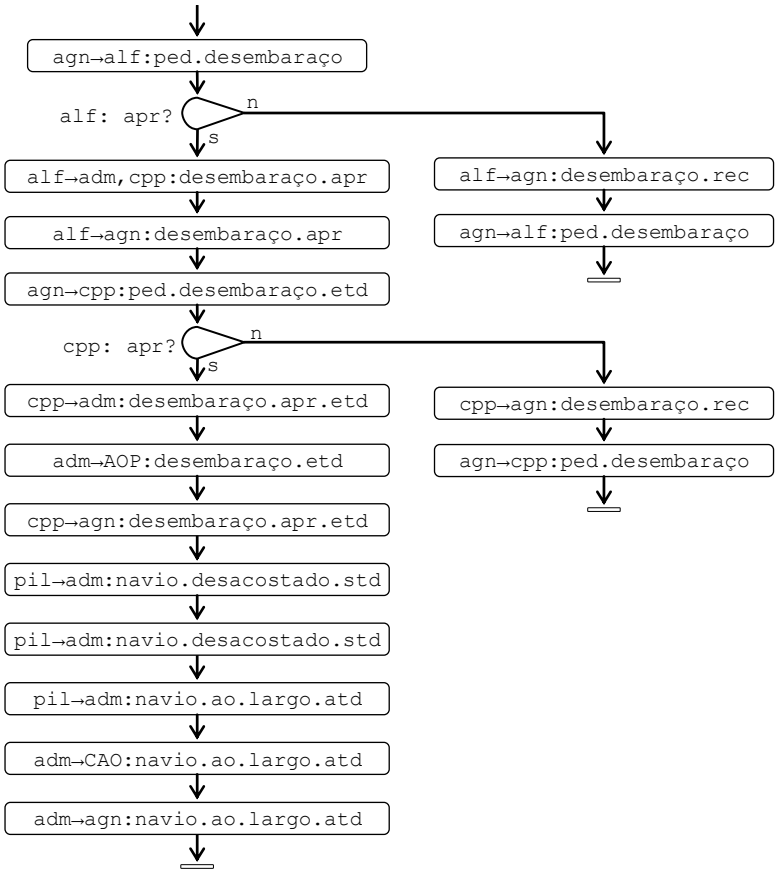


Figura 65 – Cenário marítimo: Fim de escala

6.1.3 Visualização do processo CBPEL e dos processos gerados

A visualização dos vários processos, quer do processo global em CBPEL, quer dos processos dos participantes em BPEL, utiliza uma estrutura em árvore proveniente da aplicação concebida. Remete-se para o anexo D a listagem completa, em XML, dos vários processos existentes do cenário.

Visualização do processo global em linguagem CBPEL

Nas figuras figura 66, figura 67 e figura 68 encontra-se a visualização do processo CBPEL deste cenário.

A figura 66 contém a parte inicial das declarações do processo, e a parte das actividades que correspondem ao início da escala, em que termina quando a acostagem é terminada. Esta figura, e as seguintes, seguem o processo tal como já foi descrito, apenas diferindo pelo facto de que as actividades de envio de mensagem com múltiplos destinatários foram desdobradas em actividade de envio de mensagem com um único destinatário.

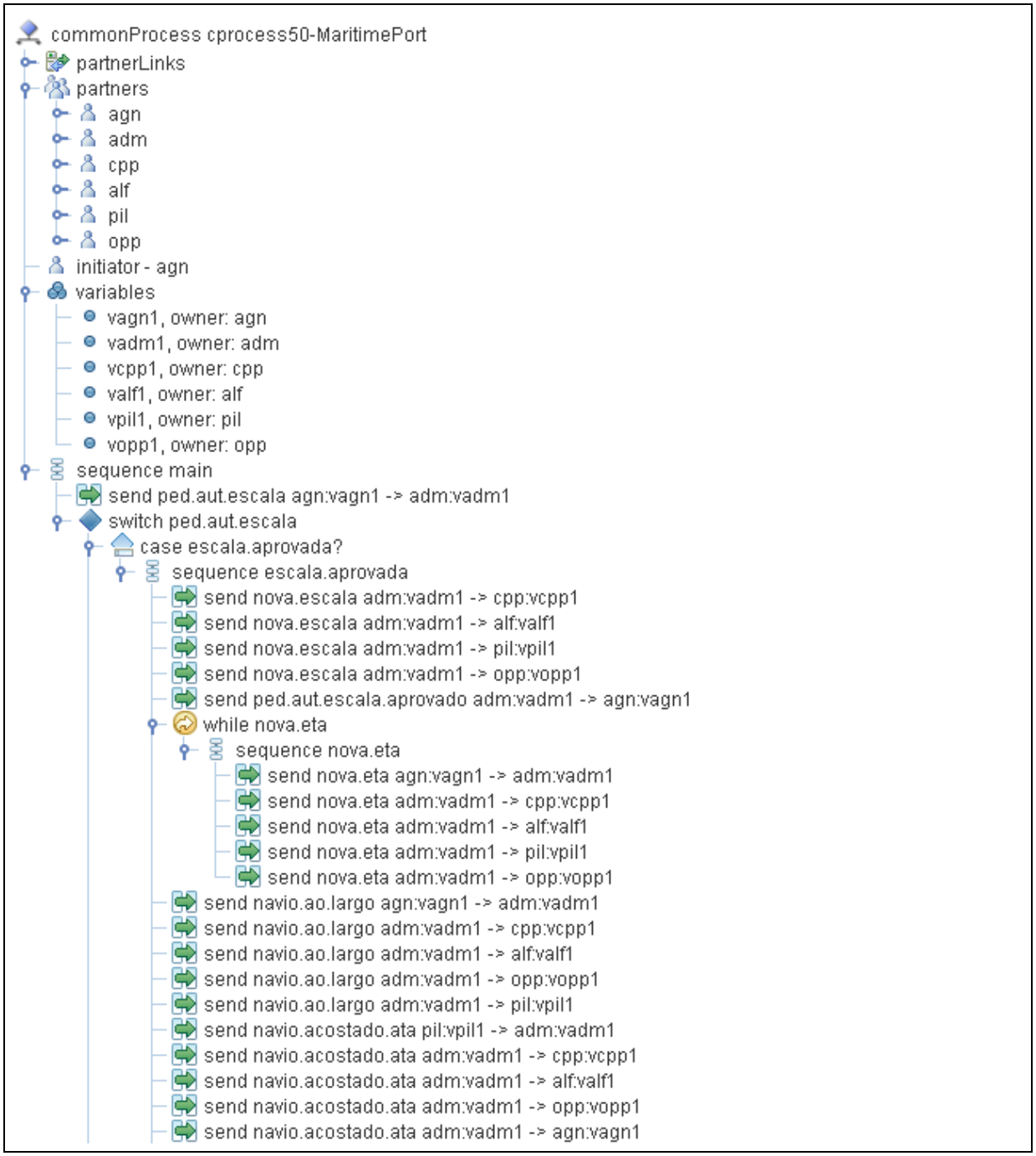


Figura 66 – Processo CBPEL do Porto Marítimo – parte 1

A figura 67 começa com as operações de escala, apresentando as operações de carga e descarga, e contém parte das operações de movimentação.

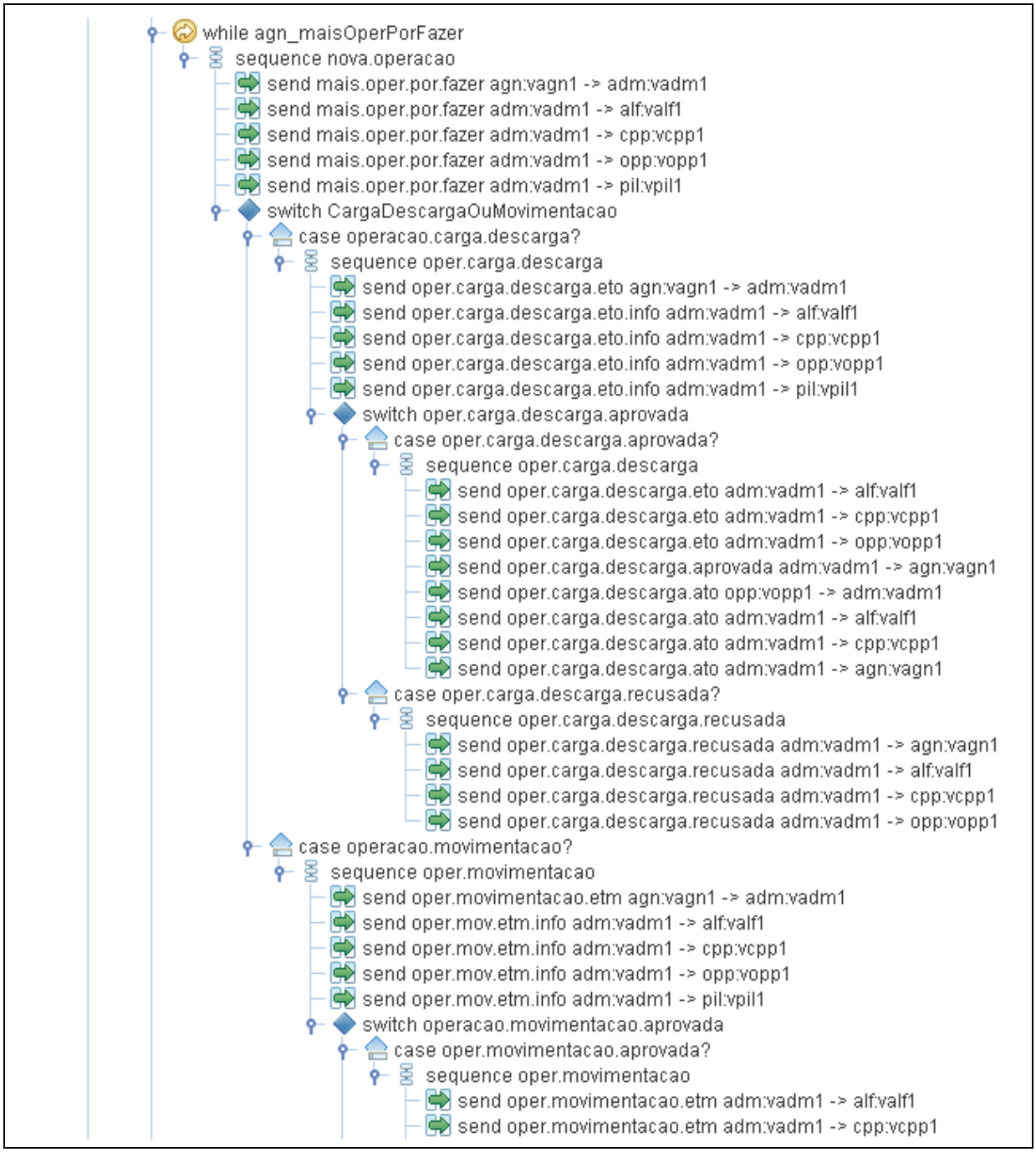


Figura 67 – Processo CBPEL do Porto Marítimo – parte 2

A figura 68 termina a parte das operações de movimentação e contém a fase de desembaraço e saída do navio do porto.

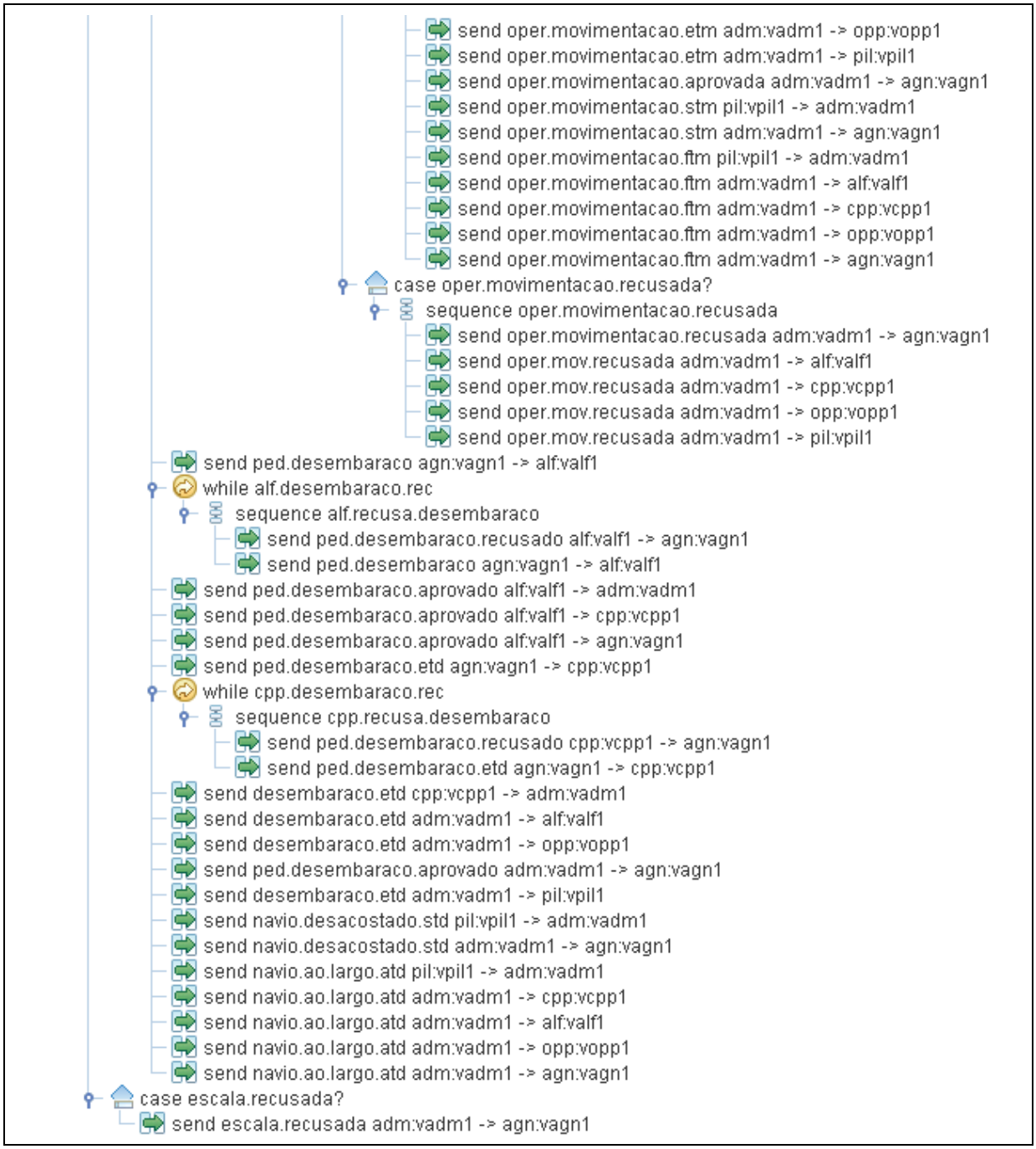


Figura 68 – Processo CBPEL do Porto Marítimo – parte 3

Segue-se, portanto, a visualização dos processos dos participantes no cenário.

Visualização do processo do Agente de Navegação

Na figura 69 encontra-se a visualização integral do processo do agente de navegação (agn).

Figura 69 – Processo do Agente de Navegação (agn)

Visualização do processo da Administração do Porto

Nas figuras figura 70, figura 71 e figura 72 encontra-se a visualização do processo da administração do porto (adm).

A figura 70 contém a parte das declarações do processo a parte inicial da escala.

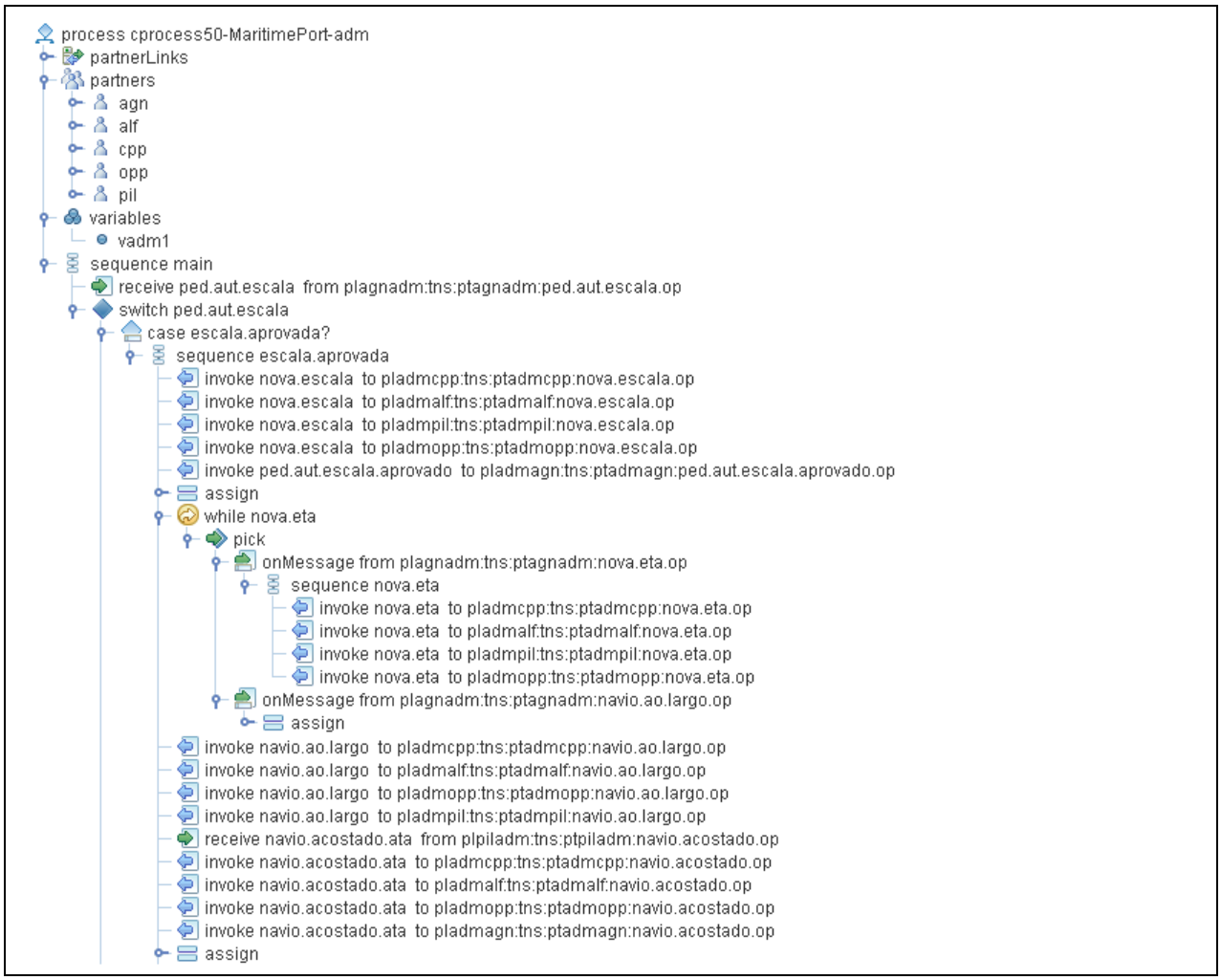


Figura 70 – Processo da Administração do Porto (adm) – parte 1

A figura 71 contém parte das operações de escala, nomeadamente as operações de carga e descarga e parte das operações de movimentação.

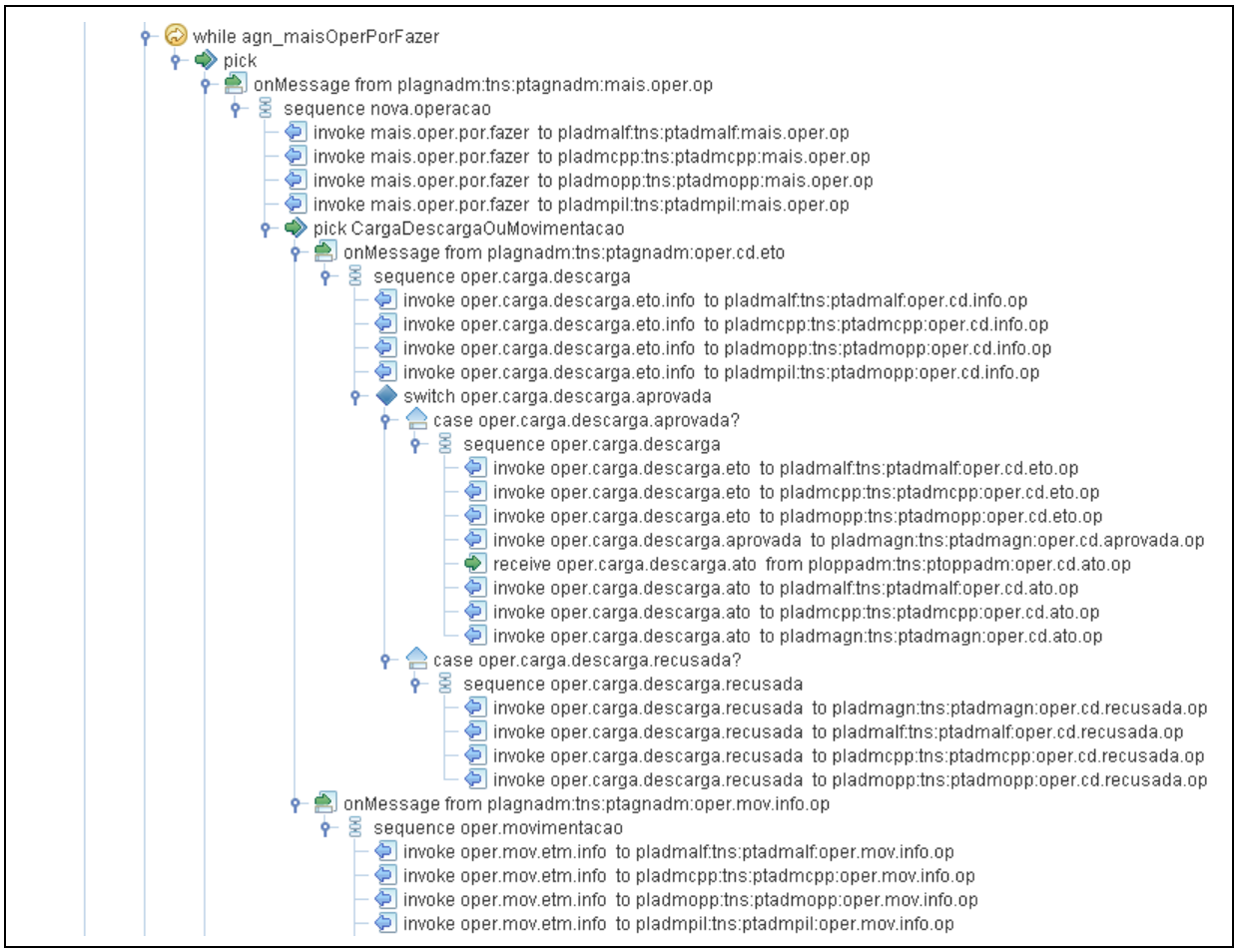


Figura 71 – Processo da Administração do Porto (adm) – parte 2

A figura 72 termina as operações de movimentação e contém a parte de desacostagem e saída do navio do porto.

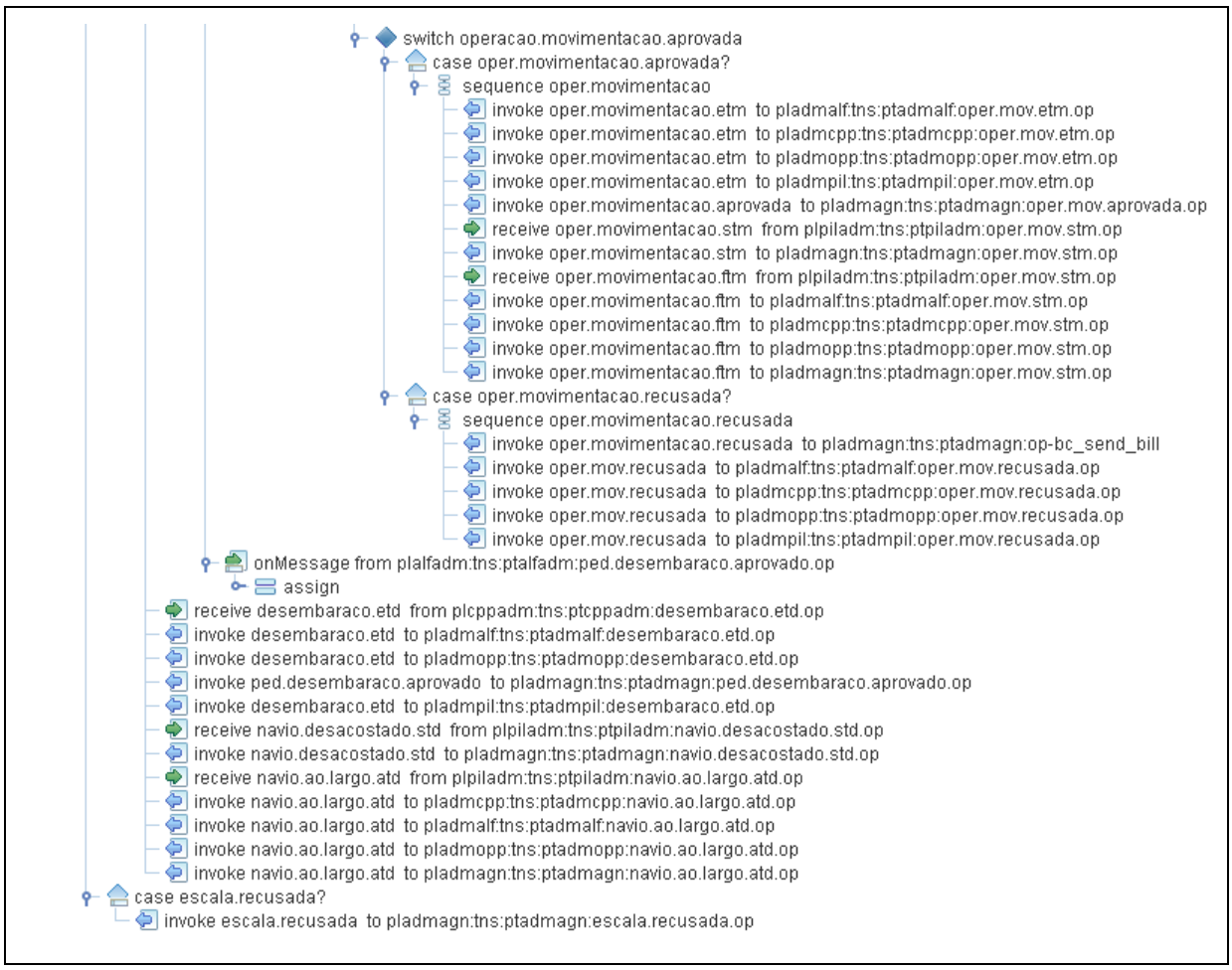


Figura 72 – Processo da Administração do Porto (adm) – parte 3

Visualização do processo da Capitania do Porto

Na figura 73 encontra-se a visualização integral do processo da capitania do porto (cpp)

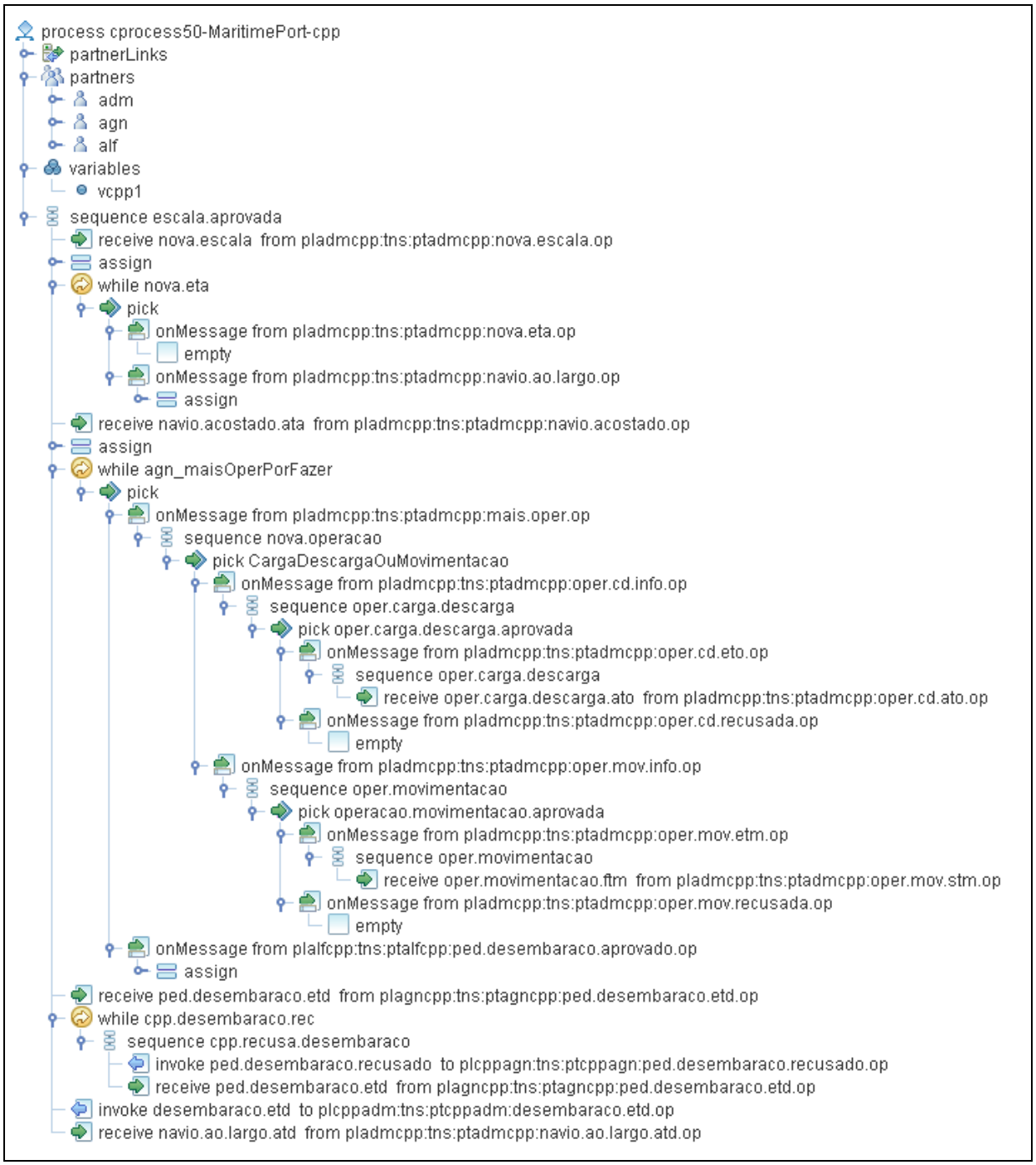


Figura 73 – Processo da Capitania do Porto (cpp)

Visualização do processo da Alfândega Portuária

Na figura 74 encontra-se a visualização integral do processo da alfândega portuária (alf).

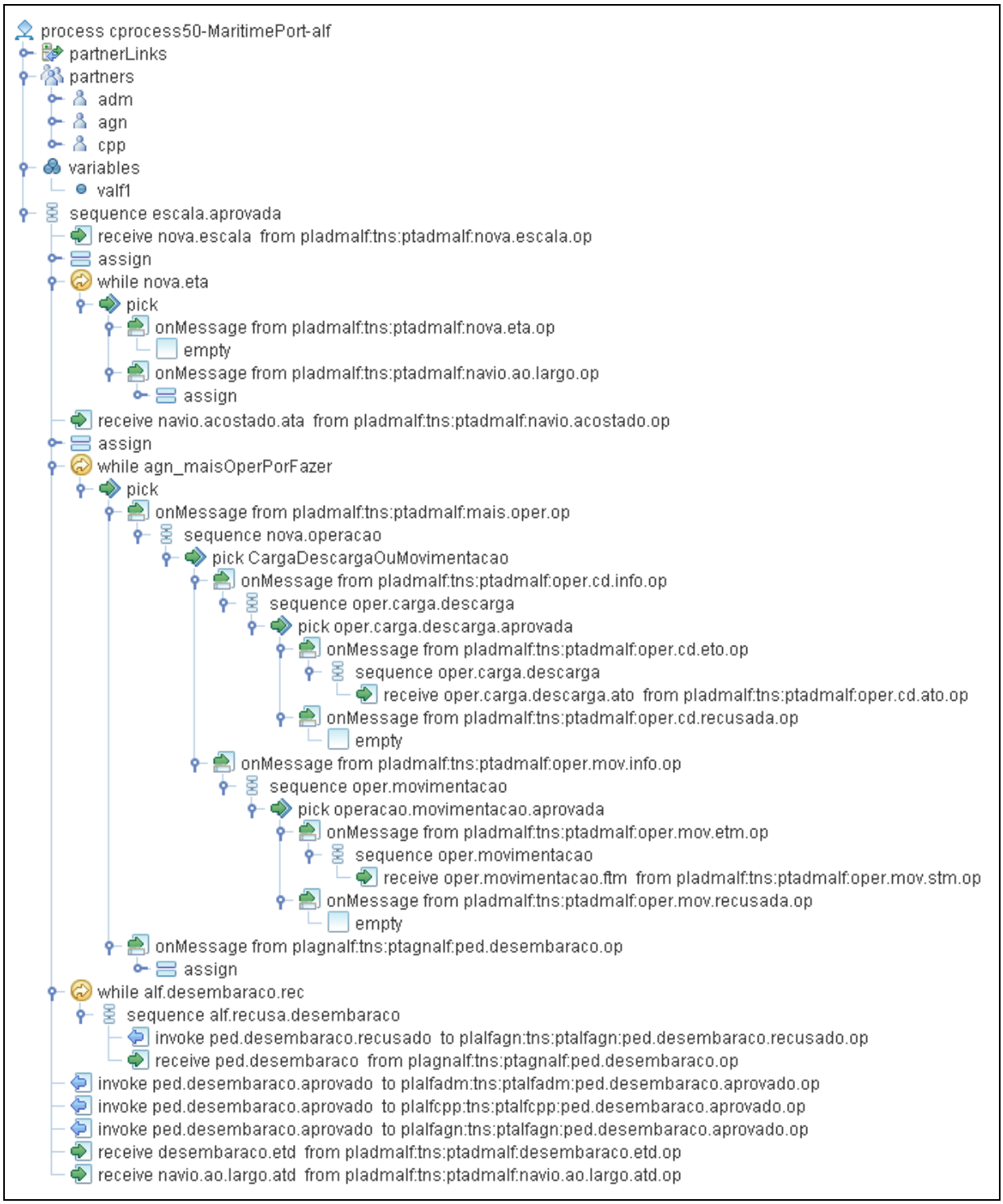


Figura 74 – Processo da Alfândega (alf)

Visualização do processo dos Pilotos

Na figura 75 encontra-se a visualização integral do processo dos pilotos (pil)

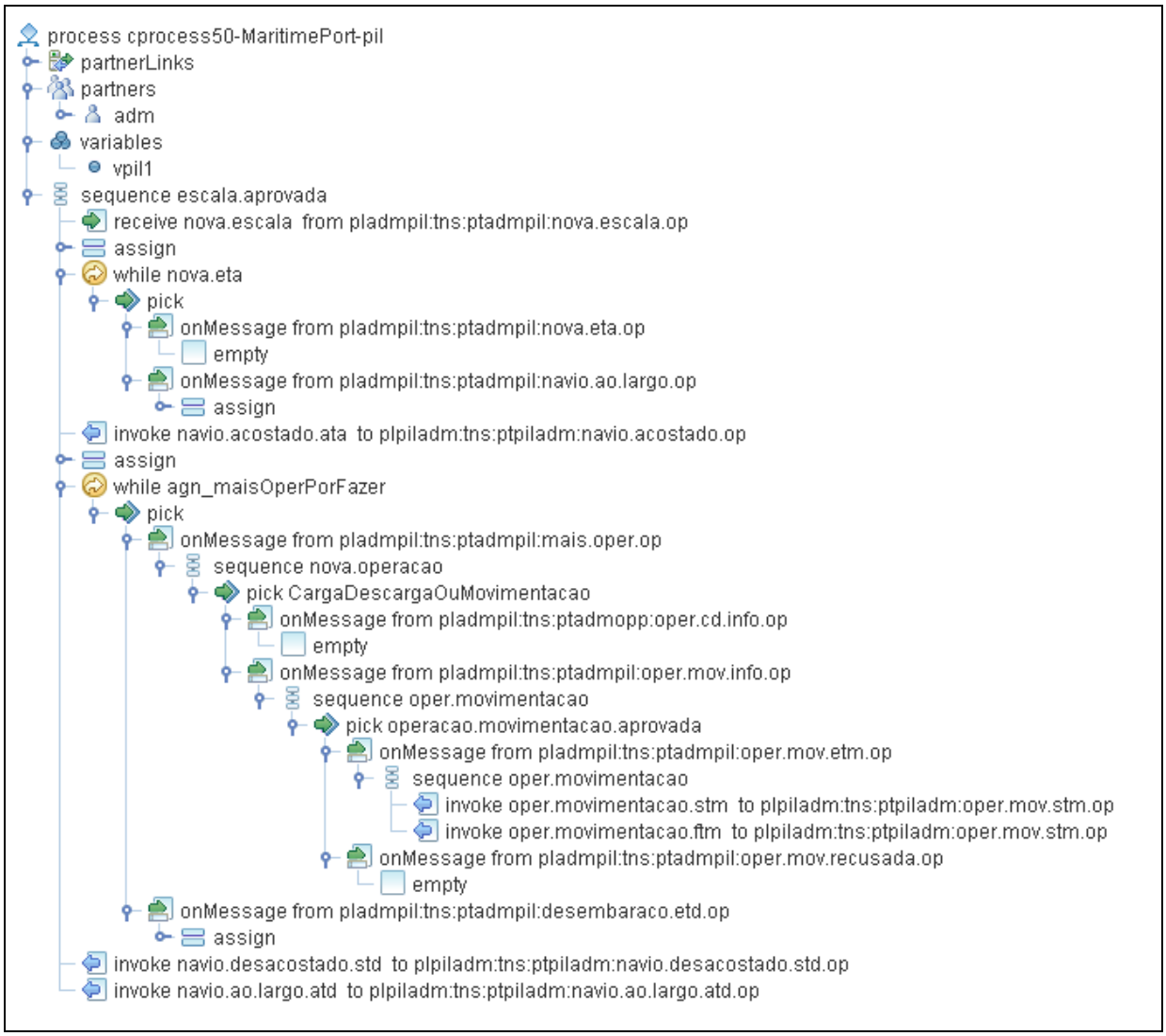


Figura 75 – Processo dos Pilotos (pil)

Visualização do processo dos Operadores Portuários

Na figura 76 encontra-se a visualização integral do processo dos operadores portuários (opp)

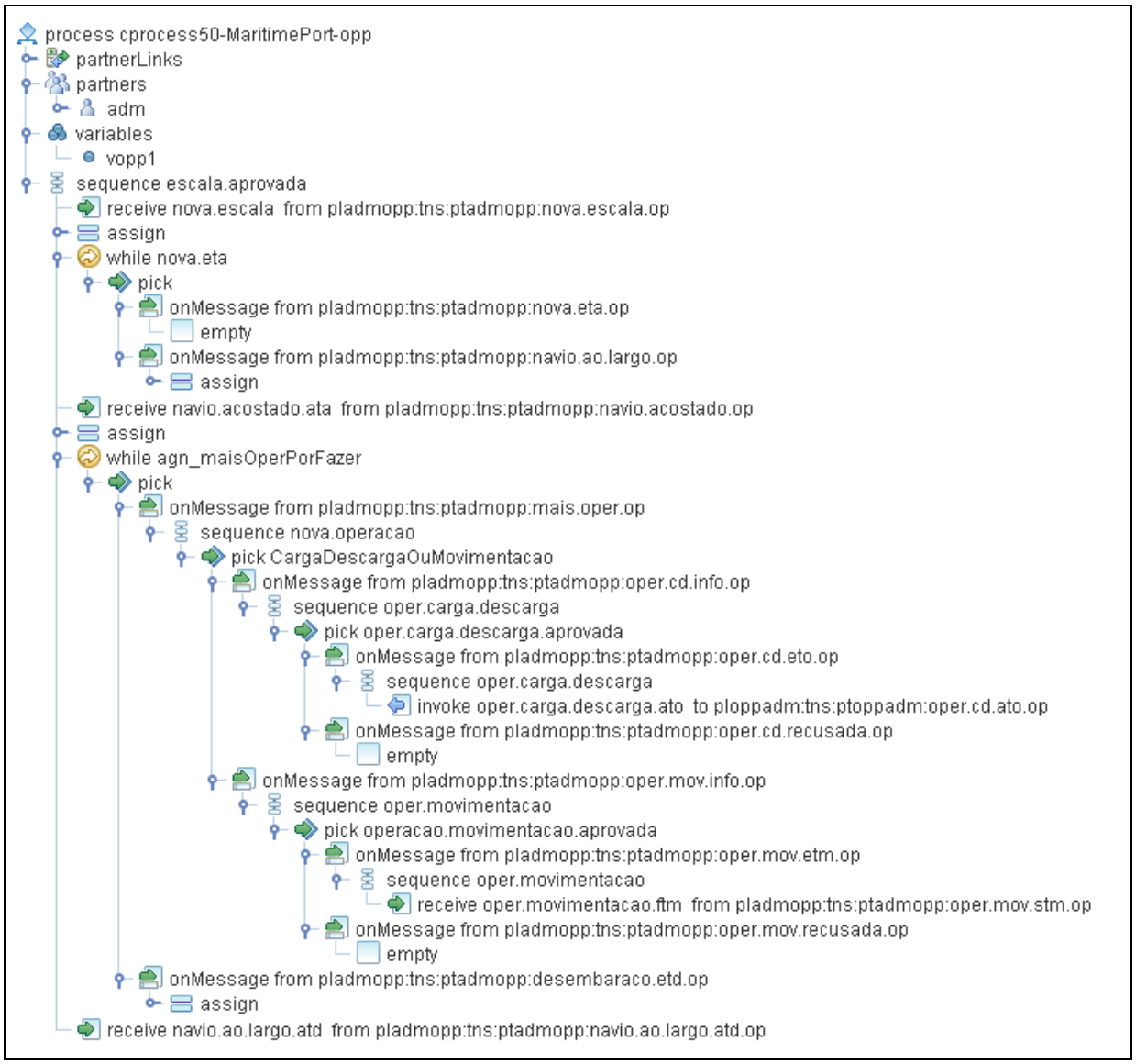


Figura 76 – Processo dos Operadores Portuários (opp)

6.2 Cenário da livraria electrónica

Este cenário foi extraído de [Aalst01] e consiste na realização de uma compra numa livraria electrónica. Inicia-se então com a indicação de compra por parte do cliente, em que a livraria, por não ter armazém próprio, transfere o pedido para o editor, e este remete-o para o expedidor (serviço de entregas) a fim de realizar a entrega final ao cliente. Irá portanto ter os seguintes participantes: o cliente (*customer*), a livraria (*bookstore*), o editor (*publisher*) e o expedidor (*shipper*).

Primeiro será feita uma descrição textual do cenário, depois uma visualização gráfica do mesmo, depois apresentados os resumos do processo em linguagem CBPEL e dos processos gerados em linguagem BPEL, primeiro numa versão sem *scopes* e depois numa outra versão com *scopes*.

6.2.1 Descrição do cenário

O cenário encontra-se descrito na figura 77 como uma rede de Petri, tal como foi descrito na sua forma original [Aalst01]. A rede contém partes que correspondem aos processos dos participantes já apresentados e a interligação entre essas partes. Os lugares que fazem essa interligação estão preenchidos a cinzento somente para salientar essa função. Da rede com todo o processo, pode gerar-se os processos dos vários participantes, separando-os pelos lugares a cinzento, e colocando um lugar de início e de fim no fluxo de cada participante.

Seguidamente descreve-se os passos existentes no cenário da figura 77, apresentando-se entre parênteses o nome da mensagem referida, uma vez que a descrição das actividades internas a cada processo não têm relevância para este trabalho. O processo inicia-se quando o cliente envia uma ordem de compra à livraria (*c_order*). Esta ordem do cliente é recebida pela livraria, a qual não possui armazenamento local de livros, pelo que remete o pedido para o editor (*b_order*). A ordem da livraria é avaliada pelo editor sendo aceitada ou rejeitada. Se a livraria recebe uma resposta negativa (b_decline), ela decide ou por procurar um editor alternativo ou por rejeitar a ordem do cliente, enviando-lhe uma mensagem (c_decline) e termina o processo. Se a livraria opta por procurar por um editor alternativo, uma nova ordem de compra é enviada para esse editor (b_order), e volta-se a repetir esta parte do processo. Se a livraria recebe uma resposta positiva do editor (b_*confirm*), o cliente é informado (c_confirm), e a livraria continua a processar a ordem do cliente. Uma vez a ordem

confirmada, a livraria envia o pedido para o expedidor (s_request), este avalia o pedido e ou aceita (*s_confirm*) ou rejeita (s_decline) o pedido de expedição. Caso rejeite, a livraria irá solicitar a outro expedidor a entrega da encomenda. Quando um expedidor aceitar a encomenda, a livraria informará o editor da identidade do expedidor (*ship_info*). O editor enviará a encomenda para o expedidor (book_to_s) e este procederá a sua entrega junto do cliente (book_to_c). O expedidor seguidamente notificará a livraria da concretização da entrega (*notification).* A livraria enviará a factura para casa do cliente (*bill*), ao que o cliente ao receber a factura e só depois de receber o livro, executará o respectivo pagamento (*payment*), o qual será recebido pela livraria encerrando o processo.

