Uma Linguagem para Representação de *Workflow* em XML com Suporte a Evolução e Versionamento de Esquemas

Fábio Zschornack, Nina Edelweiss
Universidade Federal do Rio Grande do Sul (UFRGS)
Instituto de Informática
Caixa Postal 15064 – 91501-970 – Porto Alegre, RS
{fabiozs, nina}@inf.ufrgs.br

Abstract

Workflow management systems are being used by the organizations for the automation of their business processes. Among the various workflow modeling formats, XML arises as a good alternative, allowing information interchange and being an adequate interoperation language. Business process is dynamic, because it can be modified in many ways. Therefore, the corresponding workflow schema must be flexible enough, to reflect the changes that occurred in the business process adequately. This article presents a workflow modeling language, based on XML, that is appropriate to the application of changes on a schema, among other features.

Keywords: XML, workflow modeling, workflow evolution and versioning

Resumo

Sistemas de *workflow* estão sendo utilizados pelas empresas para automatização de seus processos de negócio. Dentre as formas de modelagem de *workflow*, XML surge como uma boa alternativa, pois permite a troca de informações e também é adequada como linguagem para interoperação entre diferentes sistemas. Pelo fato de um processo de negócio ser dinâmico, ou seja, sofrer alterações pelas mais diversas razões, exige-se flexibilidade por parte do esquema de *workflow* correspondente, de tal forma que reflita adequadamente as mudanças ocorridas no processo de negócio. Este artigo apresenta uma linguagem para modelagem de *workflow*, baseada em XML, que, entre outras características, é apropriada para a aplicação de mudanças nos esquemas.

Palavras-chave: XML, modelagem de *workflow*, evolução e versionamento de *workflow*

1 Introdução

Sistemas de *workflow* estão sendo utilizados pelas empresas para a automatização dos seus processos de negócio já há algum tempo. De acordo com [11], um processo de negócio é um conjunto de atividades que coletivamente atingem um objetivo da organização, enquanto que um *workflow* é a automação de processos de negócio, parcial ou totalmente. Atualmente, esses sistemas estão ultrapassando as fronteiras das organizações, principalmente em função do crescimento do comércio eletrônico na *Web*, formado por processos inter-organizacionais [8]. Seja dentro de uma organização ou entre várias delas, a eficiente modelagem de *workflow* é determinante para uma correta execução dos respectivos processos de negócio.

XML (*eXtensible Markup Language*) [15] vem se tornando o formato padrão para a representação e o intercâmbio de dados. No aspecto de representação, pelo fato de permitir a criação de esquemas, XML possibilita o surgimento de novos padrões (SVG, MathML, etc.) que se utilizam de sua sintaxe para a descrição dos dados e do formato específicos. Este aspecto pode ser estendido, analogamente, à tecnologia de *workflow*, permitindo que os processos possam ser descritos numa linguagem que favoreça a troca de informações. Além disso, torna XML uma linguagem adequada para a interoperação de esquemas entre ferramentas diferentes.

Um processo de negócio freqüentemente é modificado de forma a se adequar a novos requisitos. Entre as causas para a realização de tais mudanças estão fatores internos (aumento de desempenho, correção de erros) ou externos (novas leis, novos requisitos de mercado) à organização. Pelo fato de esquemas de *workflow* serem representações de processos de negócio, as mudanças ocorridas nestes devem ser aplicadas também nos esquemas correspondentes (evolução estática). Além disso, um esquema que foi modificado pode ter instâncias em execução, que devem também ser adaptadas para se tornarem compatíveis com o novo esquema (evolução dinâmica) [3].

Este artigo apresenta uma linguagem de representação de *workflow* em XML, com ênfase na viabilidade de evolução e/ou versionamento dos esquemas frente a mudanças ocorridas nos processos de negócio. Na seção 2, são apresentados e discutidos alguns formatos para a modelagem de *workflow* em XML. A seção 3 trata dos aspectos inerentes a modificações de esquemas de *workflow*. A seção 4 introduz a linguagem proposta, adequada para a representação de alterações nos esquemas de *workflow*. A seção 5 apresenta aspectos relativos a modificações em documentos XML, que devem ser levados em consideração quando da modificação em um esquema de *workflow*. Por fim, a seção 6 traz algumas conclusões sobre o artigo e destaca alguns trabalhos futuros.

2 Modelos de *Workflow* em XML

A representação de *workflows* em XML é um tópico recente de pesquisa. Alguns dos mais importantes trabalhos desenvolvidos nesta área são [2, 4, 5, 12, 13].

A linguagem **XPDL** (*XML Process Definition Language*) [13] foi proposta pela WfMC (*Workflow Management Coalition*), entidade que define padrões para a comunidade que trabalha com a tecnologia de *workflow*. Essa linguagem, que está ainda em desenvolvimento, se relaciona com a Interface 1 definida pela WfMC, constituindo-se em um modelo para intercâmbio de processos de negócio entre ferramentas de modelagem diferentes. Já a linguagem **Wf-XML** [12], também definida pela WfMC, se relaciona com a Interface 4, responsável pela interoperabilidade dos processos entre vários sistemas de *workflow*. Para tanto, Wf-XML se utiliza de mensagens, permitindo que os processos possam ser executados de forma encadeada, aninhada ou paralela. Não é objetivo da Wf-XML a modelagem de fluxos de controle, o que é feita pela XPDL. A IBM, por sua vez, criou a **WSFL** (*Web Services Flow Language*) [5]. Esta linguagem é utilizada para a descrição de composições de serviços *Web*, possibilitando a modelagem dos processos de negócio e das interações entre parceiros, geralmente num ambiente inter-organizacional.

