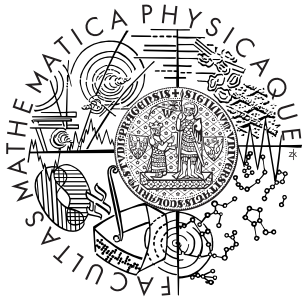
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Ján Kasarda

Konceptuálny návrh webových formulárov

MFF KSI

Vedoucí diplomové práce: Mgr. Martin Nečaský, Ph.D, MFF KSI

Studijní program: Informatika, Softwarové systémy

2009

Na tomto mieste by som chcel poďakovať rodine, ktorá ma pri mojom štúdiu podporovala a pomáhala mi. Moje poďakovanie za cenné rady patrí vedúcemu diplomovej práce Mgr. Martinovi Nečaskému, Ph.D.

Prehlasujem, že som svoju diplomovú prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe dňa
3. februára 2009

Ján Kasarda

Obsah

Název práce: Konceptuálny návrh webových formulárov
Autor: Bc. Ján Kasarda
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalárské práce: Mgr. Martin Nečaský Phd.
e-mail vedoucího: necasky@ksi.mff.cuni.cz

Abstrakt: S nárastom zložitosti webových aplikácií a software obecne rastie aj prínos použitia modelovacých techník. V tejto práci sa sústredime na modelovanie webových formulárov.

Krátkym motivačným príkladom ukážeme výhody modelovania na konceptuálnej úrovni. Zhrnieme existujúce prístupy ku konceptuálnemu modelovaniu, ktoré sú založené na UML a MDA. Predstavíme technológiu XForms. V poslednej časti predstavíme modelovanie webových formulárov na konceptuálnej úrovni a spôsob prekladu konceptuálneho modelu na XForms.

Náš prístup je založený na princípoch MDA. Cieľová doména je v prvom kroku modelovaná na konceptuálnej úrovni. V druhom kroku sú modelované jednotlivé časti bez závislosti na cieľovej technologii. Posledným krokom je generovanie XForms reprezentácie modelovaných formulárov.

Kľúčové slová: konceptuálne modelvanie, XFroms, XML, UML, MDA

Title: Conceptual Design of Web Forms

Author: Bc. Ján Kasarda
Department: Department of Software Engineering
Supervisor: Mgr. Martin Nečaský Phd.
Supervisor's e-mail address: necasky@ksi.mff.cuni.cz

Abstract: As complexity of web applications and software grows, benefits of applying modeling techniques for the development of such applications are increasing as well. In this work, we focus on modeling web forms which are an important part of current web applications.

A short motivation example will demonstrate benefits of modeling web forms at the conceptual level. We will then summarize existing approaches to conceptual modeling based on UML and Model-Driven Architecture (MDA). We will present XForms. Finally, we will discuss how to model web forms at the conceptual level and how to translate conceptual schemas of web forms to XForms representation.

Our approach is based on MDA principles. Firstly, a problem domain is modeled at the conceptual level. Secondly, each form representing a part of the domain is modeled independently of its representation in XForms. The last step is generating XForms representations of the modeled forms.

Keywords: conceptual modeling, XForms, XML, unifed modeling language (UML), model driven architecture (MDA)

Kapitola 1

Úvod

Všadeprítomný internet s pohodlným prístupom k informáciám, prepojenými systémami, existenciou otvorených štandardov a jednoduchým vytváraním obsahu vytvoril množstvo nových príležitostí. Príkladom sú bežnejšie nákupy na internete, bankové prevody či osobné blogy užívateľov. Spoločnou podmienkou pre dosiahnutie týchto benefitov sú ukazovateľe ako čas potrebný na vývoj daného systému alebo nemenej dôležitá cena údržby a následného vývoja.

Dnes najrozšírenejšou technológiou na vytáranie internetových aplikácií je HTML alebo modernejšie XHTML. Obe technológie majú rovnaký základ a filozofiu z čoho vyplývajú aj rovnaké obmedzenia a nedostatky. Nie sú navrhnuté na vytváranie dynamických formulárov, ktoré sú dnes takmer v každom systéme, nepodporujú validáciu zadávaných hodnôt a prenositeľnosť na iné druhy zariadení je prinajmenšom otázna.

Ak sa už aj podarí vytvoriť aplikáciu, ktorá spĺňa všetky potrebné parametre, bude pravdepodobne postavená na najrôznejších frameworkoch, mnohých knižniciach a možno dokonca technológiách. Takáto skladba nepríjemne vplýva cenu úpravy systému. Do úprav je často potrebné zahrnúť aj užívateľov systému, ktorí nie sú vybavený technickými znalosťami na tento proces.

Je preto dobrým začiatkom pri návrhu systémov pracovať nad úrovňou technickej implementácie a vytvoriť *konceptuálny model*, ktorý popisuje modelovaný systém. Popis na tejto úrovni zachytáva najdôležitejšie objekty v systéme ich vlastnosti a spôsob akým sú na sebe závislé. Každá súčasť modelu má vlastnú vizualizáciu. Príkladom dnes veľmi úspešného modelovacieho jazyka je UML. V tomto prípade sú objekty znázorňované ako krabičky s vlastným názvom. Atribúty objektov sú spolu s názvom a typom uvádzané

vo vnútri objektov, ku ktorým patria. Vzťahy medzi objektmi sú vyjadrované rôznymi druhmi čiar a spojníc medzi nimi.

Konceptuálny pohľad na systém sprístupňuje nové možnosti pri vývoji a technickej implementácii. Maximálne množstvo informácií o celom systéme je dostupné na jedinom mieste a to všetko v dobre definovanej štruktúrovanej podobe - pravdepodobne XML. Takýto zdroj je dokonalým odrazovým mostíkom pre generovanie fragmentov potrebných pri vývoji. Môže ísť o kusy zdrojového kódu, konfiguračné súbory, schémy popisujúce dáta či dokonca celé systémy.

Ideálnym cieľovým formátom je XML nakoľko je jednoduché zo zdrojového XML súboru generovať výstupný XML súbor v požadovanom formáte. Pri vývoji internetového informačného systému by mohlo ísť o hotové obrazovky v jazyku XHTML. Tie by museli byt doplnené napríklad o Java Script, pre zabezpečenie dynamického správania. Aj tento Java Script je možné vygenerovať, ale začína to mierne komplikovať vývoj. Ideálnym riešením je generovať jediný XML súbor, ktorý by obsahoval zobrazenie, správanie aj validáciu vstupného formuláru. Tieto podmienky spĺňa technológia XForms [2] dokonale. Navyše pridáva niektoré nové možnosti ako je napríklad komunikácia s webovými službami [1].

Modelovanie s následným generovaním častí informačného systému prináša reálne výhody pri implementácii ako je napríklad skoré zahrnutie užívateľov do vývojového cyklu, zníženie opakovanej práce spôsobenej chyba v designe či nedorozumeniami a mnohé iné. Ak by sme túto ideu dotiahli do maximálneho stavu, dalo by sa povedať, že vlastníci jednotlivých častí systému by si jeho obrazovky vytvorili sami. Ako prvý krok by popísali fungovanie svojej konkrétnej časti. V druhom kroku by boli vytvorené formuláre a obrazovky.

1.1 Príspevok

V tejto práci predstavíme možnosť použitia konceptuálneho modelovania pri vývoji internetových formulárov. Konceptuálnu stránku modelovania vysvetlíme na základe práce popisujúcej jazyk XSEM [17].

XSEM využíva dvojúrovňové modelovanie. Vyšší stupeň abstrakcie predstavuje XSEM-ER, kde je reálny svet modelovaný bez ohľadu na budúci formát dát či implementáciu. Takto vytvorené modely sú v druhom kroku prevádzané do úrovne zohľadňujúcej technologické riešenie. Pre zápis sa používa XSEM-H, ktorý je navrhnutý pre modelovanie XML dokumentov.

Model v jazyku XSEM-H teda popisuje schému XML dokumentu. Obsahuje dostatok informácii napríklad na validovanie dokumentov. Pre vytváranie a úpravu týchto dokumentov je potrebné poskytnúť užívateľom primerané nástroje. V našej práci sa preto zameriame na vytváranie vhodných webových formulárov z XSEM-H.

Ako cieľový formát formulárov sme zvolili XForms [2] konzorcia W3C[14]. Predstavíme dvojfázový algoritmus prevodu XSEM-H do XForms, ktorý rekurzívnym spôsobom vybuduje kompletný formulár. Uvedieme, kde je vhodné algoritmus doplniť užívateľom, aby bol cieľový výstup čo najviac použiteľný. Rozoberieme problémové konštrukcie, ktoré sú v technológii XForms ťažko reprezentovateľné.

V poslednej časti preberieme ukážkovú implementáciu nástroja, ktorý umožní popisovaný prevod. Hlavným cieľom je vytvoriť prototyp generovania. Nakoľko model nemusí popisovať formulár jednoznačne prevod bude realizovaný poloautomaticky. Nástroj zobrazí štruktúru prevádzaného modelu a umožní vývojárovi jednoduchým spôsobom vytvoriť formulár. Editor umožní určitú formu úpravy formulárov. Interpretácia XForms formulárov a ich vizualizácia je ponechaná na cieľové prehliadače.

1.2 Motivácia

Každá nová metóda má za úlohu priniesť zjednodušenie do každodennej práce, každý nový prístup sa snaží sprístupniť možnosti, ktoré boli doteraz nerealizovateľné. Konceptuálne modelovanie formulárov urýchli a zjednoduší vývoj internetových aplikácií. Možnosti využitia modelovania sú široké. Pre znázornenie a ukážku potencionálnych dát budeme v nasledujúcom texte pracovať s konkrétnym prípadom.

Ako príklad nám poslúži fiktívny poskytovateľ telekomunikačných a dátových služieb. Širokú základňu podnikania tvoria zákazníci, pre ktorých je určené veľké množstvo služieb od najrôznejších notifikácii cez zvýhodnenia až po balíčky. Užívatelia pomocou internetu zadávajú údaje ako adresy a objednávky. Operátor neposkytuje služby iba s využitím vlastných technických možností a preto musí jeho informačný systém komunikovať s množstvom už existujúcich systémov dodávateľov.

Integrácia rôznorodých systémov je takmer ideálna situácia pre nasadenie technológie Web services [1]. Web services (WS) je štandard, ktorý vynikajúco vyhovuje použitiu pri komunikácii počítačových systémov. Aplikácie si tak môžu vymieňať správy vo formáte XML správ. XML formát

je v posledných rokoch jedným z najpoužívanejších formátov na vymieňanie dát. Jednou z hlavných výhod je možnosť XML čítať ako jednoduchý textový formát.

Predpokladajme teda, že informačné systémy dodávateľov sú založené na technológii WS. Dodávatelia už majú informačné systémy implementované a nie je možné žiadať od nich, aby svoje systémy upravovali podľa technických požiadavok nášho operátora. Aj keď všetky spolupracujú na spoločnom cieli a pracujú s logicky rovnakými dátami, nie sú úplne zhodné. Číslo faktúry v XML zo systému účtovníctva môže byť uložené ako element *number* elementu *faktura*, no operátor ho môže ukladať ako atribút *id* elementu *faktura*. Tým vznikajú mierne odlišnosti, ktoré sa navyše môžu meniť aj v čase (napríklad kvôli zmene účtovacieho systému). Konceptuálny model je v tomto prípade veľkou výhodou, pretože je v ňom možné podobné zmeny postihnúť a upraviť potrebné entity. Konceptuálny model obsahuje dostatok informácií na vygenerovanie XML schémy popisujúcej výsledné XML dokumenty.

Zatiaľ mame pokrytú komunikáciu systémov medzi sebou čím sme umožnili operátorovi služby poskytovať. Vytvorili sme pri tom konceptuálny model, dokázali sme vygenerovať schémy popisujúce vymieňané správy a naše systémy komunikujú použitím WS. Vyvinuli sme už určité úsilie na inicializovanie vývojového a testovacieho prostredia, náš vývojový tým je zaškolený na prácu s XML a WS. Ostáva vyvinúť internetové stránky pre komunikáciu s užívateľmi takzvaný front end. Bežnou praxou býva využitie niektorej z technológií front-endov ako je Java, PHP či ASP alebo dokonca ich kombináciu. Tieto webové stránky si vyžiadajú údaje od užívateľa, prevedú ich do XML a odošlú na WS. Prečo teda nekomunikovať priamo vo forme XML aj z front-endových systémov? Jednoduchou odpoveďou bývalo samotné HTML, ktoré takúto možnosť nepodporovalo.

Tento zásadný nedostatok rieši technológia XForms [2], ktorá je navrhnutá ako moderná alternatíva k front endovým technológiám. Jej základom je opäť XML čím je možné odosielať dáta priamo v tomto formáte napr. na webové služby, ponúka pokročilé možnosti validácie vstupov bez použitia prídavných skriptovacích jazykov ako aj dynamické formuláre a stránky.

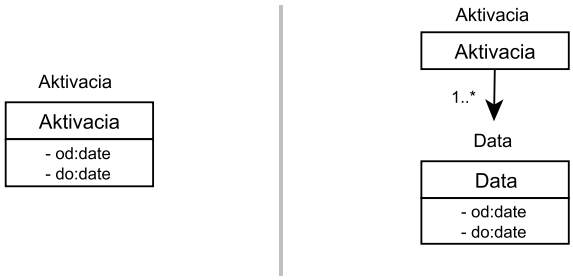
1.2.1 Príklad 1

Na záver uvedieme rozsiahlejší príklad využívajúci výhody XForms v spojení s konceptuálnym modelom. Uvažujme na chvíľu existujúcu službu, ktorou

si môže klient naplánovať aktivovanie roamingu na obdobie dovolenky v predstihu. Roaming sa mu aktivuje automaticky v prvý deň dovolenky, nemusí teda aktivovať roaming dopredu čím rozhodne ušetrí. Operátor má danú službu naimplementovanú dvoma vstupnými políčkami, kam užívateľ môže zadať dátumy kedy má byť roaming aktivovaný a kedy má byť vykonaná deaktivácia. Vďaka XForms užívateľ nemôže odostlať formulár ak nie je zadaný reťazec, ktorý by spĺňal pravidlá dátumu. Dáta sú odoslané vo formáte XML priamo na WS.

Tento prístup pokryje 99% komunikácie. Na internete sa ale nájde mnoho užívateľov, ktorý by mohli pokúsiť XML napísať ručne a poslať ho na WS. Toto XML by nebolo odosielané cez XForms formulár a nemuselo by teda spĺňať formát dátumu. Z konceptuálneho modelu sme ale vygenerovali XML schému popisujúcu validný vstup, ktorou je možné na strane WS prichádzajúce XML overiť. Tento pokus prekonať zabezpečenie by teda bol úspešne odhalený a odpoveďou by bolo oznámenie o chybe.

Jedna z možností ako je môže daný formulár namodelovať v XSEM-H je znázornená príkladom 1.1 vľavo.



Príklad 1.1: Konceptuálny model pre príklady 1 a 2

Element *Aktivacia* obsahuje atribúty *Od* a *Do*, ktoré budú obsahovať potrebné dátumy. Výsledkom prevodu uvedeného modelu bude formulár, ktorý odošle jediný element *Aktivacia*.