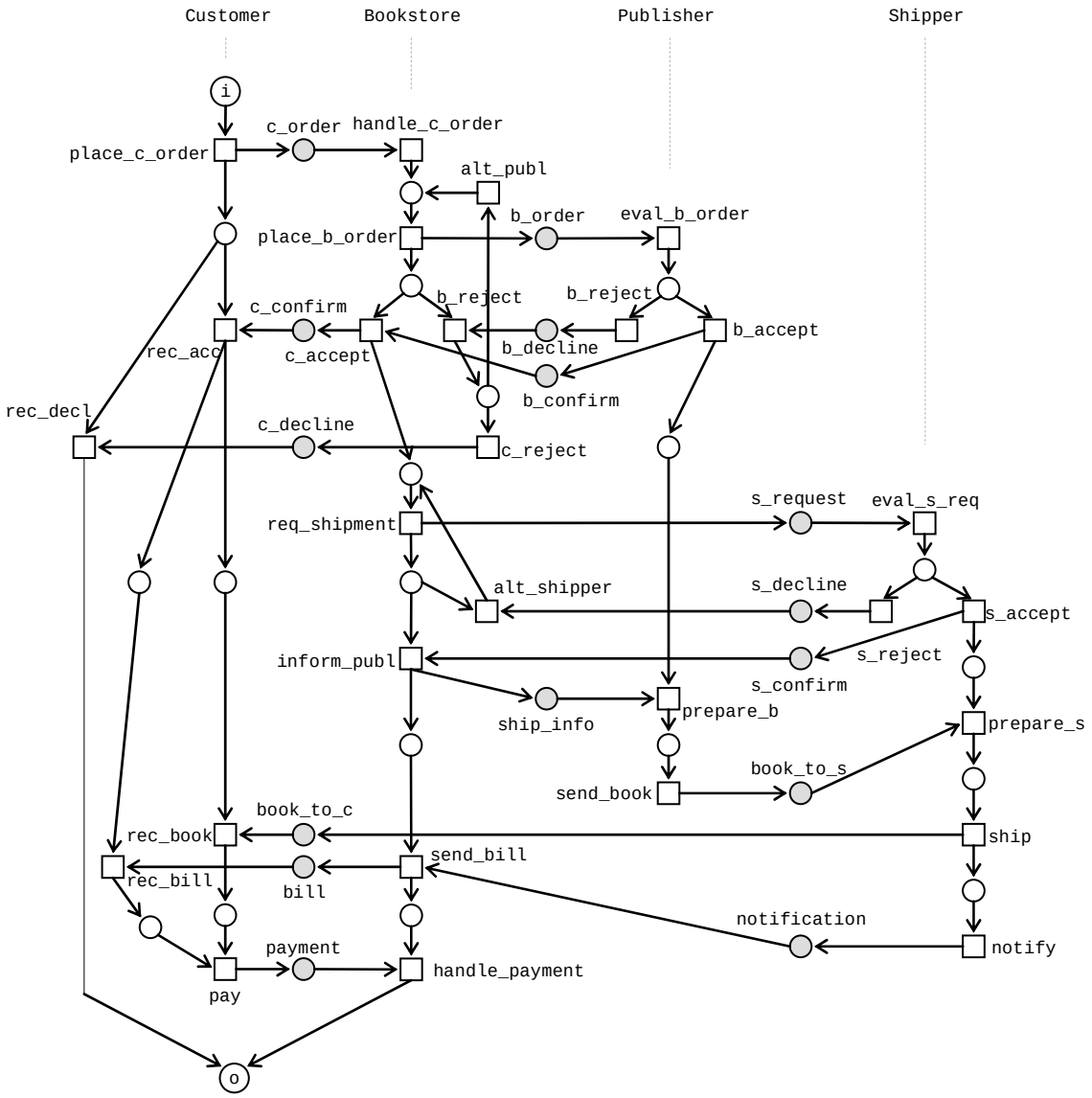


Figura 77 – Cenário da livraria electrónica

6.2.2 Descrição visual do cenário com vista global comum

Na figura 78 encontra-se a descrição comum do cenário da livraria electrónica. Esta descrição contém as interacções entre os vários participantes apresentada como um único processo. A referida figura apresenta o processo como uma rede de Petri, em que em cada lugar se refere: a uma actividade de troca de mensagem, onde se indicando o emissor, o receptor, e a mensagem (na forma: *emissor → receptor: mensagem*); ou uma actividade interna a um dos participantes (na forma: *participante: actividade*).

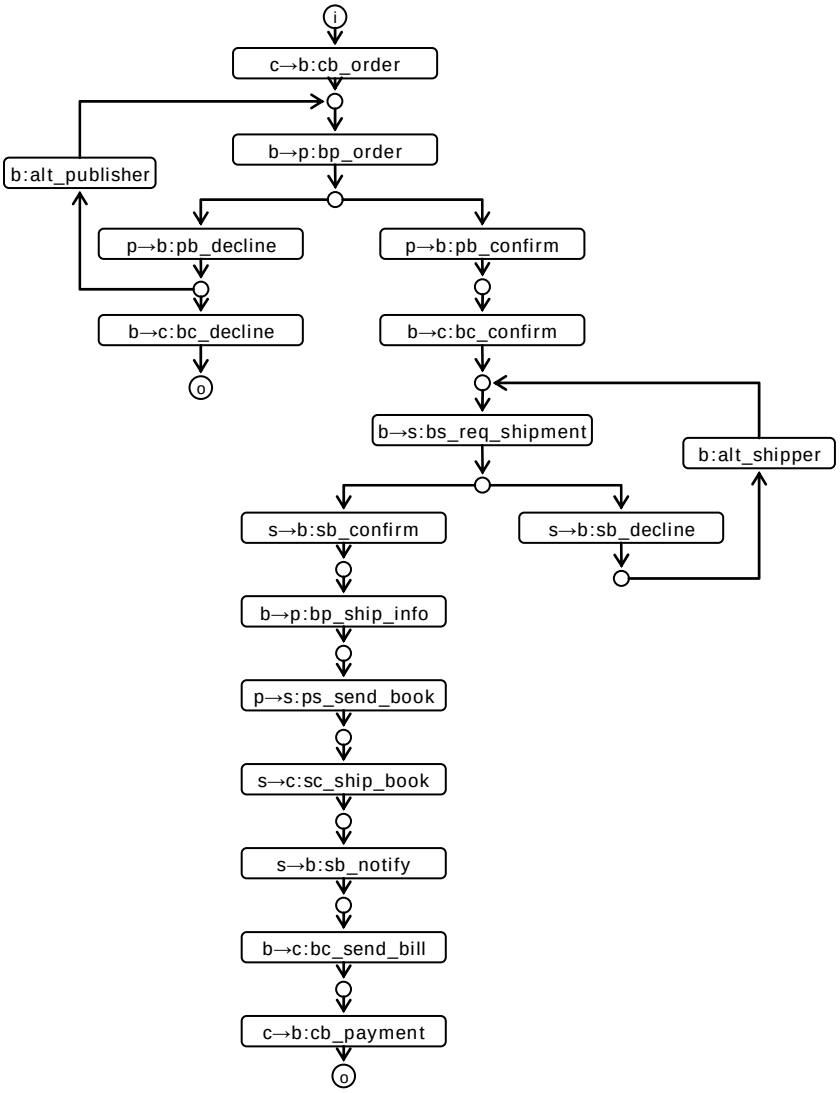


Figura 78 – Cenário da livraria electrónica descrito de forma comum

Comparando o processo descrito pela figura 77 e pela figura 78, que é o mesmo, pode-se constatar que a sua descrição, na figura 78, ficou mais compacta. No processo da figura 78 constam algumas actividades internas aos participantes, devido à necessidade de se apresentar uma rede de Petri bem construída. Considera-se que os dois pontos de

finalização da rede se encontram unidos formando um único. Neste tipo de processos o foco incide principalmente nas interacções, porque são elas que definem o contrato a estabelecer entre as várias partes envolvidas.

Este cenário irá ser implementado em CBPEL, numa primeira versão sem a utilização de *scopes*, e numa segunda versão com a utilização dos mesmos.

6.2.3 Descrição do cenário, sem a utilização de scopes

Nesta secção será implementado do cenário da livraria electrónica, em CBPEL, sem a utilização de *scopes*. Assim, na figura 79, encontra-se o cenário representado recorrendo a um diagrama com fluxo estruturado, pois a modelação em linguagem CBPEL, tendo em conta as limitações impostas neste trabalho, assim o requer. Como não é suportado a noção de escolha dinâmica de participantes, só é permitido a consulta a um *Publisher* e a um *Shipper*. Foi acrescentada a notificação *bp_cancel* de modo à *Bookstore* a notificar o *Publisher* de que a compra não irá prosseguir, caso o *Shipper* não aceite o serviço.

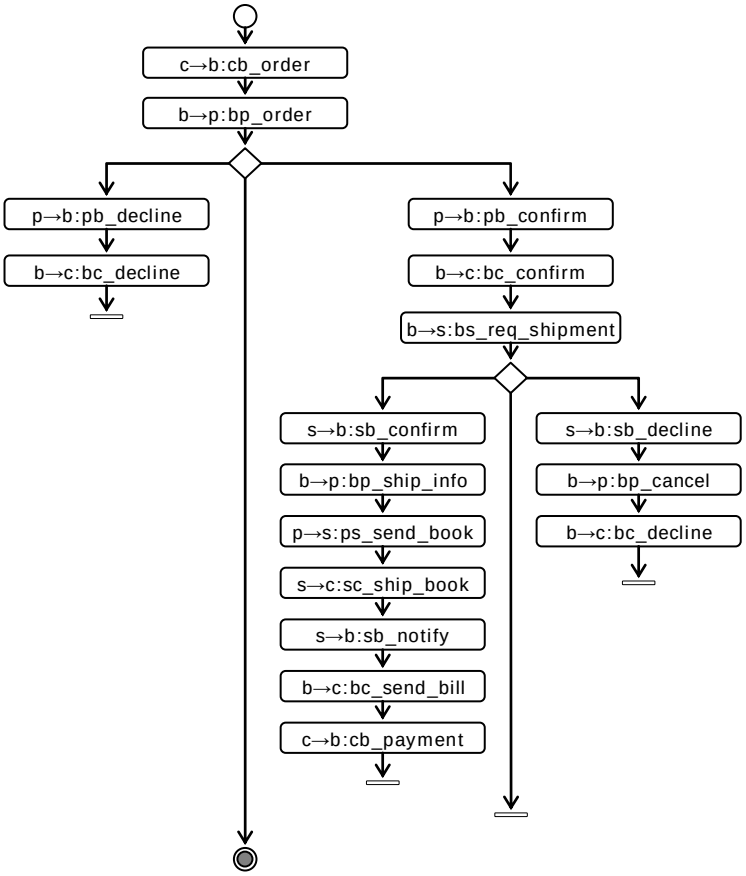


Figura 79 – Cenário da livraria electrónica com fluxo estruturado

Seguidamente será apresentado o processo CBPEL geral, e os processos BPEL dos participantes: *Customer*, *Bookstore*, *Publisher* e *Shipper*. Todos estes processos serão apresentados, de uma forma visual de seguida e em XML no anexo E. Aqui apresenta-se uma perspectiva visual desses mesmos processos, utilizando a estrutura em árvore idêntica à apresentada no cenário anterior.

Visualização do processo global em linguagem CBPEL

Na figura 80 encontra-se a visualização do processo CBPEL geral deste cenário.

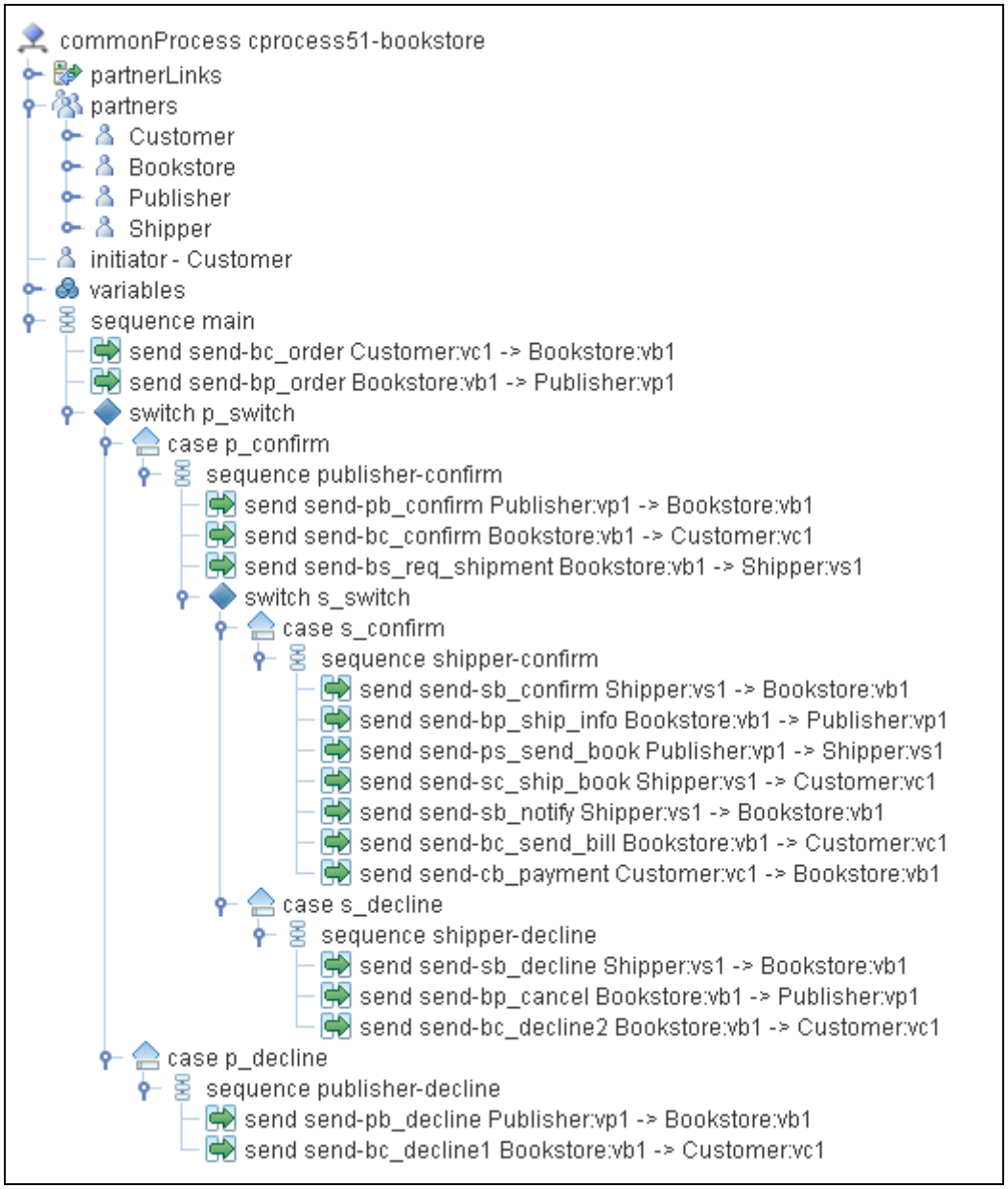


Figura 80 – Processo CBPEL da livraria sem scopes

Visualização do processo do participante Customer

Na figura 81 encontra-se a visualização do processo do participante *Customer*. Como se pode observar este participante, tem um papel passivo nas duas decisões existentes, pelo que terá de realizar uma espera diferida em ambas. Em cada uma dessas esperas, um dos ramos só contém a recepção de uma mensagem, mas como é obrigatório na linguagem BPEL a existência de pelo menos uma actividade, então teve-se de colocar a actividade de *empty* que nada faz.

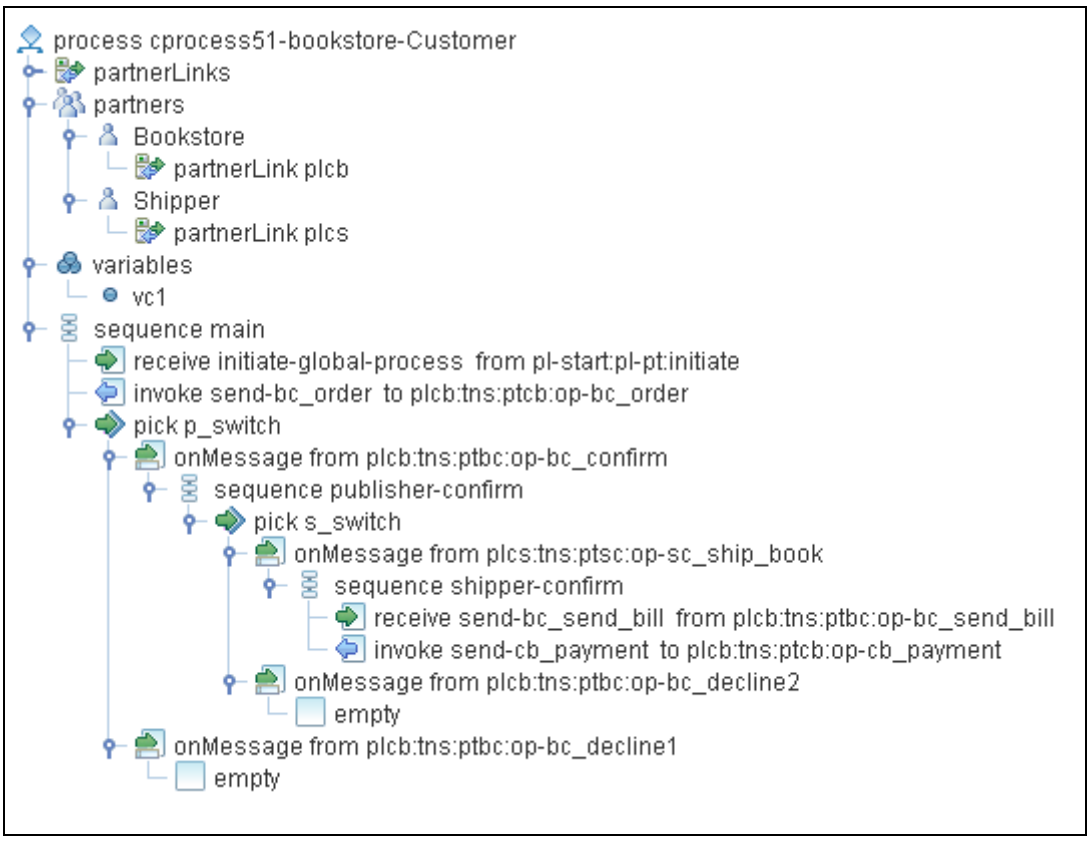


Figura 81 – Processo do Customer, na livraria sem scopes

Visualização do processo do participante Bookstore

Na figura 82 encontra-se a visualização do processo do participante *Bookstore*. Este participante também é passivo nas duas decisões pelo que o seu processo é estruturalmente idêntico ao processo do *Customer*.

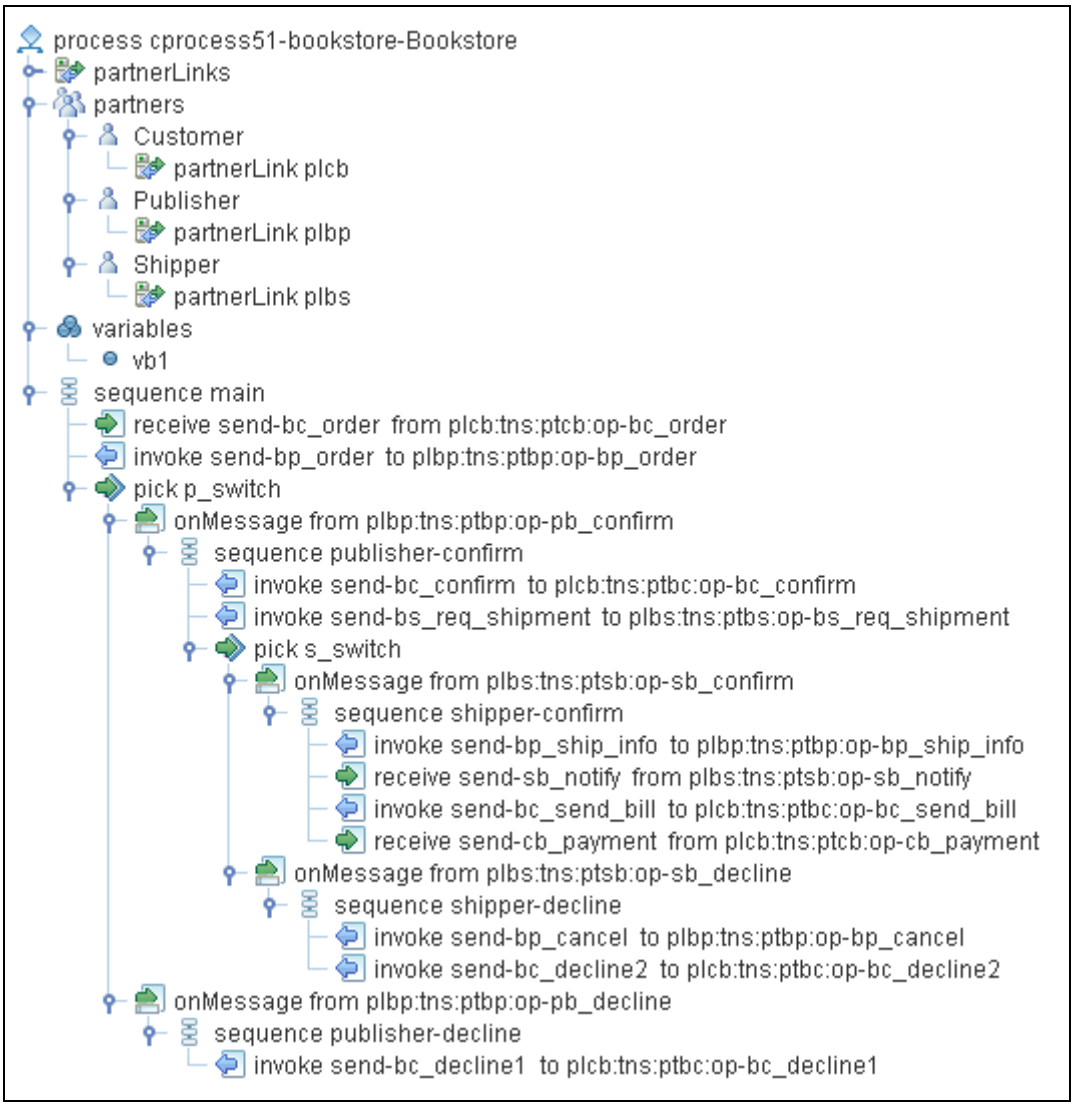


Figura 82 – Processo da Bookstore, na livraria sem scopes

Visualização do processo do participante Publisher

Na figura 83 encontra-se a visualização do processo do participante *Publisher*. Este é o participante que activamente executa a primeira decisão, pelo que esta actividade figura no seu processo como um actividade de decisão (*switch*). Na segunda decisão é também, à semelhança dos dois anteriores, um participante passivo.

Figura 83 – Processo do Publisher, na livraria sem scopes

Visualização do processo do participante Shipper

Na figura 84 encontra-se a visualização do processo do participante *Shipper*. Este participante não intervém na primeira decisão, mas executa activamente a segunda decisão.

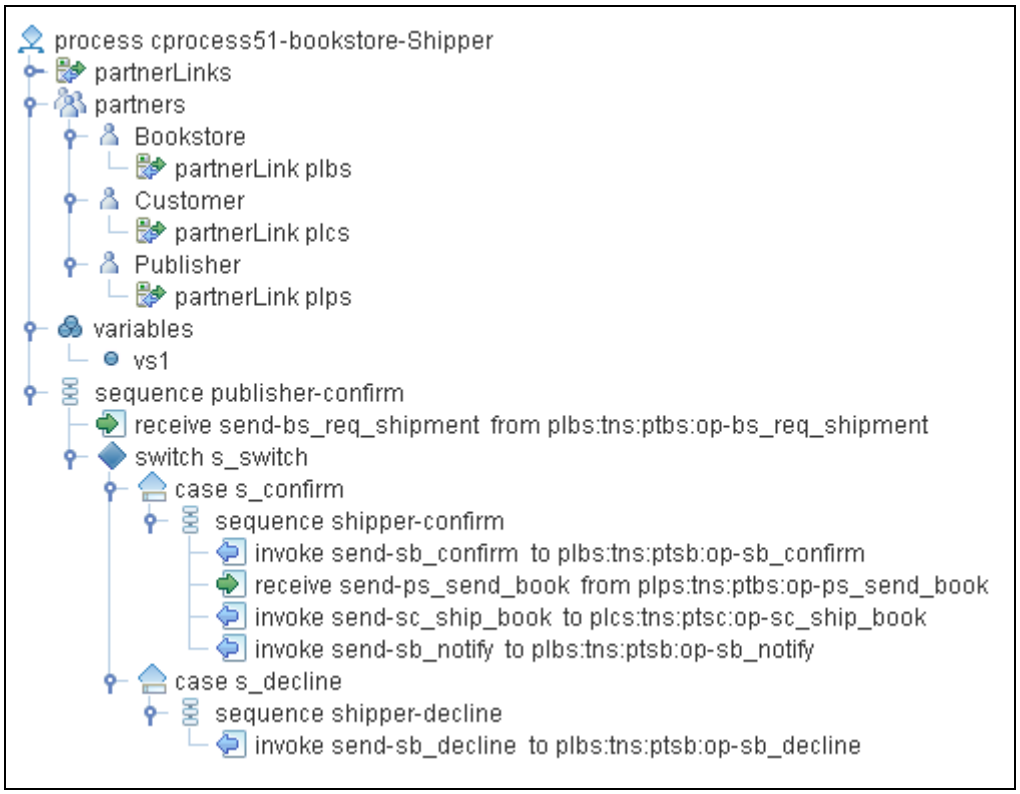


Figura 84 – Processo do Shipper, na livraria sem scopes

6.2.4 Descrição do cenário, utilizando scopes

A implementação do cenário na linguagem CBPEL utilizando *scopes* será efectuada dividindo o processo em partes, que ficarão cada uma no seu *scope*, o que irá permitir a sua compensação. A figura 85 apresenta uma descrição visual do cenário descrito nessa forma, onde do lado direito de cada *scope* se apresenta a sua rotina de compensação (ch).

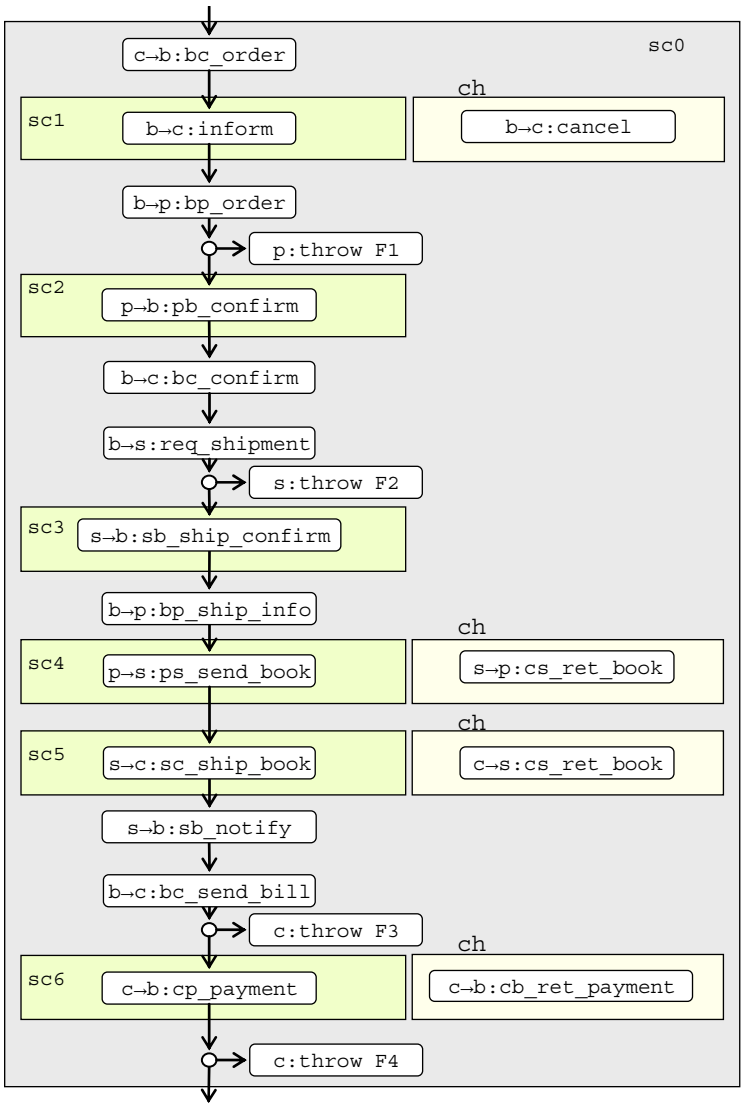


Figura 85 – Cenário da livraria electrónica em CBPEL com scopes

Os vários *scopes* são inseridos num *scope* englobante que é iniciado pelo *Customer*. Esse *scope* englobante vai ser o contexto onde as falhas serão apanhadas e onde os *scopes*, que entretanto já tiverem sido concluídos, serão compensados.

O processo descrito na figura 85 continua com a limitação de não permitir a escolha de um segundo editor ou de um segundo distribuidor, contudo acrescenta a possibilidade de o cliente devolver o artigo caso não tenha ficado satisfeito com ele quando o recebe, ou mesmo depois de o ter pago e tendo verificado que o artigo não corresponde às suas expectativas.

No caso do lançamento de uma falha, somente os participantes intervenientes nos *scopes* terminados até esse momento é que irão realizar a compensação.

Seguidamente apresenta-se o processo, da figura 85, descrito visualmente na linguagem CBPEL, seguido da visualização dos processos dos participantes em linguagem BPEL. As descrições na linguagem CBPEL e na linguagem BPEL serão realizadas novamente de forma sumária de modo a que os processos fiquem com uma dimensão razoável, ficando a sua descrição integral remetida para o anexo E.

Visualização do processo global em linguagem CBPEL

A visualização deste processo está dividida pela figura 86, que contém a parte inicial do processo e pela figura 87 que contém a parte com as actividades do processo.

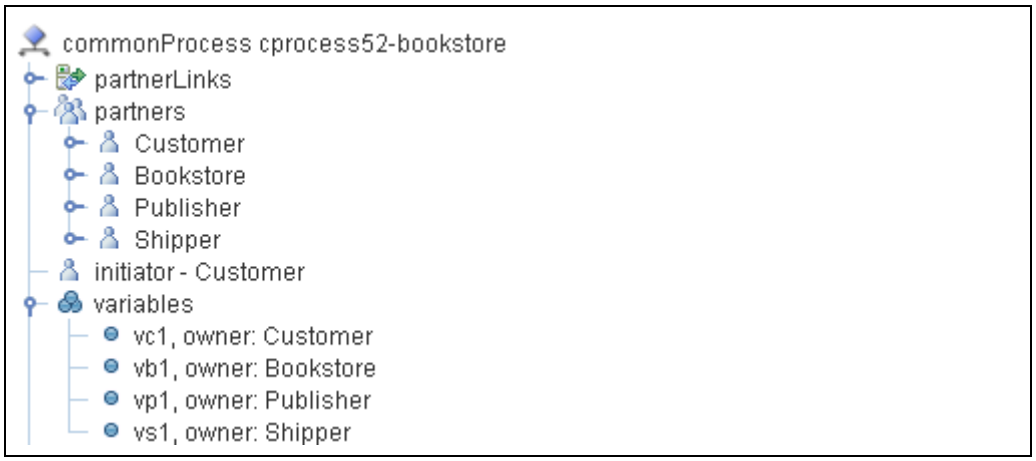


Figura 86 – Processo CBPEL da livraria com scopes – parte inicial

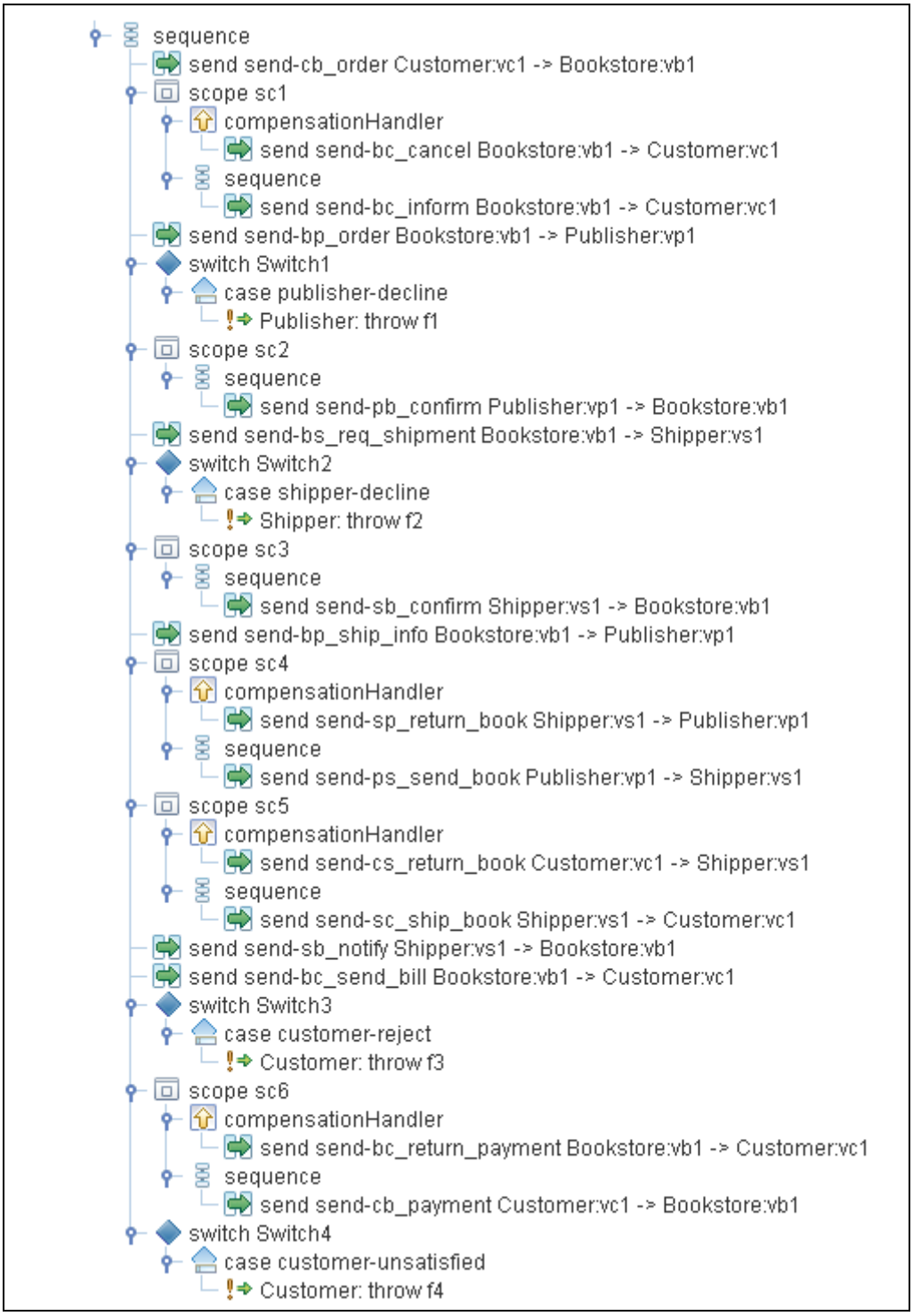


Figura 87 – Processo CBPEL da livraria com scopes – parte das actividades

Visualização do processo do participante Customer

Este processo vai estar dividido nas seguintes figuras: figura 88, que contém as declarações iniciais, e parte das actividades; figura 89, que contém a restante parte das actividades; e figura 90, que contém a visualização na íntegra das rotinas de tratamento de falhas, eventos assíncronos e compensação que cada *scope* contém.

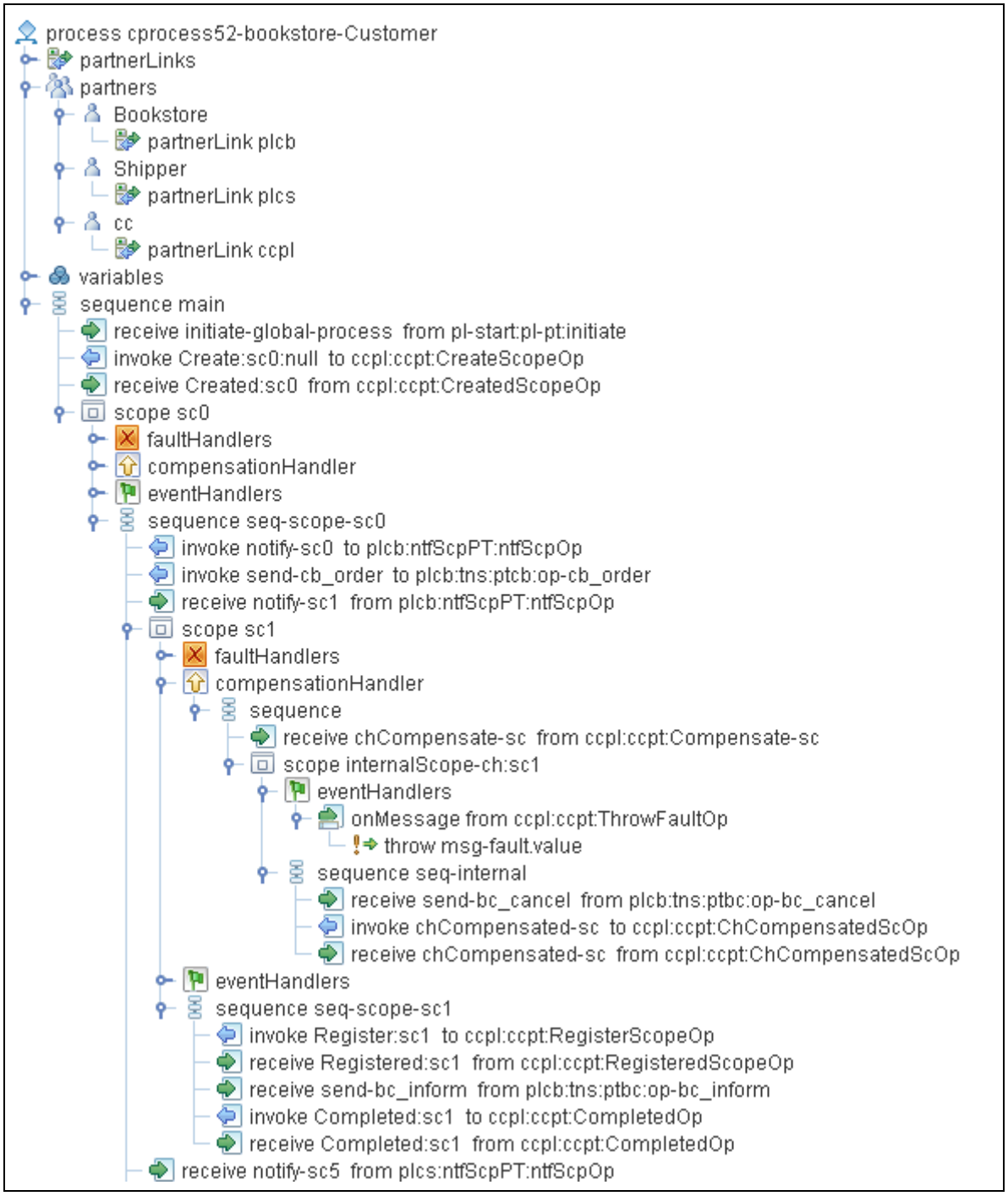


Figura 88 – Processo do Customer, na livraria com scopes – parte 1

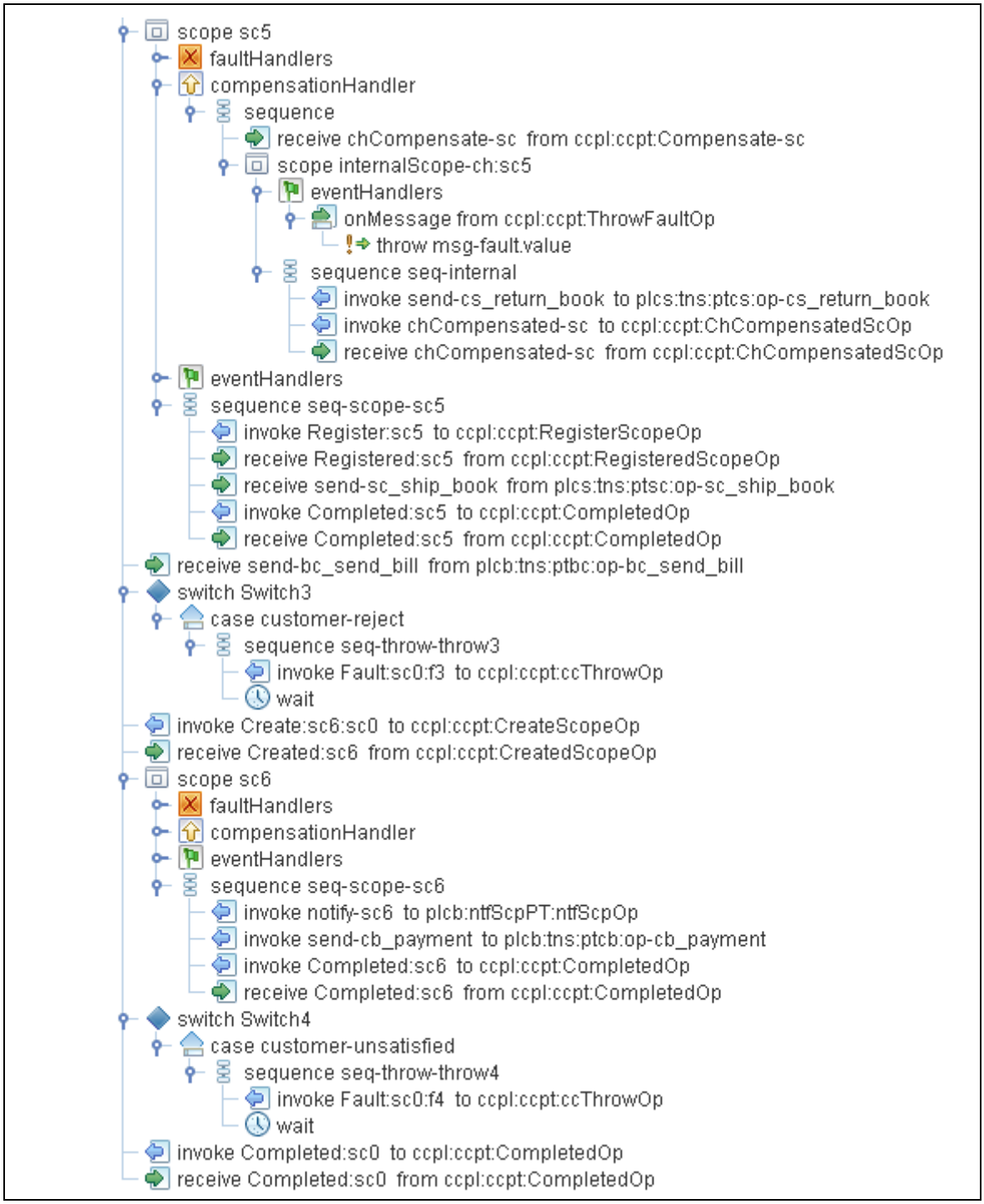


Figura 89 – Processo do Customer, na livraria com scopes – parte 2

A figura 90 mostra, portanto, as rotinas de tratamento de falhas que cada *scope* terá que conter. Uma delas destina-se a apanhar a falha de *ForcedTermination* e a outra a apanhar qualquer outra falha. Também se mostra a rotina de compensação, pois esta é a rotina de compensação por omissão, que iria aparecer nos outros participantes de forma

semelhante. Por fim mostra-se a rotina assíncrona de recepção da indicação do lançamento local de falha vinda do coordenador. Essa rotina irá existir em todos os *scopes*, e em todos os participantes. Essas rotinas daqui para a frente já não serão mais apresentadas, de forma a não alongar os processos de forma desnecessária.

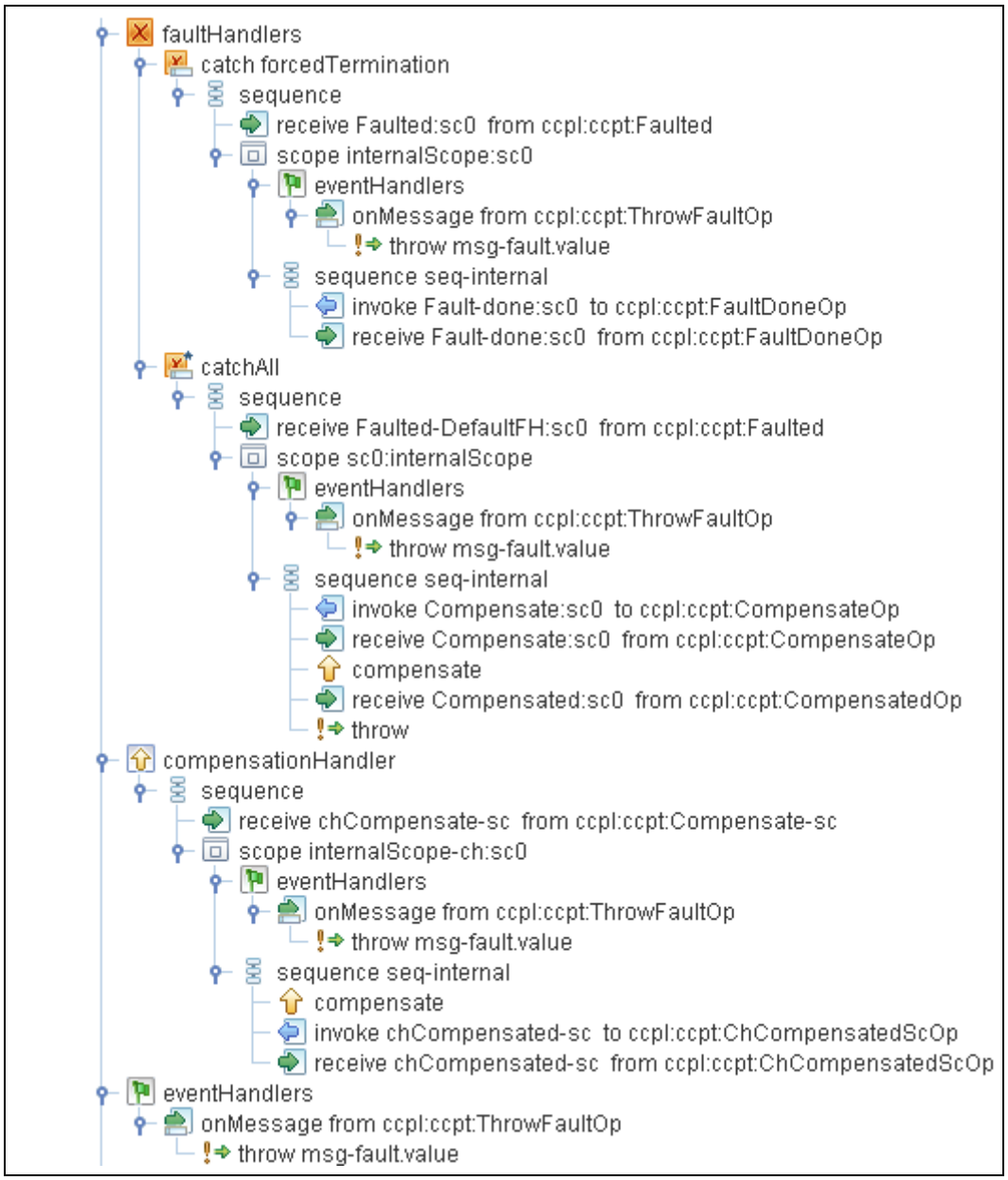


Figura 90 – Processo do Customer, na livraria com scopes – parte dos handlers

Visualização do processo do participante Bookstore

A visualização deste processo pode-se observar numa primeira parte na figura 91, e numa segunda parte na figura 92. Como já referido as rotinas de falha, compensação, e assíncronas semelhantes já apresentadas, no *Customer*, não são visualizadas.

