

Active XML Rules

Tomáš Matoušek

referát z článku

Active Rules for XML: A new paradigm for E-services

Angela Bonifati, Stefano Ceri, Stefano Paraboschi

Úvod

Active XML rules (AXR) jsou pravidla pro správu informací uložených ve formátu XML. Cílem autorů AXR je rozšířit jazyk XSLT a jazyk Lorel (dotazovací jazyk pro semistrukturovaná data) o prostředky, kterými by bylo možné reagovat na dynamické změny v XML dokumentech. Těmito prostředky jsou pravidla, která říkají, co se má stát, pokud nastane nějaká změna v daném XML dokumentu. Principiálně jsou tato pravidla analogií triggerů používaných v relačních databázových systémech. Na jednoduchých příkladech si ukážeme, že AXR mohou být efektivně využita pro rychlý vývoj elektronických služeb pracujících s XML. Poté se zaměříme na některé problémy, které je třeba vyřešit při implementaci AXR.

Aktivní pravidla

Aktivní pravidla jsou tvořena událostmi (*events*), podmínkami (*conditions*) a akcemi (*actions*). *Událost* je libovolná změna v existujícím XML dokumentu, přidání nového dokumentu do systému (např. doručení e-mailu obsahujícího XML data na server) nebo vytvoření nového dokumentu. *Podmínka* určuje, zda se má provést *akce*. Podmínka je tvořena dotazem nad XML dokumentem, jehož výsledek určuje její platnost. Je-li výsledkem dotazu prázdná množina, není podmínka splněna a akce se neprovede, jinak se akce provede. *Akce* spočívá v provedení změn v XML dokumentu nebo ve vytvoření nového dokumentu. Navíc může akce vyvolat nějakou metodu systému (lokální nebo vzdálenou¹). Příkladem může být odeslání e-mailu s daným XML obsahem. Podmínky a akce jsou v deklaracích AXR syntakticky propojeny, aby bylo možné se při deklaraci akce jednoduše odkazovat na data, která figurovala v podmínce. Deklarace AXR se tedy skládá ze dvou částí – z části definující událost (označenou klíčovým slovem EVENT) a částí definující podmínku a akci (klíčové slovo COND_ACT). V této druhé části je využito schopností jazyka XSLT definovat šablonu, podle které se generují XML data (podmínka spočívá v porovnání vzoru šablony s daty v dokumentu a akce je definována obsahem elementu šablony), resp. jazyka Lorel definovat dotaz, který v části SELECT definuje akci a v části WHERE definuje podmínku.

Události, které znamenají přidání nebo odebrání atributu nebo elementu resp. změnu jejich obsahu (jméno atributu či elementu se zachová) se nazývají mutační události (*mutation events*). Tyto události jsou označovány primitivy *insert*, *delete* a *update* a jsou vázány na elementy na zadané cestě v dokumentu. Při použití primitiva *update* jsou elementy či atributy, kterých se změna týká, uloženy do proměnných použitelných dále v podmínce nebo akci. Na tyto proměnné je možné aplikovat funkce OLD a NEW pro zjištění jejich stavu před a po změně. Dalšími událostmi mohou být libovolné systémové události, např. *login*, *logout*, apod. stejně jako u triggerů aktivních relačních databází. Pravidla mohou mít také různou granularitu stejně jako u triggerů: mohou se provést jen jednou za

všechny objekty vyhovující dané cestě (*set-oriented*) nebo se mohou provést pro každý objekt zvlášť (*node-oriented*) – analogie relační *tuple-oriented* granularity.

Deklarace pravidel v XSLT

XSLT se skládá ze sady šablon, podle kterých se transformuje zdrojový XML dokument. Každá šablona v XSLT má dvě části: vzor a obsah šablony. XSLT procesor nachází ve zdrojovém dokumentu elementy a atributy odpovídající vzorům a nahrazuje je zpracovaným obsahem šablony. Obsah šablony je procesorem zpracován protože může obsahovat řídicí elementy XSLT (podmínky, cykly apod.), po jejichž aplikaci teprve vzniknou data, která se mají nahradit za vzor. Tato výsledná data mohou, ale nemusí být v XML.

XSLT používá pro definování vzoru cestu podle standardu XPath. Na každý element zdrojového XML dokumentu může být provedena nejvýše jedna transformace. Pokud by jeden element vyhovoval více vzorům z různých šablon, rozhoduje se procesor podle priorit šablon, podle specifičnosti těchto vzorů nebo nakonec podle pořadí šablon, jak jsou uvedeny v XSLT dokumentu.