Formulár XForms k použitému modelu je uvedený v príklade 1.2 na str. 12. Dáta odoslané takýmto jednoduchým formulárom sú zobrazené na str. 13 v príklade 1.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns:j="default"
 xmlns="http://www.w3.org/1999/xhtml"
 xmlns:xforms="http://www.w3.org/2002/xforms"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ev="http://www.w3.org/2001/xml-events" >
 <head>
  <xforms:model id="Model">
   <xforms:instance id="Aktivacia">
    <Aktivacia xmlns="default" Od="" Do="" />
   </xforms:instance>
   <xforms:bind
          nodeset="instance('Aktivacia')/@Od" type="xsd:date" />
   <xforms:bind
           nodeset="instance('Aktivacia')/@Do" type="xsd:date" />
   <xforms:submission id="id"
           method="put" ref="instance('Aktivacia')"
          action="http://xformstest.org/cgi-bin/showinstance.sh"/>
   <xforms:action ev:event="xforms-ready" />
  </xforms:model>
 </head>
 <body>
  <xforms:group>
   <xforms:label>Aktivacia</xforms:label>
   <xforms:group>
    <xforms:label>Bez mena</xforms:label>
    <xforms:input incremental="true" model="Model" ref="./@Od">
     <xforms:label>Od</xforms:label>
    </xforms:input>
    <xforms:input incremental="true" model="Model" ref="./@Do">
     <xforms:label>Do</xforms:label>
    </xforms:input>
   </xforms:group>
   <xforms:group>
    <xforms:submit submission="id">
     <xforms:label>Potvrd</xforms:label>
    </xforms:submit>
   </xforms:group>
  </xforms:group>
 </body>
</html>
```

Príklad 1.2: Kompletný XForms formulár pre práklad 1

```
<?xml version="1.0" encoding="UTF-8"?>
<Aktivacia
  xmlns="default"
  xmlns:xforms="http://www.w3.org/2002/xforms"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ev="http://www.w3.org/2001/xml-events"
 Od="2004-09-01"
 Do="2009-05-07"/>
```

Príklad 1.3: Data odoslané formulárom 1

1.2.2 Príklad 2

Po určitej dobe používania služby sa ukázalo, že jej implementácia nie je postačujúca, ak si klient potrebuje aktivovať službu na niekoľko rôznych období. XForms umožňujú odosielať iba jediný element, čo znamená, že v tomto rozšírení nie je možné vystačiť si s elementom *Aktivacia*. Jednoduchá zmena konceptuálneho modelu rozšíri rozhranie služby o možnosť zadania viacerých aktivácií. Pridáme preto element *Data*, ktorý dostane obe atribúty *Od*, *Do* a v elemente *Aktivacia* bude umožnený jeho násobný výskyt.

Model XSEM-H sa zmení podľa príkladu 1.1 v pravej časti. Formulár XForms k použitému modelu je uvedený v príklade 1.4 na str.14. Odosielané XML je zobrazené v príklade 1.5.

1.2.3 Príklad 3

Operátor sa po určitom čase mohol rozhodnúť, že uvedie na trh rôzne varianty roamingu. Napríklad zvýhodnený roaming pre Európu či dátový roaming. Tým je potrebné upraviť aj službu na plánovanie roamingu. Klienti si môžu naplánovať viacero roamingov naraz, navyše je potrebné pri samotnom plánovaní zvoliť, o ktorý roaming sa jedná. V našom prístupe teda upravíme konceptuálny model. Z neho bude vygenerovaná nová schéma pre validáciu požiadaviek na strane poskytovateľa, model poslúži aj ako zdroj informácií pri vytváraní formulára XForms. Príjemnou vlastnosťou tohoto procesu je zníženie počtu potrebných programátorov. Zmeny v konceptuálnych modeloch môže vykonávať aj personál, ktorý je viac profilovaný na business či marketing.

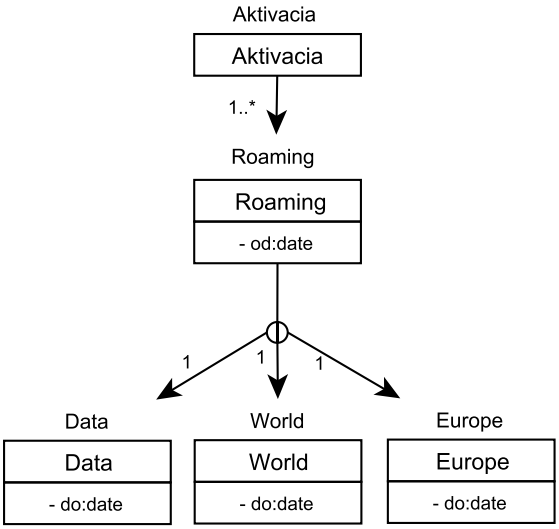
Konceptuálny model v zápise XSEM-H je uvedený v príklade 1.6. Formulár zo str. 16 si detailne rozoberieme v kapitole 2.4 na str. 20

```
<?xml version="1.0" encoding="UTF-8"?>
<html  xmlns="http://www.w3.org/1999/xhtml"  xmlns:j="default"
 xmlns:xforms="http://www.w3.org/2002/xforms"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ev="http://www.w3.org/2001/xml-events" >
 <head>
  <xforms:model id="Model">
   <xforms:instance id="Aktivacia">
    <Aktivacia xmlns="default">
     <Data Od="" Do="" />
     <Data Od="" Do="" />
    </Aktivacia>
   </xforms:instance>
   <xforms:bind nodeset="instance('Aktivacia')/j:Data/@Od" type="xsd:date" />
   <xforms:bind nodeset="instance('Aktivacia')/j:Data/@Do" type="xsd:date" />
   <xforms:bind nodeset="instance('Aktivacia')/j:Data"
           relevant="count(preceding-sibling::j:Data)&gt;0" />
   <xforms:submission id="id" method="put"
           action="http://xformstest.org/cgi-bin/showinstance.sh" ref="instance('Aktivacia')"/>
   <xforms:action ev:event="xforms-ready">
    <xforms:setindex repeat="DataID" index="2" />
   </xforms:action>
  </xforms:model>
 </head>
 <body>
  <xforms:group>
   <xforms:label>Aktivacia</xforms:label>
   <xforms:group>
    <xforms:label>Bez mena</xforms:label>
    <xforms:repeat id="DataID" nodeset="instance('Aktivacia')/j:Data" model="Model">
     <xforms:group ref=".">
      <xforms:input incremental="true" model="Model" ref="./@Od">
       <xforms:label>Od</xforms:label>
      </xforms:input>
      <xforms:input incremental="true" model="Model" ref="./@Do">
       <xforms:label>Do</xforms:label>
      </xforms:input>
     </xforms:group>
    </xforms:repeat>
   </xforms:group>
   <xforms:trigger id="DataIDadd">
    <xforms:label>DataAdd</xforms:label>
    <xforms:insert ev:event="DOMActivate" position="after" at="last()+1"
        nodeset="instance('Aktivacia')/j:Data" />
   </xforms:trigger>
   <xforms:trigger id="DataIDdel" ref="self::node()[not(count(instance('Aktivacia')/j:Data)&lt;3)]">
    <xforms:label>DataDelete</xforms:label>
    <xforms:delete ev:event="DOMActivate" at="last()" nodeset="instance('Aktivacia')/j:Data" />
   </xforms:trigger>
   <xforms:trigger id="DataIDdel" ref="self::node()[count(instance('Aktivacia')/j:Data)&lt;3]">
    <xforms:label>DataDelete</xforms:label>
   </xforms:trigger>
   <xforms:group>
    <xforms:submit submission="id">
     <xforms:label>Potvrd</xforms:label>
    </xforms:submit>
   </xforms:group>
  </xforms:group>
 </body>
</html>
```

Príklad 1.4: Kompletný XForms formulár pre príklad 2

```
<?xml version="1.0" encoding="UTF-8"?>
<Aktivacia xmlns="default"
   xmlns:xforms="http://www.w3.org/2002/xforms"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:ev="http://www.w3.org/2001/xml-events"
  <Data Od="2008-06-01" Do="2008-06-07"/>
  <Data Od="2008-06-15" Do="2008-06-21"/>
</Aktivacia>
```

Príklad 1.5: Data odoslané formulárom 2



Príklad 1.6: Konceptuálny model pre príklad 3

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"  xmlns:j="jka"
 xmlns:xf="http://www.w3.org/2002/xforms"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ev="http://www.w3.org/2001/xml-events">
 <head>
  <xf:model id="Model">
   <xf:instance id="Aktivacia">
    <Aktivacia xmlns="jka">
     <Roaming temp="Data" Do="">
      <Data Od=""/>
      <World Od=""/>
      <Europe Od=""/>
     </Roaming>
     <Roaming temp="Data" Do="">
      <Data Od=""/>
      <World Od=""/>
      <Europe Od=""/>
     </Roaming>
    </Aktivacia>
   </xf:instance>
   <xf:bind relevant="false()"
   nodeset="instance('Aktivacia')/j:Roaming/@temp"/>
   <xf:bind type="xsd:date"
   nodeset="instance('Aktivacia')/j:Roaming/j:Data/@Od"/>
   <xf:bind relevant="../@temp = 'Data'"
   nodeset="instance('Aktivacia')/j:Roaming/j:Data"/>
   <xf:bind type="xsd:date"
   nodeset="instance('Aktivacia')/j:Roaming/j:World/@Od"/>
   <xf:bind relevant="../@temp = 'World'"
   nodeset="instance('Aktivacia')/j:Roaming/j:World"/>
   <xf:bind type="xsd:date"
  nodeset="instance('Aktivacia')/j:Roaming/j:Europe/@Od"/>
   <xf:bind relevant="../@temp = 'Europe'"
   nodeset="instance('Aktivacia')/j:Roaming/j:Europe"/>
   <xf:bind type="xsd:date"
   nodeset="instance('Aktivacia')/j:Roaming/@Do"/>
   <xf:bind nodeset="instance('Aktivacia')/j:Roaming"
   relevant="count(preceding-sibling::j:Roaming)&gt;0"/>
   <xf:submission id="id" ref="instance('Aktivacia')"
   action="http://xformstest.org/cgi-bin/showinstance.sh"
   method="put"/>
   <xf:action ev:event="xforms-ready">
    <xf:setindex repeat="RoamingID" index="2"/>
   </xf:action>
  </xf:model>
 </head>
 <body>
  <xf:group>
   <xf:label>Aktivacia</xf:label>
   <xf:group>
    <xf:label>Bez mena</xf:label>
    <xf:repeat id="RoamingID"  model="Model"
     nodeset="instance('Aktivacia')/j:Roaming">
     <xf:group ref=".">
      <xf:group>
       <xf:output value="@temp"/>
       <xf:trigger>
        <xf:label>Data</xf:label>
        <xf:toggle case="Data" ev:event="DOMActivate"/>
        <xf:action ev:event="DOMActivate">
         <xf:setvalue ref="@temp">Data</xf:setvalue>
        </xf:action>
       </xf:trigger>
       <xf:trigger>
        <xf:label>World</xf:label>
        <xf:toggle case="World" ev:event="DOMActivate"/>
        <xf:action ev:event="DOMActivate">
         <xf:setvalue ref="@temp">World</xf:setvalue>
        </xf:action>
       </xf:trigger>
       <xf:trigger>
        <xf:label>Europe</xf:label>
        <xf:toggle case="Europe" ev:event="DOMActivate"/>
        <xf:action ev:event="DOMActivate">
         <xf:setvalue ref="@temp">Europe</xf:setvalue>
        </xf:action>
       </xf:trigger>
      </xf:group>
      <xf:switch>
       <xf:case id="Data">
        <xf:group ref="./j:Data">
         <xf:input incremental="true" model="Model" ref="./@Od">
          <xf:label>Od</xf:label>
         </xf:input>
        </xf:group>
       </xf:case>
       <xf:case id="World">
        <xf:group ref="./j:World">
         <xf:input incremental="true" model="Model" ref="./@Od">
          <xf:label>Od</xf:label>
         </xf:input>
        </xf:group>
       </xf:case>
       <xf:case id="Europe">
        <xf:group ref="./j:Europe">
         <xf:input incremental="true" model="Model" ref="./@Od">
          <xf:label>Od</xf:label>
         </xf:input>
        </xf:group>
       </xf:case>
      </xf:switch>
      <xf:input incremental="true" model="Model" ref="./@Do">
       <xf:label>Do</xf:label>
      </xf:input>
     </xf:group>
    </xf:repeat>
   </xf:group>
   <xf:trigger id="RoamingIDadd">
    <xf:label>RoamingAdd</xf:label>
    <xf:insert ev:event="DOMActivate" position="after"
       at="last()+1" nodeset="instance('Aktivacia')/j:Roaming"/>
   </xf:trigger>
   <xf:trigger id="RoamingIDdel"
ref="self::node()[not(count(instance('Aktivacia')/j:Roaming)&lt;3)]">
    <xf:label>RoamingDelete</xf:label>
    <xf:delete ev:event="DOMActivate" at="last()"
     nodeset="instance('Aktivacia')/j:Roaming"/>
   </xf:trigger>
   <xf:trigger id="RoamingIDdel"
    ref="self::node()[count(instance('Aktivacia')/j:Roaming)&lt;3]">
    <xf:label>RoamingDelete</xf:label>
   </xf:trigger>
   <xf:group>
    <xf:submit submission="id">
     <xf:label>Potvrd</xf:label>
    </xf:submit>
   </xf:group>
  </xf:group>
 </body>
</html>
```

Príklad 1.7: Kompletný XForms formulár pre práklad 3

```
<?xml version="1.0" encoding="UTF-8"?>
<Aktivacia xmlns="jka"
   xmlns:xforms="http://www.w3.org/2002/xforms"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:ev="http://www.w3.org/2001/xml-events" >
  <Roaming Do="2008-07-19">
    <World Od="2008-07-06"/>
  </Roaming>
  <Roaming Do="2008-08-16">
    <Data Od="2008-08-10"/>
  </Roaming>
  <Roaming Do="2008-08-23">
    <Europe Od="2008-08-17"/>
  </Roaming>
</Aktivacia>
```

Príklad 1.8: Data odoslané formulárom 3

Kapitola 2

Použité technológie a súvisiace práce

V nasledujúcej kapitole si krátko predstavíme základy technológií, ktoré v práci využívame. Technológie XML, XML Schema a XPath sú dnes už pomerne dobre známe a často používané. V kapitole 2.4 si detailnejšie predstavíme konštrukciu a fungovanie formulárov XForms. Pozrieme sa aj na aktuálny stav rôznych implementácií tohoto štandardu. V poslednej časti si ukážeme dva projekty zaoberajúce sa problematikou vytvárania formulárov XForms.

2.1 XML

Označenie technológie XML [27] je skratkou z anglického *Extensible markup language*. Je to univerzálny jazyk so značkami, ktorý vznikol z jazyka SGML. XML poskytuje možnosť textovo popisovať stromovú štruktúru dát. Značky rozdeľujú dokument na časti podobné kontejnerom, ktoré sa nazývajú *elementy* alebo uzly. Elementy môžu obsahovať ďalšie elementy ako svojich potomkov. Každý element má začiatočnú značku, kde je možné uviesť atribúty elementu. Ak nasleduje obsah, je potrebné element ukončiť jeho párovou značkou.

