De Processos Interorganizacionais CBPEL para Processos BPEL

António Teófilo
ISEL-DEETC
R. Conselheiro Emídio Navarro, 1, 1949-014 Lisboa
Tel: +351.218.317.211, Fax: +351.218.317.114
ateofilo@deetc.isel.ipl.pt

Alberto Silva
IST/INESC-ID
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
Tel: +351.21.3100307 Fax: +351.21.3145843
alberto.silva@acm.org

Resumo: A Common Business Process Execution Language (CBPEL) é uma nova especificação para a descrição de processos interorganizacionais descritos de forma global e única, e baseados em Web Services. A grande vantagem da utilização de uma descrição global e única consiste na possibilidade da geração automática dos processos das organizações participantes. Essa automatização tem por principal objectivo a redução do risco da concepção de processos interorganizacionais inconsistentes. Contudo até este momento não existe nenhum trabalho que apresente uma metodologia para a geração automática dos processos participantes. Este trabalho visa precisamente preencher essa lacuna promovendo uma reflexão sobre essa transformação, e apresentando um conjunto de regras para a sua automatização. A linguagem escolhida como destinatário da transformação foi a Business Process Execution Language (CBPEL) uma vez que é a linguagem base da linguagem CBPEL.

1 Introdução

O amadurecimento das tecnologias relacionadas com a descrição e execução de processos, e das tecnologias promotoras da interoperabilidade sobre a Internet, baseadas em XML tem proporcionado um recente interesse da comunidade científica sobre a área dos processos de negócio interorganizacionais. Esta área caracteriza-se pela descrição, execução e controlo de processos participados por diferentes organizações, e onde não existe um controle centralizado da sua execução.

Uma forma expedita de formalizar os processos interorganizacionais consiste na utilização de uma descrição global e única, contendo as actividades acordadas entre todos os participantes, unidas num único fluxo. São exemplos deste tipo de descrição as linguagens: WS-CDL (*Web Services - Choreography Description Language*) [1]; e CBPEL (*Common Business Process Execution Language*) [2]. Dessa descrição global e única dos processos interorganizacionais – que representa o contrato global acordado entre os participantes – pode-se extrair as componentes das várias organizações participantes, correspondendo a cada uma, a sua parte nesse contracto global. Cada organização, por sua iniciativa, pode modificar a sua parte nesse contracto global, acrescentando actividades privadas, desde que preserve as características do fluxo original extraído do processo global [3]. Em [4] são identificadas três regras de alteração sobre Redes de Petri: adição de um ciclo; adição de uma qualquer rede em sequência; e adição de uma qualquer rede em paralelo.

Este artigo aborda a extracção dos processos participantes, partindo de um processo global descrito na linguagem CBPEL, lançando uma reflexão sobre essa transformação, e apresentando um conjunto de regras para a sua automatização. Os processos individuais dos participantes são descritos na linguagem BPEL

In Actas das 3ª Terceiras Jornadas de Engenharia de Electrónica e Telecomunicações e de Computadores (JETC'05)

(*Business Process Execution Language for Web Services*) [5]. Essa linguagem foi a escolhida por ser a linguagem base da linguagem CBPEL, e portanto ser de forma natural uma primeira escolha.
Por uma questão de falta de espaço apresentamos seguidamente as regras de transformação a aplicar a um processo CBPEL de modo a gerar os processo BPEL participantes.

2 Transformação CBPEL para BPEL

De modo a evitar dúvidas serão utilizados na transformação os prefixos cbpel: e bpel: de modo a clarificar a identificação da linguagem utilizada. A transformação utiliza um pseudo-código baseado em conceitos vindos do DOM4J [6] dado que foi o *framework* utilizado no desenvolvimento. No início o processo CBPEL é lido para memória gerando-se uma árvore em que cada nó descreve um elemento do processo. Por cada participante aplica-se a transformação a uma cópia dessa árvore. A transformação inicia-se com o processamento do elemento cbpel:process, que depois prossegue para restantes actividades de forma hierárquica. Sendo primeiro aplicado uma acção de eliminação das actividades que não estão relacionadas com o participante em questão (função exists_partner), e em segundo aplicada a efectiva transformação da actividade na linguagem CBPEL para a(s) correspondente(s) em linguagem BPEL (função process). Por limitações de espaço apenas são referidos os aspectos mais relevantes e relativos os seguintes elementos cbpel: process, send, sequence, flow, while e switch.

2.1 Construção: cbpel:process

```
[cbpel:process] void process (bool instantiated, node bpelNode) {
  node PN = createNode(“process”, this);
  bpelNode.insertNode(SN);
  // partnerLinks
  // partners
  // variables
  // correlationSets
  activity.exists_partner(true); // clean
  activity.process(instantiated, PN);
}
```

Figura 1: [cbpel:process] Função process

A transformação do elemento cbpel:process resulta num elemento bpel:process. O elemento cbpel:process contém definições de: parceiros envolvidos (partners), ligações entre operações e o WSDL (partnerLinks), de variáveis existentes (variables), e conjuntos de correlação (correlationSets) existentes. Contudo por falta de espaço as respectivas transformações são omitidas. O processamento deste elemento limita-se ao tratamento da sua única actividade e inicia-se com a eliminação das suas sub-actividades que não estão relacionadas com o participante em questão, e prossegue com a transformação de toda a árvore de actividades. Na figura 1 encontra-se o pseudo-código da transformação deste elemento.