XSLT nyní doplníme o definici událostí, což si ukážeme na jednoduchém příkladu. Budeme pracovat s dokumentem, který obsahuje seznam cenových nabídek akcií firem (*quotation*):

```
<Quotation>
  <Name>Microsoft</>
  <Opening>$110.00</>
  <CurrentValue>$100.00</>
</Quotation>
```

Definujme AXR, které bude hlídat cenu akcie a pošle uživateli SMS zprávu na mobilní telefon, pokud se hodnota zvětší o daný počet procent. Zadání od uživatele bude vypadat takto:

```
<User>
  <MobilePhone>123-456789</>
  <RaiseFactor>0.1</>
  <QuotationMonitored>Microsoft</>
</User>
```

Pravidlo je definováno následovně:

¹ Vzdálené volání metody lze implementovat např. XML protokolem SOAP.

```

CREATE RULE CheckQuotation
PRIORITY 0
EVENT
  <xsl:variable name="Q" select="Quotation" />
  <rule:update select="$Q/CurrentValue" />
COND_ACT
  <xsl:template match="User">
    <xsl:if test="QuotationMonitored=$Q/Name and
      number($Q/CurrentValue)>number($Q/Opening)*(1+number(RaiseFactor))"
      <xsl:message>
        <sendsms>
          <to>
            <xsl:value-of select="MobilePhone" />
          </to>
          <subject>
            Alert from the Stock Market
          </subject>
          <body>
            <xsl:value-of select="QuotationMonitored" />
            has gone up
            <xsl:value-of select="RaiseFactor" />
          </body>
        </sendsms>
      </xsl:message>
    </xsl:if>
  </xsl:template>

```

V části **EVENT** je definice události pomocí elementu `<rule:update>` specifikuji element, při jehož změně (typu `update`) se provede akce. Navíc je zde uložena cesta k nadřazenému elementu do proměnné *Q*, abychom se mohli v definici akce na tento nadřazený element odkazovat.

V části **COND_ACT** je definována šablona, která je aplikována na každý element `<user>` v seznamu zadání od uživatelů. V této šabloně je test, zda aktuální hodnota sledované cenové nabídky překročila původní cenu *Opening* o více než *RaiseFactor* procent. Pokud ano, způsobí element `<xsl:message>` zaslání zprávy aplikaci, která pracuje s XML daty, a předá jí XML dokument obsažený v elementu `<xsl:message>`. Tento dokument obsahuje jednoduchou zprávu o sledované cenové nabídce.

Z tohoto příkladu je vidět, jak lze události definovat obecně. Jednoduše se do sekce **EVENT** vloží element `<rule:typ_události>`, kde typ události je *insert*, *delete*, *update*, *login*, *logout*, apod. a pomocí atributu *select* se specifikuje, jaké elementy jsou hlídány. Elementů `<rule>` může být použito i více, lze zde také definovat proměnné, jak jsme viděli v příkladu.

Deklarace pravidel v jazyce Lorel

Lorel (Lightweight Object Repository Language) je dotazovací jazyk pro semistrukturovaná data podobný jazyku SQL. Popíšeme zde jen jeho část týkající se především popisu dat dokumentu, další informace o tomto jazyku lze najít např. v [1].

V Lorelu se dotazy zapisují podobně jako v SQL (klíčová slova *select*, *from*, *where*, *delete*, *insert*, apod.). Základní syntaxe je intuitivní. Jediné, co je snad třeba zmínit je syntaxe cest v dokumentu. Lorel pracuje obecně s libovolnými semistrukturovanými daty, ale my ho budeme používat jen pro XML data, proto popis cest přizpůsobíme terminologii XML. Uvedeme nyní příklad XML dokumentu, na kterém vysvětlíme syntax cest jazyka Lorel a poté i rozšíření o události. Následující dokument uchovává informace o zakázkách.

```
<order>
  <number>234567</number>
  <total>$200.00</total>

  <seller>
    <name>Disney</name>
    <url>www.disney.com</url>
    <email>shop@disney.com</email>
  </seller>

  <items>
    <item>
      <item-type>Toys</item-type>
      <name>Mickey Mouse</name>
      <description>puppet</description>
      <size>20cm</size>
      <quantity>1</quantity>
      <list-price>$29.99</list-price>
      <our-price>$19.99</price>
      <your-save>$10.00</your-save>
    </item>
    ...
  </items>
</order>
```