Pri práci s XML je dôležitý pojem *well-formed*, ktorým sa označujú správne formátované XML dokumenty. Aby bol dokument well-formed musí spĺňať niekoľko jednoduchých podmienok:

- každý element je vhodne ukončený

- elementy sa nekrížia
- všetky elementy sú potomkami koreňového elementu

Okrem well-formed nie sú vyžadované žiadne pravidla vnárania elementov, používania atribútov či vynucovania dátových typov.

2.2 XML Schema

Jazyk XML Schema [28] je určený na popisovanie štruktúry XML dokumentov. Pomocou schémy je možné definovať aké elementy smie XML obsahovať, aké sú medzi mini vzťahy navzájom, maximálne a minimálne počty opakovaní, dátové typy elementov a atribútov ale aj mnoho iného. V jazyku XML Schema sa definujú elementy a ich typy. Typy môžu byť jednoduché alebo komplexné.

Podľa schém je možné dokumenty *validovať*. Vstupom validácie je XML dokument a jeho schéma. Výsledkom je overenie, že XML dokument spĺňa všetky podmienky zapísané v schéme prípadne zoznam problematických uzlov.

Výhodou a nevýhodou je veľká vyjadrovacia sila jazyka. Štruktúra bežne používaných XML dokumentov je takmer iste zaznamenateľná v XML Schema. Nevýhodou sa možnosti jazyka stávajú vo chvíli, keď je možné na popísanie určitej štruktúry použiť viacero konštrukcií schémy.

Popis samotný je opäť zapísaný v XML.

2.3 XPath

Štruktúra dát zapísaných v XML sa dá okrem validácie využiť aj na vyhľadávanie informácií v dokumente. Pre tento účel vznikol jazyk XPath [4]. XPath umožňuje adresovanie jednotlivých uzlov alebo množín uzlov v XML dokumente. Výraz XPath často pripomína cestu v súborovom systéme počítačov. Výraz môže byť v každom kroku doplnený o *predikát*, ktorý obmedzí výslednú skupinu vrátených uzlov. Predikáty môžu obsahovať jednoduché operácie ako aj volania vstavaných funkcií.

XPath navyše definuje rôzne osy prechádzania uzlov. Je teda možné adresovať nie len smerom od koreňa k listom ale napríklad použiť súrodencov uzlu. Vyhodnocovanie výrazu môže prebiehať od aktuálneho kontextu alebo

od koreňa dokumentu alebo dokonca je možné prehľadať celý dokument jediným výrazom.

2.4 XForms

Prevažná väčšina súčasných internetových stránok je napísaná v jazyku XHTML [26]. Tento jazyk bol vytvorený v roku 2000 konzorciom W3C, ktorého cieľom je štandardizácia jazykov a technológií používaných na internete. Používanie XHTML viedlo k vydávaniu nových a nových odporúčaní pre tento jazyk. Momentálne je aktuálna deviata verzia v poradí. Konzorcium si uvedomuje potrebu silnejšieho jazyka a preto je už vo vývoji podrobná špecifikácia technológie XForms. V tejto práci budeme pracovať s verziou XForms 1.0 [2]. No konzorcium už pracuje aj na verziách XForms 1.1 a XForms 2.0 , ktoré by mali stavať na znalostiach získaných z prvých implementácii a používania XForms.

Jazyk XForms patrí do rodiny XML jazykov. XForms nie je samostatná technológia, ale potrebuje hostiteľský jazyk, ktorým môže byť takmer ľubovoľný jazyk založený na XML. V našej práci sme zvolili XHTML, ktoré poskytne základnú štruktúru stránky. Existujú však aj iné možnosti ako napríklad SVG[25]. Hostiteľský jazyk ale nie je jediná externá technológia, ktorá bola použitá pri definícii XForms. Pre interne adresovanie prvkov bol použitý jazyk XPath[4]. Jazyk XML Schema [28] je využívaný na validáciu odosielaných dát ešte na užívateľskej strane. Niektoré časti nasledujúcich kapitol majú základ v práci [10].

2.4.1 Fungovanie XForms

Bežné formuláre na internete odosielajú na server neštruktúrované dáta. Aplikácia na serveri dostane parametre ako text, ktorý interpretuje a vyhodnocuje jeho správnosť. XForms zavádzajú nový koncept do práce s údajmi. Výrazným rozdielom je, že formulár *si je vedomí* štruktúry dát, s ktorými pracuje. Táto štruktúra je popísaná v elemente `<xforms:instance>` (viz. str. 21). Po celý čas práce s formulárom sú zadávané údaje udržiavané internej reprezentácii výsledného XML. Odosielanie formulárových dát je teda iba odoslanie serializovanej verzie tejto štruktúry. Voliteľne je možné XML pred odoslaním validovať proti dodanej XML schéme. Ušetrí sa tím serverový výkon ako aj sieťový prenos, čím sa zlepší aj užívateľská odozva.

2.4.2 Štruktúra formulárov

Jednou z hlavných myšlienok XForms je model-view-controller (MVC) prístup. XForms formulár pracuje s troma hlavnými časťami:

- model - používané dáta a ich štruktúra
- správanie - kontrola zadaných hodnôt a kalkulácie
- zobrazenie - použité vstupné polia

Toto oddelenie umožňuje vytvárať znovupoužiteľné formuláre, dodáva silné typové operácia nad formulárom, znižuje potrebu skriptovania a počet odoslaní dát na server. Dôležitým cieľom XForms bolo umožniť hardwarovú nezávislosť a umožniť tak využívanie rovnakých formulárov pre web, wap či pda.

V nasledujúcich kapitolách si detailne vysvetlíme skladbu formulárov a ich fungovanie. Čo najväčší počet konštrukcií ukážeme na príklade 1.7 str. 16. Zameriame sa primárne na vlastnosti, ktoré využijeme pri transformácii a vo výsledných formulároch. V kapitole 2.4.3 popíšeme ako sa definuje štruktúra používaných dát, úpravu správania sa formulárov a nakoniec možné spôsoby odosielania dát. Kapitola 2.4.4 potom doplní vytváranie samotného užívateľského rozhrania ako sú vstupné prvky či opakovanie sa elementov.

2.4.3 Model

Element `<xforms:model>` udržuje informácie o prvých dvoch častiach formulárov. XForms umožňuje odkazovať priamo na schémy popisujúce používané dáta uvedením ich zoznamu v atribúte `schema`. Jeho hodnotou môže byt zoznam potrebných schém. Používane údaje sú udržiavané v elemente `<xforms:instance>`. Správanie je potom popísane elementmi`<xforms:bind>` a `<xforms:submission>`. V použitom príklade je model zobrazený na riadkoch 7 až 46.

Element instance

Element `<xforms:model>` môže obsahovať niekoľko krát obsahovať element `<xforms:instance>`. Každý z nich musí byť označený unikátnym atribútom `id` a je v určitom zmysle predlohou odosielaného XML. Spôsob práce s `<xforms:instance>` povoľuje iba jediného potomka, ktorý môže mať ľubovoľný namespace. Jedinou podmienkou je, aby neporušoval zásady XML a bol

teda správne formátovaný - *well-formed*. V prípade, že sú uvedené hodnoty jednotlivých elementov alebo atribútov, budú tieto hodnoty použité ako predvyplnené dáta vo formulári.

Ak pri elemente `<xforms:instance>` uvedieme atribút `src="zdroj.xml"`, za zdroj dát bude považovaný odkazovaný XML súbor. Súbor môže byť načítaný z disku alebo získaný zo siete. Atribút `src` má prednosť pred spracovaním vnútorného obsahu elementu.

V našom príklade je zobrazený na riadkoch 8 až 21. Jediný potomok `<Aktivacia>` obsahuje dva elementy `<Roaming>`. Ak by sme neskôr nezmenili správanie formuláru elementmi `<xforms:bind>`, obe elementy by boli odoslané. Používanie zdvojených elementov je len technická pomôcka a vysvetlíme si ju v neskorších kapitolách.

Element bind

Z definície XForms je formulár rozdelený na dáta, ktoré sú popísané v `<xforms:instance>` a ovládacie prvky. Je preto potrebné špecifikovať, ktoré ovládacie prvky napĺňajú jednotlivé elementy a atribúty. Slúžia na to elementy `<xforms:bind>`.

Najdôležitejším atribútom je `nodeset`. Jeho hodnotou je XPath výraz, ktorý je používaný na adresovanie konkrétneho elementu alebo množiny elementov v instance. Takéto vytváranie väzby sa nazýva *binding*.

Pomocou prídavných atribútov je možné špecifikovať aj správanie sa formulárov. Najvýraznejšími atribútmi sú:

- *id* - jednoznačné meno elementu, na ktoré sa neskôr môžu odkazovať vstupné prvky.
- *relevant* - hodnota atribútu je vyhodnotená ako XPath výraz na boolean hodnotu. Ak je výsledná hodnota *false*, daná časť instance nie je považovaná za súčasť odosielaných dát. Znamená to teda, že nepodlieha validácii pred odoslaním ani serializácii dát. Ak je element alebo atribút označený ako nerelevantný, všetky k nemu viazané vstupné prvky musia byť vo formulári značné alebo vyradené zo zobrazovania.
- *type* - určuje typ vstupných dát. Zadávaná hodnota je plné meno typu v niektorej z odkazovaných schém alebo je to vstavaný typ jazyka XML Schema.

- *require* - zadaný XPath výraz je opäť vyhodnocovaný na boolean hodnotu. Prípade, že takto označená hodnota nie je zadaná, neprebehne žiadny pokus o odoslanie formulára.

V našom príklade sú všetky elementy `<xforms:bind>` sústredené na riadkoch 22 až 39.

Prvý element odkazuje pomocou XPath výrazu na množinu všetkých atribútov `temp` v elementoch `Roaming` a jeho atribút `relevant` ich označuje za nerelevantné v každom prípade. Na prvý pohľad sa môže vždy nerelevantný atribút zdať ako zbytočný. Avšak aj nerelevantný atribút má stále svoju hodnotu, ktorú je možné využívať v XPath výrazoch. Presne takto využijeme neskôr hodnotu tohoto pomocného atribútu.

Bind element na riadkoch 24 a 25 odkazuje na atribúty `Od` v uzloch `Data`, ktoré musia byť potomkom elementu `Roaming`. V tomto prípade je špecifikovaný typ ako `xsd:date`. Implementácia prehliadača môže túto informáciu využiť a ponúknuť pre výber hodnoty špeciálny vstupný prvok.

Posledný bind element, ktorý si detailnejšie vysvetlíme je na riadkoch 26 a 27. XPath odkazuje na uzol `Data`, ktorý je potomkom uzlu `Roaming`. Atribút `relevant` je vyhodnotený na *true* iba v prípade, že má pomocný atribút `temp` hodnotu *Data*.

Element submission

Po skončení zadávania dát je potrebné vyvolať ich odoslanie. K tomu je určený element `xforms:submission`, ktorý sa môže v elemente `xforms:model` vyskytovať niekoľko krát. Každé tlačítko na odosielanie môže odoslať iný element `xforms:instance`, použiť inú internetovú adresu ako cieľ odosielania či dokonca použiť inú serializačnú stratégiu. Rovnako je poskytovaná opačná možnosť k načítavaniu výsledných dát zo súboru teda odosielanie uložením do súboru. Nakoľko sa v tejto práci zoberáme generovaním internetových formulárov, túto možnosť nevyužijeme.

Naša potvrdzovacia akcia z riadkov 40 až 42 má jednoduchú sémantiku. Metódou `put` odosiela instance s názvom `Aktivacia` na adresu jednoduchej služby, ktorá iba zobrazí doručené dáta.

2.4.4 Užívateľské rozhranie

Špecifikácia rozdeľuje užívateľské rozhranie tri moduly:

- zhlukovanie prvkov do skupín - *group module*

- prepínanie medzi zobrazenými časťami - *switch module*
- opakovanie uzlov - *repeat module*

Oddeleným modulom sú vstupné ovládacie prvky - *form controls module*

Väzba na instance

Spoločnou vlastnosťou modulov (okrem switch modulu) je spôsob viazania sa na uzly v niektorej z dostupných šablón dát uložených v elementoch `xforms:instance`. Ak stránka obsahuje viacero elementov `xforms:model` atribút `model` nastaví správny model ako začiatok cesty pre vyhodnocovanie XPath výrazov.

Každý prvok, ktorý potrebuje adresovať uzol v instance má možnosť pomocou atribútu `bind` adresovať niektorý z elementov `xforms:bind` a tým pádom adresovať uzly v instance. Túto možnosť v našej práci nebudeme využívať. Uvažujme na chvíľu, že náš prvý bind element z riadkov 22 a 23 by mal uvedený atribút `id="IDbindElementu"`. Vstupný prvok pracujúci s hodnotou atribútu `temp` by teda vyzeral takto:

```
<xfomrs:input bind="IDbindElementu"  model="Model"/>
```

Tento vstupný prvok by však nebol nikdy zobrazený, pretože bind element ho označuje ako nerelevantný v každom prípade.

Druhý spôsob adresovania rozlišuje potrebu adresovať množinu elementov alebo adresovanie jediného elementu. Prvý prípad využíva pre zápis žiadaného XPath výrazu atribút `nodeset`. Toto je spôsob aký využíva repeat module, ktorý je použitý v príklade na riadku 53. V prípade potreby adresovať konkrétny element je potrebné použiť atribút `ref`. Group module spolu s Form controls module využívajú tento spôsob. Najjednoduchší XPath výraz použitý na riadku 55 sa môže zdať mierne zavádzajúci. Element `<xforms:repeat>`, ktorý je jeho rodičom odkazuje za použitia atribútu `nodeset` na množinu elementov, no jeho priami potomok odkazuje na jediný uzol použitím výrazu `"."` Je to možné vďaka tomu, že `<xforms:repeat>` iteruje cez množinu adresovaných uzlov a v každej iterácii nastaví aktuálny uzol ako začiatok cesty pre vyhodnocovanie XPath výrazov. Uvedený element `<xforms:group>` teda odkazuje vždy na práve spracovávaný uzol `Roaming`.

Vstupné prvky - Form controls module

XForms rozširujú vstupné prvky definované v jazyku XHTML. Špecifikácia neurčuje, ako presne má byť prvok vykreslený. Toto rozhodnutie je ponechané až na tvorcov prehliadačov. Pre plnohodnotné zariadenia s dostatočnými zobrazovacími kapacitami je možné zvoliť plné grafické zobrazenie 9napríklad niekoľko rádio buttonov pre výber jedinej možnosti). Pre mobilné zariadenia môže byť zvolená textová interpretácia 9prípadne sa kvôli úspore miesta použije combo box, ktorý zaistí rovnakú funkcionalitu, ale spotrebuje výrazne menej miesta na obrazovke). Pracuje sa aj na hlasovom prehliadači XForms.

Krátky popis základných vstupných prvkov:

- input/secret - jednoduchá implementácia vstupného políčka resp. varianta nezobrazujúca vložený text. V prípade, že je pole zviazané s uzlom typu `xsd:date`, môže byť poskytnutý iný ovládaci prvok umožnujúci pohodlnejšie vkladanie dátumu.