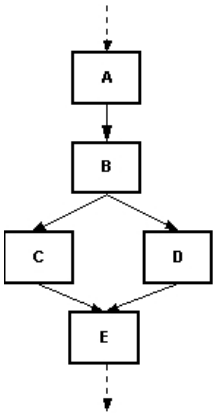
A linguagem **XRL** (*eXchangeable Routing Language*) [2] foi definida de forma a permitir o roteamento de documentos na Internet, visando a troca de informações entre parceiros de comércio eletrônico. Essa linguagem disponibiliza um conjunto de estruturas (*building blocks*) para a definição do roteamento e do fluxo de controle. A semântica de cada estrutura foi definida em termos de redes de Petri. Outro modelo é o **WQM** (*Workflow Query Model*) [4], que, da mesma forma que XRL, utiliza estruturas para a representação do fluxo de controle. No entanto, o enfoque principal dessa proposta foi a definição de consultas para os esquemas e/ou execuções baseadas segundo o modelo. Para tal finalidade, WQM utilizou-se de XML Query Algebra.

Em geral, as linguagens gráficas de modelagem de *workflow* oferecem nodos e transições (ou arcos entre nodos) como elementos principais para a construção de fluxos de controle. No entanto, em modelos XML como os apresentados anteriormente, podem existir outras alternativas de representação. Considerando como critério a forma de representação dos elementos, pode-se dividir os modelos de *workflow* em XML em duas abordagens:

- **representação explícita de transições (RET)** – a característica fundamental das linguagens pertencentes a este grupo é a presença de algum elemento XML que denote explicitamente uma transição, ligando uma atividade a outra diretamente. Para isso, tal elemento deve possuir atributos que indiquem o elemento do qual ele "parte" (origem) e o elemento no qual ele "chega" (destino). Exemplos de modelos baseados nesta abordagem são XPDL e WSFL;

- **representação implícita de transições (RIT)** – os modelos pertencentes a esta abordagem, ao contrário dos modelos RET, não possuem transições explícitas, mas são compostos de estruturas que determinam como será contruído o fluxo de controle do *workflow*. Cada estrutura possui uma semântica própria que, obviamente, deve ser conhecida pela máquina de *workflow*. XRL e WQM são exemplos de modelos que fazem parte dessa abordagem.

A figura 1 ilustra parte de um *workflow*, representado de uma forma gráfica (a), e dois modelos XML equivalentes, um representando transições explicitamente (b) e outro utilizando estruturas para indicar o fluxo de controle (c). As três representações não foram extraídas de modelos específicos, mas foram livremente definidas de forma a permitir melhor compreensão do exemplo.



(a)

```
...
<task id="A"/>
<task id="B"/>
<task id="C"/>
<task id="D"/>
<task id="E"/>
<transition from="A" to="B"/>
<transition from="B" to="C"/>
<transition from="B" to="D"/>
<transition from="C" to="E"/>
<transition from="D" to="E"/>
...
```

(b)

```
...
<seq>
   <task id="A"/>
   <task id="B"/>
   <split>
      <seq>
         <task id="C"/>
      </seq>
      <seq>
         <task id="D"/>
      </seq>
   </split>
   <task id="E"/>
</seq>
...
```

(c)

Figura 1: Representação de *workflow* em XML com transições explícitas (b) e implícitas (c), a partir de um modelo gráfico (a)

Uma análise dos modelos encontrados na literatura nos mostra que as abordagens anteriormente descritas possuem aspectos positivos e negativos. Em alguns critérios de comparação, as abordagens são opostas uma em relação a outra. Tomando como critério a liberdade de modelagem, observa-se que as linguagens que representam as transições explicitamente oferecem mais recursos ao modelador de processos para construir um *workflow*, pois praticamente não impõem restrições quanto às possíveis combinações de estruturas. Este aspecto pode ser considerado como uma desvantagem das linguagens baseadas em RIT, pois estas oferecem somente estruturas pré-definidas para a modelagem dos processos. A figura 2 ilustra um exemplo de fluxo que pode ser modelado segundo a abordagem RET, mas que os modelos de transições implícitas são incapazes de representar, na forma em que se apresenta.

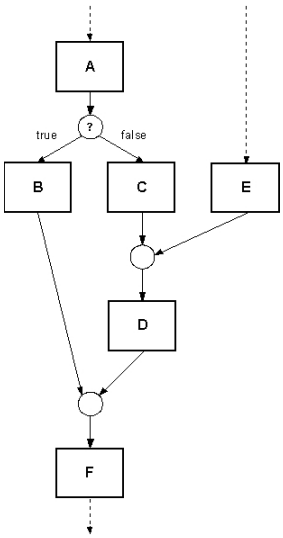


Figura 2: Fluxo que não pode ser modelado pela abordagem RIT

A verificação de correção dos esquemas é outro aspecto no qual as duas abordagens são opostas. Nas linguagens baseadas em RET, tal tarefa é complexa, porque uma DTD, neste caso, tem pouca força para especificar se um documento é correto ou não (determina somente se o documento é válido). Por outro lado, nos modelos que representam implicitamente as transições, a correção é uma tarefa trivial, pois a DTD ou *Schema* especifica como as estruturas devem estar dispostas no documento XML. Assim, o processo de validação serve também como verificação de correção.

3 Evolução e Versionamento de *Workflow*

Conforme destacado na seção 1, mudanças em processos de negócio acarretam em modificações nos esquemas de *workflow* correspondentes. Tais modificações geram sempre um novo esquema, a partir do esquema original que serviu de base. Levando-se em consideração a disponibilidade do esquema original após a aplicação de mudanças, pode-se caracterizar duas situações:

- **evolução** – nesta situação, o esquema sobre o qual foram feitas as alterações é descartado, permanecendo em seu lugar o novo esquema (figura 3a) [3];
- **versionamento** – neste caso, o esquema que sofreu modificações permanece como esquema alternativo, juntamente com o esquema gerado (figura 3b) [7].