Cesta je složena ze jmen elementů oddělených tečkou, např. `order.items.item.quantity`. V cestě lze místo její části zapsat symbol `%` nahrazující libovolnou (i prázdnou) posloupnost znaků použitelných ve jméně elementu. Např. `order.items.item.%-price` reprezentuje libovolný element libovolné položky, jehož název končí na řetězec „-price“. Symbol `#` nahrazuje libovolný (i prázdný) úsek cesty – posloupnost jmen elementů. Např. cesta `order.#.name` reprezentuje elementy `<name>` nacházející se v elementu `<order>` (v našem příkladu jméno prodejce i jména položek). Do cesty lze vložit také definici proměnné, ale tento obrat nebudeme potřebovat.

Na výše uvedeném příkladu ukážeme, jak lze pomocí AXR implementovat synchronizaci dat v pohledu na dokument (tj. dokument, který vznikne z původního dokumentu jako výsledek nějakého dotazu). Cílem je reagovat na změny v původním dokumentu tak, aby v novém dokumentu (pohledu) byla vždy aktuální data. Pohled bude obsahovat zkrácenou formu zakázek – některé údaje jsou pro jistý účel přebytečné a v pohledu proto nebudou. Jak pohled vznikl nás momentálně nezajímá, zajímá nás jen jeho obsah:

```

<short-order>
  <number>234567</number>
  <total>$200.00</total>

  <items>
    <item>
      <item-type>Toys</item-type>
      <name>Mickey Mouse</name>
      <quantity>1</quantity>
      <price>$19.99</price>
    </item>
    ...
  </items>
</short-order>

```

Předpokládejme, že se objednávky mohou měnit přidáváním, odstraňováním a úpravou položek v nich obsažených a že lze také přidávat nové objednávky nebo je mazat. Sestrojíme pravidlo R_1 reagující na vložení nové objednávky:

```

CREATE RULE R1
PRIORITY 0
EVENT
  INSERT (order O)
COND_ACT
  SELECT XML(short-order:
    SELECT nr, I, total
    FROM O.number nr, O.#.item I, O.total total
    WITH I.#)

```

V části **EVENT** je opět uveden typ události, na který se reaguje, v závorkách je název elementu a definice proměnné O zastupující tento element. V části **COND_ACT** je uveden dotaz, který zkonstruuje element <short-order> tak, že do něj vloží elementy <number>, <total> a všechny elementy <item> a jejich podelementy.

Dalším pravidlem bude pravidlo R_2 , které reaguje na odebrání položky z nějaké zakázky.

```

CREATE RULE R2
PRIORITY 0
EVENT
  DELETE (order.#.item I)
COND_ACT
  UPDATE SI-=I
  FROM short-order S, order O, S.#.item SI
  WHERE O.#.item = I
    AND O.number = S.number
    AND I.name = SI.name

```

Zde je monitorováno smazání položky I z původního dokumentu. Pokud se tak stane, je třeba najít odpovídající položku v pohledu a smazat ji (to dělá operátor -=).

Další pravidla by byla definována analogicky.

Problémy při implementaci AXR

Hlavním problémem je sledování změn XML dokumentu. Nejjednodušší možností, jak sledovat změny, je integrovat jejich detekci přímo do systému DOM a sledovat změny při volání DOM API pro práci s dokumentem. Avšak v heterogenních XML systémech pracuje s dokumentem více programů mnoha různými způsoby, nejen našimi API funkcemi (změny mohou proběhnout také mimo systém). Tedy ne každá změna může být zachycena monitorováním API funkcí. Změny se pak musí zjišťovat porovnáváním nového dokumentu s původním. To je obecný problém, který je řešen algoritmem *XML-diff*, který spáruje element nového dokumentu s odpovídajícím elementem původního dokumentu (v elementu mohlo dojít ke změně, ale sám element nebyl přidán, odebrán ani přejmenován). Pokud element původního dokumentu není spárován, pak byl odstraněn (událost typu *delete*). Pokud element nového dokumentu není spárován, pak byl přidán (událost typu *insert*). Pokud je element spárován, ale jeho obsah se liší od původního, nastane událost typu *update*. Takto lze vytvořit seznam operací (*edit script*), které musí být provedeny, aby se z původního XML dokumentu stal nový.