- range - zadávanie číselných hodnôt použitím posuvného ukazovátka tzv. *slajder*.

- select/select1 - XForms varianta výklopného zoznamu. Rozdiel v implementácii je podobný ako použitie atribútu `multiple` v bežnom HTML kóde.

- output - ovládací prvok na zobrazovanie aktuálnych hodnôt z instance prípadne iných kalkulovaných hodnôt.

  V našom príklade je na riadku 56 vložený element zobrazujúci aktuálnu hodnotu pomocného atribútu a teda aktuálne zvolenú možnosť.

- trigger - tento ovládací prvok je najčastejšie vykresľovaný ako tlačítko. Po jeho zvolení je vyvolaná udalosť `DOMActivate`. Typické použitie je vloženie elementu `<xforms:toggle>` pre zmenu aktuálne zobrazovanej možnosti v elemente `<xforms:switch>` (vid. str. 26) alebo obecnejšie vloženie elementu `<xforms:action>`, kde je možné vyvolať takmer ľubovoľnú zmenu dát a tým aj zmenu zobrazovaného formuláru.

  Element `<xforms:trigger>` z riadkov 58 až 64 kombinuje obe tieto použitia. `<xforms:toggle>` na riadku spôsobí nastavenie varianty `Data` ako práve zobrazovanej. Nakoľko v našej aplikácii ale potrebuje

vedieť, ktorá voľba je aktívna, aby sme podľa nej mohli určovať relevantnosť častí instance, potrebujeme aj druhú časť. Na riadkoch 61 až 63 nastavíme hodnotu `Data` aj do pomocného atribútu.

- submit - ovládací prvok je rovnako vykresľovaní ako tlačítko. Vyvoláva však odoslanie instance, na ktorú sa odkazuje pomocou povinného atribútu `submission`. V príklade na riadkoch 125 až 127.

Zhlukovanie - Group module

Samotné ovládacie prvky by pravdepodobne stačili na vytvorenie plne fungujúcich formulárov. Ich vývoj by ale určite nebol pohodlný, keďže neposkytujú možnosti zhlukovania sa do určitých skupín. XForms preto obsahujú celý modul, ktorý sa venuje vytváraniu štruktúrovaných formulárov. Element `xforms:group` sa dá v určitom zmysle prirovnať k elementu `div` z jazyka HTML. Vytvára hierarchiu vo vstupných poliach a dodáva dokumentu určitú formu. Neslúži už iba na úpravu zobrazovania, ale napomáha aj pri práci s XForms. Skupina elementov má podobné správanie. Ak je skupina vyhodnotená ako nerelevantná, je vyradená zo zobrazovania spolu so všetkými potomkami bez ohľadu na vyhodnotenie ich relevancie.

Všetky prvky vložené do skupiny preberajú aktuálny kontext od kontajneru. Znamená to teda, že ak sa element `<xforms:group>` z riadku 96 odkazuje na aktuálny element `Roaming`, všetky vnorené prvky odkazujú výrazom `"."` rovnako na element `Roaming`. Vstupné pole z riadku 97
v skutočnosti odkazuje na atribút `Od` elementu `Roaming`.

zgrupovanie, preberanie aktuálneho kontextu od predka spoločne správanie pri zobrazovaní.

Zobrazovanie rôznych možností - Switch module

Jednou z halvných výhod XForms je minimalizovanie potreby skriptovania. K tomu napomáha funkcionalita *switch modulu*. Jeho hlavnou úlohou je zobrazovať iba určité časti užívateľského rozhrania. Funkčne sa to podobá dobre známym "záložkám", rovnako nie je problém vytvoriť takto dojem z postupného vyplňovania formulárov na viacerých stránkach bez nutnosti odosielať formulár pri prechode. Znižuje sa tým zaťaženie serverov, užívateľský zážitok z prehliadania stránok je rovnako zlepšený vďaka takmer okamžitej odpovedi. Celý modul je tvorený troma základnými elementmi:

- switch - koreňový element obsahujúci všetky dostupné možnosti zobrazenia uložene v elemente `<xforms:case>`
- case - atribút `id` je rozhodujúci pri vyhodnocovaní, ktorý element `<xforms:case>` má byť práve aktívny. V každom momente je aktívny iba jediný element.
- toggle - podobne ako je možné akciami nastavovať hodnoty v instance, element `<xforms:toggle>` nastavuje aktuálne zobrazovanú možnosť v elemente `<xforms:switch>`. (Poznámka: aktuálne zobrazovaná možnosť nie je udržiavaná v instance ako dáta.)

Zobrazenie alebo nezobrazenie častí uložených v jednotlivých elementoch `<xforms:case>` má podobné pravidlá ako pri použití `<xforms:group>`. Nevýhodou tohoto modulu je zatiaľ nepresné fungovanie vnorených elementov `<xforms:switch>`. Problematické bolo aj spracovanie v rámci opakujúcich sa elementov.

Opakovanie - Repeat module

Potrebu externých skriptovacích technológii by mala obmedziť možnosť opakovať celé bloky formulárov. Pre tento účel slúži element `<xforms:repeat>`, ktorý je naviazaný v instance na element s premenlivým počtom opakovaní. Opakovať je možné vždy iba jeden konkrétny element.

Element z riadku 53 a 54 zopakuje svoj obsah pre každý relevantný element z kolekcie elementov `Roaming`. Pri každej iterácii je aktuálny kontext nastavený na práve opakovaný element.

XForms vo verzii 1.0 majú v určitej podobe nedoriešený stav, ak užívateľ zmaže všetky instancie opakovaného elementu. Následné pokusy o pridanie budú vždy neúspešné, pretože už nie je "predloha", podľa ktorej by bol nový uzol vytvorený. Pri generovaní sme preto vytvorili o jednu inštanciu elementu navyše (riadky 10 až 14) a označili ju ako nerelevantnú elementom `<xforms:bind>` z riadkov 39 a 40. Formulár preto pri vykreslení obsahuje iba jedno opakovanie. Pre správne fungovanie sme pri dokončení výstavby XForms stromu nastavili index aktuálneho elementu na hodnotu 2 (riadky 43 až 45).

Dôležitými časťami *repeat modulu* sú elementy `<xforms:delete>` a `<xforms:insert>`, ktoré sú vyvolávané z elementu `<xforms:trigger>`.

- insert - na pozíciu určenú atribútmi `at` a `position` vloží jednu inštanciu odkazovaného uzlu.

- delete - odstráni odkazovaný uzol z instance.

Vhodným zobrazovaním tlačítok s určitou funkčnosťou a ich variant bez funkčnosti je možné dovŕšiť nastúpenú cestu vylepšovania *repeat modulu*. Na riadkoch 114 až 123 sú zobrazené elementy `<xforms:trigger>`, ktoré majú doplňujúce sa podmienky. Je teda vždy zobrazené iba jedno tlačítko z popisom *RoamingDelete*. Podmienky sledujú minimálny počet výskytu opakovaného elementu. Ak by vymazanie elementu spôsobilo odstránenie poslednej inštancie je zobrazené tlačítko, ktoré neobsahuje `<xforms:delete>`. Samozrejme je vhodné, aby užívateľ sám rozhodol, či chce tlačítko znefunkčniť alebo ho napríklad nezobrazovať vôbec. Rovnakým spôsobom je možné implementovať kontrolu na maximálny počet opakovaní.

2.5 Dostupné XForms implementácie

Existuje niekoľko zásadne rozdielnych možností ako XForms spracovávať. Najhrubším rozdelením je server side alebo client side interpretácia XForms kódu. Výhody a nevýhody týchto prístupov sú zhrnuté v tabuľke 2.1.

V kapitolách 2.5.1 a 2.5.3 krátko opíšeme niektoré implementácie XForms processorov.

2.5.1 Client side

Mozilla plug-in

Medzi najlepšie fungujúce implementácie XForms sa určite radí zásuvný modul do prehliadača FireFox. FireFox ako plnohodnotný prehliadač dobre zvláda bežné technológie internetu ako XHTML či CSS. XForms formuláre môžu byt integrované do už fungujúcich stránok takmer bez akejkoľvek zmeny pre užívateľov. Zásuvný modul nevyžaduje zložitú inštaláciu. K fungovaniu nie je potrebné vkladať do kódu ani žiadne špeciálne objekty.

Samotné spracovanie XForms je na výbornej úrovni. Podporované sú všetky bežné konštrukcie a dokonca niekoľko naviac oproti špecifikácii. Na druhej strane nie sú do posledného detailu implementované všetky funkcie týkajúce sa štýlov pre XForms prvky. Mozilla ale pridáva mnoho vlastných štýlovacích možností, s ktorými je možné formuláre vykresľovať aj s pomocou technológie SVG. Aktuálna verzia používa niektoré deprecated low level metódy, čím trpí vykresľovanie nerelevantných prvkov.

	Výhody	Nevýhody
Client side	- prenášanie čistého XForms formuláru zrýchľuje odozvu stránok. - nie je potrebné používať skriptovacie jazyky. - všetky XForms akcie sú spracované bez spolupráce so serverom	- každá client side implementácia sa môže správať mierne odlišne prípadne rozdielne vyzerať - užívateľ si musí nainštalovať špeciálny prehliadač alebo iný program.
Server side	- XForms sa do prehliadača užívateľa vôbec neposiela - zvýšenie použiteľnosti umožní okamžité nasadenie za použitia dnešných prehliadačov	- je potrebný prehlad x XForms na XHTML - potrebné prídavné technológie ako Javascript alebo Ajax - výšená záťaž na sieť - zníženie odozvy stránok

Tabulka 2.1: Porovnanie prístupov k interpretácii

Výhodou tohoto riešenia pri vývoji aplikácii a používaní skúsenejšími užívateľmi je aj rýchla odozva tímu vývojárov. Vývojový tým promptne odpovedá na možné chyby v implementácii. Ak sa nejaká z chýb potvrdí a tým sa jej začne venovať, je chyba obyčajne odstránená v priebehu niekoľkých málo týždňov. Projekt Mozilla XForms má aj veľmi aktívnu komunitu užívateľov, testerov a prispievateľov. Denne sú dostupné najnovšie verzie zásuvného modulu.

X-Smiles

Na veľmi dobrej úrovni XForms zvláda aj prehliadač X-Smiles[29], ktorý bol v jeho začiatkoch vyvíjaný skupinou fínskych študentov. Projekt nie je zameraný priamo na XForms. Jeho cieľom je vytvoriť nový prehliadač založený na XML technológiách vhodný pre malé prenosné zariadenia. Podporovaná je napríklad možnosť používať ako hostiteľský jazyk SVG. Implementácia sa drží normy o niečo striktnejšie ako Mozilla XForms. To je ale v niektorých prípadoch na škodu funkčnosti.

Nevýhodou X-Smiles je zatiaľ nepoužitelnosť ako plnohodnotného www prehliadača. Spracovanie CSS štýlov a HTML stránok značne zaostáva za úrovňou bežných prehliadačov. Znížený užívateľský komfort robí tento prehliadač takmer nepoužiteľným. Až do uvolnenia prvej stabilnej verzie boli aktualizácie celkom pravidelné. Od verzie X-Smiles 1.0 sa vývoj akoby zastavil a neboli uvoľnené žiadne nové verzie.

Výraznou chybou je však validovanie celého elementu instance bez ohľadu na relevantné a nerelevantné časti. Formulár je preto vždy kompletne validovaný a serializovaný.

Form Faces

Zobrazovacie XForms pomocou Form Faces[30] je postavené výlučne na technológii JavaScript. Stránka je pred zobrazením JavaScriptom *preložená* z XForms do bežných vstupných polí. Všetky vlastnosti XForms formulárov by potom mali zabezpečovať skripty.

Nevýhodou je nutnosť do formulára pridávať špeciálny element, ktorý spustí transformáciu. Tento element obsahuje odkaz na adresár obsahujúci Form Faces, takže presúvanie formulárov sa stáva obtiažnejšie. Navyše každé zobrazenie vyžaduje prenost JS po sieti.

Implementácia zatiaľ nie je dokonale odladená. Aj v prípade, že po načítaní nie je zobrazená nezmyselne dlhá chybová hláška, nebýva funkčnosť dokonalá.

V niektorých prípadoch nefunguje kontrola jednoduchých typov.

MozzIE

Open source projekt portujúci Mozilla XForms do prostredia prehliadača Internet explorer. Balíčky dostupné pre Internet explorer nepridávajú žiadnu funkčnosť navyše. Za aktuálnou verziou z Firefox add-on balíčku výrazne mešká.

2.5.2 Natívne a mobilné implementácie

Objavuje sa aj viacero komerčných implementácii:

- PicoForms - http://www.picoforms.com/
- Data movil - http://www.datamovil.info
- Oracle Wireless Client - http://www.oracle.com/technology/tech/wireless/mobilebrowser.htm

2.5.3 Server side

Úplne iným prístupom k spracovaniu XForms sú serverové riešenia. Znamená to, že zdrojové súbory obsahujúce XForms nespracúva klient (užívateľov prehliadač, PDA zariadenie, ...). Všetky XForms prvky sú ešte pred odoslaním klientovi serverom *preložené* napríklad do XHTML. Dynamické vlastnosti XForms ako opakovanie prvkov či typová kontrola má byt zabezpečená inými prostriedkami ako napríklad JavaScrip-tom.

Typicky sa XForms prekladajú do technológií, ktoré sú v užívateľských prehliadačoch už dlhšie podporované. Klient tak nepotrebuje XForms žiadnym spôsobom poznať. Dokonca nemusí mať ani možnosť zistiť, že pôvodne bol formulár zapísaný v jazyku XForms. Server nakoniec dostane odpoveď od klienta, pretransformuje dáta do XML podoby a pošle ďalším častiam na spracovanie, akoby dáta poslal sám klient.

Nevýhodou server side spracovania je zložité nasadenie. Navyše sa strácajú výhody, ktoré viedli k vzniku XForms ako napríklad obmedzenie objemu posielaných dát, obmedzenie najrôznejších skriptovacích jazykov.

Orbeon

Projekt Orbeon sa venuje mnohým technológiám, ktoré ťažia zo základov jazyka XML. Open source projekt Orbeon forms sa zaoberá práve jazykom XForms. XForms formuláre zobrazované v rámci Orbeon forms zvládajú všetky základné konštrukcie. Niektoré pokročilé vlastnosti sú implementované striktne podla špecifikácie. Orbeon má podobne ako Mozilla XForms rýchle odozvy na otázky a problémy užívateľov. Do rozsiahleho mailing listu sa zapája množstvo skúsených užívateľov, vývojárov, rovnako ako samotný tým Orbeon.

Celý projekt je možné stiahnuť a nasadiť na lokálnom aplikačnom serveri. Výhodné je však použiť verejne dostupné *pieskovisko*, kde sa vzdialene nahrá formulár a je s ním možné okamžite pracovať.

Chiba

Druhým zástupcom serverového riešenia je projekt Chiba, ktorý je ako open source stiahnuteľný zo stránok SourceForge. Základom je programovací jazyk Java, ktorý XForms preloží do HTML, JavaScriptu a za použitia technológie AJAX simuluje niektoré akcie XForms.