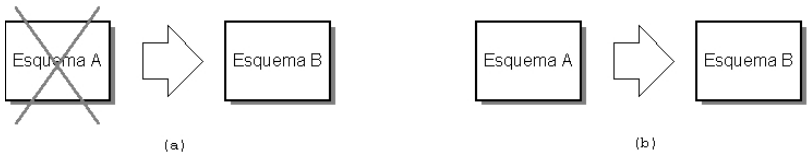


Figura 3: Esquematização de evolução (a) e versionamento (b) de esquemas

Assim sendo, a adaptação das respectivas instâncias deve ser diferente, dependendo do cenário escolhido. No caso de uma evolução, as instâncias do esquema A devem, obrigatoriamente, ser compatibilizadas com o esquema B, para que se adequem às novas regras do processo de negócio. Isso deve ocorrer porque o esquema original não está mais disponível, somente permanecendo o novo. Já em um versionamento, as instâncias podem continuar segundo o esquema A ou migrar em um momento futuro para B, uma vez que o esquema original ainda reflete o que ocorre no mundo real, não deixando de existir a partir da aplicação de mudanças.

Para que um esquema de *workflow* evolua ou seja versionado, algumas operações devem ser aplicadas sobre ele. Em ambas as situações, tais operações são as mesmas, pois, como visto na figura 3, o esquema gerado é o mesmo, independente de ter sido criado a partir de uma evolução ou de um versionamento. Dessa forma, um caso se difere do outro somente no momento em que se especifica o que acontecerá com o esquema de base (eliminação ou manutenção).

Casati et al. [3] especificam algumas primitivas para modificação do fluxo de controle em seu modelo específico. Basicamente, as operações são adição de atividade, adição de roteadores, alteração de fluxos e remoção. Por sua vez, Kradolfer e Geppert [7] definem operações para a criação e remoção de versões de esquemas, entre outras. Reichert e Dadam [9] enfatizam modificações *ad hoc*, diretamente nas instâncias, ao contrário dos trabalhos anteriores, que enfocam nas modificações nos esquemas.

Um modelo de *workflow* deve respeitar duas características importantes, para adequadamente implantar mudanças em seus esquemas:

- **facilidade de alteração** – o modelo deve permitir que mudanças possam ser realizadas sem grandes esforços por parte do modelador, preferencialmente com o auxílio de alguma linguagem;
- **correção** – todo esquema baseado no modelo deve respeitar determinadas regras sintáticas para sua construção.

Na maioria dos modelos de *workflows* estudados, são apresentados critérios de correção que orientam a construção de esquemas. Além disso, alguns incorporam a possibilidade de modificações nos mesmos. No entanto, estes dois aspectos são pouco aprofundados em modelos XML. Sendo assim, o modelo descrito a seguir combina as características desejadas, permitindo que um esquema possa ser alterado e mantendo a correção em qualquer momento, utilizando XML como linguagem para construção de *workflow*.

4 A Linguagem Proposta

A linguagem que será descrita nesta seção foi definida de forma a permitir que a evolução ou o versionamento de um esquema possa ser efetuado, observadas as características propostas no final da seção anterior (facilidade de alteração e correção). Além disso, enfoca outros aspectos, como simplicidade e clareza.

Para que todos esses aspectos pudessem ser levados em consideração, a linguagem baseou-se no trabalho de Kiepuszewski et al. [6], no qual são definidos dois conjuntos de *workflows*: os ***workflows* arbitrários** e os ***workflows* estruturados**. O conjunto dos *workflows* estruturados vem a ser um subconjunto dos primeiros, ou seja, um *workflow* estruturado também é um *workflow* arbitrário, porém o inverso nem sempre é verdadeiro.

Os *workflows* estruturados apresentam uma série de restrições de modelagem, que impedem o surgimento de várias anomalias que podem se fazer presentes nos arbitrários, como por exemplo *deadlocks* e falta de sincronismo entre ramos de execução. Como o próprio nome deixa claro, estes *workflows* são formados por estruturas, que são:

seqüência, **paralelismo**, **decisão** e **repetição**. Estas estruturas podem ser combinadas, porém com a restrição de que devem estar contidas inteiramente umas dentro das outras. Por exemplo, o fluxo ilustrado na figura 2 não pode ser caracterizado como um *workflow* estruturado, pois o ramo que contém a atividade E "entra" na estrutura condicional, o que não é permitido. Nota-se, então, uma semelhança entre os *workflows* estruturados e documentos XML em geral, pois um elemento XML também deve iniciar e finalizar dentro de outro elemento.

Assim sendo, a presente linguagem foi definida de forma a representar, em XML, *workflows* estruturados. A simplicidade é alcançada pelo conjunto de estruturas oferecido que, apesar de serem poucos, oferecem um bom poder de expressão ao modelador. Além disso, tem-se um nível adequado de clareza, pelo fato de os elementos estarem inteiramente contidos uns nos outros, como em uma estrutura hierárquica. Este aspecto também torna as tarefas de alteração e verificação de correção mais facilitadas, como será mostrado mais adiante.

Alguns *workflows* arbitrários podem ser transformados para uma representação equivalente de *workflows* estruturados. Dessa forma, eles podem também ser representados pela linguagem. Sobre transformações de *workflow*, mais detalhes podem ser encontrados em [6].

4.1 Elementos da Linguagem

A linguagem é baseada em algumas estruturas de XRL, descrita na seção 2. Além disso, foram incluídas outras características, de tal forma que os elementos da linguagem contemplem as estruturas de *workflows* estruturados, tornando assim possível a representação dos mesmos em XML. Outro aspecto importante é que somente o fluxo de controle é efetivamente modelado, pois neste se concentra grande parte das mudanças efetuadas sobre um *workflow*. Dessa forma, aspectos como fluxo de dados e modelo organizacional, por exemplo, não são contemplados na linguagem. Essa política foi adotada pois a intenção não é a de se ter uma linguagem completa ou que represente o maior número de estruturas possível, mas trabalhar sobre um conjunto menor e garantir que as mudanças nos esquemas possam ser aplicadas adequadamente sobre este conjunto. Sendo assim, esta linguagem contém estruturas suficientes, sendo apropriada para a aplicação de evolução e versionamento em seus esquemas.