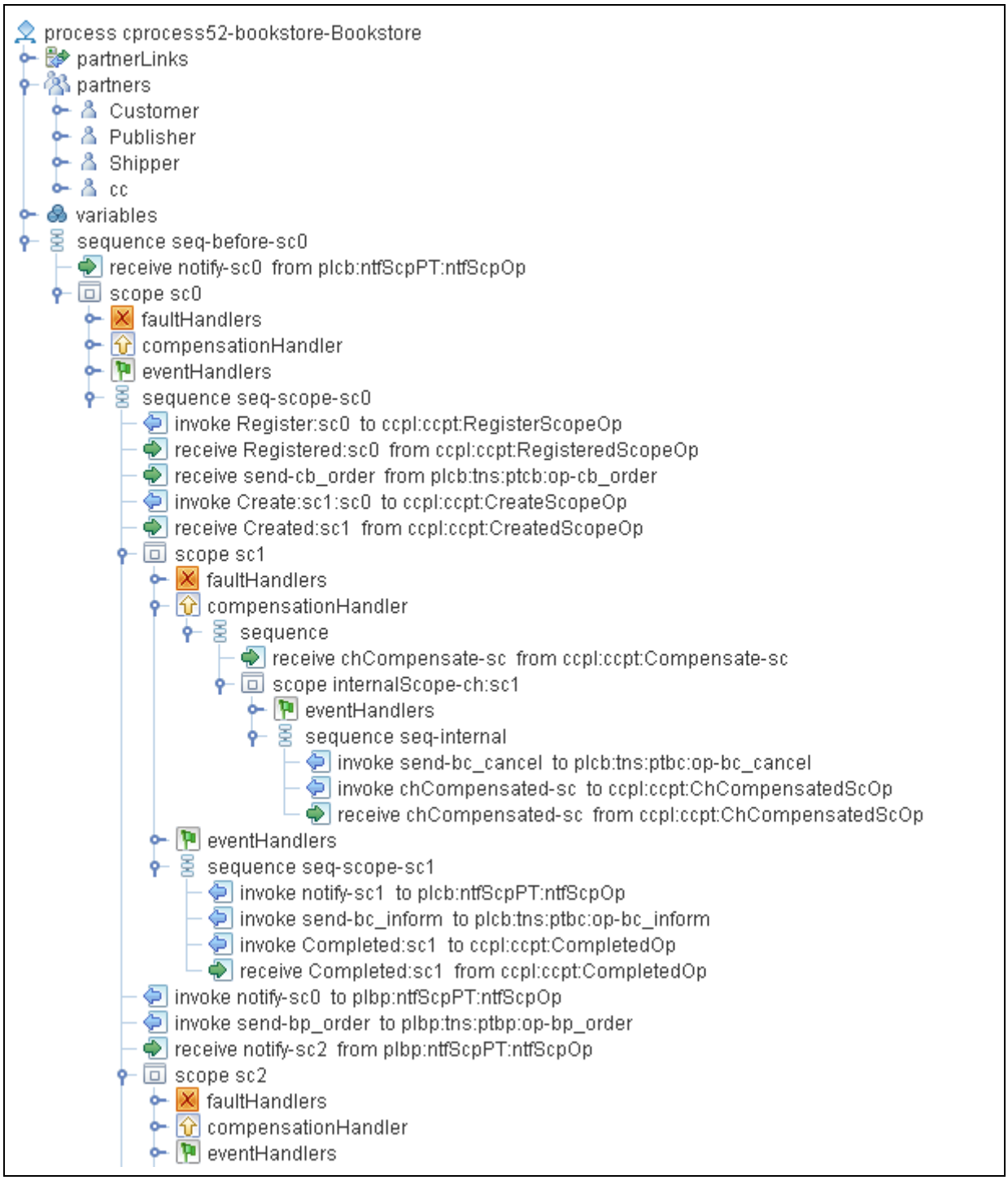


Figura 91 – Processo do Bookstore, na livraria com scopes – parte 1

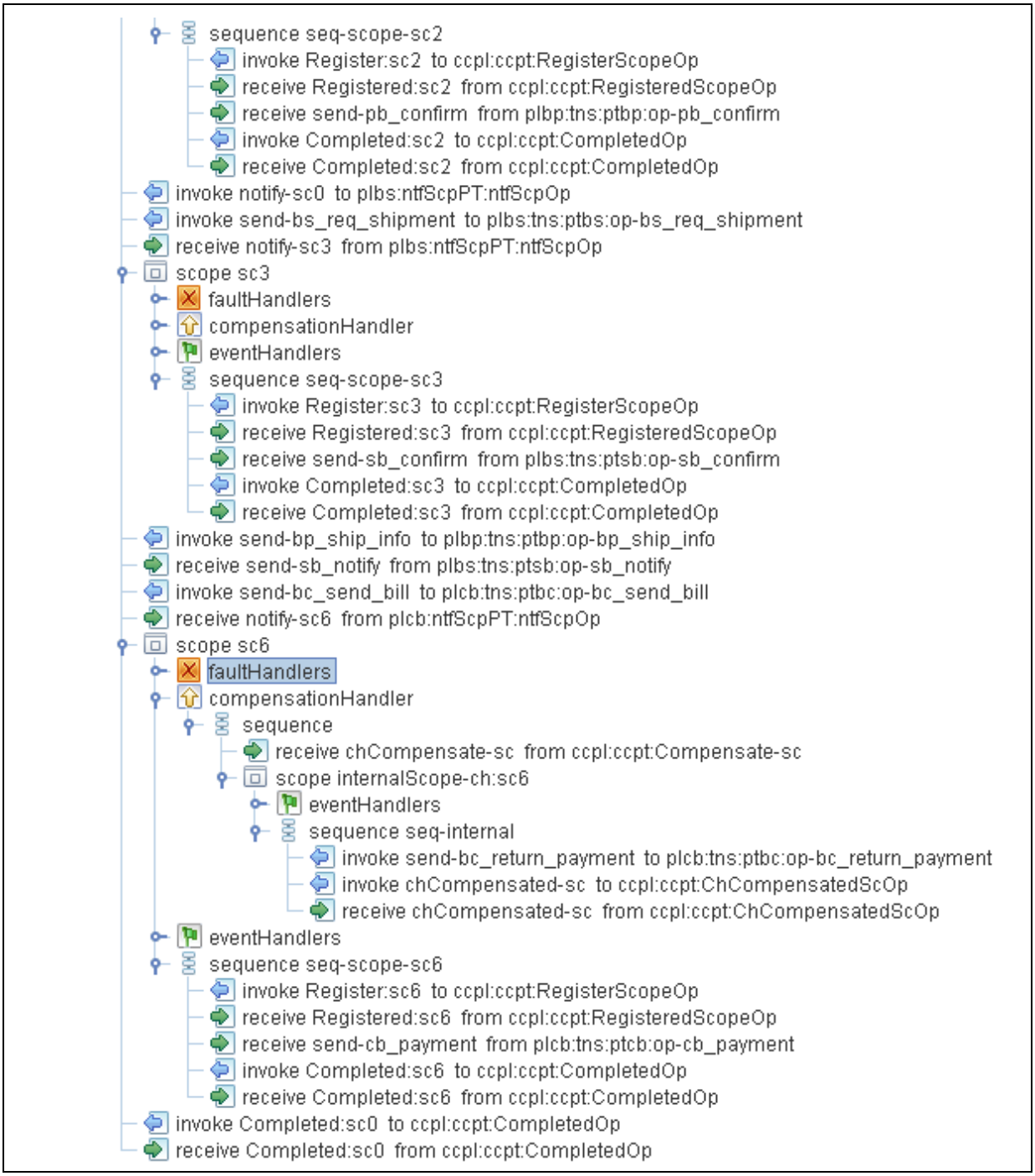


Figura 92 – Processo do Bookstore, na livraria com scopes – parte 2

Visualização do processo do participante Publisher

A visualização deste processo pode-se observar na figura 93.

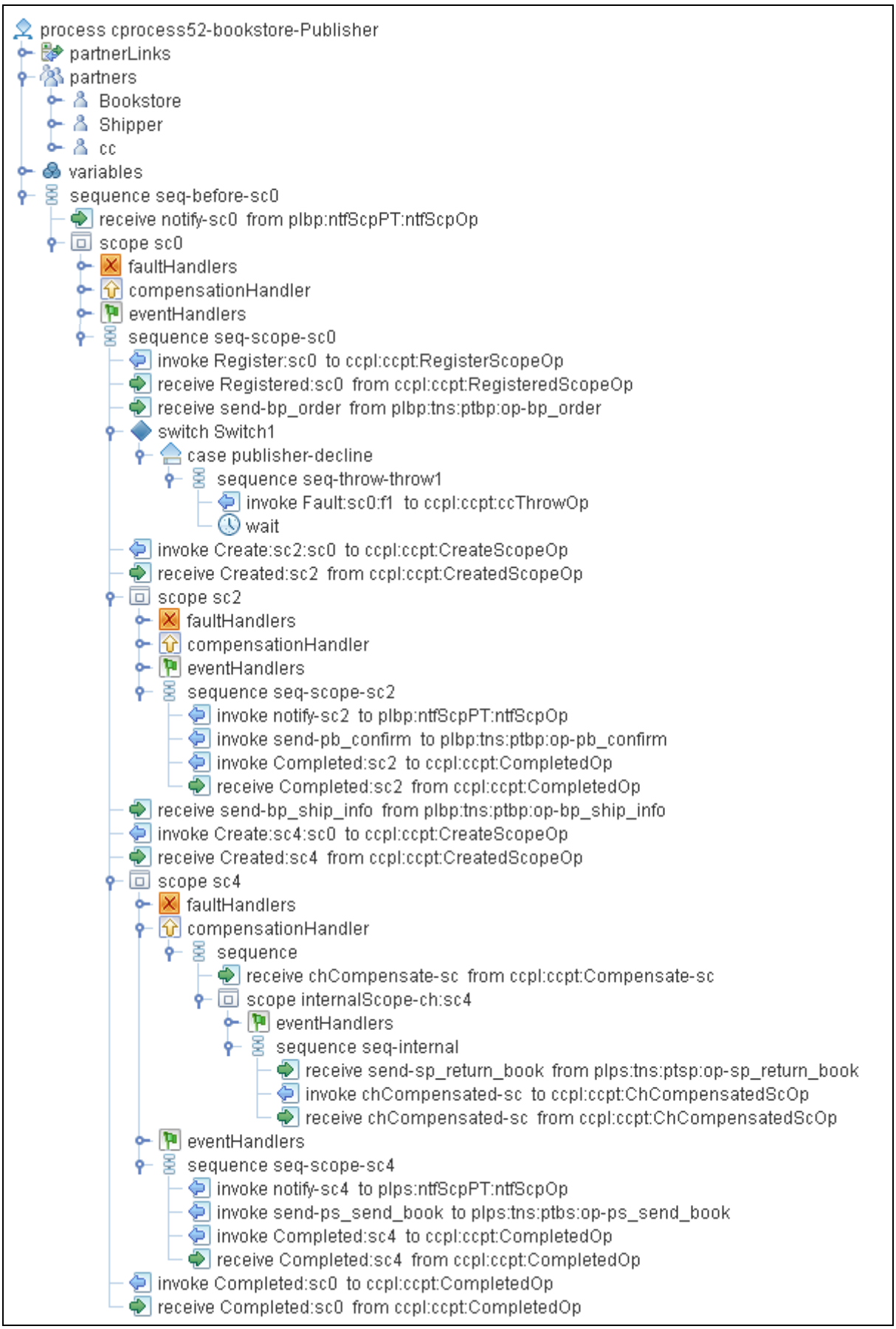


Figura 93 – Processo do Publisher, na livraria com scopes

Visualização do processo do participante Shipper

A visualização do processo deste participante pode observar-se na figura 94 e na figura 95.

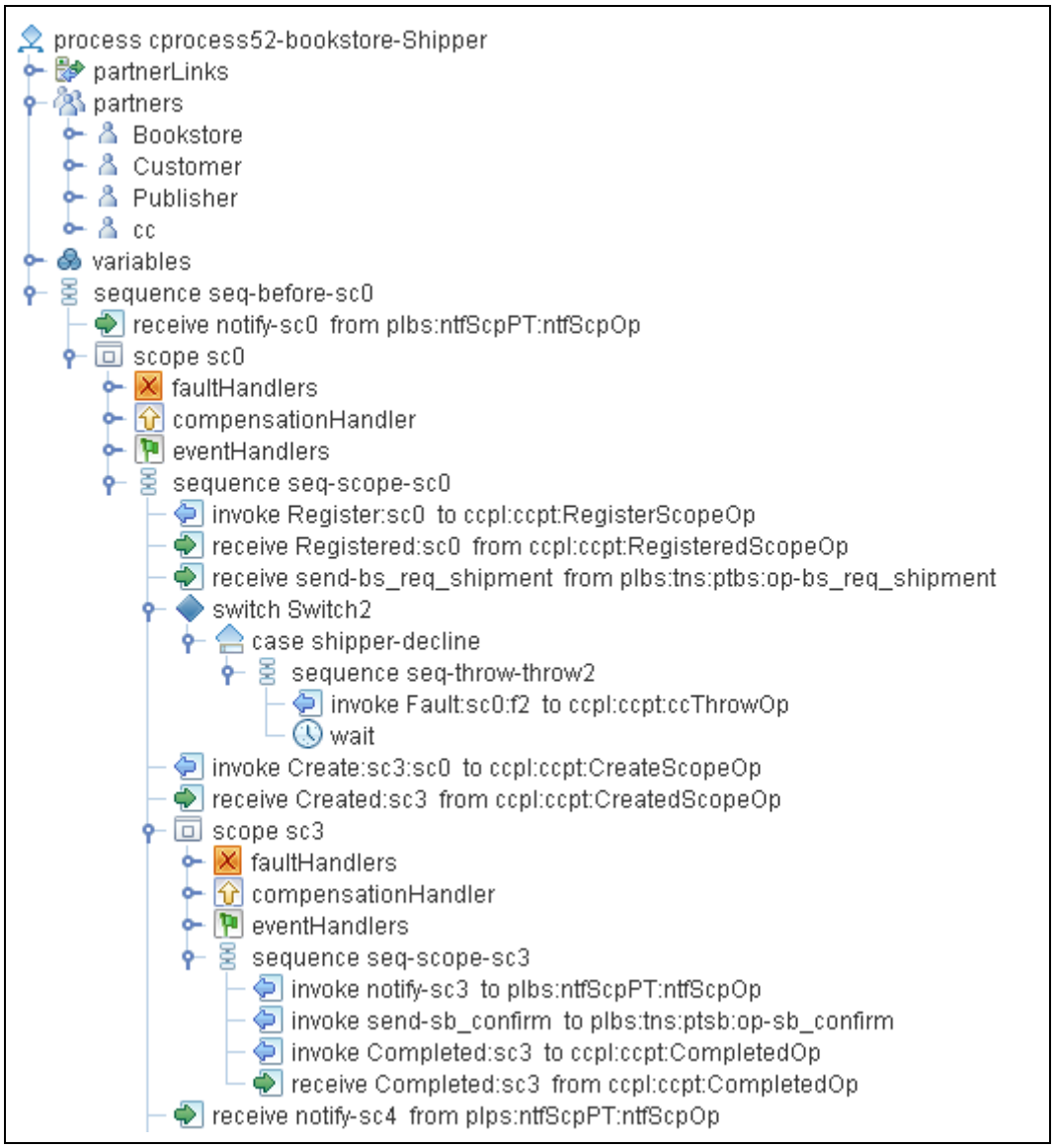


Figura 94 – Processo do Shipper, na livraria com scopes – parte 1

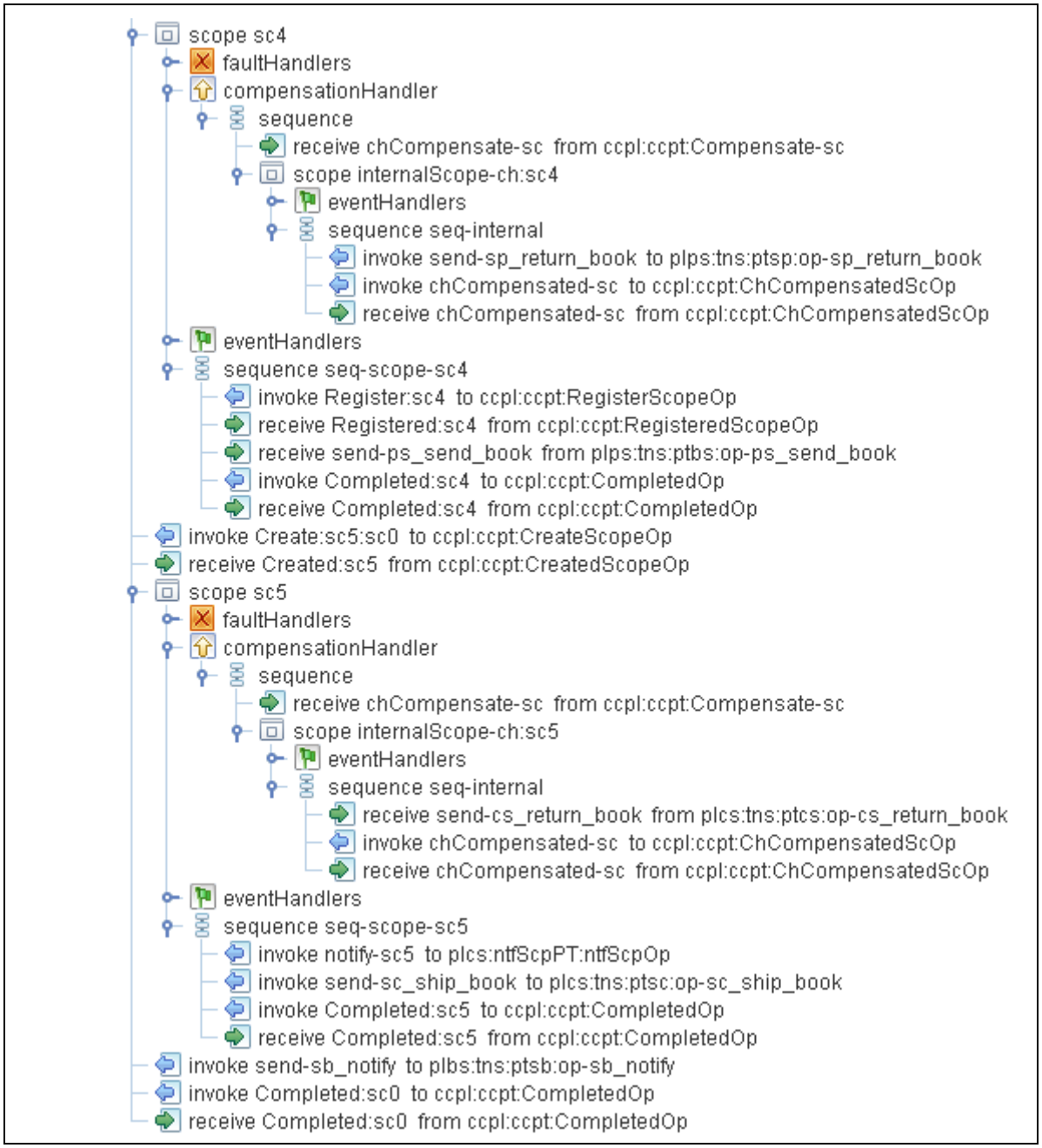


Figura 95 – Processo do Shipper, na livraria com scopes – parte 2

6.3 Conclusão

Uma primeira conclusão é que ambos os cenários foram modelados de forma global e comum pela linguagem CBPEL. A modelação apresenta algumas limitações, mas a essência dos cenários foi preservada.

Uma segunda conclusão é que dos processos concebidos em CBPEL se conseguiu gerar os processos dos participantes na linguagem BPEL. A geração apresenta algumas limitações, mas tal também não desvirtuou os processos gerados.

Uma terceira conclusão, é que a descrição comum e global pode ser bastante mais compacta que a descrição normal deste tipo de processos. Apesar de apenas no caso do cenário da livraria electrónica se apresentar o processo descrito de forma normal, na figura 77, e de forma global e comum na figura 78, pode-se generalizar que a descrição global e comum necessitará decerto de menos declarações que a descrição normal, pois só declara um fluxo e cada sua actividade de troca de mensagem terá que corresponder a duas actividades, uma em cada um dos processos dos participantes envolvidos na descrição normal.

Uma quarta conclusão, é que a utilização blocos coordenados para o controlo de falhas, é realizável. É também de se salientar a enorme diferença de extensão entre o processo concebido e os processos gerados. A complexidade dos processos gerados deve ser vista, como uma consequência da funcionalidade existente e indicadora da premência da geração automática deste tipo de processos.

Capítulo 7

Discussão

Uma vez terminada a apresentação da linguagem CBPEL, é altura para se apresentar uma reflexão acerca: do suporte proporcionado pela linguagem BPEL; da versão utilizada da linguagem BPEL; e do relacionamento entre a linguagem CBPEL e a linguagem WSCDL. Assim este capítulo contém três secções que abordam os itens já descritos bem, como uma conclusão acerca do supracitado.

7.1 Considerações sobre a linguagem BPEL

Ao longo dos capítulos 5 e 6 são identificados os seguintes aspectos que se desejariam poder modificar na linguagem BPEL, de forma a esta poder suportar processos gerados a partir de processos CBPEL:

- Que as falhas internas à plataforma, fossem geradas segundo o modelo da linguagem CBPEL;
- Que existisse forma de assegurar o sincronismo de entrada e saída nos *scopes*;
- Que existisse forma de assegurar a sequencialidade da saída dos scopes; e
- Que permitisse a definição automática de, por exemplo por intermédio de anotações da indicação de *scopes* internos nas rotinas assíncronas, de tratamento de falha e de compensação.

Sendo a linguagem BPEL, uma linguagem destinada a descrever orquestrações, ela apresenta os problemas esperados no suporte a processos envolvidos numa coreografia.

As dificuldades verificadas, poderão eventualmente ser contempladas de forma a não alterar a própria linguagem BPEL recorrendo, como já referido, a anotações.

7.2 Considerações sobre a versão da linguagem BPEL utilizada

Neste trabalho foi utilizada a versão 1.1 da linguagem BPEL [Andrews03], contudo esta linguagem desde 2007 possui a versão 2.0 [Jordan07]. Dado o demasiado comprometimento de todo o software desenvolvido de acordo com a versão 1.1, fica para trabalho futuro a sua adaptação para a versão 2.0 da linguagem BPEL. No entanto identificam-se aqui as alterações mais significativas existentes na versão 2.0, e apresenta-se uma discussão acerca das suas consequências neste trabalho. Eis portanto as alterações que a versão 2.0 da linguagem BPEL tem para com a sua versão 1.0:

- *Process* não é compensável

  O facto de um processo BPEL não ser compensável inviabiliza a utilização processos BPEL como sub-processos de outros processos BPEL. Consequentemente a utilização de processos CBPEL como sub-processos de outros processos CBPEL fica também comprometida.

- Os *onAlarm* e *OnEvent* dos *eventHandlers* têm um *scope* como actividade interna.

  A existência de um *scope*, dentro das rotinas assíncronas na BPEL, não influência o comportamento desta linguagem no suporte à linguagem CBPEL.

- Actividade de *scope* com *terminationHandler*

  O *terminationHandler* vem substituir a falha de *forcedTermination* pela utilização de uma rotina expressamente declarada para o mesmo efeito. Como a propagação de *termination* é interna à plataforma, a alteração não apresenta consequências para a tradução de CBPEL para BPEL. A CBPEL deverá então alterar a sua declaração de scope de modo a contemplar esta alteração.

- Actividade de *scope* e *process* com atributo *exitOnStandardFault*

  O atributo de *exitOnStandardFault* quando a *true*, indica se ocorrer uma falha standard, que não seja apanhada, que deve ser executado a actividade de *exit*.

Como na CBPEL não se pretende que os processos individuais terminem de forma isolada, mas sim seguindo um qualquer modelo acordado entre os participantes, este atributo não deve ser incorporado nessa linguagem.

– Actividade de *scope* com atributo *isolated* em vez de *variableAccessSerializable*

Estes atributos quando activos indicam que o acesso a recursos partilhados é serializado, e destina-se a evitar problemas de concorrência. Apesar de semelhante, a semântica de comportamento com *isolated* apresenta algumas variações face à semântica do atributo anterior. Mas tal como já referido no ponto 5.3.1 a problemática do acesso concorrencial é do domínio local aos vários participantes pelo que não é um factor pertinente para a linguagem CBPEL.

– Actividade de *if* em vez de *switch*

Esta alteração não tem consequências para a linguagem CBPEL, exceptuando a alteração lexicográfica, pois é somente uma alteração de nome e estrutura da actividade que permite uma decisão com múltiplos ramos.

– Nova actividade de *validate*

É uma actividade destinada a validar as variáveis face à sua definição. É portanto uma actividade do âmbito individual, que pode existir em CBPEL, mas devendo possuir a identificação de quem será o seu executor.

– Nova actividade de *repeatUntil*

É uma actividade semelhante ao *while*, mas em que a condição só é testada no final da execução da actividade interna. A sua tradução para BPEL, requer uma análise semelhante à existente no anexo B.

– Nova actividade de *forEach*

A actividade de *forEach* permite executar várias instâncias de um *scope*. Essas instâncias podem ser executados em série ou em paralelo, e o seu número pode ser condicionado por uma expressão e pelo número de *scopes* que entretanto tenham terminado com sucesso.

A utilização desta actividade em CBPEL implica que todos os participantes envolvidos dentro da actividade: ou soubessem qual o número de instâncias, que

seriam criadas, em que eles estariam envolvidos; ou que teriam que se preparar para receberem até um certo número máximo de mensagens. Apear de as duas situações não se afigurarem como suficientemente práticas para uma tradução automática de CBPEL para BPEL, não se exclui a possibilidade, de com um estudo mais aprofundado, de incluir futuramente esta actividade na linguagem CBPEL.

– Nova actividade de *rethrow*

  Actividade destinada a relançar a falha que originou a execução do *faultHandler* em que esta actividade se encontra e é executada. Tem a vantagem de num *faultHandler* que apanhe várias falhas, esta actividade executa automaticamente o relançamento da falha original. A sua utilização em CBPEL poderá será considerada pois a colocação de *rethrow* resultará no envio de uma mensagem de *rethrow* para o coordenador e este remeterá a falha correcta para os participantes. O coordenador não poderá enviar uma mensagem genérica também de *rethrow* para os participantes, pois estes terão que apanhar a mensagem numa rotina assíncrona que terá que existir num *scope* local para esse efeito.

– *CompensateScope* e *compensate* em vez de só compensate.

  Apenas separa as duas funcionalidades em duas actividades distintas. A CBPEL pode incorporar sem problemas esta alteração.

7.3 Considerações sobre a linguagem WSCDL

Uma vez terminada a definição da linguagem CBPEL, surge o momento de se fazer uma análise comparativa entre essa linguagem e a linguagem WSCDL.

Tal como já foi referido, a linguagem WSCDL é uma linguagem de descrição de coreografias de forma global e comum, tal como a linguagem CBPEL. De forma a se ter uma noção das suas capacidades vai ser apresentado um sumário das suas funcionalidades e indicado se a linguagem CBPEL as suporta ou não.

Segue-se, portanto, um levantamento sumário das funcionalidades descritas na especificação da linguagem WSCDL [Kavantzas05] relativas ao controlo do fluxo da coreografia em termos de: fluxo normal; fluxos de excepção; e fluxos de finalização. Cada ponto terá um comentário indicando se a referida funcionalidade é suportada, ou não, na

linguagem CBPEL em conjunto com a linguagem BPEL e se haverá alguma condicionante.

1. As coreografias podem ter um *exceptionBlock* que declara um ou mais *workunit*. Cada *workunit* pode ter o nome de uma excepção como guarda, devendo ser activo quando essa excepção ocorrer. A escolha do *workunit*, aquando da ocorrência de uma falha é baseada na ordem da sua declaração, sendo activado o primeiro que apanhe a excepção em questão.

   Na CBPEL, assim como na BPEL, permite-se a declaração de rotinas, denominadas de *faultHandler*, dedicadas ao tratamento de excepções específicas. A escolha do *faultHandler* na CBPEL/BPEL também é realizada pela escolha da primeira rotina que apanhe a excepção/falha ocorrida, segundo a ordem de declaração.

2. Pode haver um bloco que não tenha guarda. Esse bloco será executado para qualquer excepção, que não seja apanhada nos blocos com guarda.

   Na CBPEL, assim como na BPEL, permite-se a declaração de um *faultHandler* para o tratamento por omissão de uma qualquer excepção não apanhada pelas outras declarações.

3. Dentro do *exceptionBlock* de uma coreografia apenas um *workunit* pode será activado.

   Na CBPEL/BPEL o comportamento é idêntico, pois apenas um *faultHandler* pode ser activado.

4. Quando ocorre uma excepção numa coreografia, esta termina sem sucesso, passando para o estado *Unsuccessfully Completed State*, e as actividades dentro dela, incluindo as suas coreografias internas, em execução, são terminadas anormalmente.

   Na CBPEL/BPEL o comportamento é semelhante mas mais explícito, pois é esclarecido que deve ser enviado a falha de *ForcedTermination* para os *scopes* internos.

5. Uma excepção pode ter uma indicação para ser lançada nos seguintes casos: numa actividade de *assign*; numa resposta de um participante para outro, na sua

componente de envio e ou de recepção; no caso de se explicitar um tempo máximo (*timeout*) para recepção de uma resposta.

Na CBPEL/BPEL apenas é suportado o lançamento de uma excepção pela actividade de *throw*, que tem uma funcionalidade análoga à actividade de *assign,* com *causeException*, na WSCDL. O lançamento de uma excepção, em conjunto com a sua comunicação, não foi considerado, pois como todos os blocos têm comportamento coordenado, então basta que apenas um participante gere uma excepção, que todos irão receber uma notificação do coordenador para lançar a mesma excepção. A sua comunicação fica portanto redundante. A CBPEL não suporta, para já, a funcionalidade de *timeout* associado a um *scope*, ou a qualquer outra coisa, pois considerou-se que uma sincronização temporal precisa iria requerer coordenação, e que tal estudo já estaria fora da complexidade requerida para esta dissertação, dada a sua actual extensão. Contudo, o suporte a *timeouts* na WSCDL pressupõe que os participantes estão "razoavelmente bem sincronizados ao nível do segundo" [Kavantzas05], e indica que o tempo começa a contar assim que a *interaction* se inicia, não esclarecendo que tal pode ocorrer em instantes distintos nos vários participantes envolvidos. Constata-se, portanto, que a WSCDL não oferece uma forma fidedigna de implementar *timeouts*.

6. O lançamento de uma excepção é sempre local a um participante. A propagação da excepção, no caso de uma coreografia não coordenada, deve ser realizado de forma explícita entre os participantes, através de interacções com essa indicação. No caso de uma coreografia coordenada fica remetido para o protocolo de coordenação notificar os participantes que não tinham conhecimento da excepção, da ocorrência da mesma.

   Nos processos gerados na BPEL, a ocorrência de uma excepção é comunicada ao coordenador e o processo fica em espera, sendo depois realizado o lançamento local no resultado da recepção da indicação de ocorrência de excepção vinda do coordenador. Deste modo uniformiza-se o lançamento da excepção em todos os participantes, garante-se que todos são notificados, e no coordenador fica registado a ocorrência de excepção o que permite controlar o desenvolvimento do *scope* a partir desse ponto. Na WSCDL a indicação de que o coordenador notifica os participantes que não tinham conhecimento da excepção é ambíguo, porque não se esclarece como o coordenador tem conhecimento de quais são esses participantes.

7. Dentro do processamento de uma excepção, poderá ocorrer uma outra excepção, sendo esta última processada de "forma usual" [Kavantzas05].

   Considerando que, de forma usual, corresponde a que a excepção será propagada para a coreografia acima daquela onde o *exceptionBlock*/*workunit* estava a ser executado, então esse é o comportamento idêntico ao da CBPEL/BPEL.

8. Caso a coreografia não tenha o *exceptionBlock*, ou não tenha um *workunit* compatível com a excepção gerada, a coreografia entra no estado de *Closed*, e a excepção é propagada para a coreografia que a engloba. Esse procedimento é recursivo até que uma coreografia apanhe a excepção.

   Na CBPEL/BPEL o comportamento é idêntico, utilizando a noção de *scope* em vez de coreografia.

9. Uma coreografia quando termina com sucesso, por não ter mais actividades activas, desactiva o *exceptionBlock* se o tiver, activa os *finalizerBlock* que tiver e fica no estado de *Successfully Completed State*.

   Na CBPEL/BPEL o comportamento é idêntico, no entanto o suporte a rotinas de finalização com funcionalidade semelhante a um *finalizerBlock*, é limitado à rotina de compensação (*compensationHandler*).

10. Uma coreografia deve completar com sucesso, caso a sua condição de término, (*completeCondition)* assuma o valor de verdadeiro. Deve ser possível a todos os participantes na coreografia afectarem a condição de término.

    Na CBPEL/BPEL não existe a noção de se poder terminar um *scope* por afectação de uma variável. Na WSCDL, é indicado que a *completeCondition* deve poder ser afectada por todos os participantes. Só que isso não oferece nenhuma garantia de que uns deles afectem a *completeCondition* e um outro lance uma excepção vinda por exemplo de um seu *subscope* ou de uma falha de comunicação, provocando uma inconsistência no processo global. De modo geral uma coreografia só deveria terminar com sucesso após todos os seus participantes terem dado essa indicação.

11. Quando a condição de término de uma coreografia é afectada a verdadeiro, todas as actividades dentro da coreografia devem ficar desactivas, excepto os *finalizerBlock*, e a coreografia termina tal como se não houvesse nenhuma

actividade activa. Quando uma coreografia termina, todas as coreografias activas, dentro dela, devem terminar com sucesso.

Aplica-se a mesma consideração do ponto anterior.

12. A actividade de *finalize*, que permite activar um *finalizerBlock* de uma coreografia imediatamente englobada, pode ser executada no fluxo normal da coreografia, num *exceptionBlock* e num *finalizerBlock*.

    Na CBPEL/BPEL a actividade correspondente a *finalize* será a actividade *compensate* ou *compensateScope*. A segunda actividade activa a rotina de compensação de um *scope* terminado com sucesso, enquanto que a primeira activa as rotinas de compensação de todos *scopes* interiores ao *scope,* onde a actividade for executada, que foram terminados com sucesso. Os *scopes* a serem compensados têm de estar imediatamente colocados dentro do *scope* de execução da actividade de compensação. Contudo as actividades de *compensate* apenas podem ser executadas dentro de uma rotina de tratamento de falha (*faultHandler*) ou de uma rotina de compensação (*compensation*H*andler*). Verifica-se portanto que a WSCDL não possui uma actividade análoga à de *compensate*. Mas como a WSCDL assume que a finalização pode ser mais abrangente do que uma compensação, então tal requer a capacidade de se evocar a actividade de *finalize* em diferentes e mais situações do que na CBPEL/BPEL. Na WSCDL não é especificado quem deve executar o *finalize* nem é especificado como a coerência de finalização é garantida, ou seja, como os processos se comportam se uns participantes executarem a actividade de *finalize* e outros não o poderem fazer por neles ter ocorrido uma excepção.

13. Caso ocorra uma excepção num *finalizerBlock* ela será processada de "forma usual".

    Na CBPEL/BPEL, uma falha ocorrida numa rotina de compensação é propagada para o *scope* acima, para o contexto que evocou a rotina de compensação, que pode ser uma rotina de compensação ou uma rotina de tratamento de uma falha/excepção. Na WSCDL o significado de "forma usual" pode corresponder à funcionalidade já referida na CBPEL/BPEL, ou seja, propagar a excepção para o bloco acima do bloco corrente e para o contexto certo, contudo tal não é explícito na especificação.

14. Numa coreografia apenas um *finalizerBlock* pode será activado.

    Na CBPEL/BPEL tal não se coloca, pois apenas pode ser declarada uma rotina de compensação. Contudo a WSCDL permite várias rotinas de finalização, o que a CBPEL/BPEL não permite.

Ainda se assinalam as seguintes funcionalidades da linguagem WSCDL que a conjunção CBPEL / BPEL não suporta totalmente ou directamente:

- Composição de coreografias, a linguagem CBPEL não suporta a composição de processos CBPEL; e

- Canais de comunicação (*channel*), a CBPEL / BPEL não contempla a definição de uma estrutura com funcionalidade semelhante a um *channel* da WSCDL, mas permite a afectação dinâmica de pontos de interacção.

Constata-se que enquanto neste trabalho se explicita como é concretizado a tradução de processos CBPEL para BPEL e a sua consequente interacção para com o protocolo de coordenação, na especificação da linguagem WSCDL tal é deixado em aberto.

Talvez por esse motivo ainda não exista um trabalho que foque a tradução de processos WSCDL que utilizem coreografias coordenadas, para BPEL, BPMN, ou outra qualquer linguagem.

7.4 Conclusão

Uma vez apresentadas as considerações finais, resta salientar que em relação à linguagem BPEL, ela necessita de pequenos ajustes de modo a providenciar um suporte efectivo a coreografias descritas em CBPEL. Em relação à linguagem WSCDL, ela apresenta-se como uma linguagem de maior abrangência que a linguagem CBPEL, mas que essa mais valia, é no presente momento uma dificuldade, pois apresenta uma complexidade que faz com que a linguagem não tenha ainda uma verdadeira aceitação prática. É precisamente, de forma a explorar essa dificuldade, que a CBPEL propõe um modelo de coreografia mais simples, e por isso mais concretizável. Como mostra dessa concretização, são apresentados os cenários do capítulo anterior, os quais são inteiramente suportados por um simulador desenvolvido.

Capítulo 8

Conclusão

Este capítulo é composto por duas secções, uma que apresenta os aspectos relevantes deste trabalho e outra que apresenta os aspectos em aberto e trabalho futuro e que poderão ser aprofundados em futuras investigações

8.1 Sumário e aspectos relevantes

O primeiro capítulo, desta dissertação, descreveu a sua motivação, deu o enquadramento necessário, estabeleceu os objectivos e providenciou uma perspectiva deste documento. O objectivo da dissertação ficou então estabelecido de se conceber uma linguagem, baseada na linguagem BPEL, para descrever processos interorganizacionais, devendo possuir suporte a *scopes*, falhas e compensações.

No segundo capítulo foi apresentado a contextualização deste trabalho, nomeadamente a caracterização dos fluxos de trabalho intra-organizacionais e interorganizacionais, descrevendo-se os principais conceitos e aspectos da sua modelação, análise e execução.

O terceiro capítulo teve por objectivo a apresentação de como os fluxos de trabalho interorganizacionais (FTIOs), baseados na Web, podem ser suportados. Primeiro descreveu as várias linguagens existentes, e apresentou uma comparação das suas capacidades de descrição dos aspectos relevantes para os FTIOs. Por segundo, apresentou os modelos de suporte transaccional para suporte aos FTIOs e as suas implementações.

No quarto capítulo, entrou-se na parte de concretização, começando-se por descrever a parte elementar da linguagem criada, denominada de CBPEL. Foram apresentados as suas construções elementares e a sua transformação para a linguagem BPEL. Salienta-se o levantamento de situações que envolvem os vários tipos de decisões que é apresentado no anexo B. Esse levantamento apesar de não estar completo, esclarece que a transformação das decisões pode apresentar situações complexas. Salienta-se que não se conhece nenhum trabalho que seja tão extenso como este no tratamento destas situações. Os trabalhos existentes em [Mendling05a] [Xiangpeng06] [Zongyan06] apenas referem a tradução, denominada de projecção, de uma decisão como sendo outra decisão. O que implicaria que os todos participantes envolvidos teriam que ter todo o conhecimento para procederem, cada um, à tomada da decisão de igual modo. Neste trabalho considera-se que apenas um dos participantes tem informação suficiente para a tomada da decisão e que todos os outros são passivos, pois esse será o caso mais geral, porque não implica a propagação da informação que permite a tomada de decisão. A propagação da informação que permite a tomada da decisão torna os processos muito extensos e requer uma validação específica de modo a garantir que tal ocorre nos vários participantes. Os trabalhos [Mendling05] [Zongyan07] e [Hongli07] já utilizam a noção participante activo e passivo na decisão, contudo consideram que os participantes passivos terão de conter uma decisão tardia, mas não considera que o fecho dessa decisão poderá não existir dentro dos ramos da decisão. Em [Zongyan07] e [Hongli07] também se indica que qualquer participante na decisão pode ser o participante activo, sendo denominado de participante dominante, o que mostra que o pressuposto de que todos os participantes têm toda a informação para a tomada de decisão se mantém. Existe aqui alguma incoerência, pois os participantes passivos utilizam uma decisão tardia e têm a informação da tomada de decisão. Contudo a conjunção destes dois factores não faz sentido, pois por terem a informação deveriam fazer uma decisão interna e então utilizar uma recepção normal nos vários ramos. Os mesmos trabalhos avançam também para as decisões cíclicas e consideram que estas decisões têm sempre um participante activo, mas que qualquer um o pode ser desde que participe no ciclo. Novamente aqui há inconsistência, pois de modo geral o participante activo não pode ser qualquer um, porque isso implica adicionar actividades de comunicação de modo a forçar a passagem de uma informação que já seria conhecida do participante que de facto tomou a decisão. No trabalho presente em [Mendling05] a transformação de um ciclo corresponde somente a um ciclo. Em nenhum trabalho é tecido nenhum comentário à possibilidade de haver participantes novos dentro do ciclo e as consequências disso.

No quinto capítulo foi apresentado o tratamento de falhas da linguagem CBPEL. Começou-se por esclarecer como esse tratamento ocorre na linguagem base, ou seja, a linguagem BPEL. Depois apresentou-se o protocolo de coordenação adoptado e sua interligação como processos BPEL. Seguiu-se a descrição das actividades e construções na linguagem CBPEL que dão suporte ao tratamento de falhas, terminando com a transformação destes elementos para a linguagem BPEL. Deste capítulo salienta-se todo o estudo apresentado na parte da descrição do protocolo adoptado, pois é clarificado as nuances da interacção entre os processos BPEL e a figura do coordenador, e também as particularidades existentes na participação em blocos (*scopes*) encaixados. Sobre esta parte do trabalho, salienta-se a existência do trabalho presente em [Hongli07] o qual formaliza a tradução de coreografias, descritas de forma global e comum, e modeladas segundo uma linguagem denominada de *Chor*, para processos individuais descritos numa outra linguagem denominada de *Role*, a qual é inspirada nas funcionalidades da linguagem BPEL. A linguagem *Chor* permite modelar bastantes aspectos da linguagem WSCDL, sendo de realçar o suporte a coreografias, lançamento e tratamento de falhas e finalização de coreografias. Contudo, apesar de ser referido o suporte a várias rotinas de finalização, o foco é dado na existência de apenas uma e com a semântica de compensar. Assume o pressuposto que o lançamento de uma falha num processo *Chor* implica que todos os participantes recebam essa falha no mesmo bloco, que é denominado de coreografia. Não esclarecendo como tal poderá ser conseguido. O trabalho apresenta uma tradução, nele denominada de projecção, de processos de coreografias na linguagem *Chor* para processos individuais em linguagem *Role*, segundo uma correspondência quase directa em elementos de *Chor* para *Role*. Novamente, tal como visto nos elementos do capítulo anterior se assume que todos os participantes têm conhecimento de toda a informação do cenário, o que não parece realista, pois para assim ser a quantidade de mensagens a ser trocada teria de ser enorme. Como esse trabalho não aborda as coreografias coordenadas, ele não pode ser directamente comparado com o desenvolvimento existente nesta dissertação.

No sexto capítulo foram descritos dois cenários de aplicação da linguagem desenvolvida. Foi apresentado o cenário da escala de um navio num porto, que é descrito sem recurso a *scopes*. Depois foi apresentado o cenário de uma compra numa livraria electrónica, que primeiro é descrito sem *scopes*, e depois com *scopes*, falhas e compensações. Estes dois cenários deram uma perspectiva de como os processos podem ser descritos em CBPEL, e dos processos gerados em BPEL, resultantes da transformação dos primeiros. Também é de salientar a grande dimensão dos processos

aquando da utilização de *scopes*, falhas e compensações, o que evidencia que este tipo de funcionalidades requer a geração automática dos processos, caso contrário existirá uma forte possibilidade de haver inconsistências.

O sétimo capítulo visou apresentar as considerações finais. Começou com uma apreciação sobre as dificuldades de suporte dos processos CBPEL pela linguagem BPEL. Onde se constata que existe algumas limitações da linguagem BPEL em oferecer um suporte efectivo aos processos derivados da linguagem CBPEL, mas que tal resulta do facto de que a linguagem BPEL estar vocacionada para descrever orquestrações e não para suportar processos derivados de coreografias. Prosseguiu com uma apreciação das novidades introduzidas pela versão 2.0 da linguagem BPEL, face à versão 1.1 que é a versão utilizada neste trabalho. Por terceiro, apresentou as considerações sobre à linguagem WSCDL, uma vez que esta linguagem abrange as funcionalidades da linguagem CBPEL. No entanto salienta-se os desenvolvimentos executados nesta dissertação como pertinentes e clarificadores do suporte transaccional estritamente relativo a compensações, que é o modelo da linguagem BPEL, e que proporcionam uma base de referência para o suporte à linguagem WSCDL.

O oitavo capítulo, que é o presente capítulo, apresenta o presente sumário contendo o realce dos pontos mais relevantes da dissertação e termina com os aspectos em aberto e trabalho futuro.

8.2 Aspectos em aberto e trabalho futuro

Ao longo desta dissertação vários foram os aspectos que ficaram em aberto, ou que pelo menos foram limitados de forma a controlar a extensão deste trabalho. Assim realça-se os seguintes aspectos em aberto que podem constituir uma inspiração para futuras reflexões:

- Completar as regras que levem à boa construção dos processos em CBPEL, como por exemplo: depois de uma decisão, o participante que a executa deve ser o emissor da primeira interacção, em todos os ramos da decisão;

- Validar formalmente que de um processo correctamente descrito em CBPEL se gera processos locais também correctos, ou em que condições é que tal acontece;

- Adicionar à CBPEL a capacidade de composição de processos, ou seja descrever processos que utilizem outros processos CBPEL como blocos constituintes;

- Adicionar à CBPEL uma construção similar ao *channel* da WSCDL, de modo a evitar a indicação explícita dos elementos de correlação em todas as interacções e incluir algum controlo sobre a utilização da própria construção, por exemplo: número de vezes que pode ser utilizado ou que tipo de informação nele pode ser utilizada;

- Adicionar à CBPEL a capacidade para descrever fluxos não estruturados, ou seja, possuir algo equivalente aos *links* da linguagem BPEL. Determinar as implicações da sua transformação para os processos BPEL;

- Determinar de forma completa as regras de transformação das actividades de *while* e *switch* parcialmente apresentadas no anexo B, assim como para as restantes actividades da linguagem BPEL 2.0 não abordadas neste trabalho;

- Resolver os problemas de sincronismo relativos à entrada e saída nos *scopes*;

- Alterar a plataforma BPEL de modo que as falhas geradas pela própria plataforma sejam processadas da mesma forma que as falhas presentes nos processos CBPEL;

- Conceber uma forma de parametrizar, na linguagem BPEL, que permita conceber processos mais simples, para os participantes, aquando da utilização de *scopes*, falhas e compensações. A utilização de *scopes* locais em *eventHandlers*, *faultHandlers* e *compensationHandlers* deveria ser simplificada;

- Completar as consequências da participação intercalada em *scopes*;

- Contemplar a existência de temporizações associadas ao envio e respectiva resposta; e

- Analisar e implementar a tradução de processos em WSCDL para CBPEL, e vice-versa.