Veľmi príjemným je štart aplikačného servera s nasadenou serverovou časťou, ktorý je neuveriteľne rýchly. *Inštalácia* nového formuláru, ak sa tento postup dá inštaláciou nazvať, spočíva iba v skopírovaní formuláru na správne miesto. Formulár je okamžite dostupný bez potreby reštartovať server.

Engine projektu Chiba je využitý v open source komerčnom CMS projekte Alfresco, kde sa z popisu v jazyky XML Schema generujú formuláre s typickými možnosťami ako sú opakovania či validácie.

2.6 Modelovanie XForms formulárov

Firma IBM sa projektom AlphaWorks snaží rozširovať perspektívne technológie medzi vývojárov a užívateľov. Počas 10 rokov činnosti vypracovali viac ako 700 prototypov technologických návrhov. Do návrhov, vývoja a programovania sú zapojené výskumné skupiny z celého sveta. Samozrejme sa v tomto projekte zaoberajú aj technológiami na báze XML a rovnako neboli vynechané ani formuláre XForms. Vznikli tak hneď dva rôzne nástroje, ktoré si v krátkosti priblížime v ďalších častiach.

2.6.1 XML Forms Generator

XForms sú veľmi úzko spojené s technológiami XML Schema a webových služieb. Na tejto myšlienke je založený balík XML Forms Generator[31]

Generovanie formulárov je možné na základe XML instance. Prevod je plne automatický. Spolupráca užívateľa nie je potrebná. Generátor môže v prípade dostupnosti použiť XML Schema ako zdroj metadát.

Druhou možnosťou je generovanie formulárov na základe WSDL popisov internetových služieb.

XML Forms Generator je dodávaný ako zásuvný model do programovacieho prostredia Eclipse. Poskytuje užívateľom kvalitnú spoluprácu dodaného instance a jeho XML Schemy (napríklad okamžitá validácia instance proti scheme). K dispozícii je generovanie pre niekoľko systémov schopných zobrazovať XForms.

Nevýhodou je, že užívateľ nemá možnosť zasahovať do transformácie. Nie je možné pracovať s formulárom ani po transformácii.

Napriek možnostiam spoločnosti ako je IBM, ani tento generátor nepokrýva všetky možné konštrukcie jazyka XML Schema.

Návod na nainštalovanie XML Forms Generator je dostupný na stránkach IBM. Samotná inštalácia prebieha cez Update Manager prostredia Eclipse. Je však nutná registrácia.

2.6.2 XForms Visual

Zásuvný modul XForms Visual[32] slúži tiež na tvorbu XForms formulárov. Základnou myšlienkou je jednoduché umiestňovanie prvkov na formulár. Formulár je možné vytvárať aj priamym písaním kódu, kde kontextová pomoc obsahuje aj prvky XForms.

Výsledný formulár však často vyzerá výrazne ináč ako vyzerá v časti návrhu v grafickom móde. Významnú úlohu na tom má použitie kaskádových štýlov.

Výborne riešená je možnosť tvorby prvkov prislúchajúcim konkrétnym častiam instance. Samotná instance môže byť vytvorená užívateľom, vytvorená zo zdrojového XML súboru alebo schémy. Potom je možné vytvárať ovládacie prvky priamo ťahaním častí instance na formulár. Takto vytvorené prvky majú vytvorené príslušné väzbové atribúty. Vytváranie väzieb na vstupné prvky, ktoré sú na formulár najprv vložené, až potom viazané, je rovnako jednoducho riešené ťahaním.

Celkovo je aj tento editor príjemný pre užívateľa.

Kapitola 3

Modelovanie

Modelovanie je učené k zrýchleniu, zefektívneniu a zjednodušeniu vývojového procesu. Zber užívateľských požiadavok pravdepodobne neurýchli žiadna nová metóda. Jedinou cestou ako tieto informácie získať je opýtať sa užívateľov. Správne pochopenie problému je kľúčové. Od tejto chvíle najrôznejšie modely pomáhajú proces unifikovať a automatizovať. Vizuálne znázornenie riešenej domény pomáha pochopiť vzťahy objektov navzájom. Dôležitým faktorom je možnosť netechnických užívateľov zúčastňovať sa na procese špecifikácie problému. Validácia navrhovaného riešenia je možná oveľa skôr ako by mohol byť dostupný prvý prototyp či spotrebované neúmerne množstvo prostriedkov na jeho vypracovanie.

Konceptuálny model popisuje sémantiku modelovanej reality. Znamená to, že sa nesústredí na cieľovú implementáciu ale na identifikovanie objektov, ich atribútov a vzťahov medzi nimi. Je teda na vyššej úrovni abstrakcie ako databázové modely či objekty v konkrétnom programovacom jazyku.

Základnou jednotkou pri modelovaní ľubovoľného problému je *entita*. Entita predstavuje objekt z reálneho sveta ako je napríklad zákazník, zmluva či faktúra. Každý z týchto objektov má určité vlastnosti - *atribúty*. Napríklad meno, telefónne číslo pre zákazníka alebo evidenčné číslo pre papierové dokumenty. Entity so svojími atribútmi ale nezaznamenávajú všetky informácie, ktoré bude výsledný systém o objektoch zhromažďovať. *Vzťah* medzi entitami sú rovnako dôležité. Príkladom takéhoto vzťahu je jednoznačné priradenie zákazníka ku každej zmluve.

V priebehu času sa pri riešení problémov v špecifických doménach vyvíjali aj modelovacie postupy. V niektorých oblastiach sa modelovanie dostalo do veľmi pokročilých štádií a ich prínos nemožno popierať. Iné metódy sú

pomerne nové, zatiaľ neoverené dlhoročnou praxou. Najdôležitejšími vlastnosťami spoločnými pre takmer každý typ modelov sú

- *Formálne základy* Formálny popis modelu umožní odhaliť potencionálne nedostatky prípadne dokázať správnosť a úplnosť modelu.
- *Grafické znázornenie* Vizualizácia modelov výrazne napomáha ich pochopeniu a používaniu. Jednotlivé konštrukcie musia mať priradené jednoduché značky s dobre definovanou sémantikou, ktorá umožní ich bezproblémové používanie.

Najrozšírenejšie modely popíšeme v ďalšom texte - ER modelovanie v kapitole 3.1, UML v kapitole 3.2.

3.1 ER modelovanie

Autorom tejto metódy je Dr. Peter Chen. Primárnym cieľom jeho článku [5] z roku 1976 bolo zjednotenie pohľadu na oblasť sietí a relačných databáz. Navrhol v ňom konceptuálny model, ktorý modeluje reálny svet ako objekty (entity) a vzťahy medzi nimi. Z anglických slov entity a relationship vznikol názov *E-R model*.

Entita v E-R diagrame predstavuje triedu reálnych objektov, ktoré sú schopné samostatnej existencie. Za behu v cieľovom systéme existujú mnohé inštancie konkrétnych tried. Každá entita má vlastné atribúty. E-R model pracuje s rôznymi druhmi atribútov od jednoduchých a zložených až po viachodnotové kompozitné. Bližší popis týchto termínov je mimo rámec tejto práce [22].

Pre identifikaciu inštancií entít používame tzv. *kľúče*. Kľúč je množina atribútov danej entity. Entita môže mať viacero kľúčov. Každá inštancia entity je potom jednoznačne identifikovateľná v rámci ostatných inštancií pomocou kombinácie hodnôt atribútov v kľuči. V tejto práci nebudeme s kľúčmi pracovať.

Ukázali sme si zatiaľ rôzne druhy entít, ostáva nám definovať vzťahy medzi nimi.

Vzťahový typ zachytáva v akom vzťahu sú entity voči sebe. Vzťahy môžu existovať medzi entitami obecne. Jedna entita môže do jedného vzťahu vstupovať aj niekoľko násobne. Nie sú dokonca zriedkavé ani prípady rekurzie, keď je entita vo vzťahu sama so sebou.

Vzťahy je možné doplniť o kardinalitu. Kardinalita udáva opakovanie entity vo vzťahu. Nulová kardinalita vyjadruje nepovinnú účasť entity vo vzťahu.

Hierarchia ISA vzťahov vyjadruje *dedenie* medzi entitami. Každá podedená trieda v ISA hierarchii preberá atribúty a vzťahy svojho predka. Nie je možné atribúty alebo vzťahy uberať. Potomkovia sú vďaka tomu použiteľný všade, kde priamo do vzťahu vstupuje ich predok. Aby bol diagram zmysluplný nemôže v ISA hierarchii vzniknúť cyklus.

Tieto jednoduché stavebné bloky poskytujú dostatočnú abstrakciu, jednoduchosť a vyjadrovaciu silu, aby z nich boli vytvárané konceptuálne modely aj veľmi zložitých systémov. E-R modelovanie je výrazne využívané pre modelovanie databáz, kde teoretické postupy prevodu na databázové schémy umožnili plne automatický prevod.

3.2 UML

Vyžívanie UML (z anglického *Unified Modeling Language*) v návrhu software je dnes takmer denná záležitosť [20, 21]. UML svojou širokou ponukou najrôznejších diagramov ponúka možnosť popísať takmer ľubovoľnú fázu projektu od zaznamenávania požiadavok až po vývoj a nasadenie a fyzickú architektúru prostredia.

Z mnohých diagramov UML zľahka naznačíme fungovanie a použitie UML diagramu tried (*class diagram*). Ten je primárne určený na popis dát systémov resp. dát v systéme.

Ponúkané základné bloky sú veľmi podobné ako v E-R modelovaní. Základom každého diagramu je trieda so svojími atribútmi, ktorá odráža typ fyzických objektov. UML umožňuje modelovať vzťahy použitím asociácií. Viac násobné vzťahy sú modelované pomocnými triedami. Tento prístup vzdialene pripomína prevod vzťahových typov E-R diagramov na databázovú schému, keď sú vytvárané pomocné objekty.

UML nepodporuje priamo konštrukcie ako slabé entity alebo ISA hierarchie. Táto funkcionalita je ale nahradená rôznymi druhmi asociácií. Sú nimi

napríklad kompozícia alebo agregácia, existuje aj vzťah funkčne podobný ISA hierarchiám.

Miernou komplikáciou pri používaní UML je narastajúca zložitosť a používanie najrôznejších grafických notácií. Problém trochu viditeľnejší ak používame viacero typov diagramov UML. Môže sa stať, že notácia z niektorého typu diagramov značí iné vlastnosti v druhom diagrame.

3.3 Model-Driven Architecture

Model-Driven Architecture (MDA) rozširuje zavedený trend oddelenia špecifikácie funkčnosti systému od detailov využívania možností, ktoré poskytuje technologická platforma [15]. Základnými cieľmi je poskytnúť prostriedky pre:

- špecifikáciu systému nezávisle od možností poskytovaných technologickou platformou
- špecifikáciu a výber technologickej platformy
- transformáciu špecifikácie systému do implementácie nad konkrétnou technológiou.

Najdôležitejším je rozdelenie modelovania systému na rôzne úrovne vzhľadom na technológiu.

- Technológii najviac vzdialený je CIM (*computational independent model*), ktorý modeluje systém z hľadiska prostredia systému a funkčných požiadavok na systém. Detaily štruktúry sú zatiaľ skryté alebo nešpecifikované. Typickým slovníkom na tejto úrovní sú doménové pomenovania známe budúcim business vlastníkom systému.
- PIM (*platform independent model*) modeluje funkčnosť systému stále bez väzby na platformu. Časti PIM sa nemenia pri zmene platformy. Označovanie sa presunulo do pojmov bližších implementácii. Z technologických termínov je však použitá iba podmnožina vhodná pre implementáciu v niekoľkých možných cieľových platformách.
- Poslednou úrovňou je PSM (*platform specific model*), ktorý vznikne z PIM dodaním informácii o vybranej implementačnej technike. Detail a úplnosť modelu umožňuje plynulo prejsť k implementácii. Najrôznejšie

techniky prevodu dovoľujú generovať kusy výsledného systému (prípadne celý systém).

Rozdelenie modelovania na úrovne je však iba časťou teórie spojenej s MDA. Významnú časť MDA pojednáva o transformáciách medzi modelmi. Ide o spôsoby ako z CIM alebo PIM generovať PSM. PSM je možné v záverečnej fáze transformovať na zdrojový kód. Objavujú sa aj pokusy generovať kód priamo z PIM modelov. Pre tieto účely sú využívané najrôznejšie mapovania medzi modelmi navzájom či medzi modelmi a jednotlivými technologickými platformami.

Významným prínosom MDA je dosiahnutie prenesiteľnosti, interoperability a znovupoužiteľnosti modelov a kódu.

3.4 WebML

Na modelovanie internetových aplikácii je vo väčšine prípadov použitá obecná metodika, ktorá slúži rovnako dobre pre definovanie databázových aplikácii alebo hrubých klientov.

Web modeling language[1] je taliansky akademický projekt zameraný na konceptuálnu tvorbu kompletných internetových aplikácií. Projekt dopĺňa možnosti špecifické pre prostredie webových stránok, odkazov medzi nimi a mnoho ďalších. Ide napríklad o definovanie hypertextového modelu alebo personalizovaných pohľadov. Teoretický základ je detailne popísaný v článku [3]. WebML nie je iba jazykom symbolov s určitou sémantikou. Projekt pokrýva hlavne úvodné časti životného cyklu webového projektu, ktoré sú rozdelené do niekoľkých modelov.

Fáza zberu požiadavok je pomerne obecná. Hlavným prínosom je definovanie výstupov ako *skupiny užívateľov*, *dátový slovník*, *mapa webu* či *príklady použitia*. Jednotlivé artefakty budú použité pri vypracovaní ďalších modelov.

Základom aplikácie využívajúcej WebML je rovnako ako v mnohých iných metodikách *dátový model*. Pri jeho tvorbe je základom dátový slovník, ktorý môže definovať hlavné a pomocné entity. Navrhovaný dátový model je takmer zhodný s E-R modelom. Nájdeme tu všetky známe objekty ako sú entity (silné a slabé), vzťahy, atribúty a dátové typy. Metodika doporučuje najprv vytvoriť entity a vzťahy, ktoré patria logicky k sebe čím vznikajú schémy. Dotvorenie entít vyplnením atribútov je až nasledujúcim krokom.

[1] http://www.webml.org

Nasleduje tvorba hypertextového modelu, ktorý je pomerne pevne viazaný na štýl práce s webovými aplikáciami. Hypertextový model pozostáva z dvoch jednoduchších podmodelov:

- kompozičný model - definuje niekoľko skupín *jednotiek (units)*, ktoré môžu zobrazovať dáta (používajú sa tzv. *selektory*), vykonávať rôzne akcie na stránkach alebo získavať údaje od užívateľov. Vykonávanie akcií má podobnú filozofiu ako prístup CRUD (creat-read-update-delete). Jednotky sú v kompozičnom modely použité na definíciu stránok tvoriacich web. Prostredníctvom jednotiek stránky získavajú väzbu na dátový model.