Nas subseções a seguir, serão descritos os elementos e estruturas que compõem a linguagem, bem como serão feitas ligações com os elementos XRL correspondentes, quando existirem.

4.1.1 Activity

O elemento **`activity`** representa a menor unidade de trabalho da linguagem, ou seja, uma atividade propriamente dita. Corresponde ao elemento `task` em XRL. Seu formato geral é

```
<activity name="A"/>
```

É composto de um atributo **`name`**, que indica o nome da atividade. Seu formato é o de um elemento vazio (folha XML), não possuindo filhos em seu interior.

4.1.2 Subprocess

O elemento **`subprocess`** é responsável pela modularização do processo, ou seja, particiona o mesmo em processos menores. Não existe elemento similar em XRL. Seu formato geral é

```
<subprocess name="B" link="subproc.xml"/>
```

O elemento é composto de dois atributos: **`name`**, que indica o nome do subprocesso, e **`link`**, que informa a localização de outro arquivo XML que também é um processo. Da mesma forma que `activity`, `subprocess` também é um elemento vazio.

4.1.3 Process

O elemento **`process`** é o elemento raiz do documento XML, ou seja, dentro dele serão inseridos os outros elementos da linguagem. Com esse elemento, garante-se uma das restrições dos *workflows* estruturados, que exige que um processo tenha somente um ponto de início e um ponto de fim. Este elemento somente aceita um elemento filho imediato, que pode ser uma das estruturas básicas apresentadas a seguir ou um dos elementos "folha" (`activity` ou `subprocess`). Corresponde ao elemento `route` em XRL.

```
<process xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.inf.ufrgs.br/~fabiozs/workflow.xsd"
 name="P">
 ... conteúdo do processo ...
</process>
```

Este elemento é composto do atributo **`name`**, que indica o nome do processo. Além deste, também estão presentes atributos relativos ao *namespace* e *Schema* utilizados, necessários para validação.

4.1.4 Sequence

O elemento **sequence** (de mesmo nome em XRL) é uma das quatro estruturas básicas definidas em [6]. Pode aceitar várias ocorrências de cada elemento, excetuando-se `process`. Os elementos colocados dentro de `sequence` serão executados em seqüência, ou seja, na ordem em que estão dispostos no documento. A figura 4 ilustra um modelo gráfico e um trecho de documento XML equivalente, contendo um elemento `sequence` e, em seu interior, algumas atividades.

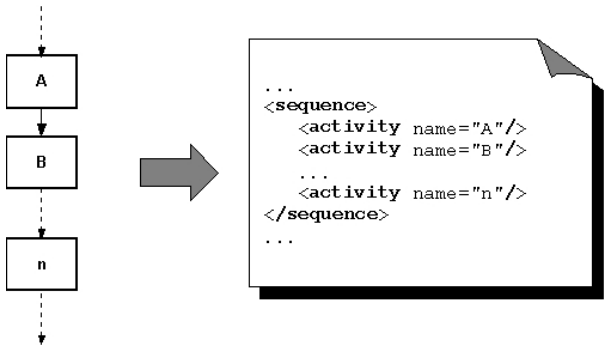


Figura 4: Esquematização do elemento sequence

É importante salientar que, nesta estrutura, a ordem dos elementos é importante, ou seja, invertendo-se a posição de quaisquer dois elementos dentro de `sequence`, tem-se um esquema diferente.

4.1.5 Parallel

O elemento **parallel** é responsável pela execução paralela de seus elementos. Em seu interior, vários elementos podem ser inseridos, assim como acontece com `sequence`. Porém, neste caso, os filhos de `parallel` serão executados em paralelo, não importando a ordem em que estão dispostos no documento. Na linguagem XRL, são disponibilizadas três formas de roteamento paralelo: `parallel_sync`, `parallel_part_sync` e `parallel_no_sync`, sendo que `parallel_sync` é a correspondente a `parallel` na presente linguagem. A figura 5 apresenta um roteamento paralelo gráfico e sua estrutura correspondente, que possui algumas atividades, sendo que todas podem ser executadas em paralelo.

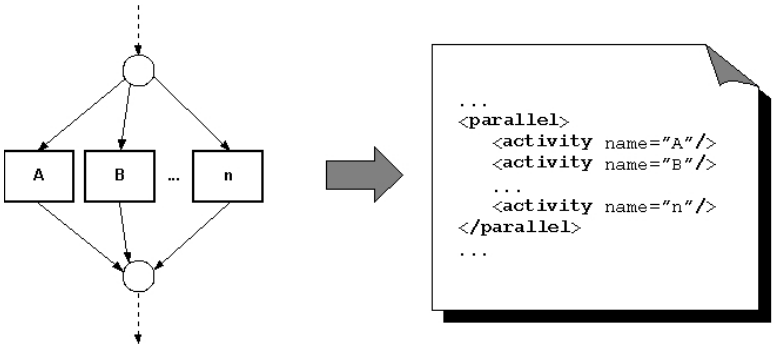


Figura 5: Esquematização do elemento parallel

Este elemento não possui atributos, somente habilitando todos os seus elementos. Ao contrário de `sequence`, no qual uma troca de elementos resulta em um documento diferente, o elemento `parallel` não impõe uma ordem pré-definida para seus filhos, sendo que uma mudança de posições entre eles não resulta em um esquema diferente.

4.1.6 Conditional

A semântica do elemento **conditional** é selecionar um dos dois possíveis ramos existentes, a partir de uma condição que deve ser testada. Sendo assim, esta estrutura possui dois únicos elementos filhos, **true** e **false**, que habilitam o ramo de execução correspondente em caso de condição verdadeira ou falsa, respectivamente. Dentro de `true` ou `false`, somente pode aparecer um dos elementos da linguagem, excetuando-se `process`. XRL disponibiliza uma estrutura `condition` para tal finalidade.

A figura 6 mostra a representação de uma estrutura condicional, de maneira gráfica e no formato da linguagem.

A condição de teste é representada por um atributo, **condition**, que recebe uma expressão booleana como entrada.