Em termos de conclusão final, a linguagem CBPEL, o protocolo de coordenação, a tradução de CBPEL para BPEL e a interligação dos processos gerados com a figura do coordenador, permitem afirmar que os objectivos iniciais de: conceber uma linguagem para descrição de processos interorganizacionais de forma global e comum, que estendesse o modelo utilizado na linguagem BPEL e também conceber uma referência para uma melhor percepção das capacidades e das incapacidades da especificação WS-

CDL, foram conseguidos. Também se tem consciência que, apesar de se ter clarificado alguns aspectos na utilização de coreografias coordenadas descritas de forma global e comum, novas questões se levantam agora e que o horizonte de investigação foi alargado com novas e interessantes expectativas.

Referências

[Aalst97] W. van der Aalst. **Verification of Workflow Nets**. In P. Azéma and G. Balbo, editors, Application and Theory of Petri Nets 1997, volume 1248 of Lecture Notes in Computer, Science, pages 407–426, Springer-Verlag, Berlin, 1997.

[Aalst98] W. van der Aalst. **Finding Errors in the Design of a Workflow Process: A Petri-net-based, Approach**. In W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors, Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98), volume 98/7 of Computing Science Reports, pages 60–81, Lisbon, Portugal, 1998, Eindhoven, University of Technology, Eindhoven.

[Aalst98b] W. van der Aalst. **The Application of Petri Nets to Workflow Management**. The Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

[Aalst99] W. van der Aalst, **Process-Oriented Architectures for Electronic Commerce and Interorganizational Workflow**, Information Systems, 24(8), pp. 639-671, 1999.

[Aalst00] W. van der Aalst, **Inheritance of Interorganizational Workflows: How to Agree to Disagree Without Loosing Control?** BETA Working Paper Series, WP 46, Eindhoven University of Technology, Eindhoven, 2000.

[Aalst01] W. van der Aalst, M. Weske, **The P2P Approach to Interorganizational Workflows**, In K.R. Dittrich, A. Geppert, M. C. Norrie (eds), Proc. Of the 13th Int. Conf. on Advanced Information Systems (CAiSE'01), Berlin, 2001

[Aalst01b] W van der Aalst, T. Basten, **Inheritance of Workflows: An approach to tackling problems related to change**, Theoretical Computer Science, 2001

[Aalst02] W. van der Aalst, M. Dumas, A. ter Hofstede, and P. Wohed, **Pattern-Based Analysis of BPML (and WSCI),** QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002

[Aalst02b] W. van der Aalst, Kees van Hee, **Workflow Management Models, Methods**, and Systems, The MIT Press, London, England, 2002

[Adam98] N. Adam, V. Atluri, and W. Huang. **Modeling and Analysis of Workflows using Petri Nets**. Journal of Intelligent Information Systems, 10(2):131–158, 1998.

[Andrews03] T. Andrews, et al., **Business Process Execution Language for Web Services (BPEL4WS)**, Version 1.0, May 2003, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf.

[Arkin01] A. Arkin, **Business Process Modeling Language (BPML),** Working Draft 0.4, BPMI, March 2001, http://www.bpmi.org/

[Arkin02] A. Arkin, et al., **Web Services Choreography Interface (WSCI),** version 1.0, W3C Note, August, 2002, http://www.w3.org/TR/wsci/

[Banerji02] A. Banerji, et al., **Web Services Conversation Language (WSCL),** version 1.0, W3C Note, March 2002, http://www.w3.org/TR/2002/NOTE-wscl10-20020314/

[Basten98] T. Basten, **In Terms of Nets: System Design with Petri Nets and Process Algebra**, PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1998.

[Benjamin95] R. Benjamin, R. Wigan, **Electronic markets and virtual value chains on the information superhighway**, Sloan Management Review, pages 62–72, 1995.

[Bernauer02] M. Bernauer, et al., **Specification of Interorganizational workflows – A comparison of Approaches**, Vienna, Austria: Institute of Software Technology and Interactive Systems, Business Informatics Group, Vienna University of Technology, 2002. Technical Report, August 2002.

[BernersLee02] T. Berners-Lee, Information Management: A Proposal, CERN, March 1989

[Box00] D. Box, et al., **Simple Object Access Protocol (SOAP)**, version 1.1, May, 2000, http://www.w3.org/TR/SOAP

[Bray98] T. Bray, et al., **Extensible Markup Language (XML)** (version 1.0) (first Edition), W3C Recommendation, February 1998, http://www.w3.org/TR/1998/REC-xml-19980210

[Bunting03] D. Bunting, et al., **Web Services Composite Application Framework (WS-CAF)**, version 1.0, July 2003

[Cabrera04] L. Cabrera, et al., **WS-Coordination / WS-BusinessActivity / WS-AtomicTransaction (WS Coordination and Transaction, WS-CT)**, IBM Microsft and Bea Editors, November 2004

[Casati01] F. Casati, A. Discenza, **Modeling and Managing Interactions among Business Processes**, Journal of Systems Integration, 10(2), pp. 145-168, 2001

[Ceponlus02] A. Ceponkus, et al., **Business Transaction Protocol (BTP),** version 1.0, OASIS Group, June 2002

[Cerf74] V. Cerf, Y. Dalal, C. Sunshine, **Specification of Internet Transmission Control Program**, RFC 675, Network Working Group, December 1974

[Clark99] J. Clark, S. DeRose, **XML Path Language (XPath),** version 1.0, 1999, http://www.w3.org/TR/1999/REC-xpath-19991116

[Clark01] J. Clark, et al., **Business Process Specification Schema (BPSS),** version 1.01, UN/CEFACT e OASIS, May, 2001, http://www.ebxml.org/specs/ebBPSS.pdf

[Chen01] Q. Chen, M. Hsu, **Inter-Enterprise Collaborative Business Process Management,** Proc. of 17th Int. Conference on Data Engineering (ICDE), pages 253–260, IEEE Computer Society, Heidelberg, Germany, April 2001

[Chiu01] D. Chiu, K. Karlapalem, Q. Li, **Views for Inter-Organization Workflow in an Ecommerce Environment**, Proc. of 9th IFIP Working Conference on Database Semantics (DS-9), Hong Kong, China, April 24-28, 2001.

[Chistensen01] E. Christensen, et al., **Web Services Description Language (WSDL)**, version 1.1, W3C Note, March, 2001, http://www.w3.org/TR/wsdl.html

[Dinkhoff95] G. Dinkhoff, V. Gruhn, **Entwicklung Workflow-Management-geeigneter Software-Systeme**, in G. Vossen, J. Becker (eds.), Geschäftsprozeßmodellierung und Workflow-Management: Modelle, Methoden, Werkzeuge, Thomson Publishing, pp. 405-421, 1995, (in german)

[Ellis80] C. Ellis, G. Nutt, **Computer science and office information systems**. ACM Computing Surveys, 12(1):27–60, March 1980.

[Elmagarmid92] A. Elmagarmid (ed.), **Database Transaction Models for Advanced Applications**, Morgan Kaufmann, 1992

[Fowler97] M. Fowler, **Analysis Patterns: Reusable Object Models**. Reading: Addison-Wesley, 1997

[Georgakopo99] D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker, **Managing Process and Service Fusion in Virtual Enterprises**, Information Systems, 24(6), pp. 429-456, 1999.

[Hammer94] M. Hammer, J. Champy, **Re-Engineering the Corporation,** A Manifesto for Business, Revolution. Addison-Wesley 1994

[Hauschildt97] D. Hauschildt, H. Verbeek, W. van der Aalst. **WOFLAN: a Petri-net-based Workflow Analyzer**, Computing Science Reports 97/12, Eindhoven University of Technology, Eindhoven, 1997.

[Hollingswo95] D. Hollingsworth, **Workflow Management Coalition: The Workflow Reference Model**, UK, Jan, 1995

[Hongli07] Y. Hongli, Z. Xiangpeng, C. Chao, and Q. Zongyan, **Exploring the Connection of Choreography and Orchestration with Exception Handling and Finalization/Compensation**, Proc. of FORTE 07, pp. 81-96, Tallinn, Estonia, 2007

[IBM96] IBM, **IBM FlowMark: Modeling Workflow**, Version 2 Release 2, Publ. No SH-19-8241-01, 1996

[Jablonski95] S. Jablonski, **Workflow-Management-Systeme: Modellierung und Architektur**, Thomsons Aktuelle Tutorien (TAT), 9, Thomson Publishing, 1995

[Jablonski97] S. Jablonski, M. Bohm, W. Schulze. (Editors) **Workflow Management: Development of Applications and Systems**. (in German) Heidelberg: dpunkt 1997

[Jordan07] D. Jordan, et al., **Business Process Execution Language for Web Services (BPEL4WS)**, Version 2.0, April 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[Kappel00] G. Kappel, S. Rausch-Schott, W. Retschitzegger, **A Framework for Workflow Management Systems Based on Objects, Rules and Roles,** ACM Computing Surveys Electronic Symposium on Object-Oriented Application Frameworks, M.E. Fayad (guest editor), March 2000.

[Kavantzas05] N. Kavantzas, et al., **Web Services Choreography Description Language (WSCDL),** version 1.0, W3C, November, 2005, http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/

[Kobielus97] J. G. Kobielus, **Workflow Strategies**, IDG Books, Foster City, CA, USA, 1997

[Koulopoulo95] T. Koulopoulos, **The Workflow Imperative: Building Real World Business Solutions**, Van Nostrand Reinhold Publishers, New York, USA, 1995

[Kreger01] H. Kreger, **Web Services Conceptual Architecture (WSCA 1.0)**, IBM Software Group, 2001

[Levine01] P. Levine, et al., **Business Process Specification Schema (BPSS v1.01)**, UN/CEFACT and OASIS, May, 2001, www.ebxml.org/specs/ebBPSS.pdf

[Leymann00] F. Leymann, D. Roller, **Production workflow: concepts and techniques**, Prentice-Hall, 2000.

[Leymann01] F. Leymann, **Web Services Flow Language (WSFL),** version 1.0, IBM, May, 2001, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[Lippe05] S. Lippe, U. Greiner, A. Barros, **A Survey on State of the Art to Facilitate Modelling of Cross-Organisational Business Processes**, Proceedings of the 2nd GI Workshop XML4BPM -- XML Interchange Formats for Business Process Management at 11th GI Conference BTW 2005, M. Nuttgens and J. Mendling eds., pages 7-22, Karlsruhe Germany, March 2005

[Lindert99] F. Lindert, W. Deiters, **Modelling inter-organizational processes with process model fragments**. GI Workshop Informatik 99, Paderborn, Germany, October 1999.

[Lima01] C. Lima, **Um Modelo Multinível de Coordenação em Ambiente de Empresa Virtual**, tese de Doutoramento, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Lisboa, 2001

[Manolescu01] D. Manolescu, **Micro-Workflow: A Workflow Architecture Supporting Compositional Object-Oriented Software Development**, PhD Thesis, University of Illinois at Urbana-Champaign, 2001

[McCready92] S. McCready, **There is no more than one kind of Work-flow Software**, in Computerworld, Nov. 2, 1992

[Mendling03] J. Mendling, M. Müller, **A Comparison of BPML and BPEL4WS**, in Proc. of the 1st Conference Berliner XML-Tage, R. Tolksdorf, R. Eckstein, eds., pages 305-316, Berlin Germany, October 2003

[Mendling04] J. Mendling, G. Neumann, M. Nüttgens, **A Comparison of XML Interchange Formats for Business Process Modelling**, in Proc. of EMISA 2004 "Informations systeme im E-Business und E-Government", F. Feltz, A. Oberweis, B. Otjacques, eds., Vol. 56 of Lecture Notes in Informatics (LNI), pages 129-140, Luxembourg, October 2004

[Mendling05] J. Mendling, M. Hafner, **From WS-CDL Choreography to BPEL Process Orchestration**. Journal of Enterprise Information Management (JEIM), vol 21, nº 5, pp 525-542, Emerald Group Publishing Limited, 2005

[Mendling05a] J. Mendling, M. Hafner, **From inter-organizational workflows to process, execution: Generating BPEL from WS-CDL**, Proc. of OTM 05, Agia Napa, Cyprus, 2005

[Meyer88] B. Meyer, **Object-Oriented Software Construction**, Prentice Hall, Englewood Cliffs, NJ, 1988

[Muehlen00] M. Muehlen, F. Klien, **AFRICA: Workflow Interoperability Based on XMLMessages**, Proceedings of CAiSE'00 workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing (IDSO'00), Stockholm, Sweden, June 5, 2000

[Muth98] P. Muth, et al., **From Centralized Workflow Specification to Distributed Workflow Execution**, Journal of Intelligent Information Systems, nº 10, 159–184, Kluwer Academic Publishers, 1998

[OMG08] Object Management Group, **Business Process Modeling Notation V1.1**, Object Management Group, January, 2008 http://www.omg.org/spec/BPMN/1.1/PDF

[Pargfriede02] K. Pargfrieder, **Interorganizational Workflow: Management, Concepts, Requirements and Approaches,** Master thesis, Johannes Kepler University, Linz, March 2002

[Peterson81] J. Peterson, **Petri net theory and the modeling of systems**, Prentice Hall, 1981

[Petri62] C. Petri, **Fundamentals of a theory of asynchronous information flow**, Proceedings of the IFIP Congress, pp. 386-390, Amsterdam, Netherlands, 1962.

[Rademacher01] T. Rademacher, et al., **DOM4J**, http://www.dom4j.org/, 2001

[RauschScho97] S. Rausch-Schott, **TriGSflow – Workflow Management Based on Active object-Oriented Database Systems and Extended Transaction Mechanisms**, PhD Thesis, University of Linz, 1997

[Shapiro02] R. Shapiro. **A comparison of XPDL, BPML and BPEL4WS**, http://xml.coverpages.org, 2002

[Shapiro08] R. Shapiro, et al., **Process Definition Interface - XML Process Definition Language (XPDL)**, WfMC-TC-1025, Version 2.1a, WfMC, USA, Oct. 10, 2008

[Sharp01] A. Sharp, P. McDermott, **Workflow Modeling: Tools for Process Improvement and Application Development**, Artech house publishers, London, England, 2001

[Shen01] M. Shen, D. Liu, **Coordinating Interorganizational Workflows Based on Process-Views**, DEXA, 2001

[Sheth99] A. Sheth, W. van der Aalst, I. Arpinar, **Processes driving the networked economy**. IEEE Concurrency, pages 18–31, July–September 1999.

[Shet93] A. Shet, M. Rusinkiewicz, **On Transactional Workflows**. Data Engineering Bulletin Vol. 16/2, 1993.

[Schulz00] K. Schulz, Z. Milosevic, **Architecting Cross-Organizational B2B Interactions**, Proceedings of the 4th International Enterprise

Distributed Object Computing Conference (EDOC 2000), pp. 92-101, Los Alamitos, CA, USA, 2000.

[Silver94] B. Silver, **Automating the Business Environment**, in T.E. White, L. Fisher (eds.), New Tools for New Times: The Workflow Paradigm - The Impact of Information Technology on Business Process Reengineering, Future Strategies Inc., pp. 129-154, 1994

[Teófilo05] A. Teófilo, A. Silva, **CBPEL - Linguagem para Definição de Processos de Negócio Interorganizacionais**, em Actas da 3ª Conferência Nacional sobre XML: Aplicações e Tecnologias Associadas (XATA2005), Braga, Portugal, Fevereiro de 2005

[Thatte01] S. Thatte: **XLANG - Web Services for Business Process Design**, Draft Specification. Microsoft, May 2001, http://www.gotdotnet.com/team/xml_wsspecs/xlangc/default.htm

[McKee01] B. Mckee, **Universal Description Discovery and Integration** (UDDI) API Specification (version 2.0), UDDI Org, June 2001, http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf

[Velosa00] H. Velosa, J. Reis, A. Silva, **Sistema de Workflow Baseado em Agentes para a Web: Workflow num porto Marítimo**. TFC-LEIC**,** IST-UTL, Portugal**,** 2000

[Wegner96] P. Wegner**, Interoperability**. In: ACM Computing Surveys, Vol. 28, No. 1, March 1996.

[Weigand98] H. Weigand, A. Ngu, **Flexible specification of interoperable transactions**. Data & Knowledge Engineering Vol. 25, 1998.

[Weske99] M. Weske, **Workflow Management Systems: Formal Foundation. Conceptual Design, Implementation Aspects**, PhD Thesis, Universitat Munster, Munster, 1999

[WfMC99] WfMC, **Workflow Management Coalition: Terminology & Glossary**, UK, Feb, 1999

[Willaert01] F. Willaert, **XML-based Frameworks and standards for B2B ECommerce**, Katholieke Universiteit Leuven, May, 2001

[Wohed02] P. Wohed, W. van der Aalst, M. Dumas, and A. ter Hofstede, **Pattern-Based Analysis of BPEL4WS**, QUT Technical report, FIT-TR-2002-04, Queensland, University of Technology, Brisbane, 2002.

[Xiangpeng06] Z. Xiangpeng, et al., **Verification of WS-CDL Choreography**, Proc. of AWCVS 06, Macao SAR, China, 2006

[Zisman77] M. Zisman, **Representation, Specification, and Automation of Office Procedures**, Ph.D. dissertation, Wharton School, University of Pennsylvania., USA, 1977

[Zongyan06] Q. Zongyan, et al., **Exploring into the Essence of Choreography**, Technical Report, Peking Univ., China, 2006

[Zongyan07] Q. Zongyan, et al., **Towards the Theoretical Foundation of Choreography**, Proc. of WWW'07, Banff, Alberta, Canada, May 8-12, 2007

Anexo A: Notação UML adaptada

A linguagem UML, nomeadamente o diagrama de classes, que é o diagrama utilizado para modelar artefactos XML, não se apresenta muito prático para descrever cenários com muitos e pequenos artefactos, dado que para cada artefacto haverá um elemento visual que o representa.

Como as linguagens descritas no capítulo 3 contêm bastantes elementos, tornou-se necessário providenciar um modo mais sintético de as descrever.

Nesse sentido adaptou-se o diagrama de classes da linguagem UML com as seguintes alterações:

- A definição de um elemento utiliza o identificador em formatação negrito. Os seus atributos são colocados imediatamente ao identificador e com formatação normal, mas precedido de dois espaços. Cada atributo pode conter o sinal '?' indicando que é opcional. Cada atributo ocupa uma linha, não existindo a parte relativa para a modelação das operações.

- Um elemento que seja uma definição local a um outro elemento, ou que seja somente utilizado dentro de um elemento, pode ser representado após os atributos do elemento no qual é referido, com o identificador a negrito, com indicação da cardinalidade envolvida se diferente de 1, prefixado com mais dois espaços e um sinal de '+', e é seguido dos seus atributos e dos seus eventuais elementos locais. Os elementos locais podem conter definições de elementos locais a eles, em que os elementos de segunda instância seguem as mesmas

regras e são prefixados com mais dois espaços. Não se permite mais do que dois níveis de definições locais, por questões de legibilidade.

- As definições de tipos, que são especializados em elementos, ou noutros tipos, têm o identificador em letra normal.

- As relações de composição e de agregação são representadas por uma ligação com um losango a cheio e são sempre respeitantes aos elementos de topo.

- As associações implementadas por valores de atributos podem ser representadas por uma ligação de composição, com o losango preenchido a branco, do referenciador para o referenciado, indicando o nome do atributo em questão.

- Utiliza-se um "E" para indicar exclusividade, que tanto pode aparecer nos atributos como nas associações. No entanto, uma exclusividade nos atributos será válida entre eles e o mesmo acontecendo nas associações. No caso de duas, ou mais, exclusividades diferentes, no mesmo género (atributos, associações), coloca-se "E1", "E2", … nos "elementos" que fazem parte dessas exclusividades. A indicação de exclusividade não esclarece se é permitido zero ocorrências, ou se deve ocorrer pelo menos uma ocorrência. Para tal, seria necessário mais uma distinção, mas optou-se por não a colocar.

Na figura A1 encontra-se um exemplo com um elemento *partners* contendo um ou mais elementos locais *partner*, que por sua vez contém o atributo *name* e zero ou mais elementos *partnerLink* locais a ele e que têm também o atributo *name*. No lado esquerdo da figura encontra-se a notação adaptada que é utilizada, enquanto que no lado direito se encontra os vários elementos descritos de forma menos compactada. Como se pode observar a representação simplificada permite uma boa economia de espaço face à representação individualizada de cada artefacto.

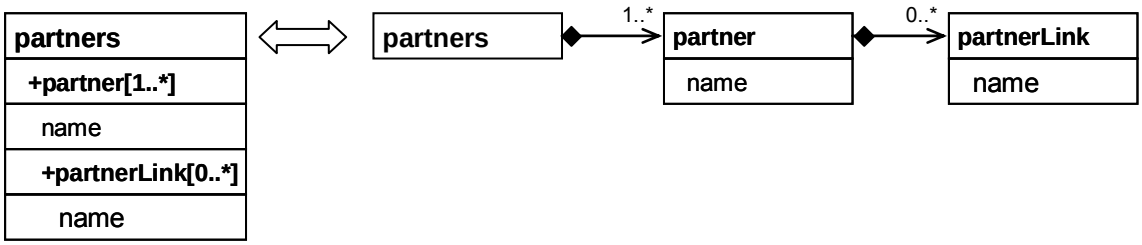


Figura A1 – Exemplo da notação UML utilizada

Anexo B: Transformação das actividades de Switch e While

Neste anexo é apresentado com mais detalhe a transformação das actividades de *switch* e de *while*, de processos em linguagem CBPEL, para as actividades correspondentes, da linguagem BPEL, na perspectiva do processo de um participante. Primeiramente será abordado a transformação da actividade de decisão múltipla *switch* e depois a transformação da actividade de execução em ciclo *while*.

Transformação da actividade de *switch*

Conforme já referido, a transformação desta actividade apresenta duas situações distintas: a situação em que o participante executa de facto a decisão múltipla designando-se neste caso de execução activa da decisão; e a situação em que o participante aguarda a recepção de uma mensagem de um qualquer ramo da decisão, designando-se neste caso de execução passiva.

No caso de execução activa, a decisão múltipla, que é uma actividade *cbpel:switch* será transposta para o processo do participante como uma actividade também de decisão múltipla *bpel:switch*, com todos os ramos que a decisão múltipla *cbpel* continha.

Seguidamente são apresentadas as várias situações identificadas e as respectivas transformações.

Decisão fechada

Na figura B1, do lado esquerdo, encontra-se uma decisão em que o executor é o participante P1. A transformação dessa actividade no participante P1 resulta numa actividade de *bpel:switch* com os vários ramos que tinha no processo CBPEL, e que pode ser observado na parte central da mesma figura. A actividade com reticências representa a eventual existência de outras actividades, e é colocada em alguns pontos somente para

se ter uma melhor percepção da possível existência de outras actividades mas que não seriam relevantes para o cenário em estudo.

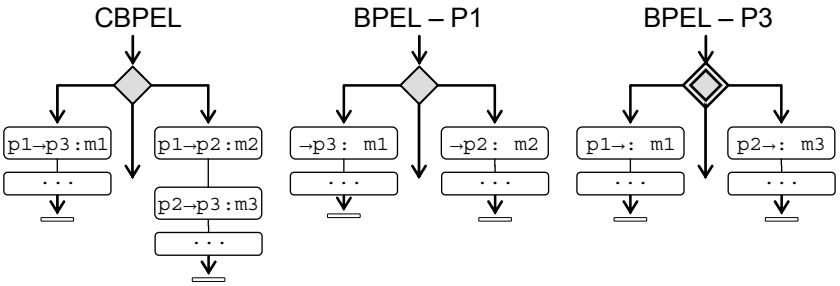


Figura B1 – Transformação de uma decisão fechada

No caso de participação passiva, a decisão múltipla será transposta para o processo do participante como uma actividade de decisão tardia *bpel:pick*, pois o participante em questão só saberá qual dos ramos foi escolhido pelo participante activo, ou por outro participante, quando receber uma primeira mensagem num dos ramos.

Na figura B1, no lado direito, encontra-se o resultado da transformação da decisão, do lado esquerdo, na perspectiva do participante P3. Este participante sendo passivo, executará uma decisão tardia, a qual está assinalada com um duplo losango, em que o ramo que receber a primeira mensagem será o executado.

Decisão aberta

Caso o participante não participe em um ou em vários ramos, será colocado um ramo vazio na respectiva decisão tardia. Caso a decisão tardia tenha um ramo vazio, esse ramo terá de receber o conjunto de actividades que contenha todas as primeiras recepções possíveis depois da decisão. Esse conjunto de actividades deverá ser copiado para o final de todos os outros ramos da decisão tardia a fim de esse conjunto de actividades não ser executado duas vezes. Esse conjunto de actividades denomina-se de fecho da decisão.

Uma decisão diz-se fechada se todos os seus ramos são fechados, ou seja, que não necessitam de um fecho. Uma decisão que contenha um ramo não fechado, ou seja, aberto, diz-se aberta e necessita de um fecho que será colocado no final de todos os seus ramos de modo a ficar fechada.

Numa decisão aberta, o procedimento de fecho depois de aplicado à própria decisão, implica uma verificação em cada um dos seus ramos, de modo a propagar o fecho por decisões que eventualmente existam dentro dos ramos.

Na figura B2 encontra-se a transformação de uma decisão aberta segundo a perspectiva do participante P3. Nessa figura e doravante, tendo em conta que se irá trabalhar com a perspectiva de um só participante, apenas será descrito no processo CBPEL as actividades referentes a esse participante, com o seu identificador e com o identificador da mensagem e na parte do processo do participante apenas será descrito nas actividades o identificador da mensagem. Como se pode observar a transformação da uma decisão aberta resultou numa decisão tardia com a cópia do fecho da decisão para os vários ramos.

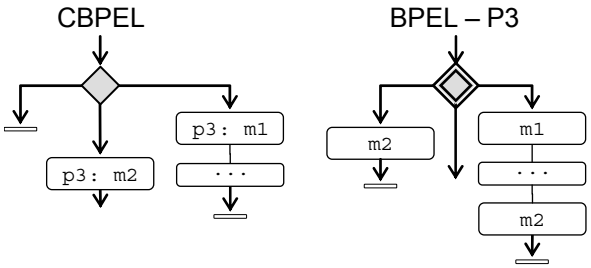


Figura B2 – Transformação de uma decisão aberta

Fluxo com várias decisões abertas

Na figura B3, no lado esquerdo, encontra-se um troço de um processo CBPEL com três decisões abertas, ou seja, todas. As decisões que sucedem às operações envolvendo as mensagens m2 e m4 têm um ramo vazio pelo que estão abertas. De facto a decisão que sucede a m2 tem os dois ramos abertos, pois o ramo da operação relativa a m4 necessita de fecho.

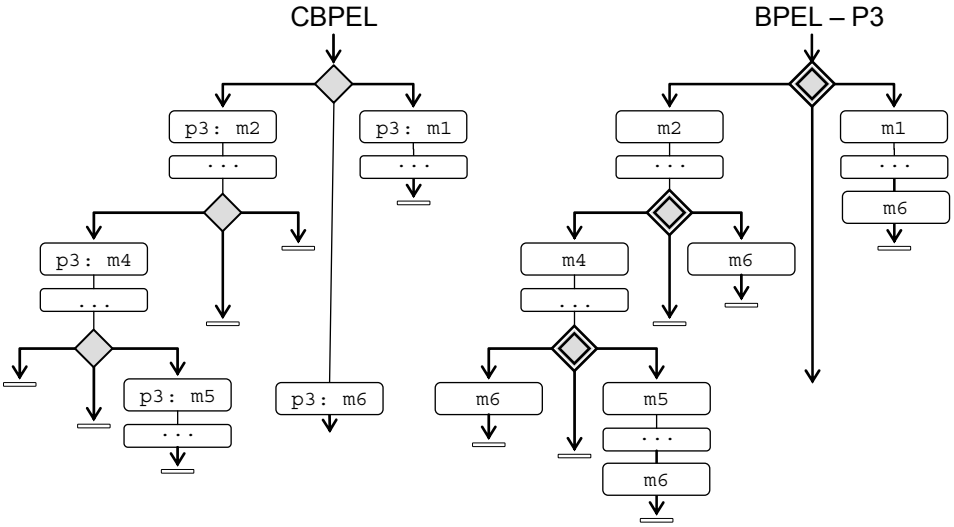


Figura B3 – Transformação de várias decisões abertas

A primeira decisão do processo, encontra-se aberta, pois o seu ramo do lado esquerdo necessita de fecho. A transformação, do referido troço, para o participante p3, está presente no lado direito da mesma figura, e terá o fecho da primeira decisão incorporado no final dos seus dois ramos. No ramo do lado esquerdo da decisão o fecho foi novamente incorporado na decisão existente. Por sua vez, o ramo do lado esquerdo também propagou o fecho para os dois ramos da decisão existentes nesse ramo.

Decisão com ramo a iniciar com outra decisão

Existe uma situação particular quando um ramo de uma decisão se inicia com outra decisão. Nesses casos deve-se incorporar a segunda decisão na primeira, ou seja, colocar os ramos da segunda decisão como ramos da primeira decisão e copiar as actividades que se seguem à segunda decisão no final desses ramos incorporados. Na figura B4 encontra-se uma situação destas, com várias decisões como primeiras actividades de ramos de outras decisões. O resultado final da transformação apresenta somente uma decisão tardia e já fechada. No entanto, esse resultado é obtido através da sistemática aplicação do fecho a todas as decisões, e depois pela incorporação sucessiva das decisões.

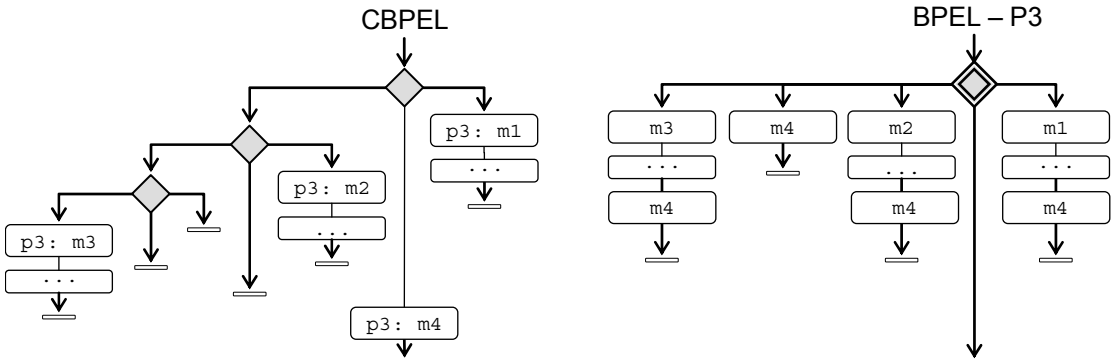


Figura B4 – Transformação de decisões seguidas de outras decisões

Decisão com fecho mais complexo

Até agora o fecho de uma decisão tem consistido numa única actividade, contudo esse não será o caso geral. De facto, o fecho de uma decisão foi definido como o conjunto de actividades que contém todas as primeiras mensagens possíveis. Dado que por agora apenas utilizamos decisões, temos na figura B5, do lado esquerdo, um exemplo de um fecho mais complexo, que envolve duas decisões.

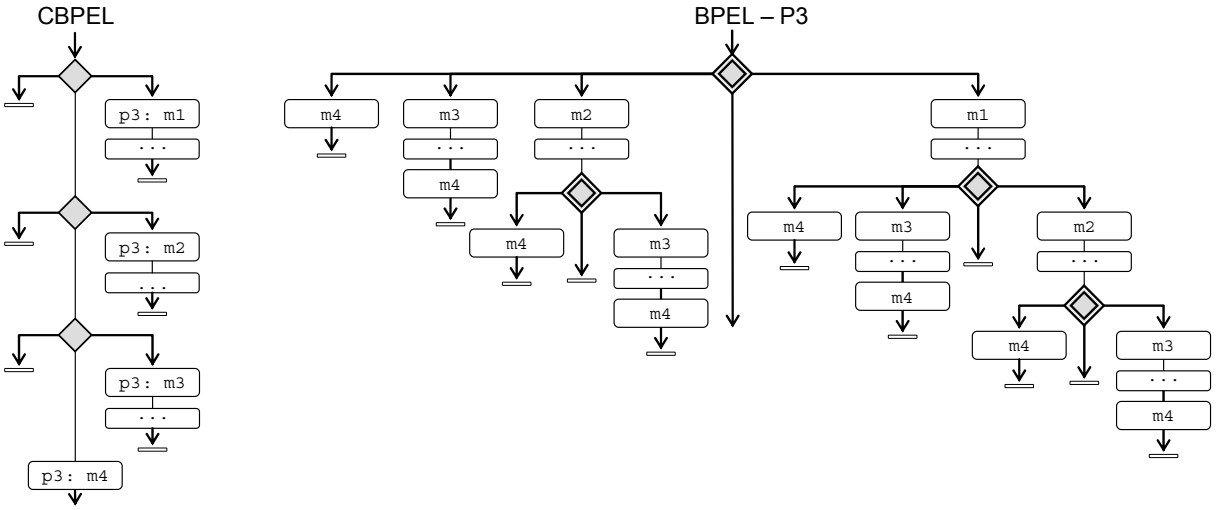


Figura B5 – Transformação de uma decisão com um fecho mais complexo

No lado esquerdo da figura B5, o fecho da primeira decisão apanha as seguintes duas decisões, só terminando na actividade da mensagem m4. O fecho da primeira decisão apanha a segunda decisão, que por sua vez também é uma decisão aberta, pelo que requer um fecho. O fecho desta apanha a terceira decisão, que também é aberta e cujo fecho consiste apenas na actividade da mensagem m4. Fechando-se a terceira decisão, essa decisão será incorporada na segunda decisão como o seu fecho. A segunda decisão já fechada será o fecho da primeira decisão resultando no troço que se encontra do lado direito da figura B5.

Do exemplo anterior pode-se verificar que o fecho pode ser mais complexo ou mais simples, mas consistirá na primeira actividade fechada, ou seja, que não necessite de mais actividades para conter todas as primeiras recepções possíveis.

Transformação da actividade de *while*

A actividade de *cbpel:while* também apresenta duas situações distintas, uma para o participante que é o seu executor e outra para os outros que participam dentro do ciclo.

Para o participante que é o seu executor esta actividade é transposta para o seu processo como uma actividade análoga, ou seja, a actividade de *bpel:switch*, sendo também transformadas as actividades dentro dos ramos da decisão.

Para os outros participantes, que participam nas actividades dentro do ciclo, eles assumem um comportamento passivo relativamente à decisão de executar o corpo do

ciclo. Pelo que, para eles, a transformação da actividade de *cbpel:while* resultará num *bpel:while* seguido de uma decisão tardia, *bpel:pick*, a qual terá as primeiras mensagens que o participante poderá receber dentro do ciclo e também as primeiras mensagens que o participante poderá receber fora do ciclo, ou seja, o fecho do ciclo. No ramo do fecho do ciclo, deverá existir uma actividade para sinalizar uma variável de modo a que o participante saia do ciclo. Essa variável deverá ser inicializada, de modo a condicionar o participante a executar o corpo do ciclo, antes da execução da actividade do ciclo *bpel:switch*.

Seguidamente são apresentadas as várias situações identificadas e as respectivas transformações.

Ciclo com final fechado

Na figura B6 encontra-se, do lado esquerdo um troço de actividades em CBPEL contendo um ciclo, e no lado direito a respectiva transformação para o participante P3.

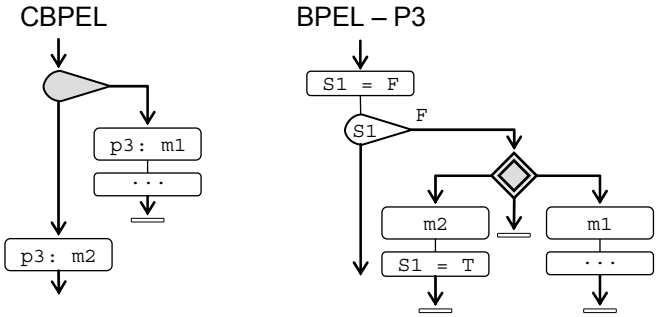


Figura B6 – Transformação de um ciclo simples

Tal como já referido o participante P3, que é passivo neste ciclo, tem de implementar um ciclo e dentro dele esperar tanto a primeira mensagem dentro do ciclo, que é m1, como a primeira mensagem fora do ciclo, que é m2. Caso receba a mensagem m1 deverá continuar com as actividades dentro do ciclo e depois voltar a testar pela recepção das duas mensagens m1 e m2. Caso receba a mensagem m2, deverá não executar as actividades dentro do ciclo, mas sim sair do mesmo. Tal é conseguido pela utilização da variável S1.

Ciclo com final aberto

Uma possível situação de acontecer com um ciclo consiste em ele ter o seu final aberto, ou seja, o seu ramo de actividades terminar necessitando de mais mensagens para clarificar qual o percurso a seguir. Neste caso, haverá três tipos de possibilidades:

receber uma mensagem da continuação do ramo dentro do ciclo; receber uma mensagem do início do ciclo; e receber uma mensagem de fora do ciclo. Caso tenha recebido uma mensagem para continuar dentro do ramo do ciclo, então depois de terminado o ramo deverá voltar ao início do ciclo. Caso tenha recebido uma mensagem de início de ciclo, deverá começar a executar o ciclo, mas depois dessa primeira mensagem já recebida. Caso tenha recebido uma mensagem de fora do ciclo, deverá sair do ciclo

Na figura B7 encontra-se a transformação de um ciclo com final aberto. Como se pode observar no início do ciclo tem-se a possibilidade de receber uma mensagem de início do ciclo, ou uma mensagem de fora do ciclo. Mas toda a parte intermédia do ramo do ciclo até à sua parte final aberta tem de ficar dentro de um outro ciclo, que termine com uma recepção tardia entre as tais três possibilidades já descritas, e que são: continuar a executar o ramo do ciclo e voltar ao início do ciclo principal; receber uma primeira mensagem do ciclo e voltar a executar o ciclo mas a partir dessa recepção; e sair do ciclo principal.

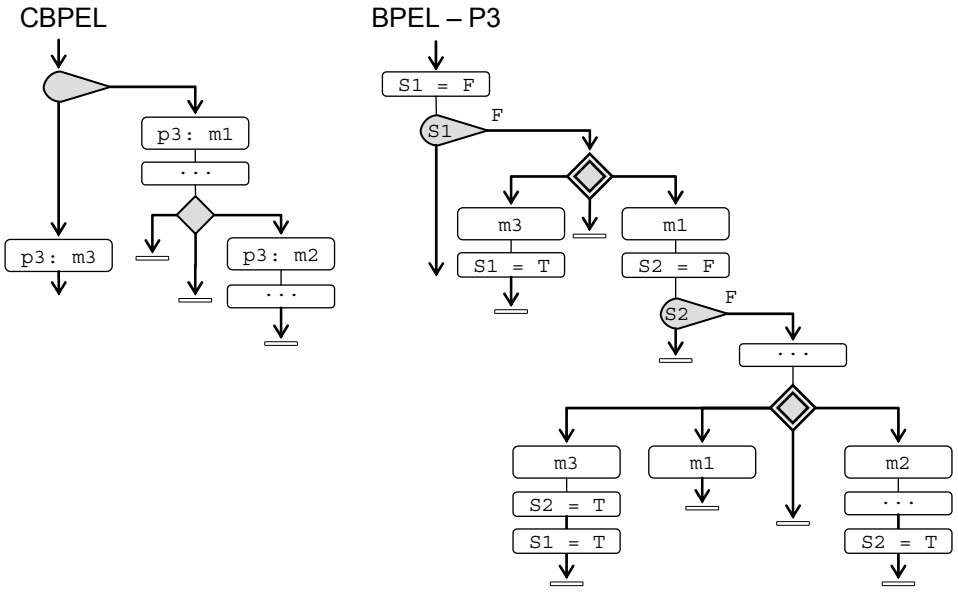


Figura B7 – Transformação de um ciclo com final aberto

Ciclo com final aberto e início com decisão aberta

Nas situações em que o final é aberto, o final mistura-se com o início do ciclo. No caso de o início também conter uma decisão aberta tem-se uma situação de vários caminhos possíveis logo na primeira decisão. No entanto, exceptuando o ramo que leva o fluxo para fora do ciclo, ou outros ramos têm de prosseguir por um troço comum, o que poderia ser implementado pela replicação do troço comum, ou como é apresentado, pela colocação do

troço comum em sequência com a decisão tardia. Esta segunda opção, implica a alteração da condição de execução do ciclo secundário (representada na figura por 'c') considerando simultaneamente a possibilidade de saída do ciclo principal e a permanência dentro do ciclo secundário.

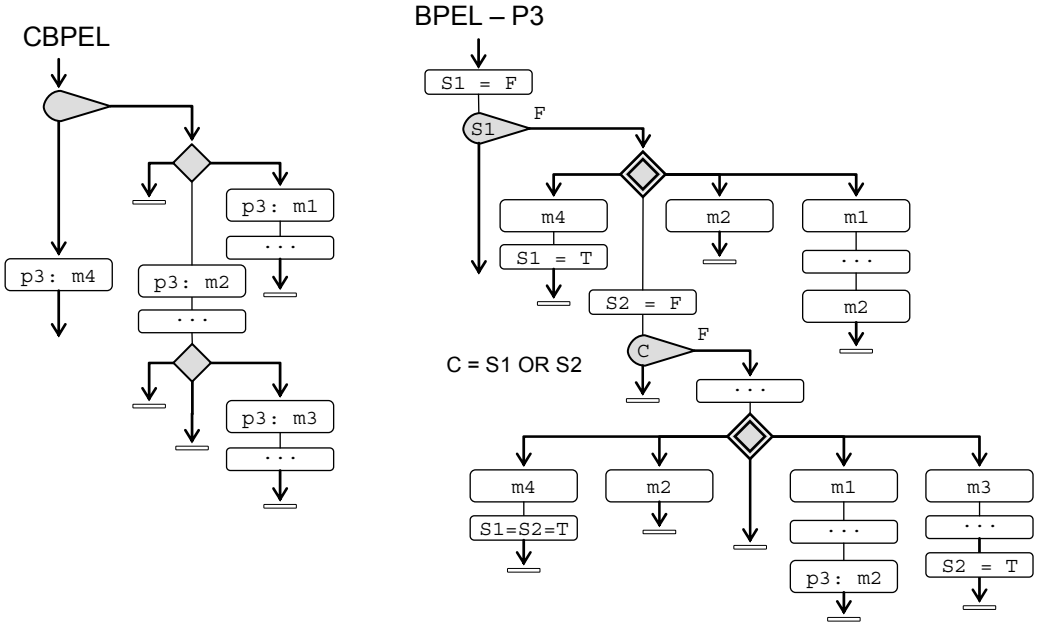


Figura B8 – Transformação de um ciclo com final aberto e início com decisão aberta

Ciclo com decisões seguidas e abertas

A transformação de um ciclo só com decisões abertas em sequência, segue o caso geral de as decisões serem incorporadas numa só, e contendo também o fecho do ciclo. Na figura B9 no lado esquerdo apresenta-se um caso destes, e no lado direito a respectiva transformação.

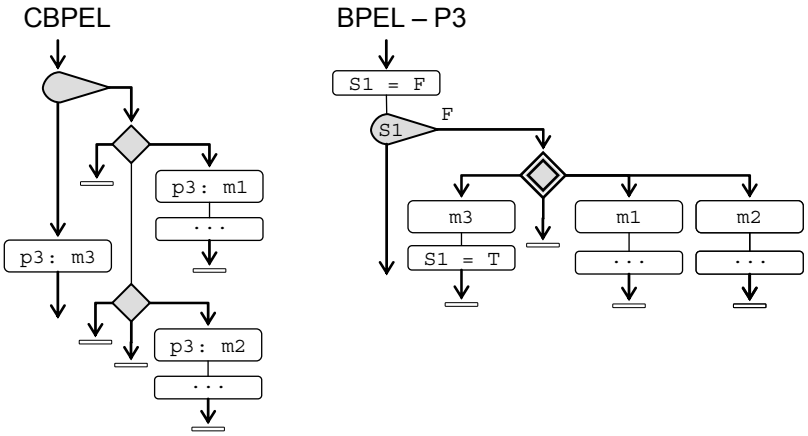


Figura B9 – Transformação de um ciclo com decisões seguidas e abertas

Uma consequência destes casos, é que não há forma de pelo fluxo se inferir quantas vezes o ciclo original é executado, apenas quantas vezes as mensagens são recebidas.

Ciclo com decisões abertas em cascata

Uma outra situação particular ocorre quando existem decisões abertas em cascata dentro de um ciclo, tal como se pode observar na figura B10 no lado esquerdo. Antes de analisar-mos a transformação desse troço, vamos aplicar o fecho possível à primeira decisão, no que resulta na figura existente do lado direito da figura B10.

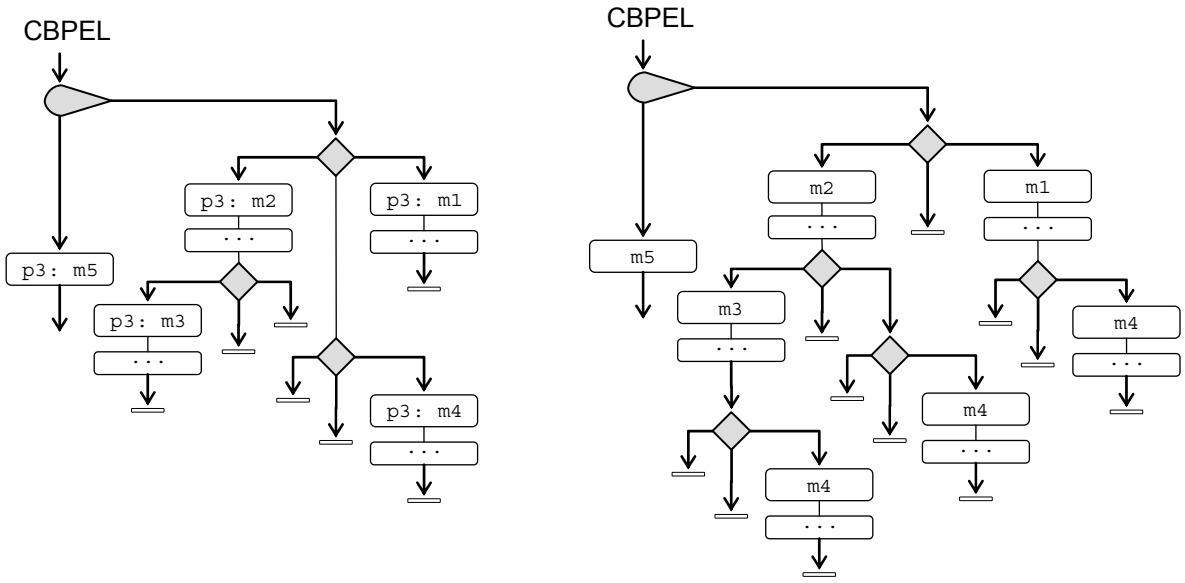


Figura B10 – Ciclo com decisões em cascata

Na figura B11 encontra-se uma transformação parcial dos troços da figura B10. Essa transformação é parcial, pois não é completa. Ela permite visualizar como os troços de fluxo poderiam continuar caso se pudesse interligar os troços já existentes. Tal configuração evitaria o crescimento repetitivo das actividades.