- navigačný model - okrem definovania väzieb medzi stránkami definuje spolupácu medzi jednotkami. Linky môžu byť nekontextové v prípade, že nenesú žiadne dáta a ich jedinou úlohou pre presmerovať užívateľa na inú obrazovku. Kontextové linky predávajú nasledujúcej stránke parametre. Parametre môžu byť určené pre celú stránku alebo konkrétnu jednotku na nej. Existuje možnosť definovať globálne parametre. Pre nastavovanie a čítanie globálnych parametrov sú definované samostatné komponenty. Takáto práca s parametrami a ich predávaním je ďalší z konceptov pomerne špecifických pre webové prostredie.

Prezentačný model špecifikuje ako budú stránky zobrazované užívateľom. Definujú sa tu rozloženia jednotiek na stránkach. Rozloženie je možné určovať na globálnej úrovni a prepoužívať ho tak pre viacero stránok. Prezentačný model je stále nezávislý na technológii, v akej sa bude aplikácia implementovať

Personalizačný model umožní identifikovaným *skupinám užívateľov* zobrazovať rôzne formátovaný obsah stránok prípadne je možné ponúkať na základe skupín rôzne časti webu. Personalizácia je dnes vyhľadávanou funkcionalitou najrôznejších internetových portálov, ktorá bola modelovaním pokrývana iba zriedka.

3.5 UWE

Metóda *UML for Web engineering (UWE)* vznikla ako rozšírenie UML pre špecifické potreby modelovania internetových aplikácii. UWE predstavuje nový UML profil zahŕňajúci nové stereotypy ako navigačný odkaz a trieda

či menu alebo index. Podrobnejší popis ako aj porovnanie s metódu *WebRE* pre zber požiadavok a ich zápisu v UML nájdeme v článkoch [12, 13]. Metóda UWE pokrýva takmer celý vývojový cyklus od zberu požiadavok až po implementácia a teda generovanie cieľového kódu.

Metodika UWE pracuje s modelmi na troch rôznych úrovniach: *CIM*, *PIM* a *PSM*. (Bližší popis týchto úrovní je uvedení v kapitole 3.3 na str. 37.)

Proces vývoja webovej aplikácie podľa UWE začína vytvorením CIM modelu. Ide o zber požiadavok funkčných ako aj požiadavok na architektúru.

Séria transformácii rozdelí CIM na niekoľko zo začiatku menších PIM modelov zachytávajúcich rôzne časti sytému ako obsah a jeho prezentácia, navigácia či business logika. PIM modely sú neskôr zlúčené do jediného veľkého PIM modelu, ktorý zobrazuje celý systém a to z hľadiska funkčného aj architektonického. Využitý je hlavne vo formálnej validácii.

Posledným krokom je transformácia na výsledný kód. Žiadny z PIM modelov ani CIM model neobsahujú informácie o konkrétnej platforme. Ponúkajú sa preto dve možné cesty ako takúto transformáciu vykonať. Prvou je získanie informácii z prídavného modelu, ktorý by riadil obecný proces transformácie. Druhou možnosťou je špecifická transformácia pre zvolený cieľový formát.

3.6 OOHDM

Nakoniec sa krátko pozrieme na metódu OOHDM (z anglického *Object oriented hypermedia design model*) [23, 24, 16].

Je to model vhodný na viacero typov úloh, medzi ktorými sú aj internetové aplikácie. Je rozdelený na štyri časti. Prvou z nich je samozrejme konceptuálne modelovanie dátovej vrstvy, kde sa znova používa syntax príbuzná UML.

Druhým a pravdepodobne najdôležitejším krokom je definícia navigačného modelu. V istom zmysle sú to znova pohľady na dátovú vrstvu. Na zápis *navigačných uzlov* sa používa dotazovací jazyk podobný jazyku SQL. Vznikajú tak nové uzly, ktoré môžu obsahovať údaje dopočítané z iných entít. K tomuto modelu patrí aj definícia niekoľkých rôznych *navigačných kontextov*. Práve touto vlastnosťou sa OOHDM výrazne odlišuje od bežného modelovania, ktoré neprihliada na špecifiká webových aplikácii.

Návrh abstraktného rozhrania sa zaoberá definíciou štruktúry aplikácie, vzťahov navigačných objektov a správania sa aplikácie. Zavádza sa pojem

abstraktného dátového pohľadu - ADV (z anglického *abstract data view*).

Finálnou fázou je implementácia. Použitím UML je možné vykonať preklad konceptuálneho modelu na databázovú schému takmer bez vynaloženého úsilia. Preklad ADV je nie až tak široko podporovaný, no koncept databázových pohľadov by nám mohol výrazne uľahčiť prácu. Implementácia rozhrania a kontextov je závislá na zvolenej technológi.

3.7 XSEM

Konceptuálny model jazyka XSEM je založený na princípoch MDA. Rozdeľuje teda modelovanie na niekoľko rôznych úrovní abstrakcie.

Prvou z nich je plaformne nezávislý model PIM *z anglického - platform independent model*. Pre grafické znázornenie je využívane UML. Nie sú však využívané všetky dostupné UML konštrukcie. Používajú sa triedy s atribútmi a binárne asociácie. PIM zachytáva modelovanú doménu bez spojitosti s konkrétnou technológiou. Úroveň abstrakcie by sa dala prirovnať k relačnému modelu databáz.

Druhú úroveň predstavuje platformne špecifický model PSM *z anglického - platform specific model*. Úroveň abstrakcie je tu o stupeň bližšie k technológii, v našom prípade XML. Grafické znázornenie je znova na základoch UML. Pre modelovanie XML štruktúr bolo nutné navrhnúť nové konštrukcie. Ich detailný popis je možné nájsť v [17].

PSM je v určitom zmysle pohľadom na PIM. Môže existovať viacero rôznych PSM pre jediný PIM. PSM teda modeluje rôzne XML štruktúry. Sémantika modelov zostáva zachovaná vďaka väzbe na PIM. Narozdiel od PIM je PSM prispôsobený priamo na modelovanie XML štruktúr. Model je preto zaznamenaný v podobe stromu. Základnými prvkami sú PSM triedy s atribútmi a asociácie medzi nimi.

PSM trieda C_{psm} reprezentuje PIM triedu C a špecifikuje ako sú jej inštancie reprezentované v modelovanej XML štruktúre. C_{psm} má rovnaké meno ako C a podmnožinu jej atribútov. Každý atribút z PIM môže mať špecifikovaný tzv. *alias*. Alias premenúva atribút z PIM vo výslednom XML. Obsahom triedy C_{psm} je usporiadaný zoznam PSM asociácii z nej vychádzajúcich. Tento zoznam môže byť prázdny a je nazývaný *obsah* triedy C_{psm}.

PSM asociácia A_{psm} je orientovaná hrana medzi triedami C_{psm}. Zdrojová trieda je označovaná ako *rodič*, cieľová ako *potomok*. Pre jednoduchosť pred-

pokladajme, že každá asociácia predstavuje vkladanie PIM tried reprezentovaných PSM triedami spojených A_{psm}.

PSM trieda C_{psm} reprezentujúca PIM triedu C modeluje štruktúru inštancie triedy C v XML dokumente ako množinu XML atribútov a sekvenciu XML elementov. XML atribúty sú modelované atribútmi PSM triedy. XML elementy sú modelované obsahom PSM triedy C_{psm}.

Príslušnosť PSM tried k sebe je vyjadrená PSM asociáciou A_{psm}. Sémantika každej PSM asociácie je popísaná konštrukciou *nesting join*, ktorá dáva do vzťahu PIM objekty a A_{psm}. Nech PSM asociácia vychádza z triedy C_{psm} a končí v triede C'_{psm}. Znamená to, že A_{psm} modeluje XML, kde je inštancia c' triedy C' obsiahnutá v XML reprezentujúcom inštanciu c triedy C.

Triedy PSM môžu mať definovaný *popisok (label)*. Atribúty a obsah PSM triedy s definovaným popiskom sú vo výslednom XML uložené v XML elemente s rovnakým názvom. Ak trieda popisok nemá jej atribúty a obsah sú propagované smerom hore do najbližšej triedy s popiskom. Existencia predka s popiskom je zaručená vďaka koreňovým triedam, ktoré musia mať popisok.

Obsahom PSM tried môže byť aj:

- *kontajner atribútov (attribute container)*. Jeho obsahom je jeden alebo viacero atribútov C_{psm}. Tieto atribúty sú vo výslednom XML vytvárané ako XML elementy.

- modelovanie výberov umožňuje *content choice*, ktorý obsahuje minimálne dve PSM asociacie vedúce z C_{psm}. Iba jediná z nich môže byť obsiahnutá v každej inštancii triedy C_{psm}.

- *štrukturálny zástupca (structural reprezentative)* Z_{psm} je PSM trieda, ktorá dedí svoj obsah a atribúty z inej PSM triedy C_{psm}. Obe triedy musia reprezentovať rovnakú PIM triedu. Z_{psm} môže mať vlastný popis.

Pre úplnosť ešte musíme presnejšie popísať sémantiku PSM asociácii. PSM sú rôznymi pohľadmi na PIM. Znamená to napríklad, že celé cesty v PIM diagramoch môžu byť v PSM vyjadrené jedinou asociáciou. Obecnejšie je preto uvažovať o popise sémantiky PSM asociácii nie ako o PIM asociáciach ale ako o cestách v PIM diagrame. (Hore uvedené zjednodušenie je stále platné, pretože PIM asociácia môže byť považovaná za cestu dĺžky 1.)

Detaily a formálne definície s podmienkami konzistencie modelu je možné nájsť v prácach [17, 18].

V nasledujúcich kapitolách budeme pracovať práve s modelom XSEM, v ktorom budeme modelovať doménu problému a formuláre samotné.

3.7.1 Projekt XCase

V našej práci budeme ďalej pracovať so zatiaľ jediným modelovacím softwarom pre jazyk XSEM, ktorým je XCase [19]. V nasledujúcej kapitole si krátko predstavíme formát, v ktorom XCase ukladá informácie.

XCase využíva na uloženie všetkých informácií o aktuálnom projekte v XML formáte. Serializovaná verzia projektu je pre človeka pomerne ťažko čitateľná. XML súbor slúži takmer výlučne iba na ukladanie sekvencií objektov. Odkazovanie prvkov medzi sebou je zabezpečené pomocou unikátnych identifikátorov (sekvenčne priradzované čísla). Súvisiace objekty tak nemusia byť sústredne v spoločnom elemente. Nepríjemným vedľajším efektom používania unikátnych identifikátorov je znížená možnosť typovej kontroly pri implementácii iného software nad týmto formátom. Rovnako manuálna úprava je mierne náročnejšia na sústredenie, kvôli možným chybám.

Serializovaný model je rozdelený na dve hlavné časti a to časť spojenú s vizualizáciou a rozmiestnením uzlov (tried) na obrazovke a časť popisujúcu model samotný.

Vizualizačná časť je v XML umiestnená až na druhom mieste. Napriek tomu ňou začíname a to preto, že s jej popisom sa nebudeme takmer vôbec zaoberať. Sú tu uložené súradnice každého objektu tak ako bol umiestnený na obrazovke v čase modelovania a to oddelene pre PIM a PSM diagramy. Vizualizácia je pre nás takmer bezvýznamná nakoľko pri generovaní sú podstatné informácie o modely. Jediné využitie v implementácii bolo jednoduché zistenie všetkých koreňových elementov práve z vizualizačnej časti.

Druhá časť popisuje všetko potrebné pre modelovanie v rozšírenom UML. Krátke pasáže obsahujú primitívne dátové typy a popis používaného UML profilu. Najvýznamnejšou časťou je popis modelu samotného.

Celý model je popísaný ako zoznam PIM tried a asociácii medzi nimi. Každá PIM trieda má okrem mena zoznamy obsahujúce:

- atribúty spolu s údajmi ako je typ alebo predvyplnená hodnota
- operácie na danej triede
- PSM triedy reprezentujúce aktuálnu PIM triedu.
- asociácie sú uložené znova cez unikátne identifikátory

Každá PSM trieda obsahuje podobný zoznam asociácii (tentokrát PSM asociácii), komponent tvoriacich obsah triedy a v neposlednom rade aj popis PSM triedy.

Nasledujúce kapitola 2.4 detailne popisuje štruktúru XForms formulárov. V jej úvode si zľahka predstavíme súvisiace technológie ako XPath či XML Schema.

Kapitola 4

Prevod do XForms

Hlavnou myšlienkou našej práce je generovanie XForms formulárov z modelu XSEM. Konceptuálny model je vytvorený najskôr na najvyššej úrovni abstrakcie a je teda zaznamenaný v XSEM-ER modeloch. Nakoľko ale XSEM-ER nie je jednoznačne prevoditeľný na XML použijeme PSM model. Každý PSM model je odvodený od PIM modelu. K jedinému PIM modelu môže existovať niekoľko rôznych PSM, ktoré sú v určitom zmysle *pohľadmi* na modely z XSEM-ER úrovne. Každý takýto pohľad definuje štruktúru XML dát, s ktorými chceme vo formulároch pracovať.

Primárnou náplňou nasledujúcej kapitoly je práve prevod XSEM-H pohľadov na XForms formuláre.

4.1 Možné spôsoby prevodu

Pri výbere spôsobu implementácie je potrebné zvažovať viacero rôznych možností. V tejto oblasti bolo publikovaných niekoľko akademických článkov, z toho niektoré obsahovali aj štatistické výsledky implementácií. Ich rozdelenie a porovnanie je možné na základe dvoch hlavných kritérií:

- Prvé možné rozdelenie je založené na použitej technológii. Prvá skupina sú technológie založené na XML ako napríklad XSLT. Druhou sú samostatné aplikácie programované v jazykoch ako Java alebo C# teda klasický OOP prístup.
  - XForms a mnohé druhy popisu dát sú založené na technológii XML. (Nakoniec aj zápis XSEM-H pohľadov je v serializovanej

podobe ukladaný vo formáte XML.) Je preto prirodzené spracovávať tieto XML súbory ďalšou technológiou založenou na XML. Vytváranie XForms za použitia XSLT je výrazne odlišný prístup k programovaniu prevodu, nakoľko sa jedná o neprocedurálne programovanie [8, 11]. Nevýhodou je, že dnešné vývojové nástroje zatiaľ neumožňujú plnohodnotne a pohodlne vyvíjať v XSLT. Rovnako si použitie XSLT technológie vynucuje plne automatický prístup, keďže neexistuje priamy spôsob ako by užívateľ mohol ovplyvňovať proces XSLT transformácie.

  – V protiklade k neprocedurálnemu prístupu stojí spomínaný OOP prístup [7]. V tomto prípade je omnoho jednoduchšie spolupracovať s užívateľom pri procese. Existujú dve možné prístupy v práci s XML: SAX a DOM. SAX je metóda založená na udalostiach vhodná pre pamäťovo náročné aplikácie vyžadujúce veľmi vysoký výkon. Toto ale nie je náš prípad a preto je možné použiť DOM prístup. Celý vstupný dokument je teda v pamäti počítača a je možné v ňom ľubovoľne prechádzať uzly a opakovane k nim pristupovať.