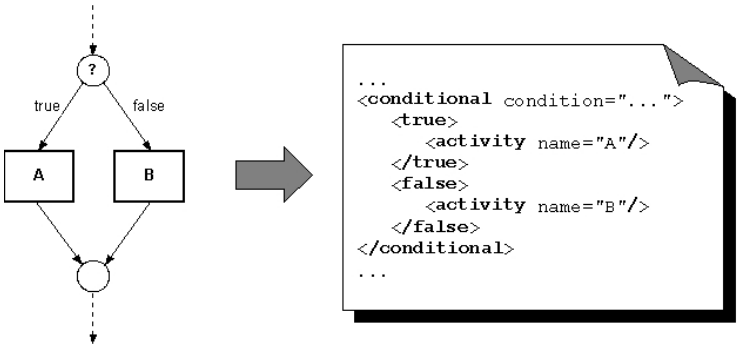


Figura 6: Esquematização do elemento `conditional`

4.1.7 Loop

O elemento **`loop`** (parcialmente representado pelo elemento `while_do` em XRL) representa um laço estruturado, no qual o elemento em seu interior será executado repetidamente. Assim como no elemento `conditional`, há um atributo **`condition`**, que armazena a expressão a ser testada. Além deste, tem-se o atributo **`type`**, cujo valor indica quando a condição será testada, antes ou após a execução do elemento interno. Em XRL, a instrução `while_do` somente permite a execução dos elementos enquanto a condição for verdadeira, ou seja, testando-a no início do laço. Na linguagem aqui apresentada, pode-se também testar a condição ao final do laço.

A figura 7 ilustra a utilização de um elemento `loop`, no qual uma seqüência de atividades será executada repetidamente.

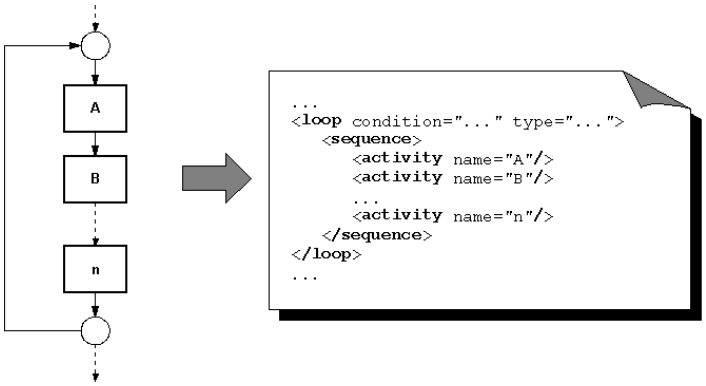


Figura 7: Esquematização do elemento `loop`

4.1.8 Outras Estruturas

As estruturas descritas anteriormente contemplam alguns elementos das ferramentas comerciais. Em [1], foram catalogados vários **padrões de *workflow***, ou seja, estruturas freqüentemente encontradas nas ferramentas, como por exemplo, múltipla escolha (*multi-choice*), junção parcial (*discriminator*), laço arbitrário (*arbitrary cycle*), entre outros padrões. A linguagem descrita nesta seção contempla apenas os padrões básicos (*sequence*, *parallel split*, *synchronization*, *exclusive choice* e *simple merge*), enquanto que os restantes ou são construídos a partir dos básicos ou não possuem representação no atual estágio. A figura 8 apresenta um caso de múltipla escolha, que pode ser equivalentemente transformado utilizando-se as estruturas paralela e condicional.

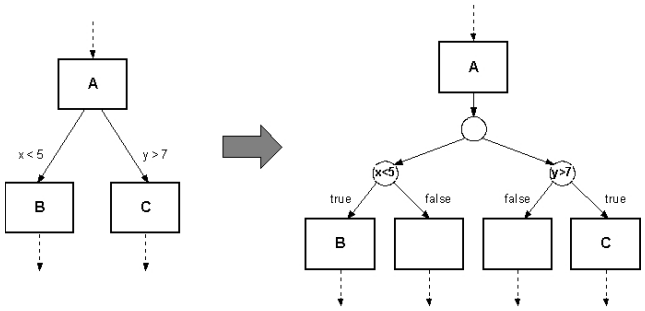
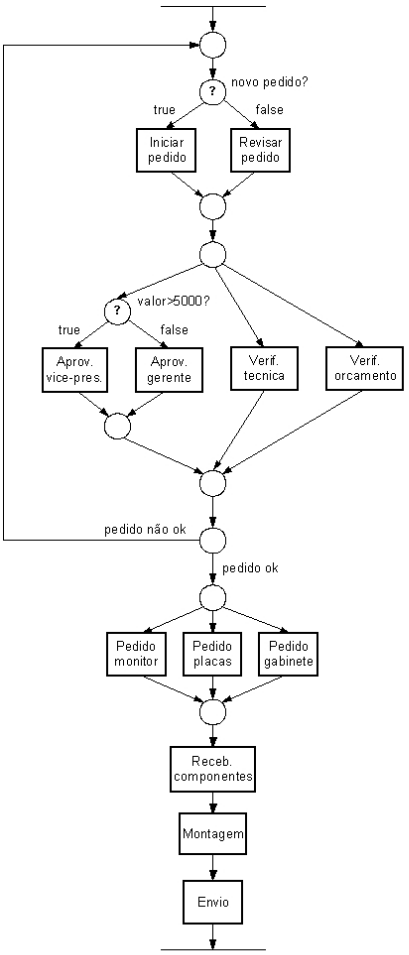


Figura 8: Representação de múltipla escolha [1]