2.2 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 a um elemento bpel:invoke ou bpel:receive respectivamente. O

pseudo-código da transformação encontra-se presente na figura 2. 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 2: [cbpel:send] Funções exists, process e onMessage

2.3 Actividade: cbpel:sequence

A actividade cbpel:sequence é a actividade que modela a execução das suas actividades internas executando-as em série. A sua transformação para a linguagem BPEL resulta num elemento bpel:sequence. Um participante não instanciado ou é instanciado com uma interacção ao nível da sequência, podendo nesse caso possuir várias actividades, ou então é instanciado dentro uma outra actividade, e nesse caso só pode existir dentro dessa actividade. O pseudo-código da transformação encontra-se presente na figura 3.

```
[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(nactivities > 1) ERRO: actividades acima da instanciação
        activity[0].process(false, bpelNode);
      return;
    }
    else
      if (!activity[0].toPartner.equals(partner))
        ERRO: erro na instanciação
  }
  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 3: [cbpel:sequence] Funções exists, process e onMessage

2.4 Actividade: cbpel:flow

A actividade cbpel:flow é a actividade que modela a execução das suas actividades internas executando-as em paralelo. Limitamos a funcionalidade deste elemento exigindo que os ramos paralelos sejam independentes, ou seja, não é possível a utilização do equivalente a bpel:link em CBPEL. A sua transformação para a linguagem BPEL resulta num elemento bpel:flow. Por uma questão prática limitamos que um participante que não esteja instanciado, só pode existir apenas num único ramo paralelo. Assim evitamos a construção de vários processos para um mesmo participante. O pseudo-código da transformação encontra-se presente na figura 4.

```
[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 {
    node FN = createNode(“flow”, this);
    bpelNode.insertNode(FN);
    for(int i=0; i<nactivities; ++i) activity[i].process(true, FN);
}.}

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

Figura 4: [cbpel:flow] Funções exists, process e onMessage

2.5 Actividade: cbpel:while

A actividade cbpel:while é a actividade que modela a execução cíclica da sua única actividade interna. Um dos participantes tomará o controle da execução do ciclo, designando-se de executor. Para esse participante o elemento cbpel:while é transformado num elemento bpel:while e a sua actividade interna processada normalmente. Para os restantes participantes, que são passivos face à decisão do primeiro, eles só sabem que existe mais uma iteração quando receberem uma mensagem que assim o indique. Contudo 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 tanto a primeira mensagem dentro do ciclo como a primeira mensagem fora do ciclo. Não é forçoso poder receber apenas uma mensagem, tanto dentro como fora do ciclo contudo essa é uma simplificação adoptada. A transformação do elemento cbpel:while para os restantes participantes (exceptuando o executor) será por um bpel:while e por um bpel:pick. Mas como o controle do ciclo é automaticamente terminado pela recepção da mensagem fora do ciclo então pode-se adicionar uma variável de controle de execução do ciclo, como pode ser observado no pseudo-código da transformação que

se encontra presente na figura 5. Os participantes instanciados dentro do while, apenas terão um processo, mas que poderá ser executado múltiplas vezes.

```
[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_executor()) {
      node WN = createNode(“while”, this);
      bpelNode.insert(WN);
      activity.process(true, WN);
    }
    else {
      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 5: [cbpel:while] Funções exists, process e onMessage

2.6 Actividade: cbpel:switch

A actividade cbpel:switch é a actividade que modela uma decisão múltipla. Novamente um dos participantes tomará o controle da decisão, designando-se de executor. Para esse participante o elemento cbpel:switch é transformado num elemento bpel:switch e as suas actividades internas cbpel:case são transformadas em bpel:case. Para os restantes participantes, que são passivos face à decisão do primeiro, eles só sabem que um ramo da decisão foi escolhido quando receberem uma mensagem que assim o indique. Mas caso um participante não exista em todas as opções da decisão, só saberá que não foi seleccionado quando receber a primeira mensagem possível fora da decisão. Pelo que os participantes que existirem em todos os ramos da decisão, devem esperar em alternativa a primeira mensagem de um desses ramos, aqueles que não existirem em todos os ramos da decisão, devem esperar pela primeira mensagem de cada ramo e mais a primeira mensagem fora da decisão. Mais uma vez a referência somente à primeira mensagem surge por uma questão de simplificação adoptada. A transformação do elemento cbpel:switch para os restantes participantes (exceptuando o executor) será por um bpel:pick e cada cbpel:case corresponderá a um bpel:onMessage, assim como a primeira recepção fora da decisão no caso dos participantes não existirem em todos os ramos da decisão. O pseudo pseudo-código desta transformação encontra-se presente na figura 6. Quando aos participantes não instanciados, e de modo a evitar por 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_executor()) {
      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 executor
      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 6: Funções [cbpel:switch] exists, process e onMessage

3 Conclusões

Este artigo apresenta uma proposta de transformação de processos CBPEL em processos BPEL. Esta proposta contém algumas limitações que devem ser estudadas de modo a definir totalmente as regras de transformação. Numa posterior fase, espera-se que a parte de tratamento de excepções e compensações seja definida na linguagem CBPEL, para se poder conceber as respectivas regras de transformação.

4 Referências

[1] Kavantzas, N., Burdett, D., e Ritzinger, G., "Web Services Choreography Description Language (WSCDL)", version 1.0, April, 2004, http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/

[2] Teófilo, A., e Silva, A., "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

[3] Aalst, W., e Weske, M., "The P2P Approach to Interorganizational Workflows", in K.R. *Proc. of the 13th Int. Conf. on Advanced Information Systems (CAiSE'01)*, Berlin, Germany, 2001

[4] Aalst, W., e BAsten, T., "Inheritance of Workflows: An approach to tackling problems related to change", *Theoretical Computer Science*, 2001

[5] Andrews, T., et al, "Business Process Execution Language for Web Services", Version 1.0, May 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[6] Rademacher, T., et al, DOM4J, http://www.dom4j.org/, 2001