Uma análise mais refinada ao fluxo revela que o ponto P4 é idêntico ao ponto P0, e que os pontos são executados ciclicamente. Essa conclusão permite reescrever todo o fluxo de um modo mais sistemático e coerente para com as construções do controlo de fluxo que estamos a utilizar.

Na figura B12 encontra-se então a transformação final em que as decisões dos pontos P13, P2 e P4 são executadas mediante o estado de uma variável. Essa variável é inicializada de modo a executar a decisão que corresponde ao final do ciclo e consequentemente ao seu início, e no final da recepção de uma mensagem coloca-se a

variável a indicar qual o troço que foi tratado, de modo a voltar ao ciclo e tratar as recepções que se seguem.

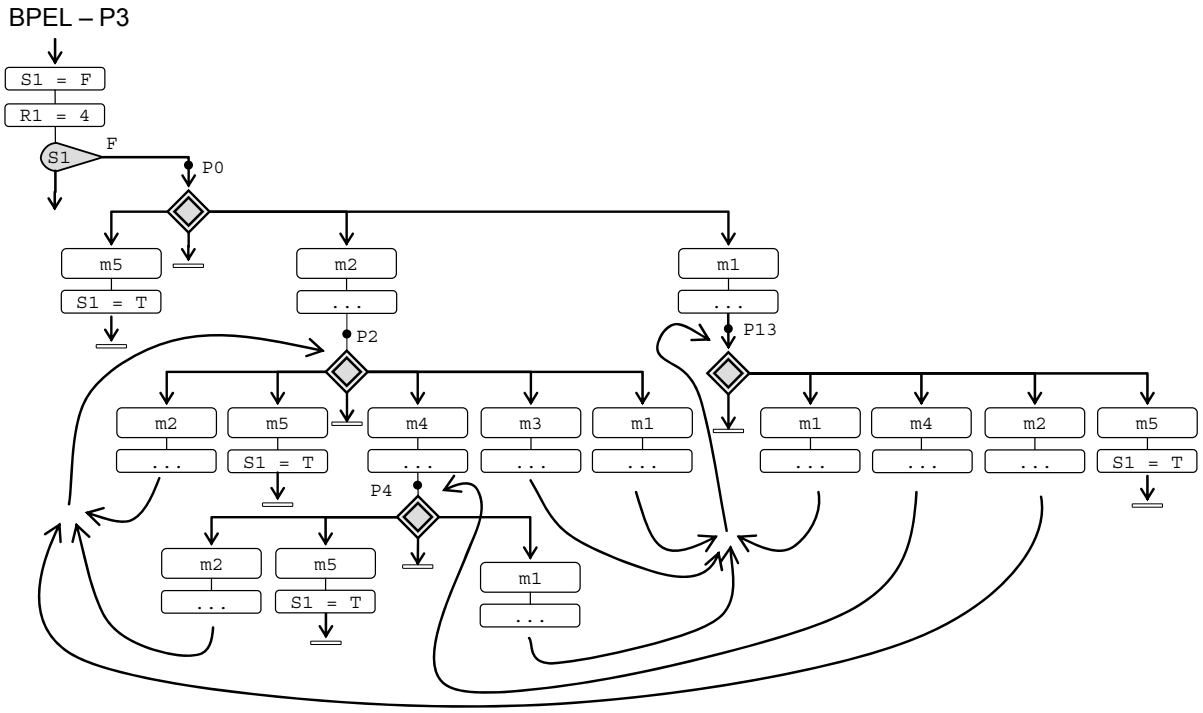


Figura B11 – Transformação incompleta de um ciclo com decisões em cascata

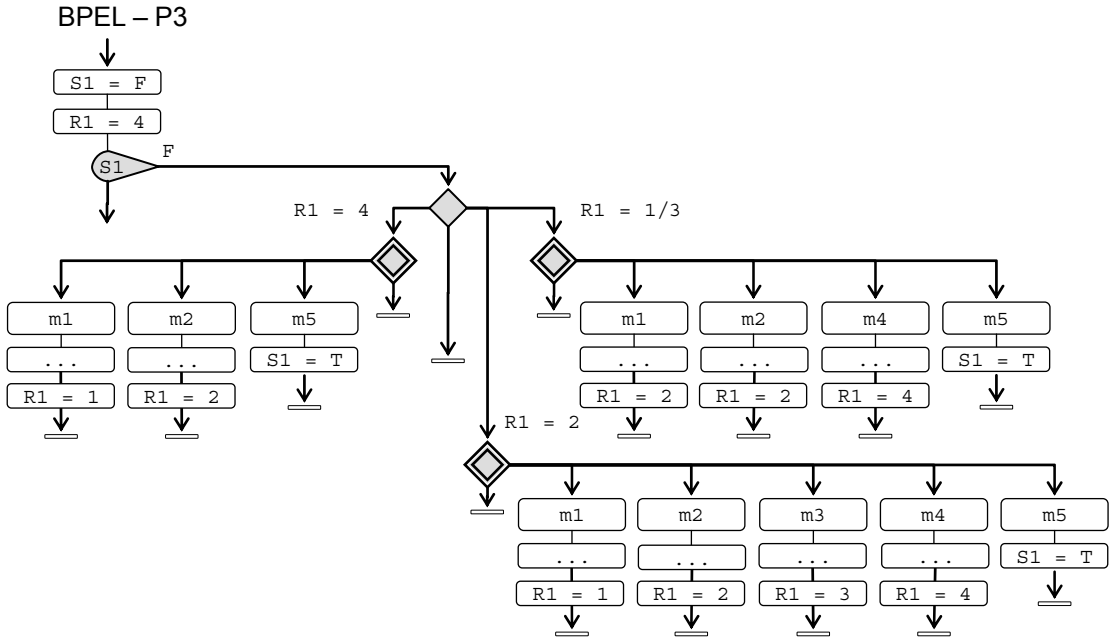


Figura B12 – Transformação de um ciclo com decisões em cascata

Ciclos com ciclo interno

No lado esquerdo da figura B13 encontra-se um primeiro exemplo de um ciclo dentro de outro ciclo. O ciclo que se encontra dentro será denominado de ciclo interno e o ciclo de fora de ciclo externo. A transformação deste exemplo, dado que o ciclo interno tem um fecho evidente, segue a metodologia já apresentada, ou seja, cada ciclo possuirá uma variável de controlo, será implementado por um ciclo e uma decisão tardia com as primeiras mensagens dentro de cada ciclo e com o fecho desse ciclo. No lado direito da mesma figura pode-se observar o resultado da transformação.

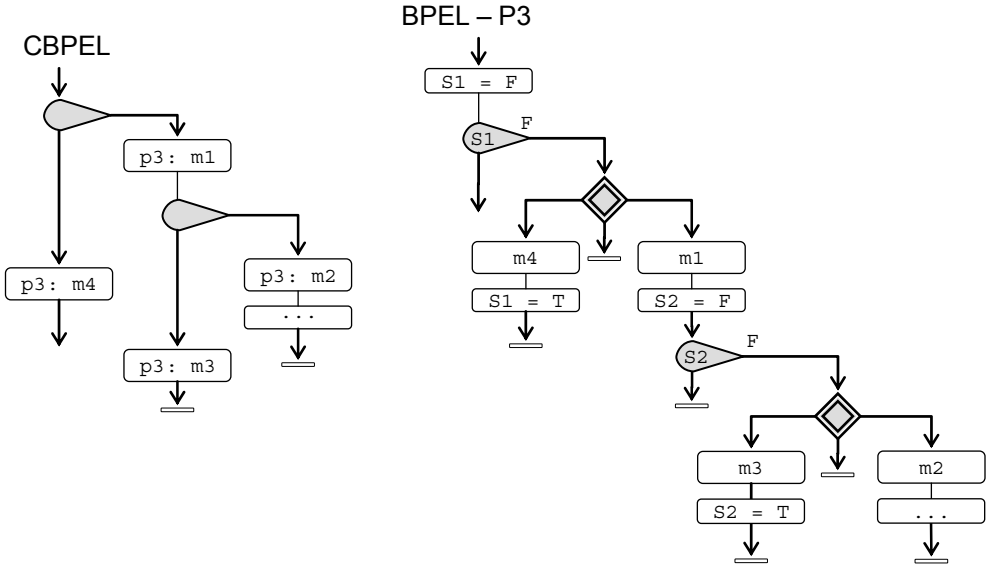


Figura B13 – Transformação de um ciclo dentro de outro ciclo

Ciclo com ciclo interno iniciado com decisão aberta

No lado esquerdo da figura B14 encontra-se uma situação de um ciclo com ciclo interno precedido de uma decisão aberta. Este caso insere-se na metodologia já descrita, pois o ciclo interno vai pertencer ao fecho da decisão e vai por isso ser colocado no final dos dois ramos da decisão. No lado direito da mesma figura encontra-se o resultado da transformação.

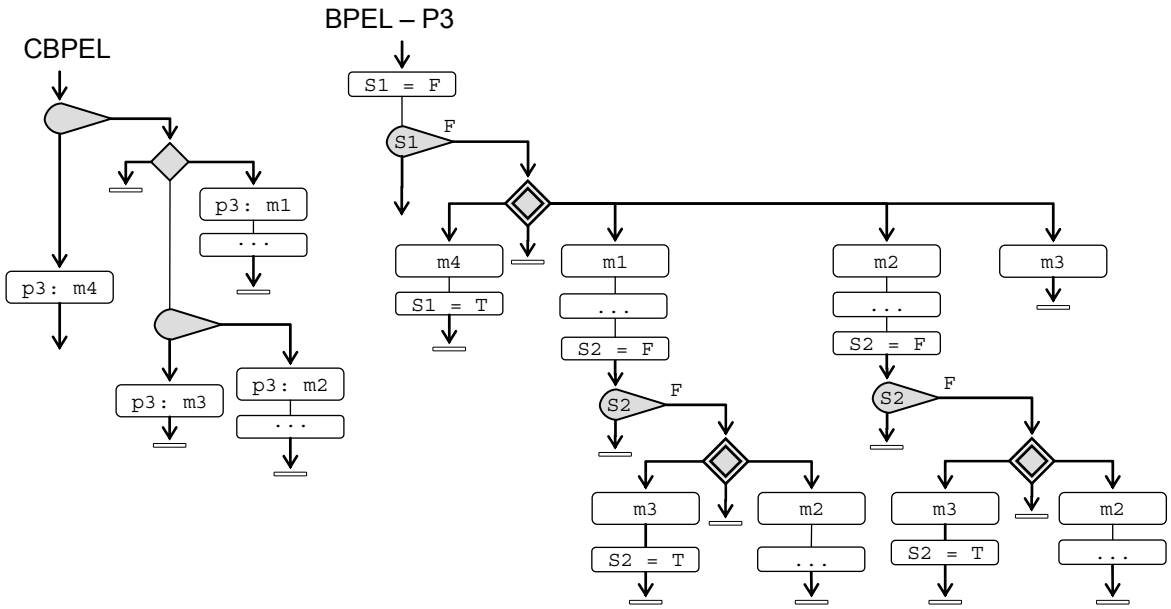


Figura B14 – Transformação de um ciclo interno precedido de decisão aberta

Ciclo com ciclo interno sem fecho

No lado esquerdo da figura B15 encontra-se uma situação de um ciclo interno que não tem um fecho directo. Neste caso o fecho do ciclo interno apanha o início do ciclo externo, incorporando nele as recepções de m1 e m3. A transformação desta situação pode ser observada no lado direito da mesma figura.

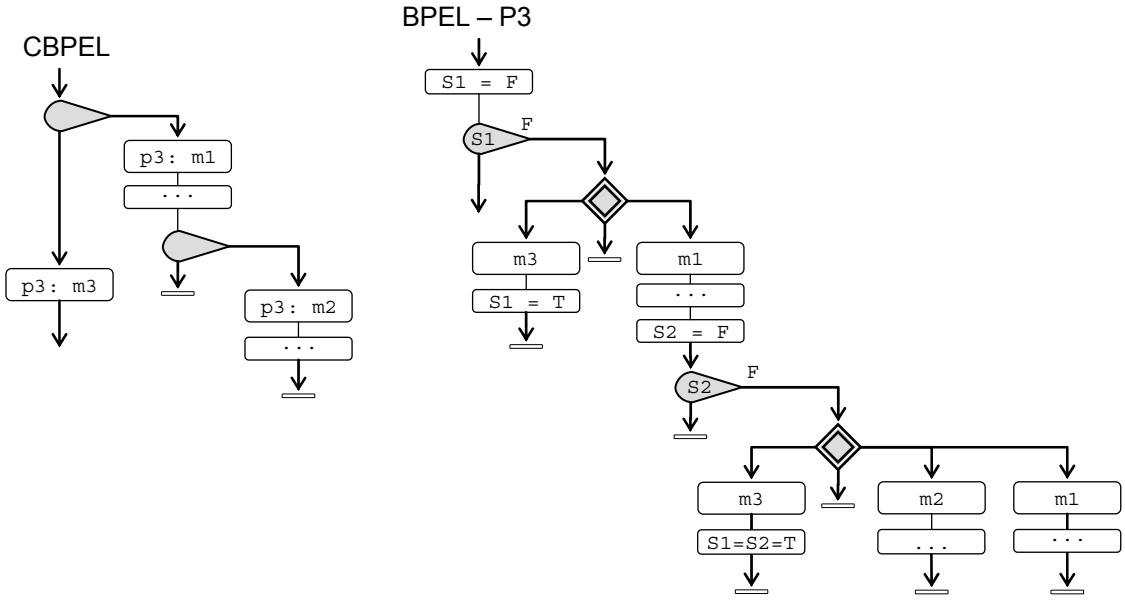


Figura B15 – Transformação de um ciclo interno sem fecho

Ciclo com ciclo interno no seu início

No lado esquerdo da figura B16 encontra-se uma situação de um ciclo que se inica com outro ciclo. A transformação do ciclo interno, resulta na situação normal de variável, ciclo e decisão tardia, mas antes dessa execução tem de se testar se há entrada no ciclo ou não, colocando numa decisão tardia à entrada do ciclo interno, e como o ciclo interno se encontra no início do ciclo externo é necessário colocar também a recepção, ou as recepções, correspondentes ao fecho do ciclo externo, neste caso m3.

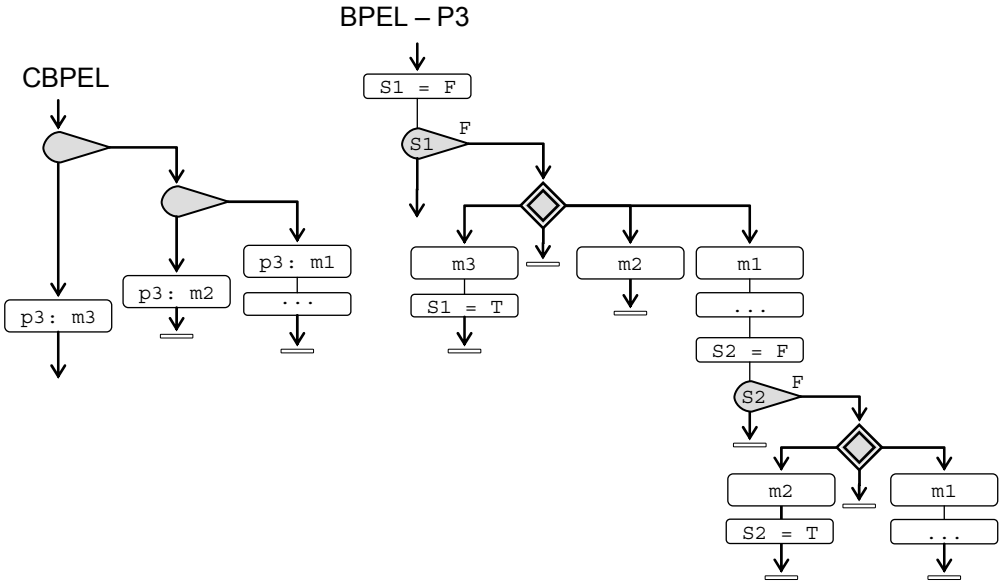


Figura B16 – Transformação de um ciclo com outro ciclo no seu início

Ciclo com ciclo interno e troço aberto

No lado esquerdo da figura B17 encontra-se uma situação de um ciclo com ciclo interno e troço de execução sem qualquer fecho. Nas situações de execução facultativa, não faz sentido pensar-se em sequencialidade, mas sim em execução alternativa. A transformação resulta portanto num ciclo com a possibilidade de recepção da mensagem da decisão inicial, das mensagens ciclo interno, e da mensagem de saída. O ciclo interno foi eliminado pois ficava directamente dentro do ciclo externo.

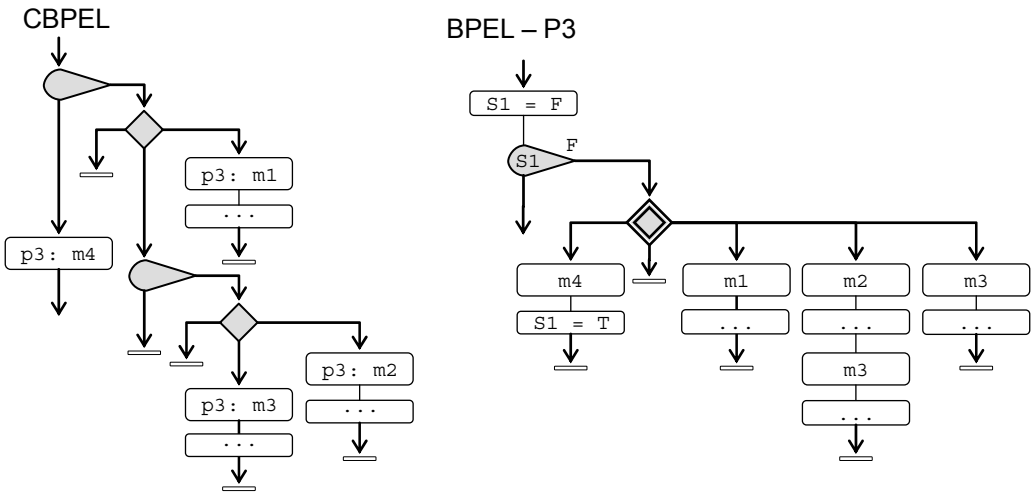


Figura B17 – Transformação de um ciclo com ciclo interno e troço sem qualquer fecho

Considerações finais

Foi apresentado uma série de situações e as respectivas transformações numa tentativa de sistematizar a transformação de fluxos com decisões e ciclos em linguagem CBPEL para a linguagem BPEL. A série de situações apresentada tenta cobrir as várias possibilidades de configuração de fluxos envolvendo decisões e ciclos, mas não garante que não haja situações não contempladas e cuja transformação requeira soluções particulares. No entanto as situações apresentadas cobrem as possibilidades mais evidentes e provavelmente as mais utilizadas de forma frequente.

Anexo C: Xml Schema da Linguagem CBPEL

Segue a descrição em XML Schema da linguagem CBPEL:

```
<?xml version="1.0" encoding="UTF-8"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:cbpel="http://schemas.xmlsoap.org/ws/2005/05/common-business-process/"
        targetNamespace="http://schemas.xmlsoap.org/ws/2005/05/common-business-process/"
        elementFormDefault="qualified">

   <import namespace="http://schemas.xmlsoap.org/wsdl/"
           schemaLocation="http://schemas.xmlsoap.org/wsdl/"/>

   <complexType name="tExtensibleElements">
      <sequence>
         <any namespace="##other" processContents="lax" minOccurs="0"
              maxOccurs="unbounded"/>
      </sequence>
      <anyAttribute namespace="##other" processContents="lax"/>
   </complexType>

   <element name="commonProcess" type="cbpel:tCommonProcess"/>
   <complexType name="tCommonProcess">
      <complexContent>
         <extension base="cbpel:tExtensibleElements">
            <sequence>
               <element ref="cbpel:partnerLinks" minOccurs="0"/>
               <element ref="cbpel:partners" minOccurs="0"/>
               <element ref="cbpel:initiator"/>
               <element ref="cbpel:variables" minOccurs="0"/>
               <element ref="cbpel:correlationSets" minOccurs="0"/>
               <group   ref="cbpel:activity"/>
            </sequence>
            <attribute name="name" type="NCName" use="required"/>
            <attribute name="targetNamespace" type="anyURI" use="required"/>
            <attribute name="queryLanguage" type="anyURI"
                       default="http://www.w3.org/TR/1999/REC-xpath-19991116"/>
            <attribute name="expressionLanguage" type="anyURI"
                       default="http://www.w3.org/TR/1999/REC-xpath-19991116"/>
         </extension>
      </complexContent>
   </complexType>
```

```
<element name="partnerLinks" type="cbpel:tPartnerLinks"/>
<complexType name="tPartnerLinks">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <sequence>
            <element ref="cbpel:partnerLink" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<element name="partnerLink" type="cbpel:tPartnerLink"/>
<complexType name="tPartnerLink">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <sequence>
            <element ref="cbpel:role" minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
         <attribute name="name" type="NCName" use="required"/>
         <attribute name="partnerLinkType" type="QName" use="required"/>
      </extension>
   </complexContent>
</complexType>

<element name="role" type="cbpel:tRole"/>
<complexType name="tRole">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <attribute name="name" type="NCName" use="required"/>
         <attribute name="partnerLinkTypeRoleName" type="NCName" use="required"/>
      </extension>
   </complexContent>
</complexType>

<element name="partners" type="cbpel:tPartners"/>
<complexType name="tPartners">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <sequence>
            <element ref="cbpel:partner" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<element name="partner" type="cbpel:tPartner"/>
<complexType name="tPartner">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <sequence>
            <element name="partnerLink" maxOccurs="unbounded">
               <complexType>
                  <complexContent>
                     <extension base="cbpel:tExtensibleElements">
                        <attribute name="name" type="NCName" use="required"/>
                        <attribute name="role" type="NCName" use="required"/>
                     </extension>
                  </complexContent>
               </complexType>
            </element>
         </sequence>
         <attribute name="name" type="NCName" use="required"/>
      </extension>
   </complexContent>
</complexType>

<element name="variables" type="cbpel:tVariables"/>
<complexType name="tVariables">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <sequence>
            <element ref="cbpel:variable" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

```
<element name="variable" type="cbpel:tVariable"/>
<complexType name="tVariable">
   <attribute name="name" type="NCName" use="required"/>
   <attribute name="messageType" type="QName" use="required"/>
   <attribute name="owner" type="NCName" use="required"/>
</complexType>

<element name="initiator" type="cbpel:tInitiator"/>
<complexType name="tInitiator">
   <attribute name="partner" type="NCName" use="required"/>
   <attribute name="initialSend" type="NCName" use="required"/>
</complexType>

<element name="correlationSets" type="cbpel:tCorrelationSets"/>
<complexType name="tCorrelationSets">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <sequence>
             <element ref="cbpel:correlationSet" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<element name="correlationSet" type="cbpel:tCorrelationSet"/>
<complexType name="tCorrelationSet">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <attribute name="properties" use="required">
            <simpleType>
               <list itemType="QName"/>
            </simpleType>
         </attribute>
         <attribute name="name" type="NCName" use="required"/>
         <attribute name="owner" type="NCName" use="required"/>
      </extension>
   </complexContent>
</complexType>

<group name="activity">
   <choice>
      <element ref="cbpel:send"/>
      <element ref="cbpel:sequence"/>
      <element ref="cbpel:switch"/>
      <element ref="cbpel:while"/>
      <element ref="cbpel:assign"/>
      <element ref="cbpel:flow"/>
      <element ref="cbpel:wait"/>
      <element ref="cbpel:empty"/>
      <element ref="cbpel:throw"/>
      <element ref="cbpel:scope"/>
      <element ref="cbpel:compensate"/>
   </choice>
</group>

<complexType name="tActivity">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <attribute name="name" type="NCName"/>
      </extension>
   </complexContent>
</complexType>

<element name="while" type="cbpel:tWhile"/>
<complexType name="tWhile">
   <complexContent>
      <extension base="cbpel:tActivity">
         <sequence>
            <group ref="cbpel:activity"/>
         </sequence>
         <attribute name="condition" type="cbpel:tBoolean-expr" use="required"/>
         <attribute name="executer" type="NCName" use="required"/>
      </extension>
   </complexContent>
</complexType>
```

```
 <element name="empty" type="cbpel:tEmpty"/>
 <complexType name="tEmpty">
    <complexContent>
       <extension base="cbpel:tActivity">
          <attribute name="executer" type="NCName"/>
       </extension>
    </complexContent>
 </complexType>

<element name="sequence" type="cbpel:tSequence"/>
 <complexType name="tSequence">
    <complexContent>
       <extension base="cbpel:tActivity">
          <sequence>
             <group ref="cbpel:activity" maxOccurs="unbounded"/>
          </sequence>
       </extension>
    </complexContent>
 </complexType>

 <element name="send" type="cbpel:tSend"/>
 <complexType name="tSend">
    <complexContent>
       <extension base="cbpel:tActivity">
          <sequence>
             <element ref="cbpel:correlations" minOccurs="0"/>
          </sequence>
          <attribute name="partnerLink" type="NCName" use="required"/>
          <attribute name="portType" type="QName" use="required"/>
          <attribute name="operation" type="NCName" use="required"/>
          <attribute name="fromPartner" type="NCName" use="required"/>
          <attribute name="toPartner" type="NCName" use="required"/>
          <attribute name="inputVariable" type="NCName" use="optional"/>
          <attribute name="outputVariable" type="NCName" use="optional"/>
          <attribute name="createInstance" type="cbpel:tBoolean" default="no"/>
       </extension>
    </complexContent>
 </complexType>

 <element name="correlations" type="cbpel:tCorrelations"/>
 <complexType name="tCorrelations">
    <complexContent>
       <extension base="cbpel:tExtensibleElements">
          <sequence>
             <element ref="cbpel:correlation" maxOccurs="unbounded"/>
          </sequence>
       </extension>
    </complexContent>
 </complexType>

 <element name="correlation" type="cbpel:tCorrelation"/>
 <complexType name="tCorrelation">
    <complexContent>
       <extension base="cbpel:tExtensibleElements">
          <attribute name="set" type="NCName" use="required"/>
          <attribute name="initiate" type="cbpel:tBoolean" default="no"/>
          <attribute name="pattern">
             <simpleType>
                <restriction base="string">
                   <enumeration value="in"/>
                   <enumeration value="out"/>
                </restriction>
             </simpleType>
          </attribute>
       </extension>
    </complexContent>
 </complexType>

 <element name="wait" type="cbpel:tWait"/>
 <complexType name="tWait">
    <complexContent>
       <extension base="cbpel:tActivity">
          <attribute name="for" type="cbpel:tDuration-expr"/>
          <attribute name="until" type="cbpel:tDeadline-expr"/>
          <attribute name="executer" type="NCName" use="required"/>
       </extension>
    </complexContent>
```

```
</complexType>

<element name="switch" type="cbpel:tSwitch"/>
<complexType name="tSwitch">
   <complexContent>
      <extension base="cbpel:tActivity">
         <sequence>
            <element name="case" maxOccurs="unbounded">
               <complexType>
                  <complexContent>
                     <extension base="cbpel:tActivityContainer">
                        <attribute name="condition" type="cbpel:tBoolean-expr"
                                   use="required"/>
                     </extension>
                  </complexContent>
               </complexType>
            </element>
            <element name="otherwise" type="cbpel:tActivityContainer" minOccurs="0"/>
         </sequence>
         <attribute name="executer" type="NCName" use="required"/>
      </extension>
   </complexContent>
</complexType>

<element name="flow" type="cbpel:tFlow"/>
<complexType name="tFlow">
   <complexContent>
      <extension base="cbpel:tActivity">
         <sequence>
            <group ref="cbpel:activity" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<element name="assign" type="cbpel:tAssign"/>
<complexType name="tAssign">
   <complexContent>
      <extension base="cbpel:tActivity">
         <sequence>
            <element ref="cbpel:copy" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<element name="copy" type="cbpel:tCopy"/>
<complexType name="tCopy">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <sequence>
            <element ref="cbpel:from"/>
            <element ref="cbpel:to"/>
         </sequence>
         <attribute name="executer" type="NCName" use="required"/>
      </extension>
   </complexContent>
</complexType>

<element name="from" type="cbpel:tFrom"/>
<complexType name="tFrom">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <attribute name="variable" type="NCName"/>
         <attribute name="part" type="NCName"/>
         <attribute name="query" type="string"/>
         <attribute name="property" type="QName"/>
         <attribute name="partnerLink" type="NCName"/>
         <attribute name="endpointReference" type="cbpel:tRoles"/>
         <attribute name="expression" type="string"/>
         <attribute name="opaque" type="cbpel:tBoolean"/>
      </extension>
   </complexContent>
</complexType>
```

```
<element name="to">
   <complexType>
      <complexContent>
         <restriction base="cbpel:tFrom">
            <attribute name="expression" type="string" use="prohibited"/>
            <attribute name="opaque" type="cbpel:tBoolean" use="prohibited"/>
            <attribute name="endpointReference" type="cbpel:tRoles" use="prohibited"/>
         </restriction>
      </complexContent>
   </complexType>
</element>
<simpleType name="tRoles">
   <restriction base="string">
      <enumeration value="partnerRole"/>
      <enumeration value="theOtherPartnerRole"/>
   </restriction>
</simpleType>

<element name="throw" type="cbpel:tThrow"/>
<complexType name="tThrow">
   <complexContent>
      <extension base="cbpel:tActivity">
         <attribute name="faultName" type="QName" use="required"/>
         <attribute name="executer" type="NCName" use="required"/>
      </extension>
   </complexContent>
</complexType>

<element name="compensate" type="cbpel:tCompensate"/>
<complexType name="tCompensate">
   <complexContent>
      <extension base="cbpel:tActivity">
         <attribute name="scope" type="NCName"/>
      </extension>
   </complexContent>
</complexType>

<element name="scope" type="cbpel:tScope"/>
<complexType name="tScope">
   <complexContent>
      <extension base="cbpel:tActivity">
         <sequence>
            <element ref="cbpel:variables" minOccurs="0"/>
            <element ref="cbpel:correlationSets" minOccurs="0"/>
            <element ref="cbpel:faultHandlers" minOccurs="0"/>
            <element ref="cbpel:compensationHandler" minOccurs="0"/>
            <element ref="cbpel:eventHandlers" minOccurs="0"/>
            <group   ref="cbpel:activity"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<element name="faultHandlers" type="cbpel:tFaultHandlers"/>
<complexType name="tFaultHandlers">
   <complexContent>
      <extension base="cbpel:tExtensibleElements">
         <sequence>
            <element ref="cbpel:catch" minOccurs="0" maxOccurs="unbounded"/>
            <element ref="cbpel:catchAll" minOccurs="0"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>

<element name="catch" type="cbpel:tCatch"/>
<complexType name="tCatch">
   <complexContent>
      <extension base="cbpel:tActivityContainer">
         <attribute name="faultName" type="QName" use="required"/>
      </extension>
   </complexContent>
</complexType>

<element name="catchAll" type="cbpel:tActivityContainer"/>
```

```
   <complexType name="tActivityContainer">
      <complexContent>
         <extension base="cbpel:tExtensibleElements">
            <sequence>
               <group ref="cbpel:activity"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>

   <element name="eventHandlers" type="cbpel:tEventHandlers" />
   <complexType name="tEventHandlers">
      <complexContent>
         <extension base="cbpel:tExtensibleElements">
            <sequence>
               <element ref="cbpel:messageHandler" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>

   <element name="messageHandler" type="cbpel:tMessageHandler"/>
   <complexType name="tMessageHandler">
      <complexContent>
         <extension base="cbpel:tActivityContainer">
            <attribute name="initiator" type="NCName" use="required"/>
         </extension>
      </complexContent>
   </complexType>

   <element name="compensationHandler" type="cbpel:tCompensationHandler"/>
   <complexType name="tCompensationHandler">
      <complexContent>
         <extension base="cbpel:tActivityContainer"/>
      </complexContent>
   </complexType>

   <simpleType name="tBoolean-expr">
      <restriction base="string"/>
   </simpleType>

   <simpleType name="tDuration-expr">
      <restriction base="string"/>
   </simpleType>

   <simpleType name="tDeadline-expr">
      <restriction base="string"/>
   </simpleType>

   <simpleType name="tBoolean">
      <restriction base="string">
         <enumeration value="yes"/>
         <enumeration value="no"/>
      </restriction>
   </simpleType>

</schema>
```

Anexo D: Codificação XML do Cenário Marítimo

Este anexo apresenta as listagens completas relativas ao cenário marítimo da escala de um navio num porto. O cenário encontra-se descrito na secção 6.1 e as listagens correspondem aos vários processos existentes no ponto 6.1.3.

De modo a evitar o processamento do conteúdo das mensagens, optou-se por não utilizar as mensagens para qualquer efeito. Deste modo, as actividades relativas à troca de informação contêm uma qualquer variável ou mesmo nenhuma e nos casos em que há informação envolvida ela é transportada no nome da actividade.

Os vários processos têm a seguinte localização:

Processo CBPEL do Porto Marítimo

```
<?xml version="1.0" encoding="UTF-8"?>

<commonProcess
   xmlns="http://schemas.xmlsoap.org/ws/2005/05/common-business-process/"
   xmlns:tns="http://meec.ist.utl.pt/cenario50" name="cprocess50-MaritimePort"
   targetNamespace="http://meec.ist.utl.pt/cenario50">
   <!-- ##### partnerLinks ### -->
   <partnerLinks>
      <partnerLink name="pladmagn" partnerLinkType="tns:pltadmagn">
         <role name="ragn" partnerLinkTypeRoleName="pltadmagn-ragn" />
         <role name="radm" partnerLinkTypeRoleName="pltadmagn-radm" />
      </partnerLink>
      <partnerLink name="plalfagn" partnerLinkType="tns:pltalfagn">
         <role name="ragn" partnerLinkTypeRoleName="pltalfagn-ragn" />
         <role name="ralf" partnerLinkTypeRoleName="pltalfagn-ralf" />
      </partnerLink>
      <partnerLink name="plcppagn" partnerLinkType="tns:pltcppagn">
         <role name="ragn" partnerLinkTypeRoleName="pltcppagn-ragn" />
         <role name="rcpp" partnerLinkTypeRoleName="pltcppagn-rcpp" />
      </partnerLink>
      <partnerLink name="plagnadm" partnerLinkType="tns:pltagnadm">
         <role name="ragn" partnerLinkTypeRoleName="pltagnadm-ragn" />
         <role name="radm" partnerLinkTypeRoleName="pltagnadm-radm" />
      </partnerLink>
      <partnerLink name="plagnalf" partnerLinkType="tns:pltagnalf">
         <role name="ragn" partnerLinkTypeRoleName="pltagnalf-ragn" />
         <role name="ralf" partnerLinkTypeRoleName="pltagnalf-ralf" />
      </partnerLink>
      <partnerLink name="plagncpp" partnerLinkType="tns:pltagncpp">
         <role name="ragn" partnerLinkTypeRoleName="pltagncpp-ragn" />
         <role name="rcpp" partnerLinkTypeRoleName="pltagncpp-rcpp" />
      </partnerLink>
      <partnerLink name="plpiladm" partnerLinkType="tns:pltpiladm">
         <role name="radm" partnerLinkTypeRoleName="pltpiladm-radm" />
         <role name="rpil" partnerLinkTypeRoleName="pltpiladm-rpil" />
      </partnerLink>
      <partnerLink name="ploppadm" partnerLinkType="tns:pltoppadm">
         <role name="radm" partnerLinkTypeRoleName="pltoppadm-radm" />
         <role name="ropp" partnerLinkTypeRoleName="pltoppadm-ropp" />
      </partnerLink>
      <partnerLink name="plalfadm" partnerLinkType="tns:pltalfadm">
         <role name="radm" partnerLinkTypeRoleName="pltalfadm-radm" />
         <role name="ralf" partnerLinkTypeRoleName="pltalfadm-ralf" />
      </partnerLink>
      <partnerLink name="plcppadm" partnerLinkType="tns:pltcppadm">
         <role name="radm" partnerLinkTypeRoleName="pltcppadm-radm" />
         <role name="rcpp" partnerLinkTypeRoleName="pltcppadm-rcpp" />
      </partnerLink>
      <partnerLink name="pladmpil" partnerLinkType="tns:pltadmpil">
         <role name="radm" partnerLinkTypeRoleName="pltadmpil-radm" />
         <role name="rcpp" partnerLinkTypeRoleName="pltadmpil-rcpp" />
         <role name="rpil" partnerLinkTypeRoleName="pltadmpil-rpil" />
      </partnerLink>
      <partnerLink name="pladmcpp" partnerLinkType="tns:pltadmcpp">
         <role name="radm" partnerLinkTypeRoleName="pltadmcpp-radm" />
         <role name="rcpp" partnerLinkTypeRoleName="pltadmcpp-rcpp" />
      </partnerLink>
      <partnerLink name="pladmalf" partnerLinkType="tns:pltadmalf">
         <role name="radm" partnerLinkTypeRoleName="pltadmalf-radm" />
         <role name="ralf" partnerLinkTypeRoleName="pltadmalf-ralf" />
      </partnerLink>
      <partnerLink name="pladmopp" partnerLinkType="tns:pltadmopp">
         <role name="radm" partnerLinkTypeRoleName="pltadmopp-radm" />
         <role name="ropp" partnerLinkTypeRoleName="pltadmopp-ropp" />
      </partnerLink>
      <partnerLink name="plalfcpp" partnerLinkType="tns:pltalfcpp">
         <role name="rcpp" partnerLinkTypeRoleName="pltalfcpp-rcpp" />
         <role name="ralf" partnerLinkTypeRoleName="pltalfcpp-ralf" />
      </partnerLink>
   </partnerLinks>
```

```
<!-- ##### partners #### -->
<partners>
   <partner name="agn">
      <partnerLink name="pladmagn" role="ragn" />
      <partnerLink name="plalfagn" role="ragn" />
      <partnerLink name="plcppagn" role="ragn" />
      <partnerLink name="plagnadm" role="ragn" />
      <partnerLink name="plagnalf" role="ragn" />
      <partnerLink name="plagncpp" role="ragn" />
   </partner>
   <partner name="adm">
      <partnerLink name="plagnadm" role="radm" />
      <partnerLink name="plpiladm" role="radm" />
      <partnerLink name="ploppadm" role="radm" />
      <partnerLink name="plalfadm" role="radm" />
      <partnerLink name="plcppadm" role="radm" />
      <partnerLink name="pladmpil" role="radm" />
      <partnerLink name="pladmcpp" role="radm" />
      <partnerLink name="pladmalf" role="radm" />
      <partnerLink name="pladmopp" role="radm" />
      <partnerLink name="pladmagn" role="radm" />
   </partner>
   <partner name="cpp">
      <partnerLink name="pladmcpp" role="rcpp" />
      <partnerLink name="plalfcpp" role="rcpp" />
      <partnerLink name="plagncpp" role="rcpp" />
      <partnerLink name="plcppagn" role="rcpp" />
      <partnerLink name="plcppadm" role="rcpp" />
      <partnerLink name="pladmpil" role="rcpp" />
   </partner>
   <partner name="alf">
      <partnerLink name="pladmalf" role="ralf" />
      <partnerLink name="plagnalf" role="ralf" />
      <partnerLink name="plalfadm" role="ralf" />
      <partnerLink name="plalfcpp" role="ralf" />
      <partnerLink name="plalfagn" role="ralf" />
   </partner>
   <partner name="pil">
      <partnerLink name="pladmpil" role="rpil" />
      <partnerLink name="plpiladm" role="rpil" />
   </partner>
   <partner name="opp">
      <partnerLink name="pladmopp" role="ropp" />
      <partnerLink name="ploppadm" role="ropp" />
   </partner>
</partners>
<!-- ##### initiator ### -->
<initiator partner="agn" initialSend="ped.aut.escala" />
<!-- ##### variables ### -->
<variables>
   <variable name="vagn1" messageType="tns:msg" owner="agn" />
   <variable name="vadm1" messageType="tns:msg" owner="adm" />
   <variable name="vcpp1" messageType="tns:msg" owner="cpp" />
   <variable name="valf1" messageType="tns:msg" owner="alf" />
   <variable name="vpil1" messageType="tns:msg" owner="pil" />
   <variable name="vopp1" messageType="tns:msg" owner="opp" />
</variables>
<!-- ##### activity ### -->
<sequence name="main">
   <send name="ped.aut.escala" partnerLink="plagnadm" portType="tns:ptagnadm"
      operation="ped.aut.escala.op" fromPartner="agn" toPartner="adm"
      inputVariable="vagn1" outputVariable="vadm1" createInstance="yes" />
   <switch name="ped.aut.escala" executer="adm">
      <case condition="escala.aprovada?">
         <sequence name="escala.aprovada">
            <send name="nova.escala" partnerLink="pladmcpp"
               portType="tns:ptadmcpp" operation="nova.escala.op"
               fromPartner="adm" toPartner="cpp"
               inputVariable="vadm1" outputVariable="vcpp1" createInstance="yes" />
            <send name="nova.escala" partnerLink="pladmalf" portType="tns:ptadmalf"
               operation="nova.escala.op" fromPartner="adm" toPartner="alf"
               inputVariable="vadm1" outputVariable="valf1" createInstance="yes" />
            <send name="nova.escala" partnerLink="pladmpil" portType="tns:ptadmpil"
               operation="nova.escala.op" fromPartner="adm" toPartner="pil"
               inputVariable="vadm1" outputVariable="vpil1" createInstance="yes" />
            <send name="nova.escala" partnerLink="pladmopp" portType="tns:ptadmopp"
               operation="nova.escala.op" fromPartner="adm" toPartner="opp"
```

```
      inputVariable="vadm1" outputVariable="vopp1" createInstance="yes" />
   <send name="ped.aut.escala.aprovado" partnerLink="pladmagn"
      portType="tns:ptadmagn" operation="ped.aut.escala.aprovado.op"
      fromPartner="adm" toPartner="agn" inputVariable="vadm1"
      outputVariable="vagn1" />
   <while name="nova.eta" condition="nova.ETA?" executer="agn">
      <sequence name="nova.eta">
         <send name="nova.eta" partnerLink="plagnadm" portType="tns:ptagnadm"
            operation="nova.eta.op" fromPartner="agn" toPartner="adm"
            inputVariable="vagn1" outputVariable="vadm1" />
         <send name="nova.eta" partnerLink="pladmcpp" portType="tns:ptadmcpp"
            operation="nova.eta.op" fromPartner="adm" toPartner="cpp"
            inputVariable="vadm1" outputVariable="vcpp1" />
         <send name="nova.eta" partnerLink="pladmalf" portType="tns:ptadmalf"
            operation="nova.eta.op" fromPartner="adm" toPartner="alf"
            inputVariable="vadm1" outputVariable="valf1" />
         <send name="nova.eta" partnerLink="pladmpil" portType="tns:ptadmpil"
            operation="nova.eta.op" fromPartner="adm" toPartner="pil"
            inputVariable="vadm1" outputVariable="vpil1" />
         <send name="nova.eta" partnerLink="pladmopp" portType="tns:ptadmopp"
            operation="nova.eta.op" fromPartner="adm" toPartner="opp"
            inputVariable="vadm1" outputVariable="vopp1" />
      </sequence>
   </while>
   <send name="navio.ao.largo" partnerLink="plagnadm" portType="tns:ptagnadm"
      operation="navio.ao.largo.op" fromPartner="agn" toPartner="adm"
      inputVariable="vagn1" outputVariable="vadm1" />
   <send name="navio.ao.largo" partnerLink="pladmcpp" portType="tns:ptadmcpp"
      operation="navio.ao.largo.op" fromPartner="adm" toPartner="cpp"
      inputVariable="vadm1" outputVariable="vcpp1" />
   <send name="navio.ao.largo" partnerLink="pladmalf" portType="tns:ptadmalf"
      operation="navio.ao.largo.op" fromPartner="adm" toPartner="alf"
      inputVariable="vadm1" outputVariable="valf1" />
   <send name="navio.ao.largo" partnerLink="pladmopp" portType="tns:ptadmopp"
      operation="navio.ao.largo.op" fromPartner="adm" toPartner="opp"
      inputVariable="vadm1" outputVariable="vopp1" />
   <send name="navio.ao.largo" partnerLink="pladmpil" portType="tns:ptadmpil"
      operation="navio.ao.largo.op" fromPartner="adm" toPartner="pil"
      inputVariable="vadm1" outputVariable="vpil1" />
   <send name="navio.acostado.ata" partnerLink="plpiladm"
      portType="tns:ptpiladm" operation="navio.acostado.op" fromPartner="pil"
      toPartner="adm" inputVariable="vpil1" outputVariable="vadm1" />
   <send name="navio.acostado.ata" partnerLink="pladmcpp"
      portType="tns:ptadmcpp" operation="navio.acostado.op" fromPartner="adm"
      toPartner="cpp" inputVariable="vadm1" outputVariable="vcpp1" />
   <send name="navio.acostado.ata" partnerLink="pladmalf"
      portType="tns:ptadmalf" operation="navio.acostado.op" fromPartner="adm"
      toPartner="alf" inputVariable="vadm1" outputVariable="valf1" />
   <send name="navio.acostado.ata" partnerLink="pladmopp"
      portType="tns:ptadmopp" operation="navio.acostado.op" fromPartner="adm"
      toPartner="opp" inputVariable="vadm1" outputVariable="vopp1" />
   <send name="navio.acostado.ata" partnerLink="pladmagn"
      portType="tns:ptadmagn" operation="navio.acostado.op" fromPartner="adm"
      toPartner="agn" inputVariable="vadm1" outputVariable="vagn1" />
   <while name="agn_maisOperPorFazer" condition="maisOperPorFazer?"
      executer="agn">
      <sequence name="nova.operacao">
         <send name="mais.oper.por.fazer" partnerLink="plagnadm"
            portType="tns:ptagnadm" operation="mais.oper.op" fromPartner="agn"
            toPartner="adm" inputVariable="vagn1" outputVariable="vadm1" />
         <send name="mais.oper.por.fazer" partnerLink="pladmalf"
            portType="tns:ptadmalf" operation="mais.oper.op" fromPartner="adm"
            toPartner="alf" inputVariable="vadm1" outputVariable="valf1" />
         <send name="mais.oper.por.fazer" partnerLink="pladmcpp"
            portType="tns:ptadmcpp" operation="mais.oper.op" fromPartner="adm"
            toPartner="cpp" inputVariable="vadm1" outputVariable="vcpp1" />
         <send name="mais.oper.por.fazer" partnerLink="pladmopp"
            portType="tns:ptadmopp" operation="mais.oper.op" fromPartner="adm"
            toPartner="opp" inputVariable="vadm1" outputVariable="vopp1" />
         <send name="mais.oper.por.fazer" partnerLink="pladmpil"
            portType="tns:ptadmpil" operation="mais.oper.op" fromPartner="adm"
            toPartner="pil" inputVariable="vadm1" outputVariable="vpil1" />
         <switch name="CargaDescargaOuMovimentacao" executer="agn">
            <case condition="operacao.carga.descarga?">
               <sequence name="oper.carga.descarga">
                  <send name="oper.carga.descarga.eto" partnerLink="plagnadm"
                     portType="tns:ptagnadm" operation="oper.cd.eto" fromPartner="agn"
                     toPartner="adm" inputVariable="vagn1" outputVariable="vadm1" />
```