4.2 Exemplo de Modelagem

Nesta subseção, é apresentado um exemplo de utilização da linguagem proposta, conforme a figura 9. Nesta figura tem-se uma representação gráfica (a) e seu esquema XML correspondente (b). O caso representa um processo de compra de um computador, composto basicamente das fases de pedido, aprovação de pedido e verificações, montagem e entrega. Este exemplo foi retirado de [4], sendo que pequenas alterações foram feitas para que se adequasse à sintaxe, especialmente para o elemento `conditional`.




```
<?xml version="1.0" encoding="iso-8859-1"?>
<process xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.inf.ufrgs.br/~fabiozs/
workflow.xsd" name="Compra de PC">
  <sequence>
    <loop condition="pedido = ok" type="repeat_until">
      <sequence>
        <conditional condition="novopedido">
          <true>
            <activity name="Iniciar pedido"/>
          </true>
          <false>
            <activity name="Revisar pedido"/>
          </false>
        </conditional>
        <parallel>
          <conditional condition="valor &gt; 5000">
            <true>
              <activity name="Aprov. vice-pres."/>
            </true>
            <false>
              <activity name="Aprov. gerente"/>
            </false>
          </conditional>
          <activity name="Verif. tecnica"/>
          <activity name="Verif. orcamento"/>
        </parallel>
      </sequence>
    </loop>
    <parallel>
      <activity name="Pedido monitor"/>
      <activity name="Pedido placas"/>
      <activity name="Pedido gabinete"/>
    </parallel>
    <activity name="Receb. componentes"/>
    <activity name="Montagem"/>
    <activity name="Envio"/>
  </sequence>
</process>
```

(b)

Figura 9: Exemplo de um fluxo de controle representado na linguagem proposta

4.3 Análise de Correção

A correção de um esquema de *workflow*, baseado na linguagem proposta nesta seção, está intimamente ligada com as regras de boa formação (*well-formedness*) e de validação (*validation*) do documento XML correspondente. Em outras palavras, é necessário que o documento seja avaliado quanto a estes critérios para que o esquema seja considerado correto.

Um documento é considerado bem-formado se segue a determinadas restrições que normatizam sua forma (aparência) [15]. As restrições se adequam perfeitamente ao formato das estruturas da linguagem, apresentadas anteriormente. Restrições como a presença de um único elemento raiz e o encaixe de um elemento inteiramente dentro de outro são algumas dessas regras respeitadas.

A validação, por sua vez, é feita comparando-se o documento com a DTD ou *Schema* correspondente, a fim de verificar se o mesmo segue regras de construção específicas. A linguagem possui um *Schema* para este propósito, na qual foram inseridas as restrições necessárias. Um exemplo é aplicado sobre o elemento `conditional`, que somente aceita como filhos diretos os elementos `true` e `false`.

Existe um grande número de ferramentas que verificam se um documento XML é bem-formado e/ou válido. Assim, não é preciso construir uma arquitetura nem algoritmos específicos para o processo de correção de esquemas.

A seguir, são apresentados alguns aspectos sobre modificações em documentos XML e, mais especificamente, em esquemas baseados na linguagem definida nesta seção.

5 Modificações em XML

Como XML está se tornando um padrão para a representação de dados e troca de informações na *Web*, inúmeras linguagens estão disponíveis para consulta em documentos. No entanto, um aspecto ainda pouco explorado por essas linguagens é a possibilidade das mesmas realizarem modificações sobre um documento.

Nas próximas subseções, são apresentados alguns tipos de modificações em documentos XML, além de propostas de linguagem de modificação de documentos, com a finalidade de identificar como as alterações em um esquema de *workflow* podem ser representadas no documento XML correspondente, tomando como base a linguagem apresentada na seção 4.

5.1 Operações de Modificação

Um documento XML pode ser alterado de muitas maneiras. Entre as principais modificações tem-se inserção, remoção, troca de nome, etc., seja de elementos, atributos ou conteúdos destes. Além disso, algumas dessas características possuem ordenação (p. ex., elementos), outras não (p. ex., atributos).

Dentre as formas possíveis de modificação de documentos, algumas são importantes, ou seja, fazem sentido, quando o documento representa um esquema de *workflow*. No caso dos esquemas construídos a partir da linguagem apresentada na seção 4, o fluxo de controle é determinado pela disposição das estruturas no documento. Assim sendo, as mudanças em *workflow*s baseados nessa linguagem se concentram em, essencialmente, alterações nos elementos do documento XML. Tem-se, então, as seguintes operações:

- **inserção** de elemento (folha ou composto) em uma determinada localização no esquema;
- **remoção** de um elemento (folha ou composto). No caso de um elemento que contenha filhos, todos também serão removidos;
- **movimentação** de elemento (folha ou composto), alterando sua posição atual no esquema;
- **troca de nome** de elemento (transformação de uma estrutura em outra).

A utilização de uma linguagem para a modificação de documentos é necessária, pois se as alterações forem feitas manualmente, podem resultar em documentos mal formados, o que ocasiona problemas de correção. Por isso, na próxima seção são apresentadas algumas propostas de linguagens de modificação, para auxílio ao modelador na tarefa de aplicação de mudanças em documentos XML e, conseqüentemente, em esquemas de *workflow*.

5.2 Linguagens para Modificação de Documentos XML

Uma linguagem para modificação de documentos XML é um "grande desejo" dos desenvolvedores e profissionais que trabalham com esta tecnologia. No entanto, a W3C, entidade que define padrões para a *Web*, até o momento não incorporou aspectos que permitam alterar documentos na sua linguagem de consulta oficial, a XQuery [16], nem apresentou algo específico para preencher esta lacuna.

Assim sendo, foram lançadas algumas propostas de linguagens de modificação, independentemente da W3C. Uma dessas propostas está descrita em [10]. Os autores desse artigo propõem a extensão da linguagem XQuery para dar suporte a alterações de documentos. A linguagem é aumentada com a inclusão de uma cláusula `UPDATE`, além das tradicionais `FOR`, `LET`, `WHERE` e `RETURN`. Naquela cláusula, são especificadas as alterações a serem feitas no documento.

Além dessa proposta, existe também a linguagem XUpdate, definida por um grupo, o XML:DB [17]. Nessa linguagem, as operações de alteração são especificadas como um documento XML. Além disso, já existem algumas implementações dessa linguagem, permitindo que a mesma possa ser testada.