- Všetky spomínané práce popisujú plne automatický prevod. Nevýhodou je, že pre doplnenie chýbajúcich informácii je niekedy potrebné poskytnúť špeciálne označované schémy. Chýbajúce údaje by mohol počas transformácie doplniť užívateľ. To nás privádza k inému pohľadu na implementačný prístup - *mieru automatizácie*.

  – plne automatické generovanie - celý proces transformácie pohľadov XSEM-H prebieha bez zásahu užívateľa. Veľkou výhodou je umožnenie dávkového spracovávania veľkého množstva modelov či napríklad transformácia na vyžiadanie až v čase, keď je formulár skutočne potrebný. Nevýhodou sú zvýšené nároky na spracovávané schémy. Častou podmienkou bývajú špecifické anotácie, ktoré musia byť pridávané užívateľmi alebo špeciálnymi krokmi, ktoré chýbajúce informácie dopĺňajú z iných zdrojov. Rovnako je znížená možnosť prispôsobovania výsledných formulárov.

  – polo automatické generovanie - proces transformácie je pod dohľadom užívateľa, ktorý môže v určitých bodoch prispôsobovať výstupný formulár napríklad rozhodovaním o správaní sa opakujúcich elementov. V prípade, že chýbajú potrebné informácie,

ktoré by zabránili úspešnému spracovaniu modelu, užívateľ doplní dáta a tým zvyšuje univerzálnosť procesu.

4.2 Popis algoritmu

Vstupom prevodu konceptuálneho modelu na XForms je model vyprodukovaný projektom XCase [19] serializovaný v podobe XML súboru. Používaný formát v konečnom dôsledku popisuje stromovú štruktúru XML dát. Jeho zápis ale nie je práve ideálne *stromový*. Je to skôr zoznam tried a uzlov, ktoré ich reprezentujú. Stromovú štruktúru teda popisuje pomocou unikátnych identifikátorov - sekvenčne priraďované čísla - a odkazmi na ne. Výhodou je zjednodušený popis takejto serializácie nakoľko nenastávajú rekurzívne vzťahy vo výslednom XML. Na druhú stranu je stále zachovaná možnosť modelovať rekurzívne vzťahy. Miernou nevýhodou je práca s inštanciami daného XML. Je potrebné pracovať s vyhľadávaním podľa spomínaných identifikátorov, čo robí programové rozhranie (API) slabším. Najdôležitejšia je však možnosť daný zápis *stromovo* prechádzať, ktorá ostala zachovaná

Výsledným produktom celého procesu je v našom prípade XForms formulár vložený do XHTML. Na zachytávanie štruktúry dát je plne využívaná stromová podstata XML (viď. str. 21). Odkazovanie pomocou identifikátorov je použité presne podľa špecifikácie XForms 1.0 [2].

Navrhovaný algoritmus pracuje v dvoch hlavných fázach:

- Prvá fáza postaví zo vstupného modelu obsah `xforms:instance`, ktorý je v istom zmysle šablónou odosielaných dát. Šablóna, ale ešte nemusí byť hotová. Jej rozšírenie môžu spôsobiť napríklad opakujúce sa prvky alebo možnosti výberu obsahu formulárov. O každom z vytvorených uzlov si uloží malé množstvo informácií, ktoré pomôžu zjednodušiť druhú fázu.

- Druhá fáza použije ako vstup výsledok predošlého kroku a vytvorí výsledné užívateľské rozhranie s väzbami na instance. Použité vstupné prvky môžu byť odvodené od dátového typu poľa, alebo zadané užívateľom.

4.2.1 Prvá fáza

Úlohou prvého kroku je vytvoriť predbežnú podobu odosielaných dát. Je potrebná pre druhú fázu na vytváranie XPath výrazov. Výhodou dopredu

zostavenej šablóny je možnosť mať v druhom kroku jednotlivé elementy predpripravené a tým pádom na nich ako na objektoch volať potrebné metódy. Šablóna nie je kompletná pretože sa môže v druhom kroku ešte zmeniť. Túto zmenu si ale bližšie vysvetlíme v časti 4.2.2 na str. 50. Takto postavená šablóna neobsahuje žiadne informácie o typoch alebo povolených hodnotách, ktoré by mohli byť získané z konceptuálneho modelu. Všetky tieto informácie ale pomôžu spresniť prevod v druhej fáze. Je preto vhodné ukladať si k jednotlivým uzlom instance pomocné informácie vhodné pre neskoršie použitie.

Rekurzívny algoritmus vyžaduje parametre:

- zdroj - uzol XSEM-H pohľadu
- cieľ - potomok výsledného elementu `xforms:instnace`

Spustenie algoritmu si vyžaduje spoluprácu s užívateľom nakoľko serializovaný XSEM-H pohľad projektu XCase môže obsahovať niekoľko vrcholových (tzv. root, top-level) uzly. Je teda na užívateľovi, aby vybral prevádzaný uzol. Prevádzať je možné iba vrcholové uzly. Ako cieľový argument je v prvej iterácii použitý jediný povolený potomok `xforms:instnace`, ktorý bude vytvorený podľa mena prevádzaného XSEM-H konštruktu a to bez ohľadu na prítomnosť popisku elementu (element label). Tento parameter sa dá prirovnať k *ukazovátku* v instance, kde sa spracovanie práve nachádza.

Prvá fáza algoritmu je veľmi jednoduchá. Od vrcholového elementu až k listom sa rekurzívne spracuje každý uzol v XSEM-H pohľade podľa jeho typu takto:

- **element s popiskom** - nový uzol v instance. Všetky atribúty a informácie o ich typoch ulož k novovytvorenému uzlu v instance. Pre každého potomka spusť algoritmus znova a ako vstupné parametre použij aktuálne spracovávaného potomka ako zdroj a novo vytvorený uzol v instance ako cieľ.
- **element bez popisku** - instance ostáva nezmenená. Všetky atribúty a informácie o ich typoch ulož v instance k uzlu, ktorý bol na vstupe ako cieľ. Pre každého potomka spusť algoritmus znova a ako vstupné parametre použij aktuálne spracovávaného potomka ako zdroj. Cieľ ostáva nezmenený a teda použij rovnaký objekt ako bol na vstupe.
- **kontainer atribútov** - pre každý atribút z kontajner vlož do aktuálneho miesta v instance nový uzol s menom odvodeným od mena atribútu.

K týmto uzlom ulož informáciu o type. V tomto mieste rekurzívne volanie vždy skončí.

- **content choice** - pre každého potomka vytvor nový uzol v instance a použij ho ako vstup pre nasledujúce volanie spolu s daným potomkom. Do aktuálneho uzlu instance ulož informáciu o tom, že bol prevádzaný uzol s možnosťou výberu. Ak má uzol typu "content choice" zmenenú kardinalitu vygeneruj upozornenie na možnú chybu v prevode (viz. str. 52).

- **content container** - nakoľko tento uzol nepridáva žiadne špecialne spracovanie je potrebné vyrobiť iba nový uzol v instance (jeho meno je odvodené od názvu kontejneru) a použiť ho ako cieľ pri spracovaný všetkých potomkov.

- **štrukturálny zástupca s popiskom** - spracuj rovnako ako element s popiskom, do spracovania zaraď aj obsah zastupovaného elementu, ktorý je získaný podľa referenčného ID.

- **štrukturálny zástupca bez popisku** - spracuj rovnako ako element bez popisku, do spracovania zaraď aj obsah zastupovaného elementu, ktorý je získaný podľa referenčného ID.

Pre zachovanie prehľadnosti je popis spracovania štrukturálnych zástupcov zjednodušený. Ak by sa tento algoritmus implementoval presne takto, viedlo by to na možné nekonečné zacyklenie. Tento prípad nastáva ak sa štrukturálny zástupca odkazuje na niektorého zo svojich predkov. V prípade, že zastupovaný element je z iného podstromu funguje aj popísaný algoritmus.

Ako posledné ostali konštrukcie union a špecializácie, ktoré nie sú v predkladanej prototypovej implementácii podporované.

Príklad 1 (Prvá fáza) *Vytvorenie instance elementu pre príklad zobrazený na strane 16 by prebiehalo takto:*

Jediný vhodný zdrojový element na transformáciu je vrcholový element Aktivacia. Element `xforms:instance` *preto dostane id Aktivacia a je do neho vložený rovnako pomenovaný koreňový uzol, ktorý je použitý ako cieľový parameter pri prvej iterácii algoritmu.*

Spracovanie uzlu Aktivacia spracuje jediného potomka a to uzol Roaming, ktorý si do druhej časti ponesie informáciu o zmenenej kardinalite. Nový uzol Roaming je preto do instance pridaný ako potomok uzlu Aktivacia. Nová

iterácia algoritmu sa postará o ďalšie spracovanie. Vstupnými parametrami sú XSEM-H uzol Roaming a nový uzol Aktivacia z instance.

Zdrojový uzol má popisok a preto je do instance prostredníctvom cieľového atribútu pridaný uzol Roaming. Atribút Do je pridaný do informácii o uzle Roaming spolu s informáciou o type. Nasleduje spracovanie elementu content choice. Nová iterácia je spustená s novým uzlom Roaming.

Všetky uzly pod content choice sú spracované ako elementy s popiskom a sú pridávané do instance ako potomkovia uzlu Roaming. Každý z nich si do druhej časti spracovania nesie informáciu o svojom atribúte Od.

Výsledná instance sa od instance z príkladu líši počtom opakovaní a neprítomnosťou atribútov. Je štruktúra je teda:

```
<xf:instance id="Aktivacia">
<Aktivacia xmlns="jka">
 <Roaming >
  <Data />
  <World />
  <Europe />
 </Roaming>
</Aktivacia>
</xf:instance>
```

4.2.2 Druhá fáza

Druhá časť prevodu modelu na formulár pokračuje prácu nad instance z prvého kroku. Základnou myšlienkou, aby si každý uzol v instance sám vygeneroval všetko potrebné pre vlastné zobrazenie. K tomu mu poslúžia informácie z prvej časti procesu. Samotná poloha uzlu v instance uchováva takmer kompletnú informáciu potrebnú pre vytváranie XPath výrazov. Informácie o type pomôžu vytvoriť artefakty pre *binding*.

Algoritmus opäť pracuje s dvoma parametrami:

- zdroj - uzol z vytvorenej instance
- cieľ - element z namespace XForms, do ktorého budú generované vstupné prvky.

Prístup na niekoľko ďalších lokácií je zabezpečený staticky. Ide napríklad o pridávanie elementov `xforms:bind`. Aj napriek tomu, že sú všetky generované v tejto časti, ich pridávanie nevyužíva ani jeden z parametrov. Sú

pridávané na jediné dobre definované miesto v elemente `xforms:model`. Podobný príklad je aj element `xforms:submission`.

Prvé spustenie algoritmu použije ako vstup jediného potomka elementu `xforms:instance`. Ako cieľ sa na začiatku použije prázdny `xforms:group`, ktorý bude po skončení obsahovať celé užívateľské rozhranie prislúchajúce prevádzanému dokumentu. (Znamená to, že teoreticky je možné mať na jednej stránke viacero oddelených dokumentov.)

Príklad 2 (Druhá fáza) *Uzol Aktivacia neobsahuje žiadne špeciálne informácie o svojom type, slúži iba ako obálka pre svojich potomkov. Vytvorená bude iba xforms skupina* `xforms:group` *s vhodným popiskom. Novovytvorená skupina sa použije ako cieľ pri ďalšom volaní algoritmu.*

Spracovanie elementu Roaming je o niečo komplikovanejšie pretože ide o spracovanie opakujúcej sa možnosti voľby obsahujúcou atribút.

Element uloženú prítomnosť atribútu `Do`*. Je preto upravený uzol v instance a je mu pridaný tento atribút. Ak by užívateľ poskytol defaultnú hodnotu, bola by zaznamenaná do instance. Pomocné údaje udávajú typ ako dátum a preto je vygenerovaný element* `xforms:bind` *s XPath výrazom odkazujúcim na tento atribút tak, aby obmedzoval typ iba na platný dátum.*

Uzol má z prvej fázy uloženú informáciu o zmenenej kardinalite. Bude preto vygenerovaný element `xforms:repeat` *s XPath výrazom odkazujúcim na element Roaming. Ako prvý potomok tohoto elementu sa vytvorí xforms skupina s XPath výrazom odkazujúcim na kontext aktuálne opakovaného prvku. Umiestnenie tejto skupiny by sa mohlo zdať bezvýznamné. Zohráva ale rolu pri vyhodnocovaní relevancie častí užívateľského rozhrania a preto je v našom príklade potrebná. Práva táto skupina bude použitá ako cieľ pri ďalšom spracovaní.*

Aby bolo možné vyvolávať akcie pridávania a odoberania sú vygenerované aj elementy `xforms:trigger`*. Ich umiestnenie môže byť rôzne podľa preferencií užívateľa. Je možné umiestniť tlačítka pre odoberanie opakovaní vedľa každého opakovania alebo napríklad umiestniť jedno globálne tlačítko, ktoré odoberie práve označené opakovanie. Podobné možnosti prispôsobenia sa týkajú tlačítok pridávania. Ich voľby sú oddelené a nemajú vplyv na seba navzájom.*

Dokončenia spracovania zmenenej kardinality vyžaduje v instance zreplikovať element. Táto kópia navyše slúži ako vzor, ak by užívateľ zmazal všetky opakovania. Aby nebola zobrazovaná je vygenerovaný `xforms:bind`*, ktorý zneplatňuje prvý výskyt. Takto nebude nikdy zobrazený ani odoslaný.*

Spracovanie elementu Roaming má pokračovať spracovaním potomkov a novým cieľom sa stal element `xforms:group` *zo spracovania opakovaní (je potomkom elementu* `xforms:repeat`*). Z prvej fázy je ale jasné, že má byť umožnený výber jedného z potomkov. Pre uloženie aktuálnej voľby je vytvorený dočasný pomocný atribút elementu Roaming. Pre tento atribút je vytvorený* `xforms:bind` *element, ktorý ho vždy zneplatní. Vďaka tomu nie je nikdy odoslaný, ale je možné používať jeho hodnotu v XPath výrazoch.*

Element `xforms:switch` *použijeme na zobrazovanie správnej časti užívateľského rozhrania. Podobe ako* `xforms:repeat` *sú vygenerované tlačítka na prepínanie aktuálnej voľby. Tlačítka používajú element* `xforms:toggle` *na prepínanie v rámci switch modulu a action module a* `xforms:setvalue` *sa postarajú o nastavenie správnej hodnoty pomocného atribútu.*

Posledná časť spracovania je naplnenie `xforms:switch` *elementu. Pre každého potomka je vytvorený element* `xforms:case` *a použitý ako cieľ. Navyše sú vygenerované elementy* `xforms:bind`*, ktoré podľa aktuálnej hodnoty pomocného atribútu zneplatňujú nezobrazené časti formuláru.*

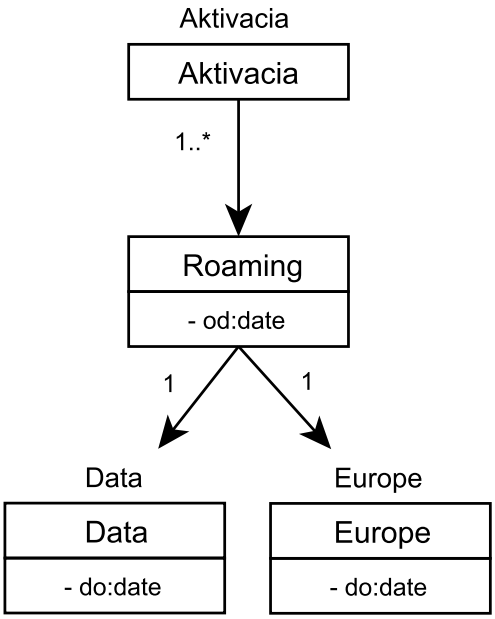
Spracovanie potomkov je už jednoduché a zopakuje tri krát jednoduché vytvorenie atribútu a `xforms:bind` *elementov.*

4.3 Analýza algoritmu

V nasledujúcej časti si popíšme niektoré konštrukcie, ktoré sa pre prevod do XForms veľmi nehodia prípadne slepé uličky objavené pri návrhu a implementácii prevodu. Nebudeme detailne rozoberať časovú ani priestorovú náročnosť programu. V kapitole 4.3.1 ukážeme, že aj celkom prirodzené a jednoduché konštrukcie ako *modelové skupiny* môžu znamenať problém. Kapitola 4.3.2 pojednáva o práci s výberom možností a možnej implementácii. Nasleduje vysvetlenie problémov so štrukturálnymi zástupcami v kapitole 4.3.3. Posledná časť ukazuje, že XForms môže v niektorých prípadoch prevýšiť aj silu jazyka XML Schema.