```
            <send name="oper.carga.descarga.eto.info" partnerLink="pladmalf"
               portType="tns:ptadmalf" operation="oper.cd.info.op"
               fromPartner="adm" toPartner="alf" inputVariable="vadm1"
               outputVariable="valf1" />
            <send name="oper.carga.descarga.eto.info" partnerLink="pladmcpp"
               portType="tns:ptadmcpp" operation="oper.cd.info.op"
               fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
               outputVariable="vcpp1" />
            <send name="oper.carga.descarga.eto.info" partnerLink="pladmopp"
               portType="tns:ptadmopp" operation="oper.cd.info.op"
               fromPartner="adm" toPartner="opp" inputVariable="vadm1"
               outputVariable="vopp1" />
            <send name="oper.carga.descarga.eto.info" partnerLink="pladmpil"
               portType="tns:ptadmopp" operation="oper.cd.info.op"
               fromPartner="adm" toPartner="pil" inputVariable="vadm1"
               outputVariable="vpil1" />
            <switch name="oper.carga.descarga.aprovada" executer="adm">
               <case condition="oper.carga.descarga.aprovada?">
                  <sequence name="oper.carga.descarga">
                     <send name="oper.carga.descarga.eto" partnerLink="pladmalf"
                        portType="tns:ptadmalf" operation="oper.cd.eto.op"
                        fromPartner="adm" toPartner="alf" inputVariable="vadm1"
                        outputVariable="valf1" />
                     <send name="oper.carga.descarga.eto" partnerLink="pladmcpp"
                        portType="tns:ptadmcpp" operation="oper.cd.eto.op"
                        fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
                        outputVariable="vcpp1" />
                     <send name="oper.carga.descarga.eto" partnerLink="pladmopp"
                        portType="tns:ptadmopp" operation="oper.cd.eto.op"
                        fromPartner="adm" toPartner="opp" inputVariable="vadm1"
                        outputVariable="vopp1" />
                     <send name="oper.carga.descarga.aprovada" partnerLink="pladmagn"
                        portType="tns:ptadmagn" operation="oper.cd.aprovada.op"
                        fromPartner="adm" toPartner="agn" inputVariable="vadm1"
                        outputVariable="vagn1" />
                     <send name="oper.carga.descarga.ato" partnerLink="ploppadm"
                        portType="tns:ptoppadm" operation="oper.cd.ato.op"
                        fromPartner="opp" toPartner="adm" inputVariable="vopp1"
                        outputVariable="vadm1" />
                     <send name="oper.carga.descarga.ato" partnerLink="pladmalf"
                        portType="tns:ptadmalf" operation="oper.cd.ato.op"
                        fromPartner="adm" toPartner="alf" inputVariable="vadm1"
                        outputVariable="valf1" />
                     <send name="oper.carga.descarga.ato" partnerLink="pladmcpp"
                        portType="tns:ptadmcpp" operation="oper.cd.ato.op"
                        fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
                        outputVariable="vcpp1" />
                     <send name="oper.carga.descarga.ato" partnerLink="pladmagn"
                        portType="tns:ptadmagn" operation="oper.cd.ato.op"
                        fromPartner="adm" toPartner="agn" inputVariable="vadm1"
                        outputVariable="vagn1" />
                  </sequence>
               </case>
               <case condition="oper.carga.descarga.recusada?">
                  <sequence name="oper.carga.descarga.recusada">
                     <send name="oper.carga.descarga.recusada" partnerLink="pladmagn"
                        portType="tns:ptadmagn" operation="oper.cd.recusada.op"
                        fromPartner="adm" toPartner="agn" inputVariable="vadm1"
                        outputVariable="vagn1" />
                     <send name="oper.carga.descarga.recusada" partnerLink="pladmalf"
                        portType="tns:ptadmalf" operation="oper.cd.recusada.op"
                        fromPartner="adm" toPartner="alf" inputVariable="vadm1"
                        outputVariable="valf1" />
                     <send name="oper.carga.descarga.recusada" partnerLink="pladmcpp"
                        portType="tns:ptadmcpp" operation="oper.cd.recusada.op"
                        fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
                        outputVariable="vcpp1" />
                     <send name="oper.carga.descarga.recusada" partnerLink="pladmopp"
                        portType="tns:ptadmopp" operation="oper.cd.recusada.op"
                        fromPartner="adm" toPartner="opp" inputVariable="vadm1"
                        outputVariable="vopp1" />
                  </sequence>
               </case>
            </switch>
         </sequence>
      </case>
```

```
<case condition="operacao.movimentacao?">
   <sequence name="oper.movimentacao">
      <send name="oper.movimentacao.etm" partnerLink="plagnadm"
         portType="tns:ptagnadm" operation="oper.mov.info.op"
         fromPartner="agn" toPartner="adm" inputVariable="vagn1"
         outputVariable="vadm1" />
      <send name="oper.mov.etm.info" partnerLink="pladmalf"
         portType="tns:ptadmalf" operation="oper.mov.info.op"
         fromPartner="adm" toPartner="alf" inputVariable="vadm1"
         outputVariable="valf1" />
      <send name="oper.mov.etm.info" partnerLink="pladmcpp"
         portType="tns:ptadmcpp" operation="oper.mov.info.op"
         fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
         outputVariable="vcpp1" />
      <send name="oper.mov.etm.info" partnerLink="pladmopp"
         portType="tns:ptadmopp" operation="oper.mov.info.op"
         fromPartner="adm" toPartner="opp" inputVariable="vadm1"
         outputVariable="vopp1" />
      <send name="oper.mov.etm.info" partnerLink="pladmpil"
         portType="tns:ptadmpil" operation="oper.mov.info.op"
         fromPartner="adm" toPartner="pil" inputVariable="vadm1"
         outputVariable="vpil1" />
      <switch name="operacao.movimentacao.aprovada" executer="adm">
         <case condition="oper.movimentacao.aprovada?">
            <sequence name="oper.movimentacao">
               <send name="oper.movimentacao.etm" partnerLink="pladmalf"
                  portType="tns:ptadmalf" operation="oper.mov.etm.op"
                  fromPartner="adm" toPartner="alf" inputVariable="vadm1"
                  outputVariable="valf1" />
               <send name="oper.movimentacao.etm" partnerLink="pladmcpp"
                  portType="tns:ptadmcpp" operation="oper.mov.etm.op"
                  fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
                  outputVariable="vcpp1" />
               <send name="oper.movimentacao.etm" partnerLink="pladmopp"
                  portType="tns:ptadmopp" operation="oper.mov.etm.op"
                  fromPartner="adm" toPartner="opp" inputVariable="vadm1"
                  outputVariable="vopp1" />
               <send name="oper.movimentacao.etm" partnerLink="pladmpil"
                  portType="tns:ptadmpil" operation="oper.mov.etm.op"
                  fromPartner="adm" toPartner="pil" inputVariable="vadm1"
                  outputVariable="vpil1" />
               <send name="oper.movimentacao.aprovada" partnerLink="pladmagn"
                  portType="tns:ptadmagn" operation="oper.mov.aprovada.op"
                  fromPartner="adm" toPartner="agn" inputVariable="vadm1"
                  outputVariable="vagn1" />
               <send name="oper.movimentacao.stm" partnerLink="plpiladm"
                  portType="tns:ptpiladm" operation="oper.mov.stm.op"
                  fromPartner="pil" toPartner="adm" inputVariable="vpil1"
                  outputVariable="vadm1" />
               <send name="oper.movimentacao.stm" partnerLink="pladmagn"
                  portType="tns:ptadmagn" operation="oper.mov.stm.op"
                  fromPartner="adm" toPartner="agn" inputVariable="vadm1"
                  outputVariable="vagn1" />
               <send name="oper.movimentacao.ftm" partnerLink="plpiladm"
                  portType="tns:ptpiladm" operation="oper.mov.stm.op"
                  fromPartner="pil" toPartner="adm" inputVariable="vpil1"
                  outputVariable="vadm1" />
               <send name="oper.movimentacao.ftm" partnerLink="pladmalf"
                  portType="tns:ptadmalf" operation="oper.mov.stm.op"
                  fromPartner="adm" toPartner="alf" inputVariable="vadm1"
                  outputVariable="valf1" />
               <send name="oper.movimentacao.ftm" partnerLink="pladmcpp"
                  portType="tns:ptadmcpp" operation="oper.mov.stm.op"
                  fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
                  outputVariable="vcpp1" />
               <send name="oper.movimentacao.ftm" partnerLink="pladmopp"
                  portType="tns:ptadmopp" operation="oper.mov.stm.op"
                  fromPartner="adm" toPartner="opp" inputVariable="vadm1"
                  outputVariable="vopp1" />
               <send name="oper.movimentacao.ftm" partnerLink="pladmagn"
                  portType="tns:ptadmagn" operation="oper.mov.stm.op"
                  fromPartner="adm" toPartner="agn" inputVariable="vadm1"
                  outputVariable="vagn1" />
            </sequence>
         </case>
```

```
                    <case condition="oper.movimentacao.recusada?">
                       <sequence name="oper.movimentacao.recusada">
                          <send name="oper.movimentacao.recusada" partnerLink="pladmagn"
                             portType="tns:ptadmagn" operation="op-bc_send_bill"
                             fromPartner="adm" toPartner="agn" inputVariable="vadm1"
                             outputVariable="vagn1" />
                          <send name="oper.mov.recusada" partnerLink="pladmalf"
                             portType="tns:ptadmalf" operation="oper.mov.recusada.op"
                             fromPartner="adm" toPartner="alf" inputVariable="vadm1"
                             outputVariable="valf1" />
                          <send name="oper.mov.recusada" partnerLink="pladmcpp"
                             portType="tns:ptadmcpp" operation="oper.mov.recusada.op"
                             fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
                             outputVariable="vcpp1" />
                          <send name="oper.mov.recusada" partnerLink="pladmopp"
                             portType="tns:ptadmopp" operation="oper.mov.recusada.op"
                             fromPartner="adm" toPartner="opp" inputVariable="vadm1"
                             outputVariable="vopp1" />
                          <send name="oper.mov.recusada" partnerLink="pladmpil"
                             portType="tns:ptadmpil" operation="oper.mov.recusada.op"
                             fromPartner="adm" toPartner="pil" inputVariable="vadm1"
                             outputVariable="vpil1" />
                       </sequence>
                    </case>
                 </switch>
              </sequence>
           </case>
        </switch>
     </sequence>
  </while>
  <send name="ped.desembaraco" partnerLink="plagnalf" portType="tns:ptagnalf"
     operation="ped.desembaraco.op" fromPartner="agn" toPartner="alf"
     inputVariable="vagn1" outputVariable="valf1" />
  <while name="alf.desembaraco.rec" condition="alf.desembaraco.recusado?"
     executer="alf">
     <sequence name="alf.recusa.desembaraco">
        <send name="ped.desembaraco.recusado" partnerLink="plalfagn"
           portType="tns:ptalfagn" operation="ped.desembaraco.recusado.op"
           fromPartner="alf" toPartner="agn" inputVariable="valf1"
           outputVariable="vagn1" />
        <send name="ped.desembaraco" partnerLink="plagnalf"
           portType="tns:ptagnalf" operation="ped.desembaraco.op"
           fromPartner="agn" toPartner="alf" inputVariable="vagn1"
           outputVariable="valf1" />
     </sequence>
  </while>
  <send name="ped.desembaraco.aprovado" partnerLink="plalfadm"
     portType="tns:ptalfadm" operation="ped.desembaraco.aprovado.op"
     fromPartner="alf" toPartner="adm" inputVariable="valf1"
     outputVariable="vadm1" />
  <send name="ped.desembaraco.aprovado" partnerLink="plalfcpp"
     portType="tns:ptalfcpp" operation="ped.desembaraco.aprovado.op"
     fromPartner="alf" toPartner="cpp" inputVariable="valf1"
     outputVariable="vcpp1" />
  <send name="ped.desembaraco.aprovado" partnerLink="plalfagn"
     portType="tns:ptalfagn" operation="ped.desembaraco.aprovado.op"
     fromPartner="alf" toPartner="agn" inputVariable="valf1"
     outputVariable="vagn1" />
  <send name="ped.desembaraco.etd" partnerLink="plagncpp"
     portType="tns:ptagncpp" operation="ped.desembaraco.etd.op"
     fromPartner="agn" toPartner="cpp" inputVariable="vagn1"
     outputVariable="vcpp1" />
  <while name="cpp.desembaraco.rec" condition="cpp.desembaraco.recusado?"
     executer="cpp">
     <sequence name="cpp.recusa.desembaraco">
        <send name="ped.desembaraco.recusado" partnerLink="plcppagn"
           portType="tns:ptcppagn" operation="ped.desembaraco.recusado.op"
           fromPartner="cpp" toPartner="agn" inputVariable="vcpp1"
           outputVariable="vagn1" />
        <send name="ped.desembaraco.etd" partnerLink="plagncpp"
           portType="tns:ptagncpp" operation="ped.desembaraco.etd.op"
           fromPartner="agn" toPartner="cpp" inputVariable="vagn1"
           outputVariable="vcpp1" />
     </sequence>
  </while>
```

```
            <send name="desembaraco.etd" partnerLink="plcppadm" portType="tns:ptcppadm"
               operation="desembaraco.etd.op" fromPartner="cpp" toPartner="adm"
               inputVariable="vcpp1" outputVariable="vadm1" />
            <send name="desembaraco.etd" partnerLink="pladmalf" portType="tns:ptadmalf"
               operation="desembaraco.etd.op" fromPartner="adm" toPartner="alf"
               inputVariable="vadm1" outputVariable="valf1" />
            <send name="desembaraco.etd" partnerLink="pladmopp" portType="tns:ptadmopp"
               operation="desembaraco.etd.op" fromPartner="adm" toPartner="opp"
               inputVariable="vadm1" outputVariable="vopp1" />
            <send name="ped.desembaraco.aprovado" partnerLink="pladmagn"
               portType="tns:ptadmagn" operation="ped.desembaraco.aprovado.op"
               fromPartner="adm" toPartner="agn" inputVariable="vadm1"
               outputVariable="vagn1" />
            <send name="desembaraco.etd" partnerLink="pladmpil" portType="tns:ptadmpil"
               operation="desembaraco.etd.op" fromPartner="adm" toPartner="pil"
               inputVariable="vadm1" outputVariable="vpil1" />
            <send name="navio.desacostado.std" partnerLink="plpiladm"
               portType="tns:ptpiladm" operation="navio.desacostado.std.op"
               fromPartner="pil" toPartner="adm" inputVariable="vpil1"
               outputVariable="vadm1" />
            <send name="navio.desacostado.std" partnerLink="pladmagn"
               portType="tns:ptadmagn" operation="navio.desacostado.std.op"
               fromPartner="adm" toPartner="agn" inputVariable="vadm1"
               outputVariable="vagn1" />
            <send name="navio.ao.largo.atd" partnerLink="plpiladm"
               portType="tns:ptpiladm" operation="navio.ao.largo.atd.op"
               fromPartner="pil" toPartner="adm" inputVariable="vpil1"
               outputVariable="vadm1" />
            <send name="navio.ao.largo.atd" partnerLink="pladmcpp"
               portType="tns:ptadmcpp" operation="navio.ao.largo.atd.op"
               fromPartner="adm" toPartner="cpp" inputVariable="vadm1"
               outputVariable="vcpp1" />
            <send name="navio.ao.largo.atd" partnerLink="pladmalf"
               portType="tns:ptadmalf" operation="navio.ao.largo.atd.op"
               fromPartner="adm" toPartner="alf" inputVariable="vadm1"
               outputVariable="valf1" />
            <send name="navio.ao.largo.atd" partnerLink="pladmopp"
               portType="tns:ptadmopp" operation="navio.ao.largo.atd.op"
               fromPartner="adm" toPartner="opp" inputVariable="vadm1"
               outputVariable="vopp1" />
            <send name="navio.ao.largo.atd" partnerLink="pladmagn"
               portType="tns:ptadmagn" operation="navio.ao.largo.atd.op"
               fromPartner="adm" toPartner="agn" inputVariable="vadm1"
               outputVariable="vagn1" />
         </sequence>
      </case>
      <case condition="escala.recusada?">
         <send name="escala.recusada" partnerLink="pladmagn" portType="tns:ptadmagn"
            operation="escala.recusada.op" fromPartner="adm" toPartner="agn"
            inputVariable="vadm1" outputVariable="vagn1" />
      </case>
   </switch>
</sequence>
</commonProcess>
```

Processo BPEL da Administração do Porto (adm)

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario50" name="cprocess50-MaritimePort-adm"
         targetNamespace="http://meec.ist.utl.pt/cenario50" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="pladmagn" partnerLinkType="tns:pltadmagn" partnerRole="pltadmagn-ragn"/>
    <partnerLink name="pladmalf" partnerLinkType="tns:pltadmalf" partnerRole="pltadmalf-ralf"/>
    <partnerLink name="pladmcpp" partnerLinkType="tns:pltadmcpp" partnerRole="pltadmcpp-rcpp"/>
    <partnerLink name="pladmopp" partnerLinkType="tns:pltadmopp" partnerRole="pltadmopp-ropp"/>
    <partnerLink name="pladmpil" partnerLinkType="tns:pltadmpil" partnerRole="pltadmpil-rpil"/>
    <partnerLink name="plagnadm" partnerLinkType="tns:pltagnadm" myRole="pltagnadm-radm"/>
    <partnerLink name="plalfadm" partnerLinkType="tns:pltalfadm" myRole="pltalfadm-radm"/>
    <partnerLink name="plcppadm" partnerLinkType="tns:pltcppadm" myRole="pltcppadm-radm"/>
    <partnerLink name="ploppadm" partnerLinkType="tns:pltoppadm" myRole="pltoppadm-radm"/>
    <partnerLink name="plpiladm" partnerLinkType="tns:pltpiladm" myRole="pltpiladm-radm"/>
```

```
</partnerLinks>
<partners>
  <partner name="agn">
    <partnerLink name="pladmagn"/>
    <partnerLink name="plagnadm"/>
  </partner>
  <partner name="alf">
    <partnerLink name="pladmalf"/>
    <partnerLink name="plalfadm"/>
  </partner>
  <partner name="cpp">
    <partnerLink name="pladmcpp"/>
    <partnerLink name="plcppadm"/>
  </partner>
  <partner name="opp">
    <partnerLink name="pladmopp"/>
    <partnerLink name="ploppadm"/>
  </partner>
  <partner name="pil">
    <partnerLink name="pladmpil"/>
    <partnerLink name="plpiladm"/>
  </partner>
</partners>
<variables>
  <variable name="vadm1" messageType="tns:msg"/>
</variables>
<sequence name="main">
  <receive name="ped.aut.escala" partnerLink="plagnadm" portType="tns:ptagnadm"
           operation="ped.aut.escala.op" variable="vadm1" createInstance="yes"/>
  <switch name="ped.aut.escala">
    <case condition="escala.aprovada?">
      <sequence name="escala.aprovada">
        <invoke name="nova.escala" partnerLink="pladmcpp" portType="tns:ptadmcpp"
                operation="nova.escala.op" inputVariable="vadm1"/>
        <invoke name="nova.escala" partnerLink="pladmalf" portType="tns:ptadmalf"
                operation="nova.escala.op" inputVariable="vadm1"/>
        <invoke name="nova.escala" partnerLink="pladmpil" portType="tns:ptadmpil"
                operation="nova.escala.op" inputVariable="vadm1"/>
        <invoke name="nova.escala" partnerLink="pladmopp" portType="tns:ptadmopp"
                operation="nova.escala.op" inputVariable="vadm1"/>
        <invoke name="ped.aut.escala.aprovado" partnerLink="pladmagn" portType="tns:ptadmagn"
                operation="ped.aut.escala.aprovado.op" inputVariable="vadm1"/>
        <assign>
          <copy>
            <from expression="false"/>
            <to variable="sair"/>
          </copy>
        </assign>
        <while name="nova.eta" condition="sair==False">
          <pick>
            <onMessage partnerLink="plagnadm" portType="tns:ptagnadm" operation="nova.eta.op"
                       variable="vadm1">
              <sequence name="nova.eta">
                <invoke name="nova.eta" partnerLink="pladmcpp" portType="tns:ptadmcpp"
                        operation="nova.eta.op" inputVariable="vadm1"/>
                <invoke name="nova.eta" partnerLink="pladmalf" portType="tns:ptadmalf"
                        operation="nova.eta.op" inputVariable="vadm1"/>
                <invoke name="nova.eta" partnerLink="pladmpil" portType="tns:ptadmpil"
                        operation="nova.eta.op" inputVariable="vadm1"/>
                <invoke name="nova.eta" partnerLink="pladmopp" portType="tns:ptadmopp"
                        operation="nova.eta.op" inputVariable="vadm1"/>
              </sequence>
            </onMessage>
            <onMessage partnerLink="plagnadm" portType="tns:ptagnadm" operation="navio.ao.largo.op"
                       variable="vadm1">
              <assign>
                <copy>
                  <from expression="true"/>
                  <to variable="sair"/>
                </copy>
              </assign>
            </onMessage>
          </pick>
        </while>
        <receive name="navio.ao.largo" partnerLink="plagnadm" portType="tns:ptagnadm"
                 operation="navio.ao.largo.op" variable="vadm1"/>
        <invoke name="navio.ao.largo" partnerLink="pladmcpp" portType="tns:ptadmcpp"
                operation="navio.ao.largo.op" inputVariable="vadm1"/>
```

```
<invoke name="navio.ao.largo" partnerLink="pladmalf" portType="tns:ptadmalf"
        operation="navio.ao.largo.op" inputVariable="vadm1"/>
<invoke name="navio.ao.largo" partnerLink="pladmopp" portType="tns:ptadmopp"
        operation="navio.ao.largo.op" inputVariable="vadm1"/>
<invoke name="navio.ao.largo" partnerLink="pladmpil" portType="tns:ptadmpil"
        operation="navio.ao.largo.op" inputVariable="vadm1"/>
<receive name="navio.acostado.ata" partnerLink="plpiladm" portType="tns:ptpiladm"
        operation="navio.acostado.op" variable="vadm1"/>
<invoke name="navio.acostado.ata" partnerLink="pladmcpp" portType="tns:ptadmcpp"
        operation="navio.acostado.op" inputVariable="vadm1"/>
<invoke name="navio.acostado.ata" partnerLink="pladmalf" portType="tns:ptadmalf"
        operation="navio.acostado.op" inputVariable="vadm1"/>
<invoke name="navio.acostado.ata" partnerLink="pladmopp" portType="tns:ptadmopp"
        operation="navio.acostado.op" inputVariable="vadm1"/>
<invoke name="navio.acostado.ata" partnerLink="pladmagn" portType="tns:ptadmagn"
        operation="navio.acostado.op" inputVariable="vadm1"/>
<assign>
  <copy>
    <from expression="false"/>
    <to variable="sair"/>
  </copy>
</assign>
<while name="agn_maisOperPorFazer" condition="sair==False">
  <pick>
    <onMessage partnerLink="plagnadm" portType="tns:ptagnadm" operation="mais.oper.op"
               variable="vadm1">
      <sequence name="nova.operacao">
        <invoke name="mais.oper.por.fazer" partnerLink="pladmalf" portType="tns:ptadmalf"
                operation="mais.oper.op" inputVariable="vadm1"/>
        <invoke name="mais.oper.por.fazer" partnerLink="pladmcpp" portType="tns:ptadmcpp"
                operation="mais.oper.op" inputVariable="vadm1"/>
        <invoke name="mais.oper.por.fazer" partnerLink="pladmopp" portType="tns:ptadmopp"
                operation="mais.oper.op" inputVariable="vadm1"/>
        <invoke name="mais.oper.por.fazer" partnerLink="pladmpil" portType="tns:ptadmpil"
                operation="mais.oper.op" inputVariable="vadm1"/>
        <pick name="CargaDescargaOuMovimentacao">
          <onMessage partnerLink="plagnadm" portType="tns:ptagnadm" operation="oper.cd.eto"
                     variable="vadm1">
            <sequence name="oper.carga.descarga">
              <invoke name="oper.carga.descarga.eto.info" partnerLink="pladmalf"
                     portType="tns:ptadmalf" operation="oper.cd.info.op" inputVariable="vadm1"/>
              <invoke name="oper.carga.descarga.eto.info" partnerLink="pladmcpp"
                     portType="tns:ptadmcpp" operation="oper.cd.info.op" inputVariable="vadm1"/>
              <invoke name="oper.carga.descarga.eto.info" partnerLink="pladmopp"
                     portType="tns:ptadmopp" operation="oper.cd.info.op" inputVariable="vadm1"/>
              <invoke name="oper.carga.descarga.eto.info" partnerLink="pladmpil"
                     portType="tns:ptadmopp" operation="oper.cd.info.op" inputVariable="vadm1"/>
              <switch name="oper.carga.descarga.aprovada">
                <case condition="oper.carga.descarga.aprovada?">
                  <sequence name="oper.carga.descarga">
                    <invoke name="oper.carga.descarga.eto" partnerLink="pladmalf"
                      portType="tns:ptadmalf" operation="oper.cd.eto.op" inputVariable="vadm1"/>
                    <invoke name="oper.carga.descarga.eto" partnerLink="pladmcpp"
                      portType="tns:ptadmcpp" operation="oper.cd.eto.op" inputVariable="vadm1"/>
                    <invoke name="oper.carga.descarga.eto" partnerLink="pladmopp"
                      portType="tns:ptadmopp" operation="oper.cd.eto.op" inputVariable="vadm1"/>
                    <invoke name="oper.carga.descarga.aprovada" partnerLink="pladmagn"
                  portType="tns:ptadmagn" operation="oper.cd.aprovada.op" inputVariable="vadm1"/>
                    <receive name="oper.carga.descarga.ato" partnerLink="ploppadm"
                      portType="tns:ptoppadm" operation="oper.cd.ato.op" variable="vadm1"/>
                    <invoke name="oper.carga.descarga.ato" partnerLink="pladmalf"
                      portType="tns:ptadmalf" operation="oper.cd.ato.op" inputVariable="vadm1"/>
                    <invoke name="oper.carga.descarga.ato" partnerLink="pladmcpp"
                      portType="tns:ptadmcpp" operation="oper.cd.ato.op" inputVariable="vadm1"/>
                    <invoke name="oper.carga.descarga.ato" partnerLink="pladmagn"
                      portType="tns:ptadmagn" operation="oper.cd.ato.op" inputVariable="vadm1"/>
                  </sequence>
                </case>
                <case condition="oper.carga.descarga.recusada?">
                  <sequence name="oper.carga.descarga.recusada">
                    <invoke name="oper.carga.descarga.recusada" partnerLink="pladmagn"
                  portType="tns:ptadmagn" operation="oper.cd.recusada.op" inputVariable="vadm1"/>
                    <invoke name="oper.carga.descarga.recusada" partnerLink="pladmalf"
                  portType="tns:ptadmalf" operation="oper.cd.recusada.op" inputVariable="vadm1"/>
                    <invoke name="oper.carga.descarga.recusada" partnerLink="pladmcpp"
                  portType="tns:ptadmcpp" operation="oper.cd.recusada.op" inputVariable="vadm1"/>
                    <invoke name="oper.carga.descarga.recusada" partnerLink="pladmopp"
                  portType="tns:ptadmopp" operation="oper.cd.recusada.op" inputVariable="vadm1"/>
```

```
                  </sequence>
                </case>
              </switch>
            </sequence>
          </onMessage>
          <onMessage partnerLink="plagnadm" portType="tns:ptagnadm" operation="oper.mov.info.op"
                     variable="vadm1">
            <sequence name="oper.movimentacao">
              <invoke name="oper.mov.etm.info" partnerLink="pladmalf" portType="tns:ptadmalf"
                      operation="oper.mov.info.op" inputVariable="vadm1"/>
              <invoke name="oper.mov.etm.info" partnerLink="pladmcpp" portType="tns:ptadmcpp"
                      operation="oper.mov.info.op" inputVariable="vadm1"/>
              <invoke name="oper.mov.etm.info" partnerLink="pladmopp" portType="tns:ptadmopp"
                      operation="oper.mov.info.op" inputVariable="vadm1"/>
              <invoke name="oper.mov.etm.info" partnerLink="pladmpil" portType="tns:ptadmpil"
                      operation="oper.mov.info.op" inputVariable="vadm1"/>
              <switch name="operacao.movimentacao.aprovada">
                <case condition="oper.movimentacao.aprovada?">
                  <sequence name="oper.movimentacao">
                    <invoke name="oper.movimentacao.etm" partnerLink="pladmalf"
                     portType="tns:ptadmalf" operation="oper.mov.etm.op" inputVariable="vadm1"/>
                    <invoke name="oper.movimentacao.etm" partnerLink="pladmcpp"
                     portType="tns:ptadmcpp" operation="oper.mov.etm.op" inputVariable="vadm1"/>
                    <invoke name="oper.movimentacao.etm" partnerLink="pladmopp"
                     portType="tns:ptadmopp" operation="oper.mov.etm.op" inputVariable="vadm1"/>
                    <invoke name="oper.movimentacao.etm" partnerLink="pladmpil"
                     portType="tns:ptadmpil" operation="oper.mov.etm.op" inputVariable="vadm1"/>
                    <invoke name="oper.movimentacao.aprovada" partnerLink="pladmagn"
                portType="tns:ptadmagn" operation="oper.mov.aprovada.op" inputVariable="vadm1"/>
                    <receive name="oper.movimentacao.stm" partnerLink="plpiladm"
                     portType="tns:ptpiladm" operation="oper.mov.stm.op" variable="vadm1"/>
                    <invoke name="oper.movimentacao.stm" partnerLink="pladmagn"
                     portType="tns:ptadmagn" operation="oper.mov.stm.op" inputVariable="vadm1"/>
                    <receive name="oper.movimentacao.ftm" partnerLink="plpiladm"
                     portType="tns:ptpiladm" operation="oper.mov.stm.op" variable="vadm1"/>
                    <invoke name="oper.movimentacao.ftm" partnerLink="pladmalf"
                     portType="tns:ptadmalf" operation="oper.mov.stm.op" inputVariable="vadm1"/>
                    <invoke name="oper.movimentacao.ftm" partnerLink="pladmcpp"
                     portType="tns:ptadmcpp" operation="oper.mov.stm.op" inputVariable="vadm1"/>
                    <invoke name="oper.movimentacao.ftm" partnerLink="pladmopp"
                     portType="tns:ptadmopp" operation="oper.mov.stm.op" inputVariable="vadm1"/>
                    <invoke name="oper.movimentacao.ftm" partnerLink="pladmagn"
                     portType="tns:ptadmagn" operation="oper.mov.stm.op" inputVariable="vadm1"/>
                  </sequence>
                </case>
                <case condition="oper.movimentacao.recusada?">
                  <sequence name="oper.movimentacao.recusada">
                    <invoke name="oper.movimentacao.recusada" partnerLink="pladmagn"
                     portType="tns:ptadmagn" operation="op-bc_send_bill" inputVariable="vadm1"/>
                    <invoke name="oper.mov.recusada" partnerLink="pladmalf"
                portType="tns:ptadmalf" operation="oper.mov.recusada.op" inputVariable="vadm1"/>
                    <invoke name="oper.mov.recusada" partnerLink="pladmcpp"
                portType="tns:ptadmcpp" operation="oper.mov.recusada.op" inputVariable="vadm1"/>
                    <invoke name="oper.mov.recusada" partnerLink="pladmopp"
                portType="tns:ptadmopp" operation="oper.mov.recusada.op" inputVariable="vadm1"/>
                    <invoke name="oper.mov.recusada" partnerLink="pladmpil"
                portType="tns:ptadmpil" operation="oper.mov.recusada.op" inputVariable="vadm1"/>
                  </sequence>
                </case>
              </switch>
            </sequence>
          </onMessage>
        </pick>
      </sequence>
    </onMessage>
    <onMessage partnerLink="plalfadm" portType="tns:ptalfadm"
               operation="ped.desembaraco.aprovado.op" variable="vadm1">
      <assign>
        <copy>
          <from expression="true"/>
          <to variable="sair"/>
        </copy>
      </assign>
    </onMessage>
  </pick>
</while>
<receive name="ped.desembaraco.aprovado" partnerLink="plalfadm" portType="tns:ptalfadm"
         operation="ped.desembaraco.aprovado.op" variable="vadm1"/>
```

```
          <receive name="desembaraco.etd" partnerLink="plcppadm" portType="tns:ptcppadm"
                  operation="desembaraco.etd.op" variable="vadm1"/>
          <invoke name="desembaraco.etd" partnerLink="pladmalf" portType="tns:ptadmalf"
                  operation="desembaraco.etd.op" inputVariable="vadm1"/>
          <invoke name="desembaraco.etd" partnerLink="pladmopp" portType="tns:ptadmopp"
                  operation="desembaraco.etd.op" inputVariable="vadm1"/>
          <invoke name="ped.desembaraco.aprovado" partnerLink="pladmagn" portType="tns:ptadmagn"
                  operation="ped.desembaraco.aprovado.op" inputVariable="vadm1"/>
          <invoke name="desembaraco.etd" partnerLink="pladmpil" portType="tns:ptadmpil"
                  operation="desembaraco.etd.op" inputVariable="vadm1"/>
          <receive name="navio.desacostado.std" partnerLink="plpiladm" portType="tns:ptpiladm"
                  operation="navio.desacostado.std.op" variable="vadm1"/>
          <invoke name="navio.desacostado.std" partnerLink="pladmagn" portType="tns:ptadmagn"
                  operation="navio.desacostado.std.op" inputVariable="vadm1"/>
          <receive name="navio.ao.largo.atd" partnerLink="plpiladm" portType="tns:ptpiladm"
                  operation="navio.ao.largo.atd.op" variable="vadm1"/>
          <invoke name="navio.ao.largo.atd" partnerLink="pladmcpp" portType="tns:ptadmcpp"
                  operation="navio.ao.largo.atd.op" inputVariable="vadm1"/>
          <invoke name="navio.ao.largo.atd" partnerLink="pladmalf" portType="tns:ptadmalf"
                  operation="navio.ao.largo.atd.op" inputVariable="vadm1"/>
          <invoke name="navio.ao.largo.atd" partnerLink="pladmopp" portType="tns:ptadmopp"
                  operation="navio.ao.largo.atd.op" inputVariable="vadm1"/>
          <invoke name="navio.ao.largo.atd" partnerLink="pladmagn" portType="tns:ptadmagn"
                  operation="navio.ao.largo.atd.op" inputVariable="vadm1"/>
        </sequence>
      </case>
      <case condition="escala.recusada?">
        <invoke name="escala.recusada" partnerLink="pladmagn" portType="tns:ptadmagn"
                operation="escala.recusada.op" inputVariable="vadm1"/>
      </case>
    </switch>
  </sequence>
</process>
```

Processo BPEL do Agente de Navegação (agn)

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario50" name="cprocess50-MaritimePort-agn"
         targetNamespace="http://meec.ist.utl.pt/cenario50" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="pladmagn" partnerLinkType="tns:pltadmagn" myRole="pltadmagn-ragn"/>
    <partnerLink name="plagnadm" partnerLinkType="tns:pltagnadm" partnerRole="pltagnadm-radm"/>
    <partnerLink name="plagnalf" partnerLinkType="tns:pltagnalf" partnerRole="pltagnalf-ralf"/>
    <partnerLink name="plagncpp" partnerLinkType="tns:pltagncpp" partnerRole="pltagncpp-rcpp"/>
    <partnerLink name="plalfagn" partnerLinkType="tns:pltalfagn" myRole="pltalfagn-ragn"/>
    <partnerLink name="plcppagn" partnerLinkType="tns:pltcppagn" myRole="pltcppagn-ragn"/>
    <partnerLink name="pl-start" partnerLinkType="tns:pltadmagn" myRole="pltadmagn-ragn"/>
  </partnerLinks>
  <partners>
    <partner name="adm">
      <partnerLink name="pladmagn"/>
      <partnerLink name="plagnadm"/>
    </partner>
    <partner name="alf">
      <partnerLink name="plagnalf"/>
      <partnerLink name="plalfagn"/>
    </partner>
    <partner name="cpp">
      <partnerLink name="plagncpp"/>
      <partnerLink name="plcppagn"/>
    </partner>
  </partners>
  <variables>
    <variable name="vagn1" messageType="tns:msg"/>
  </variables>
  <sequence name="main">
    <receive name="initiate-global-process" partnerLink="pl-start" portType="pl-pt" operation="initiate"
             variable="vagn1" createInstance="yes"/>
    <invoke name="ped.aut.escala" partnerLink="plagnadm" portType="tns:ptagnadm"
            operation="ped.aut.escala.op" inputVariable="vagn1"/>
```

```
<pick name="ped.aut.escala">
  <onMessage partnerLink="pladmagn" portType="tns:ptadmagn" operation="ped.aut.escala.aprovado.op"
             variable="vagn1">
    <sequence name="escala.aprovada">
      <while name="nova.eta" condition="nova.ETA?">
        <sequence name="nova.eta">
          <invoke name="nova.eta" partnerLink="plagnadm" portType="tns:ptagnadm"
                  operation="nova.eta.op" inputVariable="vagn1"/>
        </sequence>
      </while>
      <invoke name="navio.ao.largo" partnerLink="plagnadm" portType="tns:ptagnadm"
              operation="navio.ao.largo.op" inputVariable="vagn1"/>
      <receive name="navio.acostado.ata" partnerLink="pladmagn" portType="tns:ptadmagn"
               operation="navio.acostado.op" variable="vagn1"/>
      <while name="agn_maisOperPorFazer" condition="maisOperPorFazer?">
        <sequence name="nova.operacao">
          <invoke name="mais.oper.por.fazer" partnerLink="plagnadm" portType="tns:ptagnadm"
                  operation="mais.oper.op" inputVariable="vagn1"/>
          <switch name="CargaDescargaOuMovimentacao">
            <case condition="operacao.carga.descarga?">
              <sequence name="oper.carga.descarga">
                <invoke name="oper.carga.descarga.eto" partnerLink="plagnadm" portType="tns:ptagnadm"
                        operation="oper.cd.eto" inputVariable="vagn1"/>
                <pick name="oper.carga.descarga.aprovada">
                  <onMessage partnerLink="pladmagn" portType="tns:ptadmagn"
                             operation="oper.cd.aprovada.op" variable="vagn1">
                    <sequence name="oper.carga.descarga">
                      <receive name="oper.carga.descarga.ato" partnerLink="pladmagn"
                            portType="tns:ptadmagn" operation="oper.cd.ato.op" variable="vagn1"/>
                    </sequence>
                  </onMessage>
                  <onMessage partnerLink="pladmagn" portType="tns:ptadmagn"
                             operation="oper.cd.recusada.op" variable="vagn1">
                    <sequence name="oper.carga.descarga.recusada">
                      <empty/>
                    </sequence>
                  </onMessage>
                </pick>
              </sequence>
            </case>
            <case condition="operacao.movimentacao?">
              <sequence name="oper.movimentacao">
                <invoke name="oper.movimentacao.etm" partnerLink="plagnadm" portType="tns:ptagnadm"
                        operation="oper.mov.info.op" inputVariable="vagn1"/>
                <pick name="operacao.movimentacao.aprovada">
                  <onMessage partnerLink="pladmagn" portType="tns:ptadmagn"
                             operation="oper.mov.aprovada.op" variable="vagn1">
                    <sequence name="oper.movimentacao">
                      <receive name="oper.movimentacao.stm" partnerLink="pladmagn"
                           portType="tns:ptadmagn" operation="oper.mov.stm.op" variable="vagn1"/>
                      <receive name="oper.movimentacao.ftm" partnerLink="pladmagn"
                           portType="tns:ptadmagn" operation="oper.mov.stm.op" variable="vagn1"/>
                    </sequence>
                  </onMessage>
                  <onMessage partnerLink="pladmagn" portType="tns:ptadmagn"
                             operation="op-bc_send_bill" variable="vagn1">
                    <sequence name="oper.movimentacao.recusada">
                      <empty/>
                    </sequence>
                  </onMessage>
                </pick>
              </sequence>
            </case>
          </switch>
        </sequence>
      </while>
      <invoke name="ped.desembaraco" partnerLink="plagnalf" portType="tns:ptagnalf"
              operation="ped.desembaraco.op" inputVariable="vagn1"/>
      <assign>
        <copy>
          <from expression="false"/>
          <to variable="sair"/>
        </copy>
      </assign>
      <while name="alf.desembaraco.rec" condition="sair==False">
        <pick>
          <onMessage partnerLink="plalfagn" portType="tns:ptalfagn"
                     operation="ped.desembaraco.recusado.op" variable="vagn1">
```

```
                <sequence name="alf.recusa.desembaraco">
                  <invoke name="ped.desembaraco" partnerLink="plagnalf" portType="tns:ptagnalf"
                          operation="ped.desembaraco.op" inputVariable="vagn1"/>
                </sequence>
              </onMessage>
              <onMessage partnerLink="plalfagn" portType="tns:ptalfagn"
                         operation="ped.desembaraco.aprovado.op" variable="vagn1">
                <assign>
                  <copy>
                    <from expression="true"/>
                    <to variable="sair"/>
                  </copy>
                </assign>
              </onMessage>
            </pick>
          </while>
          <receive name="ped.desembaraco.aprovado" partnerLink="plalfagn" portType="tns:ptalfagn"
                   operation="ped.desembaraco.aprovado.op" variable="vagn1"/>
          <invoke name="ped.desembaraco.etd" partnerLink="plagncpp" portType="tns:ptagncpp"
                  operation="ped.desembaraco.etd.op" inputVariable="vagn1"/>
          <assign>
            <copy>
              <from expression="false"/>
              <to variable="sair"/>
            </copy>
          </assign>
          <while name="cpp.desembaraco.rec" condition="sair==False">
            <pick>
              <onMessage partnerLink="plcppagn" portType="tns:ptcppagn"
                         operation="ped.desembaraco.recusado.op" variable="vagn1">
                <sequence name="cpp.recusa.desembaraco">
                  <invoke name="ped.desembaraco.etd" partnerLink="plagncpp" portType="tns:ptagncpp"
                          operation="ped.desembaraco.etd.op" inputVariable="vagn1"/>
                </sequence>
              </onMessage>
              <onMessage partnerLink="pladmagn" portType="tns:ptadmagn"
                         operation="ped.desembaraco.aprovado.op" variable="vagn1">
                <assign>
                  <copy>
                    <from expression="true"/>
                    <to variable="sair"/>
                  </copy>
                </assign>
              </onMessage>
            </pick>
          </while>
          <receive name="ped.desembaraco.aprovado" partnerLink="pladmagn" portType="tns:ptadmagn"
                   operation="ped.desembaraco.aprovado.op" variable="vagn1"/>
          <receive name="navio.desacostado.std" partnerLink="pladmagn" portType="tns:ptadmagn"
                   operation="navio.desacostado.std.op" variable="vagn1"/>
          <receive name="navio.ao.largo.atd" partnerLink="pladmagn" portType="tns:ptadmagn"
                   operation="navio.ao.largo.atd.op" variable="vagn1"/>
        </sequence>
      </onMessage>
      <onMessage partnerLink="pladmagn" portType="tns:ptadmagn" operation="escala.recusada.op"
                 variable="vagn1">
        <empty/>
      </onMessage>
    </pick>
  </sequence>
</process>
```

Processo BPEL da Alfândega (alf)

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario50" name="cprocess50-MaritimePort-alf"
         targetNamespace="http://meec.ist.utl.pt/cenario50" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="pladmalf" partnerLinkType="tns:pltadmalf" myRole="pltadmalf-ralf"/>
    <partnerLink name="plagnalf" partnerLinkType="tns:pltagnalf" myRole="pltagnalf-ralf"/>
    <partnerLink name="plalfadm" partnerLinkType="tns:pltalfadm" partnerRole="pltalfadm-radm"/>
    <partnerLink name="plalfagn" partnerLinkType="tns:pltalfagn" partnerRole="pltalfagn-ragn"/>
```

```
  <partnerLink name="plalfcpp" partnerLinkType="tns:pltalfcpp" partnerRole="pltalfcpp-rcpp"/>
</partnerLinks>
<partners>
  <partner name="adm">
    <partnerLink name="pladmalf"/>
    <partnerLink name="plalfadm"/>
  </partner>
  <partner name="agn">
    <partnerLink name="plagnalf"/>
    <partnerLink name="plalfagn"/>
  </partner>
  <partner name="cpp">
    <partnerLink name="plalfcpp"/>
  </partner>
</partners>
<variables>
  <variable name="valf1" messageType="tns:msg"/>
</variables>
<sequence name="escala.aprovada">
  <receive name="nova.escala" partnerLink="pladmalf" portType="tns:ptadmalf" operation="nova.escala.op"
           variable="valf1" createInstance="yes"/>
  <assign>
    <copy>
      <from expression="false"/>
      <to variable="sair"/>
    </copy>
  </assign>
  <while name="nova.eta" condition="sair==False">
    <pick>
      <onMessage partnerLink="pladmalf" portType="tns:ptadmalf" operation="nova.eta.op"
                 variable="valf1">
        <sequence name="nova.eta">
          <empty/>
        </sequence>
      </onMessage>
      <onMessage partnerLink="pladmalf" portType="tns:ptadmalf" operation="navio.ao.largo.op"
                 variable="valf1">
        <assign>
          <copy>
            <from expression="true"/>
            <to variable="sair"/>
          </copy>
        </assign>
      </onMessage>
    </pick>
  </while>
  <receive name="navio.ao.largo" partnerLink="pladmalf" portType="tns:ptadmalf"
           operation="navio.ao.largo.op" variable="valf1"/>
  <receive name="navio.acostado.ata" partnerLink="pladmalf" portType="tns:ptadmalf"
           operation="navio.acostado.op" variable="valf1"/>
  <assign>
    <copy>
      <from expression="false"/>
      <to variable="sair"/>
    </copy>
  </assign>
  <while name="agn_maisOperPorFazer" condition="sair==False">
    <pick>
      <onMessage partnerLink="pladmalf" portType="tns:ptadmalf" operation="mais.oper.op"
                 variable="valf1">
        <sequence name="nova.operacao">
          <pick name="CargaDescargaOuMovimentacao">
            <onMessage partnerLink="pladmalf" portType="tns:ptadmalf" operation="oper.cd.info.op"
                       variable="valf1">
              <sequence name="oper.carga.descarga">
                <pick name="oper.carga.descarga.aprovada">
                  <onMessage partnerLink="pladmalf" portType="tns:ptadmalf" operation="oper.cd.eto.op"
                             variable="valf1">
                    <sequence name="oper.carga.descarga">
                      <receive name="oper.carga.descarga.ato" partnerLink="pladmalf"
                               portType="tns:ptadmalf" operation="oper.cd.ato.op" variable="valf1"/>
                    </sequence>
                  </onMessage>
                  <onMessage partnerLink="pladmalf" portType="tns:ptadmalf"
                             operation="oper.cd.recusada.op" variable="valf1">
                    <sequence name="oper.carga.descarga.recusada">
                      <empty/>
                    </sequence>
```

```
                  </onMessage>
                </pick>
              </sequence>
            </onMessage>
            <onMessage partnerLink="pladmalf" portType="tns:ptadmalf" operation="oper.mov.info.op"
                       variable="valf1">
              <sequence name="oper.movimentacao">
                <pick name="operacao.movimentacao.aprovada">
                  <onMessage partnerLink="pladmalf" portType="tns:ptadmalf" operation="oper.mov.etm.op"
                             variable="valf1">
                    <sequence name="oper.movimentacao">
                      <receive name="oper.movimentacao.ftm" partnerLink="pladmalf"
                               portType="tns:ptadmalf" operation="oper.mov.stm.op" variable="valf1"/>
                    </sequence>
                  </onMessage>
                  <onMessage partnerLink="pladmalf" portType="tns:ptadmalf"
                             operation="oper.mov.recusada.op" variable="valf1">
                    <sequence name="oper.movimentacao.recusada">
                      <empty/>
                    </sequence>
                  </onMessage>
                </pick>
              </sequence>
            </onMessage>
          </pick>
        </sequence>
      </onMessage>
      <onMessage partnerLink="plagnalf" portType="tns:ptagnalf" operation="ped.desembaraco.op"
                 variable="valf1">
        <assign>
          <copy>
            <from expression="true"/>
            <to variable="sair"/>
          </copy>
        </assign>
      </onMessage>
    </pick>
  </while>
  <receive name="ped.desembaraco" partnerLink="plagnalf" portType="tns:ptagnalf"
           operation="ped.desembaraco.op" variable="valf1"/>
  <while name="alf.desembaraco.rec" condition="alf.desembaraco.recusado?">
    <sequence name="alf.recusa.desembaraco">
      <invoke name="ped.desembaraco.recusado" partnerLink="plalfagn" portType="tns:ptalfagn"
              operation="ped.desembaraco.recusado.op" inputVariable="valf1"/>
      <receive name="ped.desembaraco" partnerLink="plagnalf" portType="tns:ptagnalf"
               operation="ped.desembaraco.op" variable="valf1"/>
    </sequence>
  </while>
  <invoke name="ped.desembaraco.aprovado" partnerLink="plalfadm" portType="tns:ptalfadm"
          operation="ped.desembaraco.aprovado.op" inputVariable="valf1"/>
  <invoke name="ped.desembaraco.aprovado" partnerLink="plalfcpp" portType="tns:ptalfcpp"
          operation="ped.desembaraco.aprovado.op" inputVariable="valf1"/>
  <invoke name="ped.desembaraco.aprovado" partnerLink="plalfagn" portType="tns:ptalfagn"
          operation="ped.desembaraco.aprovado.op" inputVariable="valf1"/>
  <receive name="desembaraco.etd" partnerLink="pladmalf" portType="tns:ptadmalf"
          operation="desembaraco.etd.op" variable="valf1"/>
  <receive name="navio.ao.largo.atd" partnerLink="pladmalf" portType="tns:ptadmalf"
          operation="navio.ao.largo.atd.op" variable="valf1"/>
 </sequence>
</process>
```