A partir da análise das propostas, escolheu-se XUpdate como linguagem para a representação das operações de modificação em esquemas de *workflow* em XML pois, além de já possuir implementações, representa as operações no mesmo formato dos "dados" a serem modificados, ou seja, em XML.

5.2.1 Linguagem XUpdate

Como descrito anteriormente, a linguagem XUpdate incorpora as alterações a serem realizadas num documento em outro documento XML. Utilizando-se um mesmo documento, múltiplas modificações podem ser expressas, como por exemplo, a inserção de dois elementos em lugares distintos e a posterior remoção de outro elemento. Para a seleção dos componentes que serão alterados, XUpdate utiliza a linguagem XPath [14], linguagem para expressões de caminho da W3C.

A figura 10 apresenta um documento XUpdate (a), que contém uma operação de inserção de elemento no documento XML (b). O resultado é o documento com o novo elemento inserido (c).

As modificações realizadas com XUpdate garantem que o documento continua bem formado, no entanto, podem torná-lo inválido. Por isso, o documento modificado deve ser novamente validado, para que eventuais problemas possam ser detectados e corrigidos.

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
   xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:insert-after select="/amigos/amigo[1]/aniversario">
      <xupdate:element name="fone">3331-3131</xupdate:element>
   </xupdate:insert-after>
</xupdate:modifications>
```

(a)

```
<?xml version="1.0"?>
<amigos>
   <amigo>
      <nome>Jose Silva</nome>
      <aniversario dia="10" mes="11" ano="1975"/>
      <cidade>Porto Alegre</cidade>
      <pais>Brasil</pais>
   </amigo>
</amigos>
```

(b)

```
<?xml version="1.0"?>
<amigos>
   <amigo>
      <nome>Jose Silva</nome>
      <aniversario dia="10" mes="11" ano="1975"/>
      <fone>3331-3131</fone>
      <cidade>Porto Alegre</cidade>
      <pais>Brasil</pais>
   </amigo>
</amigos>
```

(c)

Figura 10: Modificação de um documento XML com a linguagem XUpdate

5.3 Mudanças no Esquema de *Workflow* com a Linguagem XUpdate

Como destacado na subseção 5.1, para que o fluxo de controle em um esquema de *workflow* representado de acordo com a linguagem proposta seja alterado, as operações devem ser efetuadas tomando-se como parâmetros elementos ou estruturas básicas. Sendo assim, dentre as várias operações oferecidas por XUpdate, as seguintes são utilizadas:

- **`insert-after`** (e **`insert-before`**) – operação que inclui um elemento ou uma estrutura depois (ou antes) de outro elemento;
- **`remove`** – operação que exclui um elemento ou uma estrutura do documento;
- **`rename`** – modifica o nome de um elemento.

Como destacado anteriomente, a aplicação dessas operações pode resultar em um esquema inválido. Por exemplo, a remoção de um elemento obrigatório deve ser acusada após uma revalidação do documento. Quando casos como esse acontecem, a modificação deve ser desfeita integralmente, ou deve-se recuperar o esquema antigo, de forma que se tenha, novamente, o esquema correto.

Qualquer estrutura pode ser inserida num esquema, a exceção de `process`. Uma operação de inserção não causa problemas de validade se o elemento for inserido dentro de `sequence` ou `parallel`, em qualquer posição, pois estes aceitam mais de um filho imediato. Também será aceito dentro de `conditional`, desde que seja um elemento `false` e não haja outro elemento igual a este previamente. Já as estruturas `loop` e `process` não aceitam inserções em seu interior.

No caso da remoção, os elementos vazios (`activity` e `subprocess`) podem ser retirados, desde que hajam outros elementos adjacentes, ou seja, filhos do mesmo elemento pai. Isso ocorre porque, dentro de uma estrutura, é obrigatória a presença de, pelos menos, um elemento. No caso específico de `parallel`, exige-se dois elementos no mínimo. Com relação a remoção de estruturas, a mesma regra é válida. No caso particular de `conditional`, o elemento `false` pode ser removido, porém o elemento `true` nunca pode ser retirado, por ser obrigatório. A única estrutura presente dentro de `process` e `loop` não pode ser removida.

A operação de troca de nome de um elemento somente é permitida nos casos de `sequence` para `parallel` e vice-versa, pois estes elementos são sintaticamente parecidos, não possuindo atributos e podendo conter vários filhos. Com essa operação, a semântica de execução dos filhos se altera. No caso de um elemento `sequence` passar para `parallel`, os elementos internos não serão mais executados em seqüência, mas em paralelo. O efeito contrário também é verdadeiro. Para que outras trocas possam ser realizadas, outras operações devem ser feitas anteriormente, de forma que as estruturas possam se adequar às particularidades do novo elemento.

Outra operação que deve ser analisada é a movimentação de elemento ou estrutura para uma outra posição no documento. Essa operação pode ser considerada uma operação composta, visto que basicamente é formada por uma remoção e uma inclusão. A estrutura que será movida deve, primeiramente, ser copiada para uma variável, através do elemento **`variable`**. A estrutura armazenada pode então ser colocada em outro ponto, por meio do elemento **`value-of`** de XUpdate, responsável por recuperar o valor de uma variável. Os dois elementos especificados atuam em conjunto, armazenando e recuperando estruturas.

A mudança de valores em atributos (no caso dos elementos `activity`, `subprocess`, `process`, `conditional` e `loop`) não é considerada, visto que alterações em atributos não modificam o formato da estrutura e, por consequência, o fluxo de controle.

A figura 11 ilustra um exemplo de modificação do esquema apresentado na subseção 4.2, com a utilização das operações de XUpdate. Num certo momento, decidiu-se que todas as verificações, independentemente de valor do pedido, seriam feitas pelo gerente. A atividade **Aprov. gerente** agora é executada paralelamente a **Verif. tecnica** e **Verif. orcamento**. Deve-se então extrair a atividade de aprovação pelo gerente do roteamento condicional, remover o mesmo e inserir tal atividade diretamente no roteamento paralelo. Na figura 11, tem-se um trecho do esquema original (b), um trecho do esquema modificado (c) e o documento que contém as instruções XUpdate (a).

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
   xmlns:xupdate="http://www.xmldb.org/xupdate">

   <!-- seleção da atividade Aprov. gerente -->
   <xupdate:variable name="gerente" select="/process/sequence/
    loop/sequence/parallel/conditional/false/
    activity[@name='Aprov. gerente']"/>

   <!-- remoção da estrutura condicional -->
   <xupdate:remove select="/process/sequence/loop/sequence/
    parallel/conditional"/>

   <!-- inserção de Aprov. gerente antes de Verif. tecnica -->
   <xupdate:insert-before select="/process/sequence/loop/
    sequence/parallel/activity[@name='Verif. tecnica']">
      <xupdate:value-of select="$gerente"/>
   </xupdate:insert-before>

</xupdate:modifications>
```