Dalším problémem jsou konflikty pravidel. Obecně může být vyvoláno několik pravidel najednou. V jakém pořadí se mají pravidla vykonat? Je třeba definovat lineární uspořádání na množině všech pravidel. Toto uspořádání může být uživatelem explicitně nastaveno pomocí klíčového slova *PRIORITY* nebo může být určeno systémem (např. podle času, kdy byla pravidla vytvořena apod.). Uživatelské uspořádání může být částečné a na lineární je doplněno uspořádáním systémovým.

Je-li určeno uspořádání pravidel, pak se pravidla vykonávají podle tohoto uspořádání. Vykonávání pravidla *P* ale může způsobit spuštění jiného pravidla, které má vyšší prioritu (je v daném uspořádání větší). V tom okamžiku je provádění pravidla *P* pozastaveno a je nejdříve vykonáno pravidlo s vyšší prioritou. Teprve, když neexistuje jiné pozastavené pravidlo, které by mělo vyšší prioritu než *P*, pokračuje vykonávání *P*. Podobně je definováno vykonávání triggerů ve standardu SQL3. Existuje ale možnost, že toto vykonávání pravidel nikdy neskončí. Postačující podmínkou konečnosti výpočtu je acykličnost grafu, jehož vrcholy tvoří pravidla a hrany vedou mezi takovými dvěma pravidly, že první může vyvolat druhé. Tento problém byl prostudován již v aktivních relačních databázích a výsledky jsou aplikovatelné i na XML dokumenty.

Když už víme v jakém pořadí pravidla vykonáme, je třeba vyřešit další otázku. Co se stane, když pravidlo vykonané na daném elementu jako první element odstraní? Další pravidla by pak pracovala s elementem, který už neexistuje. Před vykonáním každého pravidla je však možné zkontrolovat, zda je element stále součástí dokumentu a tím taková pravidla eliminovat.

Implementace AXR v XSLT

Systém AXR založený na XML se skládá ze tří komponent: modul na zjišťování rozdílů mezi různými verzemi XML dokumentů (*offline event detector*), XSLT procesor a modul spouštějící události (*rule engine*). První dvě komponenty již existují, stačí je přizpůsobit pro práci s AXR, poslední komponenta musí být nově implementována, ale její složitost není velká. Navíc je možné integrovat detektor změn do DOM pomocí událostí, které jsou definovány v DOM Level 2 a optimalizovat tak detekci změn při použití DOM API.

Je třeba dodat, že není možno použít XSLT procesor pro vyhodnocování konfliktů v pravidlech. XSLT procesor totiž vybírá jen jednu šablonu, která se zpracuje (podle specifčnosti cesty popisující element a poté podle pořadí šablon) a ostatní ignoruje. Neznal by tedy priority dané uživatelem a ani by nezpracovával všechna pravidla. Proto je třeba vytvořit vlastní zpracování pořadí aktivních pravidel a XSLT procesor nechat zpracovávat jen sekce EVENT a COND_ACT těchto pravidel.

Implementace AXR v jazyce Lorel

Implementace v Lorelu může využít přístupy známé z databázových systémů, jelikož dotazy v části COND_ACT mohou být zpracovávány stejně jako jiné dotazy. Je třeba pouze přidat *rule engine* (podobný tomu, který je implementován v aktivních relačních databázích pro triggeru) a *offline event detector (XML-diff)*.

Závěr

Aktivní XML pravidla lze použít pro rychlý vývoj elektronických služeb založených na práci s XML dokumenty pomocí XSLT nebo jazyka Lorel. Definují reakce na změny v XML dokumentech obdobně jako triggeru definují reakce aktivního DBMS na změny provedené v relacích. Aktivní pravidla jsou v mnohém podobná triggerům a lze proto využít znalostí z relačních DBMS. Vynořují se zde ale některé nové problémy, které je třeba řešit a které autoři původního článku pouze naznačili a hodlají se jejich řešením dále zabývat. Autoři již také implementovali systém aktivních pravidel nad systémem Lore.

Literatura

- [0] Active Rules for XML: A new paradigm for E-services
Angela Bonifati, Stefano Ceri, Stefano Paraboschi
- [1] Serge Abiteboul, Dallon Quass, Jason McHugh, Jennifer Widom, Janet Weiner:
The Lorel Query Language for Semistructured Data,
Department of Computer Science, Stanford University