Processo BPEL da Capitania do Porto (cpp)

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario50" name="cprocess50-MaritimePort-cpp"
         targetNamespace="http://meec.ist.utl.pt/cenario50" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="pladmcpp" partnerLinkType="tns:pltadmcpp" myRole="pltadmcpp-rcpp"/>
    <partnerLink name="plagncpp" partnerLinkType="tns:pltagncpp" myRole="pltagncpp-rcpp"/>
    <partnerLink name="plalfcpp" partnerLinkType="tns:pltalfcpp" myRole="pltalfcpp-rcpp"/>
    <partnerLink name="plcppadm" partnerLinkType="tns:pltcppadm" partnerRole="pltcppadm-radm"/>
    <partnerLink name="plcppagn" partnerLinkType="tns:pltcppagn" partnerRole="pltcppagn-ragn"/>
```

```
</partnerLinks>
<partners>
  <partner name="adm">
    <partnerLink name="pladmcpp"/>
    <partnerLink name="plcppadm"/>
  </partner>
  <partner name="agn">
    <partnerLink name="plagncpp"/>
    <partnerLink name="plcppagn"/>
  </partner>
  <partner name="alf">
    <partnerLink name="plalfcpp"/>
  </partner>
</partners>
<variables>
  <variable name="vcpp1" messageType="tns:msg"/>
</variables>
<sequence name="escala.aprovada">
  <receive name="nova.escala" partnerLink="pladmcpp" portType="tns:ptadmcpp" operation="nova.escala.op"
           variable="vcpp1" createInstance="yes"/>
  <assign>
    <copy>
      <from expression="false"/>
      <to variable="sair"/>
    </copy>
  </assign>
  <while name="nova.eta" condition="sair==False">
    <pick>
      <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp" operation="nova.eta.op"
                 variable="vcpp1">
        <sequence name="nova.eta">
          <empty>
        </sequence>
      </onMessage>
      <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp" operation="navio.ao.largo.op"
                 variable="vcpp1">
        <assign>
          <copy>
            <from expression="true"/>
            <to variable="sair"/>
          </copy>
        </assign>
      </onMessage>
    </pick>
  </while>
  <receive name="navio.ao.largo" partnerLink="pladmcpp" portType="tns:ptadmcpp"
           operation="navio.ao.largo.op" variable="vcpp1"/>
  <receive name="navio.acostado.ata" partnerLink="pladmcpp" portType="tns:ptadmcpp"
           operation="navio.acostado.op" variable="vcpp1"/>
  <assign>
    <copy>
      <from expression="false"/>
      <to variable="sair"/>
    </copy>
  </assign>
  <while name="agn_maisOperPorFazer" condition="sair==False">
    <pick>
      <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp" operation="mais.oper.op"
                 variable="vcpp1">
        <sequence name="nova.operacao">
          <pick name="CargaDescargaOuMovimentacao">
            <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp" operation="oper.cd.info.op"
                       variable="vcpp1">
              <sequence name="oper.carga.descarga">
                <pick name="oper.carga.descarga.aprovada">
                  <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp" operation="oper.cd.eto.op"
                             variable="vcpp1">
                    <sequence name="oper.carga.descarga">
                      <receive name="oper.carga.descarga.ato" partnerLink="pladmcpp"
                               portType="tns:ptadmcpp" operation="oper.cd.ato.op" variable="vcpp1"/>
                    </sequence>
                  </onMessage>
                  <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp"
                             operation="oper.cd.recusada.op" variable="vcpp1">
                    <sequence name="oper.carga.descarga.recusada">
                      <empty/>
                    </sequence>
                  </onMessage>
```

```
                </pick>
              </sequence>
            </onMessage>
            <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp" operation="oper.mov.info.op"
                       variable="vcpp1">
              <sequence name="oper.movimentacao">
                <pick name="operacao.movimentacao.aprovada">
                  <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp" operation="oper.mov.etm.op"
                             variable="vcpp1">
                    <sequence name="oper.movimentacao">
                      <receive name="oper.movimentacao.ftm" partnerLink="pladmcpp"
                               portType="tns:ptadmcpp" operation="oper.mov.stm.op" variable="vcpp1"/>
                    </sequence>
                  </onMessage>
                  <onMessage partnerLink="pladmcpp" portType="tns:ptadmcpp"
                             operation="oper.mov.recusada.op" variable="vcpp1">
                    <sequence name="oper.movimentacao.recusada">
                      <empty/>
                    </sequence>
                  </onMessage>
                </pick>
              </sequence>
            </onMessage>
          </pick>
        </sequence>
      </onMessage>
      <onMessage partnerLink="plalfcpp" portType="tns:ptalfcpp" operation="ped.desembaraco.aprovado.op"
                 variable="vcpp1">
        <assign>
          <copy>
            <from expression="true"/>
            <to variable="sair"/>
          </copy>
        </assign>
      </onMessage>
    </pick>
  </while>
  <receive name="ped.desembaraco.aprovado" partnerLink="plalfcpp" portType="tns:ptalfcpp"
           operation="ped.desembaraco.aprovado.op" variable="vcpp1"/>
  <receive name="ped.desembaraco.etd" partnerLink="plagncpp" portType="tns:ptagncpp"
           operation="ped.desembaraco.etd.op" variable="vcpp1"/>
  <while name="cpp.desembaraco.rec" condition="cpp.desembaraco.recusado?">
    <sequence name="cpp.recusa.desembaraco">
      <invoke name="ped.desembaraco.recusado" partnerLink="plcppagn" portType="tns:ptcppagn"
              operation="ped.desembaraco.recusado.op" inputVariable="vcpp1"/>
      <receive name="ped.desembaraco.etd" partnerLink="plagncpp" portType="tns:ptagncpp"
               operation="ped.desembaraco.etd.op" variable="vcpp1"/>
    </sequence>
  </while>
  <invoke name="desembaraco.etd" partnerLink="plcppadm" portType="tns:ptcppadm"
          operation="desembaraco.etd.op" inputVariable="vcpp1"/>
  <receive name="navio.ao.largo.atd" partnerLink="pladmcpp" portType="tns:ptadmcpp"
           operation="navio.ao.largo.atd.op" variable="vcpp1"/>
 </sequence>
</process>
```

Processo BPEL dos Operadores Portuários (opp)

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario50" name="cprocess50-MaritimePort-opp"
         targetNamespace="http://meec.ist.utl.pt/cenario50" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="pladmopp" partnerLinkType="tns:pltadmopp" myRole="pltadmopp-ropp"/>
    <partnerLink name="ploppadm" partnerLinkType="tns:pltoppadm" partnerRole="pltoppadm-radm"/>
  </partnerLinks>
  <partners>
    <partner name="adm">
      <partnerLink name="pladmopp"/>
      <partnerLink name="ploppadm"/>
    </partner>
  </partners>
  <variables>
```

```
  <variable name="vopp1" messageType="tns:msg"/>
</variables>
<sequence name="escala.aprovada">
  <receive name="nova.escala" partnerLink="pladmopp" portType="tns:ptadmopp" operation="nova.escala.op"
           variable="vopp1" createInstance="yes"/>
  <assign>
    <copy>
      <from expression="false"/>
      <to variable="sair"/>
    </copy>
  </assign>
  <while name="nova.eta" condition="sair==False">
    <pick>
      <onMessage partnerLink="pladmopp" portType="tns:ptadmopp" operation="nova.eta.op"
                 variable="vopp1">
        <sequence name="nova.eta">
          <empty/>
        </sequence>
      </onMessage>
      <onMessage partnerLink="pladmopp" portType="tns:ptadmopp" operation="navio.ao.largo.op"
                 variable="vopp1">
        <assign>
          <copy>
            <from expression="true"/>
            <to variable="sair"/>
          </copy>
        </assign>
      </onMessage>
    </pick>
  </while>
  <receive name="navio.ao.largo" partnerLink="pladmopp" portType="tns:ptadmopp"
           operation="navio.ao.largo.op" variable="vopp1"/>
  <receive name="navio.acostado.ata" partnerLink="pladmopp" portType="tns:ptadmopp"
           operation="navio.acostado.op" variable="vopp1"/>
  <assign>
    <copy>
      <from expression="false"/>
      <to variable="sair"/>
    </copy>
  </assign>
  <while name="agn_maisOperPorFazer" condition="sair==False">
    <pick>
      <onMessage partnerLink="pladmopp" portType="tns:ptadmopp" operation="mais.oper.op"
                 variable="vopp1">
        <sequence name="nova.operacao">
          <pick name="CargaDescargaOuMovimentacao">
            <onMessage partnerLink="pladmopp" portType="tns:ptadmopp" operation="oper.cd.info.op"
                       variable="vopp1">
              <sequence name="oper.carga.descarga">
                <pick name="oper.carga.descarga.aprovada">
                  <onMessage partnerLink="pladmopp" portType="tns:ptadmopp" operation="oper.cd.eto.op"
                             variable="vopp1">
                    <sequence name="oper.carga.descarga">
                      <invoke name="oper.carga.descarga.ato" partnerLink="ploppadm"
                              portType="tns:ptoppadm" operation="oper.cd.ato.op" inputVariable="vopp1"/>
                    </sequence>
                  </onMessage>
                  <onMessage partnerLink="pladmopp" portType="tns:ptadmopp"
                             operation="oper.cd.recusada.op" variable="vopp1">
                    <sequence name="oper.carga.descarga.recusada">
                      <empty/>
                    </sequence>
                  </onMessage>
                </pick>
              </sequence>
            </onMessage>
            <onMessage partnerLink="pladmopp" portType="tns:ptadmopp" operation="oper.mov.info.op"
                       variable="vopp1">
              <sequence name="oper.movimentacao">
                <pick name="operacao.movimentacao.aprovada">
                  <onMessage partnerLink="pladmopp" portType="tns:ptadmopp" operation="oper.mov.etm.op"
                             variable="vopp1">
                    <sequence name="oper.movimentacao">
                      <receive name="oper.movimentacao.ftm" partnerLink="pladmopp"
                               portType="tns:ptadmopp" operation="oper.mov.stm.op" variable="vopp1"/>
                    </sequence>
                  </onMessage>
                  <onMessage partnerLink="pladmopp" portType="tns:ptadmopp"
```

```
                                operation="oper.mov.recusada.op" variable="vopp1">
                        <sequence name="oper.movimentacao.recusada">
                          <empty/>
                        </sequence>
                      </onMessage>
                    </pick>
                  </sequence>
                </onMessage>
              </pick>
            </sequence>
          </onMessage>
          <onMessage partnerLink="pladmopp" portType="tns:ptadmopp" operation="desembaraco.etd.op"
                     variable="vopp1">
            <assign>
              <copy>
                <from expression="true"/>
                <to variable="sair"/>
              </copy>
            </assign>
          </onMessage>
        </pick>
      </while>
      <receive name="desembaraco.etd" partnerLink="pladmopp" portType="tns:ptadmopp"
               operation="desembaraco.etd.op" variable="vopp1"/>
      <receive name="navio.ao.largo.atd" partnerLink="pladmopp" portType="tns:ptadmopp"
               operation="navio.ao.largo.atd.op" variable="vopp1"/>
    </sequence>
</process>
```

Processo BPEL dos Pilotos (pil)

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario50" name="cprocess50-MaritimePort-pil"
         targetNamespace="http://meec.ist.utl.pt/cenario50" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="pladmpil" partnerLinkType="tns:pltadmpil" myRole="pltadmpil-rpil"/>
    <partnerLink name="plpiladm" partnerLinkType="tns:pltpiladm" partnerRole="pltpiladm-radm"/>
  </partnerLinks>
  <partners>
    <partner name="adm">
      <partnerLink name="pladmpil"/>
      <partnerLink name="plpiladm"/>
    </partner>
  </partners>
  <variables>
    <variable name="vpil1" messageType="tns:msg"/>
  </variables>
  <sequence name="escala.aprovada">
    <receive name="nova.escala" partnerLink="pladmpil" portType="tns:ptadmpil" operation="nova.escala.op"
             variable="vpil1" createInstance="yes"/>
    <assign>
      <copy>
        <from expression="false"/>
        <to variable="sair"/>
      </copy>
    </assign>
    <while name="nova.eta" condition="sair==False">
      <pick>
        <onMessage partnerLink="pladmpil" portType="tns:ptadmpil" operation="nova.eta.op"
                   variable="vpil1">
          <sequence name="nova.eta">
            <empty/>
          </sequence>
        </onMessage>
        <onMessage partnerLink="pladmpil" portType="tns:ptadmpil" operation="navio.ao.largo.op"
                   variable="vpil1">
          <assign>
            <copy>
              <from expression="true"/>
              <to variable="sair"/>
            </copy>
          </assign>
```

```
        </onMessage>
      </pick>
    </while>
    <receive name="navio.ao.largo" partnerLink="pladmpil" portType="tns:ptadmpil"
             operation="navio.ao.largo.op" variable="vpil1"/>
    <invoke name="navio.acostado.ata" partnerLink="plpiladm" portType="tns:ptpiladm"
            operation="navio.acostado.op" inputVariable="vpil1"/>
    <assign>
      <copy>
        <from expression="false"/>
        <to variable="sair"/>
      </copy>
    </assign>
    <while name="agn_maisOperPorFazer" condition="sair==False">
      <pick>
        <onMessage partnerLink="pladmpil" portType="tns:ptadmpil" operation="mais.oper.op"
                   variable="vpil1">
          <sequence name="nova.operacao">
            <pick name="CargaDescargaOuMovimentacao">
              <onMessage partnerLink="pladmpil" portType="tns:ptadmopp" operation="oper.cd.info.op"
                         variable="vpil1">
                <sequence name="oper.carga.descarga">
                  <empty/>
                </sequence>
              </onMessage>
              <onMessage partnerLink="pladmpil" portType="tns:ptadmpil" operation="oper.mov.info.op"
                         variable="vpil1">
                <sequence name="oper.movimentacao">
                  <pick name="operacao.movimentacao.aprovada">
                    <onMessage partnerLink="pladmpil" portType="tns:ptadmpil" operation="oper.mov.etm.op"
                               variable="vpil1">
                      <sequence name="oper.movimentacao">
                        <invoke name="oper.movimentacao.stm" partnerLink="plpiladm"
                                portType="tns:ptpiladm" operation="oper.mov.stm.op" inputVariable="vpil1"/>
                        <invoke name="oper.movimentacao.ftm" partnerLink="plpiladm"
                                portType="tns:ptpiladm" operation="oper.mov.stm.op" inputVariable="vpil1"/>
                      </sequence>
                    </onMessage>
                    <onMessage partnerLink="pladmpil" portType="tns:ptadmpil"
                               operation="oper.mov.recusada.op" variable="vpil1">
                      <sequence name="oper.movimentacao.recusada">
                        <empty/>
                      </sequence>
                    </onMessage>
                  </pick>
                </sequence>
              </onMessage>
            </pick>
          </sequence>
        </onMessage>
        <onMessage partnerLink="pladmpil" portType="tns:ptadmpil" operation="desembaraco.etd.op"
                   variable="vpil1">
          <assign>
            <copy>
              <from expression="true"/>
              <to variable="sair"/>
            </copy>
          </assign>
        </onMessage>
      </pick>
    </while>
    <receive name="desembaraco.etd" partnerLink="pladmpil" portType="tns:ptadmpil"
             operation="desembaraco.etd.op" variable="vpil1"/>
    <invoke name="navio.desacostado.std" partnerLink="plpiladm" portType="tns:ptpiladm"
            operation="navio.desacostado.std.op" inputVariable="vpil1"/>
    <invoke name="navio.ao.largo.atd" partnerLink="plpiladm" portType="tns:ptpiladm"
            operation="navio.ao.largo.atd.op" inputVariable="vpil1"/>
  </sequence>
</process>
```

Anexo E: Codificação XML do Cenário da Livraria

Este anexo apresenta as listagens completas relativas ao cenário da livraria electrónica. O cenário encontra-se descrito na secção 6.2 e primeiramente são apresentadas as listagens dos vários processos existentes na versão sem *scopes* e que constam na subsecção 6.2.3, sendo depois apresentadas as listagens dos vários processos existentes na versão com *scopes* e que constam na subsecção 6.2.4.

A informação a ser transportada pelas mensagens segue o mecanismo indicado no anexo anterior.

Os vários processos têm a seguinte localização:

Cenário da Bookstore sem scopes

Processo CBPEL do cenário da Bookstore sem scopes

```
<commonProcess name="cprocess51-bookstore"
      xmlns="http://schemas.xmlsoap.org/ws/2005/05/common-business-process/"
      targetNamespace="http://meec.ist.utl.pt/cenario51" xmlns:tns="http://meec.ist.utl.pt/cenario51">

   <!-- ##### partnerLinks ### -->
   <partnerLinks>

      <partnerLink name="plcb" partnerLinkType="tns:pltcb">
         <role name="rpc" partnerLinkTypeRoleName="pltrncb-c" />
         <role name="rpb" partnerLinkTypeRoleName="pltrncb-b" />
      </partnerLink>

      <partnerLink name="plcs" partnerLinkType="tns:pltcs">
         <role name="rpc" partnerLinkTypeRoleName="pltrncs-c" />
         <role name="rps" partnerLinkTypeRoleName="pltrncs-s" />
      </partnerLink>

      <partnerLink name="plbp" partnerLinkType="tns:pltbp">
         <role name="rpb" partnerLinkTypeRoleName="pltrnbp-b" />
         <role name="rpp" partnerLinkTypeRoleName="pltrnbp-p" />
      </partnerLink>

      <partnerLink name="plps" partnerLinkType="tns:pltps">
         <role name="rpp" partnerLinkTypeRoleName="pltrnps-p" />
         <role name="rps" partnerLinkTypeRoleName="pltrn4ps-s" />
      </partnerLink>

      <partnerLink name="plbs" partnerLinkType="tns:pltbs">
         <role name="rpb" partnerLinkTypeRoleName="pltrnbs-b" />
         <role name="rps" partnerLinkTypeRoleName="pltrnbs-s" />
      </partnerLink>

   </partnerLinks>


   <!-- ##### partners #### -->
   <partners>

      <partner name="Customer">
         <partnerLink name="plcb" role="rpc" />
         <partnerLink name="plcs" role="rpc" />
      </partner>

      <partner name="Bookstore">
         <partnerLink name="plcb" role="rpb" />
         <partnerLink name="plbp" role="rpb" />
         <partnerLink name="plbs" role="rpb" />
      </partner>

      <partner name="Publisher">
         <partnerLink name="plbp" role="rpp" />
         <partnerLink name="plps" role="rpp" />
      </partner>

      <partner name="Shipper">
         <partnerLink name="plbs" role="rps" />
         <partnerLink name="plps" role="rps" />
         <partnerLink name="plcs" role="rps" />
      </partner>

   </partners>

   <!-- ##### initiator ### -->
   <initiator partner="Customer" initialSend="send-bc_order" />
```

```
<!-- ##### variables ### -->
<variables>
   <variable name="vc1" messageType="tns:msgOrder" owner="Customer" />
   <variable name="vb1" messageType="tns:msgOrder" owner="Bookstore" />
   <variable name="vp1" messageType="tns:msgOrder" owner="Publisher" />
   <variable name="vs1" messageType="tns:msgOrder" owner="Shipper" />
</variables>

<!-- ##### activity ### -->
<sequence name="main">

   <send name="send-bc_order" partnerLink="plcb" portType="tns:ptcb"
      operation="op-bc_order" fromPartner="Customer" toPartner="Bookstore"
      inputVariable="vc1" outputVariable="vb1" createInstance="yes" />

   <send name="send-bp_order" partnerLink="plbp" portType="tns:ptbp"
      operation="op-bp_order" fromPartner="Bookstore" toPartner="Publisher"
      inputVariable="vb1" outputVariable="vp1" createInstance="yes" />

   <switch name="p_switch" executer="Publisher">
      <case condition="p_confirm">
         <sequence name="publisher-confirm">
            <send name="send-pb_confirm" partnerLink="plbp" portType="tns:ptbp"
               operation="op-pb_confirm" fromPartner="Publisher" toPartner="Bookstore"
               inputVariable="vp1" outputVariable="vb1" />

            <send name="send-bc_confirm" partnerLink="plcb" portType="tns:ptbc"
               operation="op-bc_confirm" fromPartner="Bookstore" toPartner="Customer"
               inputVariable="vb1" outputVariable="vc1" />

            <send name="send-bs_req_shipment" partnerLink="plbs" portType="tns:ptbs"
               operation="op-bs_req_shipment" fromPartner="Bookstore" toPartner="Shipper"
               inputVariable="vb1" outputVariable="vs1" createInstance="yes" />

            <switch name="s_switch" executer="Shipper">

               <case condition="s_confirm">
                  <sequence name="shipper-confirm">
                     <send name="send-sb_confirm" partnerLink="plbs" portType="tns:ptsb"
                        operation="op-sb_confirm" fromPartner="Shipper" toPartner="Bookstore"
                        inputVariable="vs1" outputVariable="vb1" />

                     <send name="send-bp_ship_info" partnerLink="plbp" portType="tns:ptbp"
                        operation="op-bp_ship_info" fromPartner="Bookstore" toPartner="Publisher"
                        inputVariable="vb1" outputVariable="vp1" />

                     <send name="send-ps_send_book" partnerLink="plps" portType="tns:ptbs"
                        operation="op-ps_send_book" fromPartner="Publisher" toPartner="Shipper"
                        inputVariable="vp1" outputVariable="vs1" />

                     <send name="send-sc_ship_book" partnerLink="plcs" portType="tns:ptsc"
                        operation="op-sc_ship_book" fromPartner="Shipper" toPartner="Customer"
                        inputVariable="vs1" outputVariable="vc1" />

                     <send name="send-sb_notify" partnerLink="plbs" portType="tns:ptsb"
                        operation="op-sb_notify" fromPartner="Shipper" toPartner="Bookstore"
                        inputVariable="vs1" outputVariable="vb1" />

                     <send name="send-bc_send_bill" partnerLink="plcb" portType="tns:ptbc"
                        operation="op-bc_send_bill" fromPartner="Bookstore" toPartner="Customer"
                        inputVariable="vb1" outputVariable="vc1" />

                     <send name="send-cb_payment" partnerLink="plcb" portType="tns:ptcb"
                        operation="op-cb_payment" fromPartner="Customer" toPartner="Bookstore"
                        inputVariable="vc1" outputVariable="vb1" />
                  </sequence>
               </case>

               <case condition="s_decline">
                  <sequence name="shipper-decline">
                     <send name="send-sb_decline" partnerLink="plbs" portType="tns:ptsb"
                        operation="op-sb_decline" fromPartner="Shipper" toPartner="Bookstore"
                        inputVariable="vs1" outputVariable="vb1" />

                     <send name="send-bp_cancel" partnerLink="plbp" portType="tns:ptbp"
                        operation="op-bp_cancel" fromPartner="Bookstore" toPartner="Publisher"
                        inputVariable="vb1" outputVariable="vp1" />
```

```
                        <send name="send-bc_decline2" partnerLink="plcb" portType="tns:ptbc"
                           operation="op-bc_decline2" fromPartner="Bookstore" toPartner="Customer"
                           inputVariable="vb1" outputVariable="vc1" />
                     </sequence>
                  </case>
               </switch>

            </sequence>
         </case>

         <case condition="p_decline">
            <sequence name="publisher-decline">

               <send name="send-pb_decline" partnerLink="plbp" portType="tns:ptbp"
                  operation="op-pb_decline" fromPartner="Publisher" toPartner="Bookstore"
                  inputVariable="vp1" outputVariable="vb1" />

               <send name="send-bc_decline1" partnerLink="plcb" portType="tns:ptbc"
                  operation="op-bc_decline1" fromPartner="Bookstore" toPartner="Customer"
                  inputVariable="vb1" outputVariable="vc1" />

            </sequence>
         </case>
      </switch>

   </sequence>

</commonProcess>
```

Processo BPEL da Bookstore

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario51" name="cprocess51-bookstore-Bookstore"
         targetNamespace="http://meec.ist.utl.pt/cenario51" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="plbp" partnerLinkType="tns:pltbp" myRole="pltrnbp-b" partnerRole="pltrnbp-p"/>
    <partnerLink name="plbs" partnerLinkType="tns:pltbs" myRole="pltrnbs-b" partnerRole="pltrnbs-s"/>
    <partnerLink name="plcb" partnerLinkType="tns:pltcb" myRole="pltrncb-b" partnerRole="pltrncb-c"/>
  </partnerLinks>
  <partners>
    <partner name="Customer">
      <partnerLink name="plcb"/>
    </partner>
    <partner name="Publisher">
      <partnerLink name="plbp"/>
    </partner>
    <partner name="Shipper">
      <partnerLink name="plbs"/>
    </partner>
  </partners>
  <variables>
    <variable name="vb1" messageType="tns:msgOrder"/>
  </variables>
  <sequence name="main">
    <receive name="send-bc_order" partnerLink="plcb" portType="tns:ptcb" operation="op-bc_order"
             variable="vb1" createInstance="yes"/>
    <invoke name="send-bp_order" partnerLink="plbp" portType="tns:ptbp" operation="op-bp_order"
            inputVariable="vb1"/>
    <pick name="p_switch">
      <onMessage partnerLink="plbp" portType="tns:ptbp" operation="op-pb_confirm" variable="vb1">
        <sequence name="publisher-confirm">
          <invoke name="send-bc_confirm" partnerLink="plcb" portType="tns:ptbc" operation="op-bc_confirm"
                  inputVariable="vb1"/>
          <invoke name="send-bs_req_shipment" partnerLink="plbs" portType="tns:ptbs"
                  operation="op-bs_req_shipment" inputVariable="vb1"/>
          <pick name="s_switch">
            <onMessage partnerLink="plbs" portType="tns:ptsb" operation="op-sb_confirm" variable="vb1">
              <sequence name="shipper-confirm">
                <invoke name="send-bp_ship_info" partnerLink="plbp" portType="tns:ptbp"
                        operation="op-bp_ship_info" inputVariable="vb1"/>
                <receive name="send-sb_notify" partnerLink="plbs" portType="tns:ptsb"
                        operation="op-sb_notify" variable="vb1"/>
                <invoke name="send-bc_send_bill" partnerLink="plcb" portType="tns:ptbc"
```

```
                        operation="op-bc_send_bill" inputVariable="vb1"/>
                <receive name="send-cb_payment" partnerLink="plcb" portType="tns:ptcb"
                        operation="op-cb_payment" variable="vb1"/>
              </sequence>
            </onMessage>
            <onMessage partnerLink="plbs" portType="tns:ptsb" operation="op-sb_decline" variable="vb1">
              <sequence name="shipper-decline">
                <invoke name="send-bp_cancel" partnerLink="plbp" portType="tns:ptbp"
                        operation="op-bp_cancel" inputVariable="vb1"/>
                <invoke name="send-bc_decline2" partnerLink="plcb" portType="tns:ptbc"
                        operation="op-bc_decline2" inputVariable="vb1"/>
              </sequence>
            </onMessage>
          </pick>
        </sequence>
      </onMessage>
      <onMessage partnerLink="plbp" portType="tns:ptbp" operation="op-pb_decline" variable="vb1">
        <sequence name="publisher-decline">
          <invoke name="send-bc_decline1" partnerLink="plcb" portType="tns:ptbc"
                  operation="op-bc_decline1" inputVariable="vb1"/>
        </sequence>
      </onMessage>
    </pick>
  </sequence>
</process>
```

Processo BPEL do Customer

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario51" name="cprocess51-bookstore-Customer"
         targetNamespace="http://meec.ist.utl.pt/cenario51" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="plcb" partnerLinkType="tns:pltcb" myRole="pltrncb-c" partnerRole="pltrncb-b"/>
    <partnerLink name="plcs" partnerLinkType="tns:pltcs" myRole="pltrncs-c"/>
    <partnerLink name="pl-start" partnerLinkType="tns:pltcb" myRole="pltrncb-c"/>
  </partnerLinks>
  <partners>
    <partner name="Bookstore">
      <partnerLink name="plcb"/>
    </partner>
    <partner name="Shipper">
      <partnerLink name="plcs"/>
    </partner>
  </partners>
  <variables>
    <variable name="vc1" messageType="tns:msgOrder"/>
  </variables>
  <sequence name="main">
    <receive name="initiate-global-process" partnerLink="pl-start" portType="pl-pt" operation="initiate"
             variable="vc1" createInstance="yes"/>
    <invoke name="send-bc_order" partnerLink="plcb" portType="tns:ptcb" operation="op-bc_order"
            inputVariable="vc1"/>
    <pick name="p_switch">
      <onMessage partnerLink="plcb" portType="tns:ptbc" operation="op-bc_confirm" variable="vc1">
        <sequence name="publisher-confirm">
          <pick name="s_switch">
            <onMessage partnerLink="plcs" portType="tns:ptsc" operation="op-sc_ship_book" variable="vc1">
              <sequence name="shipper-confirm">
                <receive name="send-bc_send_bill" partnerLink="plcb" portType="tns:ptbc"
                        operation="op-bc_send_bill" variable="vc1"/>
                <invoke name="send-cb_payment" partnerLink="plcb" portType="tns:ptcb"
                        operation="op-cb_payment" inputVariable="vc1"/>
              </sequence>
            </onMessage>
            <onMessage partnerLink="plcb" portType="tns:ptbc" operation="op-bc_decline2" variable="vc1">
              <empty/>
            </onMessage>
          </pick>
        </sequence>
      </onMessage>
      <onMessage partnerLink="plcb" portType="tns:ptbc" operation="op-bc_decline1" variable="vc1">
        <empty/>
      </onMessage>
```

```
    </pick>
  </sequence>
</process>
```

Processo BPEL do Publisher

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario51" name="cprocess51-bookstore-Publisher"
         targetNamespace="http://meec.ist.utl.pt/cenario51" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="plbp" partnerLinkType="tns:pltbp" myRole="pltrnbp-p" partnerRole="pltrnbp-b"/>
    <partnerLink name="plps" partnerLinkType="tns:pltps" partnerRole="pltrn4ps-s"/>
  </partnerLinks>
  <partners>
    <partner name="Bookstore">
      <partnerLink name="plbp"/>
    </partner>
    <partner name="Shipper">
      <partnerLink name="plps"/>
    </partner>
  </partners>
  <variables>
    <variable name="vp1" messageType="tns:msgOrder"/>
  </variables>
  <sequence name="main">
    <receive name="send-bp_order" partnerLink="plbp" portType="tns:ptbp" operation="op-bp_order"
             variable="vp1" createInstance="yes"/>
    <switch name="p_switch">
      <case condition="p_confirm">
        <sequence name="publisher-confirm">
          <invoke name="send-pb_confirm" partnerLink="plbp" portType="tns:ptbp" operation="op-pb_confirm"
                  inputVariable="vp1"/>
          <pick name="s_switch">
            <onMessage partnerLink="plbp" portType="tns:ptbp" operation="op-bp_ship_info" variable="vp1">
              <sequence name="shipper-confirm">
                <invoke name="send-ps_send_book" partnerLink="plps" portType="tns:ptbs"
                        operation="op-ps_send_book" inputVariable="vp1"/>
              </sequence>
            </onMessage>
            <onMessage partnerLink="plbp" portType="tns:ptbp" operation="op-bp_cancel" variable="vp1">
              <empty/>
            </onMessage>
          </pick>
        </sequence>
      </case>
      <case condition="p_decline">
        <sequence name="publisher-decline">
          <invoke name="send-pb_decline" partnerLink="plbp" portType="tns:ptbp" operation="op-pb_decline"
                  inputVariable="vp1"/>
        </sequence>
      </case>
    </switch>
  </sequence>
</process>
```

Processo BPEL do Shipper

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario51" name="cprocess51-bookstore-Shipper"
         targetNamespace="http://meec.ist.utl.pt/cenario51" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="plbs" partnerLinkType="tns:pltbs" myRole="pltrnbs-s" partnerRole="pltrnbs-b"/>
    <partnerLink name="plcs" partnerLinkType="tns:pltcs" partnerRole="pltrncs-c"/>
    <partnerLink name="plps" partnerLinkType="tns:pltps" myRole="pltrn4ps-s"/>
  </partnerLinks>
  <partners>
    <partner name="Bookstore">
```

```
      <partnerLink name="plbs"/>
    </partner>
    <partner name="Customer">
      <partnerLink name="plcs"/>
    </partner>
    <partner name="Publisher">
      <partnerLink name="plps"/>
    </partner>
  </partners>
  <variables>
    <variable name="vs1" messageType="tns:msgOrder"/>
  </variables>
  <sequence name="publisher-confirm">
    <receive name="send-bs_req_shipment" partnerLink="plbs" portType="tns:ptbs"
             operation="op-bs_req_shipment" variable="vs1" createInstance="yes"/>
    <switch name="s_switch">
      <case condition="s_confirm">
        <sequence name="shipper-confirm">
          <invoke name="send-sb_confirm" partnerLink="plbs" portType="tns:ptsb" operation="op-sb_confirm"
                  inputVariable="vs1"/>
          <receive name="send-ps_send_book" partnerLink="plps" portType="tns:ptbs"
                  operation="op-ps_send_book" variable="vs1"/>
          <invoke name="send-sc_ship_book" partnerLink="plcs" portType="tns:ptsc"
                  operation="op-sc_ship_book" inputVariable="vs1"/>
          <invoke name="send-sb_notify" partnerLink="plbs" portType="tns:ptsb" operation="op-sb_notify"
                  inputVariable="vs1"/>
        </sequence>
      </case>
      <case condition="s_decline">
        <sequence name="shipper-decline">
          <invoke name="send-sb_decline" partnerLink="plbs" portType="tns:ptsb" operation="op-sb_decline"
                  inputVariable="vs1"/>
        </sequence>
      </case>
    </switch>
  </sequence>
</process>
```

Cenário da Bookstore com scopes

Processo CBPEL do cenário da Bookstore com scopes

```
<commonProcess name="cprocess52-bookstore"
       xmlns="http://schemas.xmlsoap.org/ws/2005/05/common-business-process/"
       targetNamespace="http://meec.ist.utl.pt/cenario52" xmlns:tns="http://meec.ist.utl.pt/cenario52">

   <!-- ##### partnerLinks ### -->
   <partnerLinks>

      <partnerLink name="plcb" partnerLinkType="tns:pltcb">
         <role name="rpc" partnerLinkTypeRoleName="pltrncb-c" />
         <role name="rpb" partnerLinkTypeRoleName="pltrncb-b" />
      </partnerLink>

      <partnerLink name="plcs" partnerLinkType="tns:pltcs">
         <role name="rpc" partnerLinkTypeRoleName="pltrncs-c" />
         <role name="rps" partnerLinkTypeRoleName="pltrncs-s" />
      </partnerLink>

      <partnerLink name="plbp" partnerLinkType="tns:pltbp">
         <role name="rpb" partnerLinkTypeRoleName="pltrnbp-b" />
         <role name="rpp" partnerLinkTypeRoleName="pltrnbp-p" />
      </partnerLink>

      <partnerLink name="plps" partnerLinkType="tns:pltps">
         <role name="rpp" partnerLinkTypeRoleName="pltrnps-p" />
         <role name="rps" partnerLinkTypeRoleName="pltrn4ps-s" />
      </partnerLink>

      <partnerLink name="plbs" partnerLinkType="tns:pltbs">
```

```
      <role name="rpb" partnerLinkTypeRoleName="pltrnbs-b" />
      <role name="rps" partnerLinkTypeRoleName="pltrnbs-s" />
   </partnerLink>

</partnerLinks>


<!-- ##### partners #### -->
<partners>

   <partner name="Customer">
      <partnerLink name="plcb" role="rpc" />
      <partnerLink name="plcs" role="rpc" />
   </partner>

   <partner name="Bookstore">
      <partnerLink name="plcb" role="rpb" />
      <partnerLink name="plbp" role="rpb" />
      <partnerLink name="plbs" role="rpb" />
   </partner>

   <partner name="Publisher">
      <partnerLink name="plbp" role="rpp" />
      <partnerLink name="plps" role="rpp" />
   </partner>

   <partner name="Shipper">
      <partnerLink name="plbs" role="rps" />
      <partnerLink name="plps" role="rps" />
      <partnerLink name="plcs" role="rps" />
   </partner>

</partners>

<!-- ##### initiator ### -->
<initiator partner="Customer" initialSend="send-cb_order" />

<!-- ##### variables ### -->
<variables>
   <variable name="vc1" messageType="tns:msgOrder" owner="Customer" />
   <variable name="vb1" messageType="tns:msgOrder" owner="Bookstore" />
   <variable name="vp1" messageType="tns:msgOrder" owner="Publisher" />
   <variable name="vs1" messageType="tns:msgOrder" owner="Shipper" />
</variables>

<!-- ##### activity ### -->
<sequence name="main">

   <!--
      <send name="send-cbi" partnerLink="plcb" portType="tns:ptcb"
      operation="op-1" fromPartner="Customer" toPartner="Bookstore"
      inputVariable="vc1" outputVariable="vb1" createInstance="yes" />

      <send name="send-bci" partnerLink="plcb" portType="tns:ptbc"
      operation="op-1" fromPartner="Bookstore" toPartner="Customer"
      inputVariable="vb1" outputVariable="vc1" />
   -->
   <scope name="sc0">

      <sequence>

         <send name="send-cb_order" partnerLink="plcb" portType="tns:ptcb"
            operation="op-cb_order" fromPartner="Customer" toPartner="Bookstore"
            inputVariable="vc1" outputVariable="vb1" createInstance="yes"/>

         <scope name="sc1">
            <compensationHandler>
               <send name="send-bc_cancel" partnerLink="plcb" portType="tns:ptbc"
                  operation="op-bc_cancel" fromPartner="Bookstore" toPartner="Customer"
                  inputVariable="vb1" outputVariable="vc1" />
            </compensationHandler>
            <sequence>
               <send name="send-bc_inform" partnerLink="plcb" portType="tns:ptbc"
                  operation="op-bc_inform" fromPartner="Bookstore" toPartner="Customer"
                  inputVariable="vb1" outputVariable="vc1" />
            </sequence>
         </scope>
```

```
<send name="send-bp_order" partnerLink="plbp" portType="tns:ptbp"
   operation="op-bp_order" fromPartner="Bookstore" toPartner="Publisher"
   inputVariable="vb1" outputVariable="vp1" createInstance="yes" />

<switch name="Switch1" executer="Publisher">
   <case condition="publisher-decline">
      <throw name="throw1" faultName="f1" executer="Publisher" />
   </case>
</switch>

<scope name="sc2">
   <sequence>
      <send name="send-pb_confirm" partnerLink="plbp" portType="tns:ptbp"
         operation="op-pb_confirm" fromPartner="Publisher" toPartner="Bookstore"
         inputVariable="vp1" outputVariable="vb1" />
   </sequence>
</scope>

<send name="send-bs_req_shipment" partnerLink="plbs" portType="tns:ptbs"
   operation="op-bs_req_shipment" fromPartner="Bookstore" toPartner="Shipper"
   inputVariable="vb1" outputVariable="vs1" createInstance="yes" />

<switch name="Switch2" executer="Shipper">
   <case condition="shipper-decline">
      <throw name="throw2" faultName="f2" executer ="Shipper" />
   </case>
</switch>

<scope name="sc3">
   <sequence>
      <send name="send-sb_confirm" partnerLink="plbs" portType="tns:ptsb"
         operation="op-sb_confirm" fromPartner="Shipper" toPartner="Bookstore"
         inputVariable="vs1" outputVariable="vb1" />
   </sequence>
</scope>

<send name="send-bp_ship_info" partnerLink="plbp" portType="tns:ptbp"
   operation="op-bp_ship_info" fromPartner="Bookstore" toPartner="Publisher"
   inputVariable="vb1" outputVariable="vp1" />

<scope name="sc4">
   <compensationHandler>
      <send name="send-sp_return_book" partnerLink="plps" portType="tns:ptsp"
         operation="op-sp_return_book" fromPartner="Shipper" toPartner="Publisher"
         inputVariable="vs1" outputVariable="vp1" />
   </compensationHandler>
   <sequence>
      <send name="send-ps_send_book" partnerLink="plps" portType="tns:ptbs"
         operation="op-ps_send_book" fromPartner="Publisher" toPartner="Shipper"
         inputVariable="vp1" outputVariable="vs1" />
   </sequence>
</scope>

<scope name="sc5">
   <compensationHandler>
      <send name="send-cs_return_book" partnerLink="plcs" portType="tns:ptcs"
         operation="op-cs_return_book" fromPartner="Customer" toPartner="Shipper"
         inputVariable="vc1" outputVariable="vs1" />
   </compensationHandler>
   <sequence>
      <send name="send-sc_ship_book" partnerLink="plcs" portType="tns:ptsc"
         operation="op-sc_ship_book" fromPartner="Shipper" toPartner="Customer"
         inputVariable="vs1" outputVariable="vc1" />
   </sequence>
</scope>

<send name="send-sb_notify" partnerLink="plbs" portType="tns:ptsb"
   operation="op-sb_notify" fromPartner="Shipper" toPartner="Bookstore"
   inputVariable="vs1" outputVariable="vb1" />

<send name="send-bc_send_bill" partnerLink="plcb" portType="tns:ptbc"
   operation="op-bc_send_bill" fromPartner="Bookstore" toPartner="Customer"
   inputVariable="vb1" outputVariable="vc1" />

<switch name="Switch3" executer="Customer">
   <case condition="customer-reject">
      <throw name="throw3" faultName="f3" executer ="Customer" />
   </case>
```

```
            </switch>

            <scope name="sc6">
               <compensationHandler>
                  <send name="send-bc_return_payment" partnerLink="plcb" portType="tns:ptbc"
                     operation="op-bc_return_payment" fromPartner="Bookstore" toPartner="Customer"
                     inputVariable="vb1" outputVariable="vc1" />
               </compensationHandler>
               <sequence>
                  <send name="send-cb_payment" partnerLink="plcb" portType="tns:ptcb"
                     operation="op-cb_payment" fromPartner="Customer" toPartner="Bookstore"
                     inputVariable="vc1" outputVariable="vb1" />
               </sequence>
            </scope>

            <switch name="Switch4" executer="Customer">
               <case condition="customer-unsatisfied">
                  <throw name="throw4" faultName="f4" executer ="Customer" />
               </case>
            </switch>