(a)

```
...
<parallel>
  <conditional condition="valor &gt; 5000">
    <true>
      <activity name="Aprov. vice-pres."/>
    </true>
    <false>
      <activity name="Aprov. gerente"/>
    </false>
  </conditional>
  <activity name="Verif. tecnica"/>
  <activity name="Verif. orcamento"/>
</parallel>
...
```

(b)

```
...
<parallel>
  <activity name="Aprov. gerente"/>
  <activity name="Verif. tecnica"/>
  <activity name="Verif. orcamento"/>
</parallel>
...
```

(c)

Figura 11: Exemplo de modificação em um esquema de *workflow* com a linguagem XUpdate

6 Conclusões e Trabalhos Futuros

Este artigo apresentou uma alternativa para representação de *workflows* em XML apropriada para suportar evolução e versionamento de esquemas. A linguagem proposta se enquadra na abordagem de representação implícita de transições (RIT), que foi considerada a mais adequada para a análise de modificações, pois a correção dos fluxos de controle dos modelos que se baseiam nesta abordagem se torna uma tarefa bastante facilitada. Outro aspecto favorável é a facilidade com que as mudanças podem ser efetuadas no esquema, em função da estruturação que os elementos possuem no documento XML correspondente. As mudanças podem ser efetuadas, através de alguma linguagem que implemente operações de modificação de documentos XML, como por exemplo XUpdate. Foi mostrado que, apesar do grande número de modificações possíveis nos documentos, somente alguns tipos são necessários para a alteração do fluxo de controle de um esquema de *workflow*. As alterações mantém o documento bem formado, porém podem gerar esquemas inválidos, sendo necessária uma revalidação do respectivo documento para a verificação de correção do novo esquema.

Como trabalhos futuros, devem ser definidos critérios para a adequação de instâncias, tanto em uma evolução quanto em um versionamento, tomando como base a linguagem proposta neste artigo. Além disso, será estudada uma extensão para a linguagem XUpdate, de forma a implementar mais facilmente algumas operações de modificação de esquemas de *workflow*.

Agradecimentos

Os autores agradecem a CAPES, pelo apoio financeiro a este trabalho.

Referências

[1] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. Technical report, Eindhoven University of Technology, Eindhoven, 2002. Available at: http://tmitwww.tm.tue.nl/research/patterns. Last access: April 2002.

[2] W.M.P. van der Aalst and A. Kumar. XML Based Schema Definition for Support of Inter-organizational Workflow. Technical report, University of Colorado, Boulder, 2000. Available at: http://spot.colorado.edu/∼akhil/pubs.html. Last access: January 2002.

[3] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data & Knowledge Engineering*, 24(3):211–238, January 1998.

[4] V. Christophides, R. Hull, and A. Kumar. Querying and Splicing of XML Workflows. In *Proc. of the 9th International Conference on Cooperative Information Systems*, pages 386–402, Trento, Italy, September 2001. Berlin, Springer-Verlag.

[5] IBM Corporation. Web Services Flow Language (WSFL), 2001. Available at: http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf. Last access: April 2002.

[6] B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On Structured Workflow Modelling. In *Proc. of the 12th Conference on Advanced Information Systems Engineering*, pages 431–445, Stockholm, Sweden, June 2000. Berlin, Springer-Verlag.

[7] M. Kradolfer and A. Geppert. Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration. In *Proc. of the 4th International Conference on Cooperative Information Systems*, pages 104–114, Edinburgh, Scotland, September 1999. Los Alamitos, IEEE Computer Society Press.

[8] A. Kumar and J.L. Zhao. Workflow Support for Electronic Commerce Applications. *Decision Support Systems*, 32(3):265–278, January 2002.

[9] M. Reichert and P. Dadam. ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, March/April 1998.

[10] I. Tatarinov, Z.G. Ives, A.Y. Halevy, and D.S. Weld. Updating XML. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 413–424, Santa Barbara, USA, May 2001. New York, ACM Press.

[11] Workflow Management Coalition. Terminology & Glossary, 1999. Available at: http://www.wfmc.org. Last access: December 2001.

[12] Workflow Management Coalition. Workflow Standard - Interoperability Wf-XML Binding, 2001. Available at: http://www.wfmc.org. Last access: March 2002.

[13] Workflow Management Coalition. XML Process Definition Language, 2002. Available at: http://www.wfmc.org. Last access: March 2002.

[14] World Wide Web Consortium. XML Path Language (XPath), 1999. Available at: http://www.w3.org/TR/xpath. Last access: May 2002.

[15] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition), 2000. Available at: http://www.w3.org/TR/REC-xml. Last access: March 2002.

[16] World Wide Web Consortium. XQuery 1.0: An XML Query Language, 2002. Available at: http://www.w3.org/TR/xquery. Last access: May 2002.

[17] XML:DB. XUpdate - XML Update Language, 2000. Available at: http://www.xmldb.org/xupdate. Last access: May 2002.