         </sequence>

      </scope>

   </sequence>

</commonProcess>
```

Processo BPEL da Bookstore

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario52" name="cprocess52-bookstore-Bookstore"
         targetNamespace="http://meec.ist.utl.pt/cenario52" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="plbp" partnerLinkType="tns:pltbp" myRole="pltrnbp-b" partnerRole="pltrnbp-p"/>
    <partnerLink name="plbs" partnerLinkType="tns:pltbs" myRole="pltrnbs-b" partnerRole="pltrnbs-s"/>
    <partnerLink name="plcb" partnerLinkType="tns:pltcb" myRole="pltrncb-b" partnerRole="pltrncb-c"/>
    <partnerLink name="ccpl" partnerLinkType="tns:plt-Bookstorecc" partnerRole="pltrncc"
                 myRole="pltrnBookstore"/>
  </partnerLinks>
  <partners>
    <partner name="Customer">
      <partnerLink name="plcb"/>
    </partner>
    <partner name="Publisher">
      <partnerLink name="plbp"/>
    </partner>
    <partner name="Shipper">
      <partnerLink name="plbs"/>
    </partner>
    <partner name="cc">
      <partnerLink name="ccpl"/>
    </partner>
  </partners>
  <variables>
    <variable name="vb1" messageType="tns:msgOrder"/>
  </variables>
  <sequence name="seq-before-sc0">
    <receive name="notify-sc0" partnerLink="plcb" portType="ntfScpPT" operation="ntfScpOp" variable=""/>
    <scope name="sc0">
      <faultHandlers>
        <catch faultName="forcedTermination">
          <sequence>
            <receive name="Faulted:sc0" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                     variable=""/>
            <scope name="internalScope:sc0">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
```

```
          <sequence name="seq-internal">
            <invoke name="Fault-done:sc0" partnerLink="ccpl" portType="ccpt" operation="FaultDoneOp"
                    inputVariable=""/>
            <receive name="Fault-done:sc0" partnerLink="ccpl" portType="ccpt" operation="FaultDoneOp"
                     variable=""/>
          </sequence>
        </scope>
      </sequence>
    </catch>
    <catchAll>
      <sequence>
        <receive name="Faulted-DefaultFH:sc0" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                 variable=""/>
        <scope name="sc0:internalScope">
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                       variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
            <invoke name="Compensate:sc0" partnerLink="ccpl" portType="ccpt" operation="CompensateOp"
                    inputVariable=""/>
            <receive name="Compensate:sc0" partnerLink="ccpl" portType="ccpt" operation="CompensateOp"
                    variable=""/>
            <compensate/>
            <receive name="Compensated:sc0" partnerLink="ccpl" portType="ccpt"
                    operation="CompensatedOp" variable=""/>
            <throw/>
          </sequence>
        </scope>
      </sequence>
    </catchAll>
  </faultHandlers>
  <compensationHandler name="default-compHandler">
    <sequence>
      <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
               variable=""/>
      <scope name="internalScope-ch:sc0">
        <eventHandlers>
          <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
            <throw faultName="msg-fault.value"/>
          </onMessage>
        </eventHandlers>
        <sequence name="seq-internal">
          <compensate/>
          <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" inputVariable=""/>
          <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" variable=""/>
        </sequence>
      </scope>
    </sequence>
  </compensationHandler>
  <eventHandlers>
    <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
      <throw faultName="msg-fault.value"/>
    </onMessage>
  </eventHandlers>
  <sequence name="seq-scope-sc0">
    <invoke name="Register:sc0" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
            inputVariable=""/>
    <receive name="Registered:sc0" partnerLink="ccpl" portType="ccpt" operation="RegisteredScopeOp"
            variable=""/>
    <receive name="send-cb_order" partnerLink="plcb" portType="tns:ptcb" operation="op-cb_order"
            variable="vb1" createInstance="yes"/>
    <invoke name="Create:sc1:sc0" partnerLink="ccpl" portType="ccpt" operation="CreateScopeOp"
            inputVariable=""/>
    <receive name="Created:sc1" partnerLink="ccpl" portType="ccpt" operation="CreatedScopeOp"
            variable=""/>
    <scope name="sc1">
      <faultHandlers>
        <catch faultName="forcedTermination">
          <sequence>
            <receive name="Faulted:sc1" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                     variable=""/>
            <scope name="internalScope:sc1">
              <eventHandlers>
```

```
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Fault-done:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" inputVariable=""/>
                <receive name="Fault-done:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" variable=""/>
              </sequence>
            </scope>
          </sequence>
        </catch>
        <catchAll>
          <sequence>
            <receive name="Faulted-DefaultFH:sc1" partnerLink="ccpl" portType="ccpt"
                     operation="Faulted" variable=""/>
            <scope name="sc1:internalScope">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Compensate:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="CompensateOp" inputVariable=""/>
                <receive name="Compensate:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="CompensateOp" variable=""/>
                <compensate/>
                <receive name="Compensated:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="CompensatedOp" variable=""/>
                <throw/>
              </sequence>
            </scope>
          </sequence>
        </catchAll>
      </faultHandlers>
      <compensationHandler>
        <sequence>
          <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                   variable=""/>
          <scope name="internalScope-ch:sc1">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="send-bc_cancel" partnerLink="plcb" portType="tns:ptbc"
                      operation="op-bc_cancel" inputVariable="vb1"/>
              <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                      operation="ChCompensatedScOp" inputVariable=""/>
              <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                      operation="ChCompensatedScOp" variable=""/>
            </sequence>
          </scope>
        </sequence>
      </compensationHandler>
      <eventHandlers>
        <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
          <throw faultName="msg-fault.value"/>
        </onMessage>
      </eventHandlers>
      <sequence name="seq-scope-sc1">
        <invoke name="notify-sc1" partnerLink="plcb" portType="ntfScpPT" operation="ntfScpOp"
                inputVariable="varntf"/>
        <invoke name="send-bc_inform" partnerLink="plcb" portType="tns:ptbc" operation="op-bc_inform"
                inputVariable="vb1"/>
        <invoke name="Completed:sc1" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                inputVariable=""/>
        <receive name="Completed:sc1" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                variable=""/>
      </sequence>
    </scope>
    <invoke name="notify-sc0" partnerLink="plbp" portType="ntfScpPT" operation="ntfScpOp"
```

```
        inputVariable="varntf"/>
<invoke name="send-bp_order" partnerLink="plbp" portType="tns:ptbp" operation="op-bp_order"
        inputVariable="vb1"/>
<receive name="notify-sc2" partnerLink="plbp" portType="ntfScpPT" operation="ntfScpOp"
        variable=""/>
<scope name="sc2">
  <faultHandlers>
    <catch faultName="forcedTermination">
      <sequence>
        <receive name="Faulted:sc2" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                 variable=""/>
        <scope name="internalScope:sc2">
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                       variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
            <invoke name="Fault-done:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="FaultDoneOp" inputVariable=""/>
            <receive name="Fault-done:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="FaultDoneOp" variable=""/>
          </sequence>
        </scope>
      </sequence>
    </catch>
    <catchAll>
      <sequence>
        <receive name="Faulted-DefaultFH:sc2" partnerLink="ccpl" portType="ccpt"
                 operation="Faulted" variable=""/>
        <scope name="sc2:internalScope">
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                       variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
            <invoke name="Compensate:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="CompensateOp" inputVariable=""/>
            <receive name="Compensate:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="CompensateOp" variable=""/>
            <compensate/>
            <receive name="Compensated:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="CompensatedOp" variable=""/>
            <throw/>
          </sequence>
        </scope>
      </sequence>
    </catchAll>
  </faultHandlers>
  <compensationHandler name="default-compHandler">
    <sequence>
      <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
               variable=""/>
      <scope name="internalScope-ch:sc2">
        <eventHandlers>
          <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                     variable="msg-fault">
            <throw faultName="msg-fault.value"/>
          </onMessage>
        </eventHandlers>
        <sequence name="seq-internal">
          <compensate/>
          <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" inputVariable=""/>
          <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" variable=""/>
        </sequence>
      </scope>
    </sequence>
  </compensationHandler>
  <eventHandlers>
    <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
      <throw faultName="msg-fault.value"/>
    </onMessage>
  </eventHandlers>
```

```
    <sequence name="seq-scope-sc2">
      <invoke name="Register:sc2" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
              inputVariable=""/>
      <receive name="Registered:sc2" partnerLink="ccpl" portType="ccpt"
              operation="RegisteredScopeOp" variable=""/>
      <receive name="send-pb_confirm" partnerLink="plbp" portType="tns:ptbp"
              operation="op-pb_confirm" variable="vb1"/>
      <invoke name="Completed:sc2" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              inputVariable=""/>
      <receive name="Completed:sc2" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              variable=""/>
    </sequence>
  </scope>
  <invoke name="notify-sc0" partnerLink="plbs" portType="ntfScpPT" operation="ntfScpOp"
          inputVariable="varntf"/>
  <invoke name="send-bs_req_shipment" partnerLink="plbs" portType="tns:ptbs"
          operation="op-bs_req_shipment" inputVariable="vb1"/>
  <receive name="notify-sc3" partnerLink="plbs" portType="ntfScpPT" operation="ntfScpOp"
          variable=""/>
  <scope name="sc3">
    <faultHandlers>
      <catch faultName="forcedTermination">
        <sequence>
          <receive name="Faulted:sc3" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                  variable=""/>
          <scope name="internalScope:sc3">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Fault-done:sc3" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" inputVariable=""/>
              <receive name="Fault-done:sc3" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" variable=""/>
            </sequence>
          </scope>
        </sequence>
      </catch>
      <catchAll>
        <sequence>
          <receive name="Faulted-DefaultFH:sc3" partnerLink="ccpl" portType="ccpt"
                  operation="Faulted" variable=""/>
          <scope name="sc3:internalScope">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Compensate:sc3" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" inputVariable=""/>
              <receive name="Compensate:sc3" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" variable=""/>
              <compensate/>
              <receive name="Compensated:sc3" partnerLink="ccpl" portType="ccpt"
                      operation="CompensatedOp" variable=""/>
              <throw/>
            </sequence>
          </scope>
        </sequence>
      </catchAll>
    </faultHandlers>
    <compensationHandler name="default-compHandler">
      <sequence>
        <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                variable=""/>
        <scope name="internalScope-ch:sc3">
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                       variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
```

```
            <compensate/>
            <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" inputVariable=""/>
            <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" variable=""/>
          </sequence>
        </scope>
      </sequence>
    </compensationHandler>
    <eventHandlers>
      <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
        <throw faultName="msg-fault.value"/>
      </onMessage>
    </eventHandlers>
    <sequence name="seq-scope-sc3">
      <invoke name="Register:sc3" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
              inputVariable=""/>
      <receive name="Registered:sc3" partnerLink="ccpl" portType="ccpt"
              operation="RegisteredScopeOp" variable=""/>
      <receive name="send-sb_confirm" partnerLink="plbs" portType="tns:ptsb"
              operation="op-sb_confirm" variable="vb1"/>
      <invoke name="Completed:sc3" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              inputVariable=""/>
      <receive name="Completed:sc3" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              variable=""/>
    </sequence>
  </scope>
  <invoke name="send-bp_ship_info" partnerLink="plbp" portType="tns:ptbp"
          operation="op-bp_ship_info" inputVariable="vb1"/>
  <receive name="send-sb_notify" partnerLink="plbs" portType="tns:ptsb" operation="op-sb_notify"
          variable="vb1"/>
  <invoke name="send-bc_send_bill" partnerLink="plcb" portType="tns:ptbc"
          operation="op-bc_send_bill" inputVariable="vb1"/>
  <receive name="notify-sc6" partnerLink="plcb" portType="ntfScpPT" operation="ntfScpOp"
          variable=""/>
  <scope name="sc6">
    <faultHandlers>
      <catch faultName="forcedTermination">
        <sequence>
          <receive name="Faulted:sc6" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                  variable=""/>
          <scope name="internalScope:sc6">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                          variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Fault-done:sc6" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" inputVariable=""/>
              <receive name="Fault-done:sc6" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" variable=""/>
            </sequence>
          </scope>
        </sequence>
      </catch>
      <catchAll>
        <sequence>
          <receive name="Faulted-DefaultFH:sc6" partnerLink="ccpl" portType="ccpt"
                  operation="Faulted" variable=""/>
          <scope name="sc6:internalScope">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                          variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Compensate:sc6" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" inputVariable=""/>
              <receive name="Compensate:sc6" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" variable=""/>
              <compensate/>
              <receive name="Compensated:sc6" partnerLink="ccpl" portType="ccpt"
                      operation="CompensatedOp" variable=""/>
              <throw/>
            </sequence>
```

```
                </scope>
              </sequence>
            </catchAll>
          </faultHandlers>
          <compensationHandler>
            <sequence>
              <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                       variable=""/>
              <scope name="internalScope-ch:sc6">
                <eventHandlers>
                  <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                             variable="msg-fault">
                    <throw faultName="msg-fault.value"/>
                  </onMessage>
                </eventHandlers>
                <sequence name="seq-internal">
                  <invoke name="send-bc_return_payment" partnerLink="plcb" portType="tns:ptbc"
                          operation="op-bc_return_payment" inputVariable="vb1"/>
                  <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                          operation="ChCompensatedScOp" inputVariable=""/>
                  <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                          operation="ChCompensatedScOp" variable=""/>
                </sequence>
              </scope>
            </sequence>
          </compensationHandler>
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-scope-sc6">
            <invoke name="Register:sc6" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
                    inputVariable=""/>
            <receive name="Registered:sc6" partnerLink="ccpl" portType="ccpt"
                    operation="RegisteredScopeOp" variable=""/>
            <receive name="send-cb_payment" partnerLink="plcb" portType="tns:ptcb"
                    operation="op-cb_payment" variable="vb1"/>
            <invoke name="Completed:sc6" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                    inputVariable=""/>
            <receive name="Completed:sc6" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                    variable=""/>
          </sequence>
        </scope>
        <invoke name="Completed:sc0" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                inputVariable=""/>
        <receive name="Completed:sc0" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                variable=""/>
      </sequence>
    </scope>
  </sequence>
</process>
```

Processo BPEL do Customer

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario52" name="cprocess52-bookstore-Customer"
         targetNamespace="http://meec.ist.utl.pt/cenario52" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="plcb" partnerLinkType="tns:pltcb" myRole="pltrncb-c" partnerRole="pltrncb-b"/>
    <partnerLink name="plcs" partnerLinkType="tns:pltcs" myRole="pltrncs-c" partnerRole="pltrncs-s"/>
    <partnerLink name="pl-start" partnerLinkType="tns:pltcb" myRole="pltrncb-c"/>
    <partnerLink name="ccpl" partnerLinkType="tns:plt-Customercc" partnerRole="pltrncc"
                 myRole="pltrnCustomer"/>
  </partnerLinks>
  <partners>
    <partner name="Bookstore">
      <partnerLink name="plcb"/>
    </partner>
    <partner name="Shipper">
      <partnerLink name="plcs"/>
    </partner>
    <partner name="cc">
```

```
    <partnerLink name="ccpl"/>
  </partner>
</partners>
<variables>
  <variable name="vc1" messageType="tns:msgOrder"/>
</variables>
<sequence name="main">
  <receive name="initiate-global-process" partnerLink="pl-start" portType="pl-pt" operation="initiate"
          variable="vc1" createInstance="yes"/>
  <invoke name="Create:sc0:null" partnerLink="ccpl" portType="ccpt" operation="CreateScopeOp"
          inputVariable=""/>
  <receive name="Created:sc0" partnerLink="ccpl" portType="ccpt" operation="CreatedScopeOp"
          variable=""/>
  <scope name="sc0">
    <faultHandlers>
      <catch faultName="forcedTermination">
        <sequence>
          <receive name="Faulted:sc0" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                   variable=""/>
          <scope name="internalScope:sc0">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Fault-done:sc0" partnerLink="ccpl" portType="ccpt" operation="FaultDoneOp"
                      inputVariable=""/>
              <receive name="Fault-done:sc0" partnerLink="ccpl" portType="ccpt" operation="FaultDoneOp"
                       variable=""/>
            </sequence>
          </scope>
        </sequence>
      </catch>
      <catchAll>
        <sequence>
          <receive name="Faulted-DefaultFH:sc0" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                   variable=""/>
          <scope name="sc0:internalScope">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Compensate:sc0" partnerLink="ccpl" portType="ccpt" operation="CompensateOp"
                      inputVariable=""/>
              <receive name="Compensate:sc0" partnerLink="ccpl" portType="ccpt" operation="CompensateOp"
                      variable=""/>
              <compensate/>
              <receive name="Compensated:sc0" partnerLink="ccpl" portType="ccpt"
                      operation="CompensatedOp" variable=""/>
              <throw/>
            </sequence>
          </scope>
        </sequence>
      </catchAll>
    </faultHandlers>
    <compensationHandler name="default-compHandler">
      <sequence>
        <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                 variable=""/>
        <scope name="internalScope-ch:sc0">
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
            <compensate/>
            <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" inputVariable=""/>
            <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" variable=""/>
          </sequence>
        </scope>
```

```
    </sequence>
  </compensationHandler>
  <eventHandlers>
    <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
      <throw faultName="msg-fault.value"/>
    </onMessage>
  </eventHandlers>
  <sequence name="seq-scope-sc0">
    <invoke name="notify-sc0" partnerLink="plcb" portType="ntfScpPT" operation="ntfScpOp"
            inputVariable="varntf"/>
    <invoke name="send-cb_order" partnerLink="plcb" portType="tns:ptcb" operation="op-cb_order"
            inputVariable="vc1"/>
    <receive name="notify-sc1" partnerLink="plcb" portType="ntfScpPT" operation="ntfScpOp"
            variable=""/>
    <scope name="sc1">
      <faultHandlers>
        <catch faultName="forcedTermination">
          <sequence>
            <receive name="Faulted:sc1" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                     variable=""/>
            <scope name="internalScope:sc1">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Fault-done:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" inputVariable=""/>
                <receive name="Fault-done:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" variable=""/>
              </sequence>
            </scope>
          </sequence>
        </catch>
        <catchAll>
          <sequence>
            <receive name="Faulted-DefaultFH:sc1" partnerLink="ccpl" portType="ccpt"
                     operation="Faulted" variable=""/>
            <scope name="sc1:internalScope">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Compensate:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="CompensateOp" inputVariable=""/>
                <receive name="Compensate:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="CompensateOp" variable=""/>
                <compensate/>
                <receive name="Compensated:sc1" partnerLink="ccpl" portType="ccpt"
                        operation="CompensatedOp" variable=""/>
                <throw/>
              </sequence>
            </scope>
          </sequence>
        </catchAll>
      </faultHandlers>
      <compensationHandler>
        <sequence>
          <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                   variable=""/>
          <scope name="internalScope-ch:sc1">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <receive name="send-bc_cancel" partnerLink="plcb" portType="tns:ptbc"
                       operation="op-bc_cancel" variable="vc1"/>
              <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                       operation="ChCompensatedScOp" inputVariable=""/>
              <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
```

```
                    operation="ChCompensatedScOp" variable=""/>
          </sequence>
        </scope>
      </sequence>
    </compensationHandler>
    <eventHandlers>
      <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
        <throw faultName="msg-fault.value"/>
      </onMessage>
    </eventHandlers>
    <sequence name="seq-scope-sc1">
      <invoke name="Register:sc1" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
              inputVariable=""/>
      <receive name="Registered:sc1" partnerLink="ccpl" portType="ccpt"
              operation="RegisteredScopeOp" variable=""/>
      <receive name="send-bc_inform" partnerLink="plcb" portType="tns:ptbc" operation="op-bc_inform"
              variable="vc1"/>
      <invoke name="Completed:sc1" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              inputVariable=""/>
      <receive name="Completed:sc1" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              variable=""/>
    </sequence>
  </scope>
  <receive name="notify-sc5" partnerLink="plcs" portType="ntfScpPT" operation="ntfScpOp"
           variable=""/>
  <scope name="sc5">
    <faultHandlers>
      <catch faultName="forcedTermination">
        <sequence>
          <receive name="Faulted:sc5" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                   variable=""/>
          <scope name="internalScope:sc5">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Fault-done:sc5" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" inputVariable=""/>
              <receive name="Fault-done:sc5" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" variable=""/>
            </sequence>
          </scope>
        </sequence>
      </catch>
      <catchAll>
        <sequence>
          <receive name="Faulted-DefaultFH:sc5" partnerLink="ccpl" portType="ccpt"
                   operation="Faulted" variable=""/>
          <scope name="sc5:internalScope">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Compensate:sc5" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" inputVariable=""/>
              <receive name="Compensate:sc5" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" variable=""/>
              <compensate/>
              <receive name="Compensated:sc5" partnerLink="ccpl" portType="ccpt"
                      operation="CompensatedOp" variable=""/>
              <throw/>
            </sequence>
          </scope>
        </sequence>
      </catchAll>
    </faultHandlers>
    <compensationHandler>
      <sequence>
        <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                 variable=""/>
        <scope name="internalScope-ch:sc5">
          <eventHandlers>
```

```
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                       variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
            <invoke name="send-cs_return_book" partnerLink="plcs" portType="tns:ptcs"
                    operation="op-cs_return_book" inputVariable="vc1"/>
            <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" inputVariable=""/>
            <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" variable=""/>
          </sequence>
        </scope>
      </sequence>
    </compensationHandler>
    <eventHandlers>
      <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
        <throw faultName="msg-fault.value"/>
      </onMessage>
    </eventHandlers>
    <sequence name="seq-scope-sc5">
      <invoke name="Register:sc5" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
              inputVariable=""/>
      <receive name="Registered:sc5" partnerLink="ccpl" portType="ccpt"
              operation="RegisteredScopeOp" variable=""/>
      <receive name="send-sc_ship_book" partnerLink="plcs" portType="tns:ptsc"
              operation="op-sc_ship_book" variable="vc1"/>
      <invoke name="Completed:sc5" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              inputVariable=""/>
      <receive name="Completed:sc5" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              variable=""/>
    </sequence>
  </scope>
  <receive name="send-bc_send_bill" partnerLink="plcb" portType="tns:ptbc"
           operation="op-bc_send_bill" variable="vc1"/>
  <switch name="Switch3">
    <case condition="customer-reject">
      <sequence name="seq-throw-throw3">
        <invoke name="Fault:sc0:f3" partnerLink="ccpl" portType="ccpt" operation="ccThrowOp"
                inputVariable=""/>
        <wait for="forever"/>
      </sequence>
    </case>
  </switch>
  <invoke name="Create:sc6:sc0" partnerLink="ccpl" portType="ccpt" operation="CreateScopeOp"
          inputVariable=""/>
  <receive name="Created:sc6" partnerLink="ccpl" portType="ccpt" operation="CreatedScopeOp"
          variable=""/>
  <scope name="sc6">
    <faultHandlers>
      <catch faultName="forcedTermination">
        <sequence>
          <receive name="Faulted:sc6" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                   variable=""/>
          <scope name="internalScope:sc6">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Fault-done:sc6" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" inputVariable=""/>
              <receive name="Fault-done:sc6" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" variable=""/>
            </sequence>
          </scope>
        </sequence>
      </catch>
      <catchAll>
        <sequence>
          <receive name="Faulted-DefaultFH:sc6" partnerLink="ccpl" portType="ccpt"
                   operation="Faulted" variable=""/>
          <scope name="sc6:internalScope">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
```

```
                        variable="msg-fault">
                    <throw faultName="msg-fault.value"/>
                  </onMessage>
                </eventHandlers>
                <sequence name="seq-internal">
                  <invoke name="Compensate:sc6" partnerLink="ccpl" portType="ccpt"
                          operation="CompensateOp" inputVariable=""/>
                  <receive name="Compensate:sc6" partnerLink="ccpl" portType="ccpt"
                          operation="CompensateOp" variable=""/>
                  <compensate/>
                  <receive name="Compensated:sc6" partnerLink="ccpl" portType="ccpt"
                          operation="CompensatedOp" variable=""/>
                  <throw/>
                </sequence>
              </scope>
            </sequence>
          </catchAll>
        </faultHandlers>
        <compensationHandler>
          <sequence>
            <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                     variable=""/>
            <scope name="internalScope-ch:sc6">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <receive name="send-bc_return_payment" partnerLink="plcb" portType="tns:ptbc"
                        operation="op-bc_return_payment" variable="vc1"/>
                <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                        operation="ChCompensatedScOp" inputVariable=""/>
                <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                        operation="ChCompensatedScOp" variable=""/>
              </sequence>
            </scope>
          </sequence>
        </compensationHandler>
        <eventHandlers>
          <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
            <throw faultName="msg-fault.value"/>
          </onMessage>
        </eventHandlers>
        <sequence name="seq-scope-sc6">
          <invoke name="notify-sc6" partnerLink="plcb" portType="ntfScpPT" operation="ntfScpOp"
                  inputVariable="varntf"/>
          <invoke name="send-cb_payment" partnerLink="plcb" portType="tns:ptcb"
                  operation="op-cb_payment" inputVariable="vc1"/>
          <invoke name="Completed:sc6" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                  inputVariable=""/>
          <receive name="Completed:sc6" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                  variable=""/>
        </sequence>
      </scope>
      <switch name="Switch4">
        <case condition="customer-unsatisfied">
          <sequence name="seq-throw-throw4">
            <invoke name="Fault:sc0:f4" partnerLink="ccpl" portType="ccpt" operation="ccThrowOp"
                    inputVariable=""/>
            <wait for="forever"/>
          </sequence>
        </case>
      </switch>
      <invoke name="Completed:sc0" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              inputVariable=""/>
      <receive name="Completed:sc0" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              variable=""/>
    </sequence>
  </scope>
</sequence>
</process>
```

Processo BPEL do Publisher

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario52" name="cprocess52-bookstore-Publisher"
         targetNamespace="http://meec.ist.utl.pt/cenario52" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="plbp" partnerLinkType="tns:pltbp" myRole="pltrnbp-p" partnerRole="pltrnbp-b"/>
    <partnerLink name="plps" partnerLinkType="tns:pltps" myRole="pltrnps-p" partnerRole="pltrn4ps-s"/>
    <partnerLink name="ccpl" partnerLinkType="tns:plt-Publishercc" partnerRole="pltrncc"
                 myRole="pltrnPublisher"/>
  </partnerLinks>
  <partners>
    <partner name="Bookstore">
      <partnerLink name="plbp"/>
    </partner>
    <partner name="Shipper">
      <partnerLink name="plps"/>
    </partner>
    <partner name="cc">
      <partnerLink name="ccpl"/>
    </partner>
  </partners>
  <variables>
    <variable name="vp1" messageType="tns:msgOrder"/>
  </variables>
  <sequence name="seq-before-sc0">
    <receive name="notify-sc0" partnerLink="plbp" portType="ntfScpPT" operation="ntfScpOp" variable=""/>
    <scope name="sc0">
      <faultHandlers>
        <catch faultName="forcedTermination">
          <sequence>
            <receive name="Faulted:sc0" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                     variable=""/>
            <scope name="internalScope:sc0">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Fault-done:sc0" partnerLink="ccpl" portType="ccpt" operation="FaultDoneOp"
                        inputVariable=""/>
                <receive name="Fault-done:sc0" partnerLink="ccpl" portType="ccpt" operation="FaultDoneOp"
                        variable=""/>
              </sequence>
            </scope>
          </sequence>
        </catch>
        <catchAll>
          <sequence>
            <receive name="Faulted-DefaultFH:sc0" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                     variable=""/>
            <scope name="sc0:internalScope">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Compensate:sc0" partnerLink="ccpl" portType="ccpt" operation="CompensateOp"
                        inputVariable=""/>
                <receive name="Compensate:sc0" partnerLink="ccpl" portType="ccpt" operation="CompensateOp"
                        variable=""/>
                <compensate/>
                <receive name="Compensated:sc0" partnerLink="ccpl" portType="ccpt"
                        operation="CompensatedOp" variable=""/>
                <throw/>
              </sequence>
            </scope>
          </sequence>
        </catchAll>
      </faultHandlers>
      <compensationHandler name="default-compHandler">
```

```
    <sequence>
      <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
               variable=""/>
      <scope name="internalScope-ch:sc0">
        <eventHandlers>
          <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
            <throw faultName="msg-fault.value"/>
          </onMessage>
        </eventHandlers>
        <sequence name="seq-internal">
          <compensate/>
          <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" inputVariable=""/>
          <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" variable=""/>
        </sequence>
      </scope>
    </sequence>
  </compensationHandler>
  <eventHandlers>
    <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
      <throw faultName="msg-fault.value"/>
    </onMessage>
  </eventHandlers>
  <sequence name="seq-scope-sc0">
    <invoke name="Register:sc0" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
            inputVariable=""/>
    <receive name="Registered:sc0" partnerLink="ccpl" portType="ccpt" operation="RegisteredScopeOp"
            variable=""/>
    <receive name="send-bp_order" partnerLink="plbp" portType="tns:ptbp" operation="op-bp_order"
            variable="vp1" createInstance="yes"/>
    <switch name="Switch1">
      <case condition="publisher-decline">
        <sequence name="seq-throw-throw1">
          <invoke name="Fault:sc0:f1" partnerLink="ccpl" portType="ccpt" operation="ccThrowOp"
                  inputVariable=""/>
          <wait for="forever"/>
        </sequence>
      </case>
    </switch>
    <invoke name="Create:sc2:sc0" partnerLink="ccpl" portType="ccpt" operation="CreateScopeOp"
            inputVariable=""/>
    <receive name="Created:sc2" partnerLink="ccpl" portType="ccpt" operation="CreatedScopeOp"
            variable=""/>
    <scope name="sc2">
      <faultHandlers>
        <catch faultName="forcedTermination">
          <sequence>
            <receive name="Faulted:sc2" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                    variable=""/>
            <scope name="internalScope:sc2">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Fault-done:sc2" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" inputVariable=""/>
                <receive name="Fault-done:sc2" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" variable=""/>
              </sequence>
            </scope>
          </sequence>
        </catch>
        <catchAll>
          <sequence>
            <receive name="Faulted-DefaultFH:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="Faulted" variable=""/>
            <scope name="sc2:internalScope">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
```

```
            <invoke name="Compensate:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="CompensateOp" inputVariable=""/>
            <receive name="Compensate:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="CompensateOp" variable=""/>
            <compensate/>
            <receive name="Compensated:sc2" partnerLink="ccpl" portType="ccpt"
                    operation="CompensatedOp" variable=""/>
            <throw/>
          </sequence>
        </scope>
      </sequence>
    </catchAll>
  </faultHandlers>
  <compensationHandler name="default-compHandler">
    <sequence>
      <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
               variable=""/>
      <scope name="internalScope-ch:sc2">
        <eventHandlers>
          <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                     variable="msg-fault">
            <throw faultName="msg-fault.value"/>
          </onMessage>
        </eventHandlers>
        <sequence name="seq-internal">
          <compensate/>
          <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" inputVariable=""/>
          <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" variable=""/>
        </sequence>
      </scope>
    </sequence>
  </compensationHandler>
  <eventHandlers>
    <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
      <throw faultName="msg-fault.value"/>
    </onMessage>
  </eventHandlers>
  <sequence name="seq-scope-sc2">
    <invoke name="notify-sc2" partnerLink="plbp" portType="ntfScpPT" operation="ntfScpOp"
            inputVariable="varntf"/>
    <invoke name="send-pb_confirm" partnerLink="plbp" portType="tns:ptbp"
            operation="op-pb_confirm" inputVariable="vp1"/>
    <invoke name="Completed:sc2" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
            inputVariable=""/>
    <receive name="Completed:sc2" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
            variable=""/>
  </sequence>
</scope>
<receive name="send-bp_ship_info" partnerLink="plbp" portType="tns:ptbp"
        operation="op-bp_ship_info" variable="vp1"/>
<invoke name="Create:sc4:sc0" partnerLink="ccpl" portType="ccpt" operation="CreateScopeOp"
        inputVariable=""/>
<receive name="Created:sc4" partnerLink="ccpl" portType="ccpt" operation="CreatedScopeOp"
         variable=""/>
<scope name="sc4">
  <faultHandlers>
    <catch faultName="forcedTermination">
      <sequence>
        <receive name="Faulted:sc4" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                 variable=""/>
        <scope name="internalScope:sc4">
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                       variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
            <invoke name="Fault-done:sc4" partnerLink="ccpl" portType="ccpt"
                    operation="FaultDoneOp" inputVariable=""/>
            <receive name="Fault-done:sc4" partnerLink="ccpl" portType="ccpt"
                    operation="FaultDoneOp" variable=""/>
          </sequence>
        </scope>
      </sequence>
    </catch>
```

```
            <catchAll>
              <sequence>
                <receive name="Faulted-DefaultFH:sc4" partnerLink="ccpl" portType="ccpt"
                         operation="Faulted" variable=""/>
                <scope name="sc4:internalScope">
                  <eventHandlers>
                    <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                               variable="msg-fault">
                      <throw faultName="msg-fault.value"/>
                    </onMessage>
                  </eventHandlers>
                  <sequence name="seq-internal">
                    <invoke name="Compensate:sc4" partnerLink="ccpl" portType="ccpt"
                            operation="CompensateOp" inputVariable=""/>
                    <receive name="Compensate:sc4" partnerLink="ccpl" portType="ccpt"
                            operation="CompensateOp" variable=""/>
                    <compensate/>
                    <receive name="Compensated:sc4" partnerLink="ccpl" portType="ccpt"
                            operation="CompensatedOp" variable=""/>
                    <throw/>
                  </sequence>
                </scope>
              </sequence>
            </catchAll>
          </faultHandlers>
          <compensationHandler>
            <sequence>
              <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                       variable=""/>
              <scope name="internalScope-ch:sc4">
                <eventHandlers>
                  <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                             variable="msg-fault">
                    <throw faultName="msg-fault.value"/>
                  </onMessage>
                </eventHandlers>
                <sequence name="seq-internal">
                  <receive name="send-sp_return_book" partnerLink="plps" portType="tns:ptsp"
                           operation="op-sp_return_book" variable="vp1"/>
                  <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                           operation="ChCompensatedScOp" inputVariable=""/>
                  <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                           operation="ChCompensatedScOp" variable=""/>
                </sequence>
              </scope>
            </sequence>
          </compensationHandler>
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-scope-sc4">
            <invoke name="notify-sc4" partnerLink="plps" portType="ntfScpPT" operation="ntfScpOp"
                    inputVariable="varntf"/>
            <invoke name="send-ps_send_book" partnerLink="plps" portType="tns:ptbs"
                    operation="op-ps_send_book" inputVariable="vp1"/>
            <invoke name="Completed:sc4" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                    inputVariable=""/>
            <receive name="Completed:sc4" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                    variable=""/>
          </sequence>
        </scope>
        <invoke name="Completed:sc0" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                inputVariable=""/>
        <receive name="Completed:sc0" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                variable=""/>
      </sequence>
    </scope>
  </sequence>
</process>
```

Processo BPEL do Shipper

```
<?xml version="1.0" encoding="UTF-8"?>

<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:tns="http://meec.ist.utl.pt/cenario52" name="cprocess52-bookstore-Shipper"
         targetNamespace="http://meec.ist.utl.pt/cenario52" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink name="plbs" partnerLinkType="tns:pltbs" myRole="pltrnbs-s" partnerRole="pltrnbs-b"/>
    <partnerLink name="plcs" partnerLinkType="tns:pltcs" myRole="pltrncs-s" partnerRole="pltrncs-c"/>
    <partnerLink name="plps" partnerLinkType="tns:pltps" myRole="pltrn4ps-s" partnerRole="pltrnps-p"/>
    <partnerLink name="ccpl" partnerLinkType="tns:plt-Shippercc" partnerRole="pltrncc"
                 myRole="pltrnShipper"/>
  </partnerLinks>
  <partners>
    <partner name="Bookstore">
      <partnerLink name="plbs"/>
    </partner>
    <partner name="Customer">
      <partnerLink name="plcs"/>
    </partner>
    <partner name="Publisher">
      <partnerLink name="plps"/>
    </partner>
    <partner name="cc">
      <partnerLink name="ccpl"/>
    </partner>
  </partners>
  <variables>
    <variable name="vs1" messageType="tns:msgOrder"/>
  </variables>
  <sequence name="seq-before-sc0">
    <receive name="notify-sc0" partnerLink="plbs" portType="ntfScpPT" operation="ntfScpOp" variable=""/>
    <scope name="sc0">
      <faultHandlers>
        <catch faultName="forcedTermination">
          <sequence>
            <receive name="Faulted:sc0" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                     variable=""/>
            <scope name="internalScope:sc0">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Fault-done:sc0" partnerLink="ccpl" portType="ccpt" operation="FaultDoneOp"
                        inputVariable=""/>
                <receive name="Fault-done:sc0" partnerLink="ccpl" portType="ccpt" operation="FaultDoneOp"
                        variable=""/>
              </sequence>
            </scope>
          </sequence>
        </catch>
        <catchAll>
          <sequence>
            <receive name="Faulted-DefaultFH:sc0" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                     variable=""/>
            <scope name="sc0:internalScope">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Compensate:sc0" partnerLink="ccpl" portType="ccpt" operation="CompensateOp"
                        inputVariable=""/>
                <receive name="Compensate:sc0" partnerLink="ccpl" portType="ccpt" operation="CompensateOp"
                        variable=""/>
                <compensate/>
                <receive name="Compensated:sc0" partnerLink="ccpl" portType="ccpt"
                        operation="CompensatedOp" variable=""/>
                <throw/>
              </sequence>
            </scope>
```

```
      </sequence>
    </catchAll>
  </faultHandlers>
  <compensationHandler name="default-compHandler">
    <sequence>
      <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
               variable=""/>
      <scope name="internalScope-ch:sc0">
        <eventHandlers>
          <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
            <throw faultName="msg-fault.value"/>
          </onMessage>
        </eventHandlers>
        <sequence name="seq-internal">
          <compensate/>
          <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" inputVariable=""/>
          <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                  operation="ChCompensatedScOp" variable=""/>
        </sequence>
      </scope>
    </sequence>
  </compensationHandler>
  <eventHandlers>
    <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
      <throw faultName="msg-fault.value"/>
    </onMessage>
  </eventHandlers>
  <sequence name="seq-scope-sc0">
    <invoke name="Register:sc0" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
            inputVariable=""/>
    <receive name="Registered:sc0" partnerLink="ccpl" portType="ccpt" operation="RegisteredScopeOp"
            variable=""/>
    <receive name="send-bs_req_shipment" partnerLink="plbs" portType="tns:ptbs"
            operation="op-bs_req_shipment" variable="vs1" createInstance="yes"/>
    <switch name="Switch2">
      <case condition="shipper-decline">
        <sequence name="seq-throw-throw2">
          <invoke name="Fault:sc0:f2" partnerLink="ccpl" portType="ccpt" operation="ccThrowOp"
                  inputVariable=""/>
          <wait for="forever"/>
        </sequence>
      </case>
    </switch>
    <invoke name="Create:sc3:sc0" partnerLink="ccpl" portType="ccpt" operation="CreateScopeOp"
            inputVariable=""/>
    <receive name="Created:sc3" partnerLink="ccpl" portType="ccpt" operation="CreatedScopeOp"
            variable=""/>
    <scope name="sc3">
      <faultHandlers>
        <catch faultName="forcedTermination">
          <sequence>
            <receive name="Faulted:sc3" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                    variable=""/>
            <scope name="internalScope:sc3">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Fault-done:sc3" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" inputVariable=""/>
                <receive name="Fault-done:sc3" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" variable=""/>
              </sequence>
            </scope>
          </sequence>
        </catch>
        <catchAll>
          <sequence>
            <receive name="Faulted-DefaultFH:sc3" partnerLink="ccpl" portType="ccpt"
                    operation="Faulted" variable=""/>
            <scope name="sc3:internalScope">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
```

```
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Compensate:sc3" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" inputVariable=""/>
              <receive name="Compensate:sc3" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" variable=""/>
              <compensate/>
              <receive name="Compensated:sc3" partnerLink="ccpl" portType="ccpt"
                      operation="CompensatedOp" variable=""/>
              <throw/>
            </sequence>
          </scope>
        </sequence>
      </catchAll>
    </faultHandlers>
    <compensationHandler name="default-compHandler">
      <sequence>
        <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                 variable=""/>
        <scope name="internalScope-ch:sc3">
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                       variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
            <compensate/>
            <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" inputVariable=""/>
            <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" variable=""/>
          </sequence>
        </scope>
      </sequence>
    </compensationHandler>
    <eventHandlers>
      <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
        <throw faultName="msg-fault.value"/>
      </onMessage>
    </eventHandlers>
    <sequence name="seq-scope-sc3">
      <invoke name="notify-sc3" partnerLink="plbs" portType="ntfScpPT" operation="ntfScpOp"
              inputVariable="varntf"/>
      <invoke name="send-sb_confirm" partnerLink="plbs" portType="tns:ptsb"
              operation="op-sb_confirm" inputVariable="vs1"/>
      <invoke name="Completed:sc3" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              inputVariable=""/>
      <receive name="Completed:sc3" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              variable=""/>
    </sequence>
  </scope>
  <receive name="notify-sc4" partnerLink="plps" portType="ntfScpPT" operation="ntfScpOp"
           variable=""/>
  <scope name="sc4">
    <faultHandlers>
      <catch faultName="forcedTermination">
        <sequence>
          <receive name="Faulted:sc4" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                   variable=""/>
          <scope name="internalScope:sc4">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Fault-done:sc4" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" inputVariable=""/>
              <receive name="Fault-done:sc4" partnerLink="ccpl" portType="ccpt"
                      operation="FaultDoneOp" variable=""/>
            </sequence>
          </scope>
        </sequence>
      </catch>
```

```
      <catchAll>
        <sequence>
          <receive name="Faulted-DefaultFH:sc4" partnerLink="ccpl" portType="ccpt"
                   operation="Faulted" variable=""/>
          <scope name="sc4:internalScope">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <invoke name="Compensate:sc4" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" inputVariable=""/>
              <receive name="Compensate:sc4" partnerLink="ccpl" portType="ccpt"
                      operation="CompensateOp" variable=""/>
              <compensate/>
              <receive name="Compensated:sc4" partnerLink="ccpl" portType="ccpt"
                      operation="CompensatedOp" variable=""/>
              <throw/>
            </sequence>
          </scope>
        </sequence>
      </catchAll>
    </faultHandlers>
    <compensationHandler>
      <sequence>
        <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                 variable=""/>
        <scope name="internalScope-ch:sc4">
          <eventHandlers>
            <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                       variable="msg-fault">
              <throw faultName="msg-fault.value"/>
            </onMessage>
          </eventHandlers>
          <sequence name="seq-internal">
            <invoke name="send-sp_return_book" partnerLink="plps" portType="tns:ptsp"
                    operation="op-sp_return_book" inputVariable="vs1"/>
            <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" inputVariable=""/>
            <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                    operation="ChCompensatedScOp" variable=""/>
          </sequence>
        </scope>
      </sequence>
    </compensationHandler>
    <eventHandlers>
      <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
        <throw faultName="msg-fault.value"/>
      </onMessage>
    </eventHandlers>
    <sequence name="seq-scope-sc4">
      <invoke name="Register:sc4" partnerLink="ccpl" portType="ccpt" operation="RegisterScopeOp"
              inputVariable=""/>
      <receive name="Registered:sc4" partnerLink="ccpl" portType="ccpt"
              operation="RegisteredScopeOp" variable=""/>
      <receive name="send-ps_send_book" partnerLink="plps" portType="tns:ptbs"
              operation="op-ps_send_book" variable="vs1"/>
      <invoke name="Completed:sc4" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              inputVariable=""/>
      <receive name="Completed:sc4" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
              variable=""/>
    </sequence>
  </scope>
  <invoke name="Create:sc5:sc0" partnerLink="ccpl" portType="ccpt" operation="CreateScopeOp"
          inputVariable=""/>
  <receive name="Created:sc5" partnerLink="ccpl" portType="ccpt" operation="CreatedScopeOp"
          variable=""/>
  <scope name="sc5">
    <faultHandlers>
      <catch faultName="forcedTermination">
        <sequence>
          <receive name="Faulted:sc5" partnerLink="ccpl" portType="ccpt" operation="Faulted"
                   variable=""/>
          <scope name="internalScope:sc5">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
```

```
                        variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Fault-done:sc5" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" inputVariable=""/>
                <receive name="Fault-done:sc5" partnerLink="ccpl" portType="ccpt"
                        operation="FaultDoneOp" variable=""/>
              </sequence>
            </scope>
          </sequence>
        </catch>
        <catchAll>
          <sequence>
            <receive name="Faulted-DefaultFH:sc5" partnerLink="ccpl" portType="ccpt"
                     operation="Faulted" variable=""/>
            <scope name="sc5:internalScope">
              <eventHandlers>
                <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                           variable="msg-fault">
                  <throw faultName="msg-fault.value"/>
                </onMessage>
              </eventHandlers>
              <sequence name="seq-internal">
                <invoke name="Compensate:sc5" partnerLink="ccpl" portType="ccpt"
                        operation="CompensateOp" inputVariable=""/>
                <receive name="Compensate:sc5" partnerLink="ccpl" portType="ccpt"
                         operation="CompensateOp" variable=""/>
                <compensate/>
                <receive name="Compensated:sc5" partnerLink="ccpl" portType="ccpt"
                         operation="CompensatedOp" variable=""/>
                <throw/>
              </sequence>
            </scope>
          </sequence>
        </catchAll>
      </faultHandlers>
      <compensationHandler>
        <sequence>
          <receive name="chCompensate-sc" partnerLink="ccpl" portType="ccpt" operation="Compensate-sc"
                   variable=""/>
          <scope name="internalScope-ch:sc5">
            <eventHandlers>
              <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp"
                         variable="msg-fault">
                <throw faultName="msg-fault.value"/>
              </onMessage>
            </eventHandlers>
            <sequence name="seq-internal">
              <receive name="send-cs_return_book" partnerLink="plcs" portType="tns:ptcs"
                       operation="op-cs_return_book" variable="vs1"/>
              <invoke name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                      operation="ChCompensatedScOp" inputVariable=""/>
              <receive name="chCompensated-sc" partnerLink="ccpl" portType="ccpt"
                      operation="ChCompensatedScOp" variable=""/>
            </sequence>
          </scope>
        </sequence>
      </compensationHandler>
      <eventHandlers>
        <onMessage partnerLink="ccpl" portType="ccpt" operation="ThrowFaultOp" variable="msg-fault">
          <throw faultName="msg-fault.value"/>
        </onMessage>
      </eventHandlers>
      <sequence name="seq-scope-sc5">
        <invoke name="notify-sc5" partnerLink="plcs" portType="ntfScpPT" operation="ntfScpOp"
                inputVariable="varntf"/>
        <invoke name="send-sc_ship_book" partnerLink="plcs" portType="tns:ptsc"
                operation="op-sc_ship_book" inputVariable="vs1"/>
        <invoke name="Completed:sc5" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                inputVariable=""/>
        <receive name="Completed:sc5" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                 variable=""/>
      </sequence>
    </scope>
    <invoke name="send-sb_notify" partnerLink="plbs" portType="tns:ptsb" operation="op-sb_notify"
            inputVariable="vs1"/>
```

```
        <invoke name="Completed:sc0" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                inputVariable=""/>
        <receive name="Completed:sc0" partnerLink="ccpl" portType="ccpt" operation="CompletedOp"
                variable=""/>
      </sequence>
    </scope>
  </sequence>
</process>
```