4.3.1 Opakovanie modelovej skupiny

Pre XSEM-H je hrana vedúca do uzlu bez popisku celkom bezproblémová konštrukcia. Prevod konštrukcie do XForms je možný až do chvíle, keď je na danej hrane zmenená kardinalita. Príklad takejto skladby xml pohľadu je na obrázku 4.3.1, ktorý znázorňuje iné namodelovanie druhej verzie nášho



Príklad 4.1: Konceptuálny model opakovania modelovej skupiny

motivačného príkladu. Hrana vedúca z uzlu *Aktivacia* má zmenenú kardinalitu a vedie do uzlu *Roaming*, ktorý nemá popisok. Výsledkom dát podla tohto modelu by bolo napríklad nasledujúce XML:

```
<Aktivacia>
   <Od>2009-01-01</Od>
   <Do>2009-02-02</Do>
   <Od>2009-03-03</Od>
   <Do>2009-05-04</Do>
</Aktivacia>
```

Na prvý pohľad sa zdá, že podobnú konštrukciu bude možné v XForms opakovať. Použitím niekoľkých insert operácii po stlačení tlačítka bude možné vložiť jeden element `Od` a jeden element `Do`. Napríklad takto:

```
<xforms:trigger id="pridaj">
  <xforms:label>Pridaj</xf:label>
```

```
  <xforms:insert ev:event="DOMActivate" position="after"
       at="last()+1" nodeset="Od"/>
  <xforms:insert ev:event="DOMActivate" position="after"
       at="last()+1" nodeset="Do"/>
</xforms:trigger>
```

Problém však nastáva pri pridávaní, pretože obe elementy `xforms:insert` sú naviazané na vlastnú množinu uzlov a teda funkcia last vráti pozíciu za posledný uzol `Od` resp. `Do`. Tým by pridávanie nedodržalo striedanie sa elementov, ale pridávané elementy by vytvorili skupinu elementov jedného a potom druhého mena.

Rovnaký problém nastáva už pri samotnom zobrazovaní. Predstavme si, že vyššie uvedené XML poslúži ako vstupné dáta. Užívateľské rozhranie musí zobraziť opakujúcu sa skupinu. Neexistencia spoločného predka dvoch opakovaných uzol znemožňuje vytvorenie XPath výrazu pre `xforms:repeat`. Pokus použiť element `xforms:repeat` niekoľko krát za sebou zlyhá rovnako ako pridávanie na nesprávne poradie elementov `Od` resp. `Do`.

4.3.2 Content choice

V príkladoch a popise algoritmu pracujeme s možnosťou zvoliť si jednu z ponúkaných možností a odoslať len príslušnú časť formulárových dát. Predstavené riešenie by sa mohlo zdať ťažkopádne a príliš komplikované. Pokúsime sa teda priblížiť niekoľko problémov pri práci s možnosťami.

Umožnenie prepínania medzi možnosťami má dve základné prístupy:

- switch module - využitie elementov `xforms:switch`, `xforms:case` a `xforms:toggle` na prepínanie aktuálne zobrazenej časti formuláru.
- prepínanie na základe dát v instance (*model based switching*) - ak je uzol v instance vyhodnotený ako nerelevantný všetky časti užívateľského rozhrania musia byť podľa špecifikácie vyradené zo zobrazovania.

Ako si však ukážeme použitie iba jedného z týchto prístupov nestačí na pokrytie všetkých našich požiadavok na formulár.

Použitie switch modulu

Táto defaultná možnosť v rámci XForms sa javí ako ideálne riešenie. Formulár je prirodzeným spôsobom rozdelený na potrebné časti. Jednoduché

vyvolanie akcie so správnym ID vyradí nepotrebnú časť formuláru zo zobrazovania a nastaví inú ako relevantnú pre zobrazovanie.

Nevýhodou je odosielanie dát, ktoré žiadnym spôsobom neberie do úvahy zobrazené a nezobrazené elementy. Aj napriek tomu, že je nejaká časť formuláru umiestnená v nezobrazenej časti je považovaná za relevantnú a ako taká je serializovaná pri odosielaní dát. Toto správanie sa môže zdať nelogické v kontexte našeho použitie. Nie je to ale chyba špecifikácie, ale zámer jej tvorcov.

Toto správanie je výhodne ak chceme docieliť správanie sa podobné sprievodcom (*wizards*). Rôzne stránky sú umiestnené do rôznych elementov `xforms:case`, čím je zobrazená iba jedna stránka v nejakom okamžiku. Tlačítka na stránkach iba prepínajú práve zobrazenú časť na nasledujúcu resp. predošlú stránku. Užívateľ tým má pocit *listovania* po stránkach, ktoré je navyše veľmi rýchle, takmer s okamžitou odpoveďou. Pri listovaní nie sú žiadne dáta odosielané na server. Odosielanie spustí až tlačítko na poslednej stránke, ktoré odošle dáta vyplnené v celom formulári teda na všetkých stránkach. Iným príkladom výhodnosti switch modulu je možnosť vytvoriť tzv. *záložky*.

Prepínanie na základe dát

Prepínanie na základe dát znamená využiť definované správanie sa XForms k nerelevantným prvkom. Tie sú vyradené zo zobrazenia a odosielania. Vďaka tomu by malo byť možné prepínať zobrazenie a zároveň riadiť serializáciu odosielaných dát.

Formulár by obsahoval pomocný atribút, kde by sa aktuálna hodnota ukladala. Podľa tejto hodnoty je v XPath výrazoch riadená relevancia jednotlivých možností. Tento pomocný atribút ale nepatrí do odosielaných dát. Je preto označený ako nerelevantný. To síce spôsobí, že nebude odoslaný, ale ako nerelevantný spôsobí nezobrazenie akéhokoľvek ovládacieho prvku, ktorý by nastavoval jeho hodnotu.

Riešením by mohlo byť uchovávanie pomocných hodnôt v oddelenej instance, ktorá by sa nikdy neodosielala. XPath výrazy by sa jednoducho odkazovali na inú instance pomocou funkcie *instance()*. Nad jednoduchými príkladmi funguje pomocná instance výborne. Formuláre sa správajú podľa modelu, zobrazenie aj odosielanie je korektné. Toto riešenie funguje pre motivačný príklad, kde neopakujeme objednávky.

Jednoduché pridanie opakovaní v konceptuálnom modely spôsobí ne-

funkčnosť pomocnej instance. Presnejšie formulár vygenerovaný týmto spôsobom by sa nesprával podľa očakávaní. Nakoľko všetky opakovania by sa odkazovali na jediný pomocný atribút v instance, všetky opakovania by menili svoje možnosti naraz a vždy na rovnakú hodnotu. Riešením by bolo v pomocnej instance určitým spôsobom kopírovať opakovania z formulára a udržiavať pomocné hodnoty v duplikátnom strome. Toto riešenie je ale pomerne komplikované.

Udržiavanie dát v oddelenej instance je teda nefunkčné alebo príliš komplikované. Zamerajme sa teda na možnosť uložiť aktuálnu voľbu v jedinej instance spolu s dátami k odoslaniu.

Pomocné atribúty by mohli byť relevantné po celu dobu práce s formulárom. To by vyžadovalo špeciálne spracovanie akcie *submit*, počas ktorého by tieto atribúty boli odstránené. Toto riešenie je pomerne komplikované. Nepoužiteľným ho však robí až skutočnosť, že na okamžik odosielania by mohol formulár zmeniť svoje zobrazenie kvôli zmene relevancie atribútov. Navyše by sa pri neúspešnom pokuse o odoslanie dát mohli stratiť užívateľské aktuálne voľby.

Výsledným riešením je kombinácia oboch prístupov. Základom je model based switching, kvôli riadeniu odosielaných dát. O prepínanie užívateľského rozhrania sa stará switch modul. (Switch modul by mohol byt nahradený elementmi `xforms:group`, ktoré by mali atribút `ref` naplnený XPath výrazom kontrolujúcim hodnotu pomocného atribútu. Použitie switch modulu je ale omnoho prehľadnejšie.)

4.3.3 Štrukturálny zástupca ako vlastný potomok

Výrazným obmedzením XForms je nemožnosť pracovať so stromovými dátami. Na internetovú formulárovú technológiu je to ale celkom kuriózny požiadavok. Formuláre slúžia v drvivej väčšine prípadov na zbieranie jednoduchých dát ako sú registračné formuláre či objednávky, kde sa niektoré záznamy opakujú.

Namodelovať ale rekurzívnu štruktúru v konceptuálnom modelovaní nie je najmenší problém. Ak uvážime príklad, kde má uzol jediného potomka, ktorý je jeho štrukturálnym zástupcom dopracujeme sa k rekurzii už na dvoch uzloch. Zaujímavejší je samozrejme príklad, kde má rovnakých potomkov dvoch, čo vedie na istú formu *binárneho stromu*.

Prípad s rekurziou nastáva vždy ak sa štrukturálny zástupca odkazuje na niektorého zo svojich predkov.

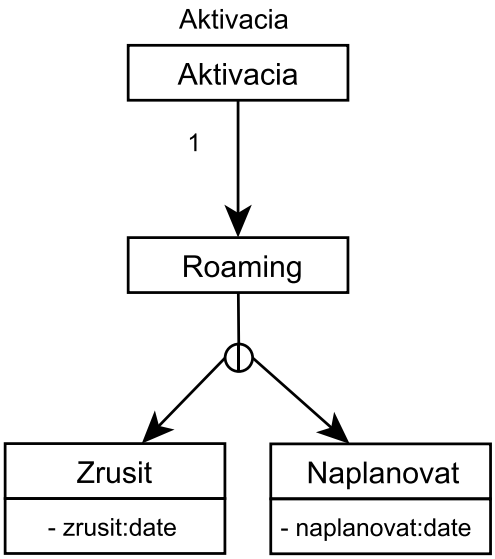
4.3.4 Výber atribútov

Nie vždy je však prevod do XForms iba obmedzujúci. Nasledujúca kapitola ukáže príklad, kde je XForms formulár vygenerovaný z konceptuálneho modelu, ktorý nemá ekvivalentný preklad do jazyka XML Schema. Možnosť zaradiť atribúty do rozhodovacieho uzlu `xsd:choice` v XML Schema neexistuje. Pre XForms je to ale obyčajný prípad, kde sa použije pomocný atribút. Rozdiel je iba v tom, že tentokrát budú elementy `xforms:bind` určovať relevanciu jednotlivých atribútov.

Nasledujúci príklad je premodelovanie najjednoduchšej verzie motivačného príkladu. Snažíme sa odosielať informáciu o zrušení alebo aktivovaní vo vhodne pomenovanom atribúte.

Implementácia, ktorá je súčasťou tejto práce zatiaľ prevod tejto konštrukcie nepodporuje.

Konceptuálny model je znázornený v príklade 4.2. Príslušný formulár je zobrazený na str. 58



Príklad 4.2: Konceptuálny model výberu atribútov

```
<?xml version="1.0" encoding="UTF-8"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:j="jka">
 <head>
  <xforms:model id="Model">
   <xforms:instance id="Aktivacia">
    <Aktivacia xmlns="jka">
     <Roaming temp="zrusit" zrusit="" naplanovat=""/>
    </Aktivacia>
   </xforms:instance>
   <xforms:bind
    nodeset="instance('Aktivacia')/j:Roaming/@temp"
    relevant="false()" />
   <xforms:bind
    nodeset="instance('Aktivacia')/j:Roaming/@zrusit"
    type="xsd:date"
    relevant="../@temp='zrusit'" />
   <xforms:bind
    nodeset="instance('Aktivacia')/j:Roaming/@naplanovat"
    type="xsd:date"
    relevant="../@temp='naplanovat'" />
   <xforms:submission
    id="id"
    action="http://xformstest.org/cgi-bin/showinstance.sh"
    ref="instance('Aktivacia')"
    method="put" />
  </xforms:model>
 </head>
 <body>
  <xforms:group>
   <xforms:label>Aktivacia</xforms:label>
   <xforms:repeat id="RoamingID" nodeset="instance('Aktivacia')/j:Roaming" model="Model">
    <xforms:group ref=".">
     <xforms:output value="@temp" />
     <xforms:trigger>
      <xforms:label>zrusit</xforms:label>
      <xforms:toggle case="zrusit" ev:event="DOMActivate" />
      <xforms:action ev:event="DOMActivate">
       <xforms:setvalue ref="@temp">zrusit</xforms:setvalue>
      </xforms:action>
     </xforms:trigger>
     <xforms:trigger>
      <xforms:label>naplanovat</xforms:label>
      <xforms:toggle case="naplanovat" ev:event="DOMActivate" />
      <xforms:action ev:event="DOMActivate">
       <xforms:setvalue ref="@temp">naplanovat</xforms:setvalue>
      </xforms:action>
     </xforms:trigger>
     <xforms:switch>
      <xforms:case id="zrusit">
       <xforms:input ref="@zrusit" />
      </xforms:case>
      <xforms:case id="naplanovat">
       <xforms:input ref="@naplanovat" />
      </xforms:case>
     </xforms:switch>
    </xforms:group>
   </xforms:repeat>
   <xforms:submit submission="id">
    <xforms:label>Potvrd</xforms:label>
   </xforms:submit>
  </xforms:group>
 </body>
</html>
```

Príklad 4.3: Formulár výber atribútov

4.4 Metodológia

Niektoré z prác spomínaných v kapitole 3 sú koncipované ako popis procesu vytvárania software. Práce rôzne pokrývajú životný cyklus software, hlavne webových aplikácií. Niektoré začínajú zberom a zaznamenaním užívateľských požiadavok či analýzou nepokrývajú však implementáciu. Iné začínajú až vo fáze modelovania domény a pokrývajú aj implementáciu.

Napriek tomu, že naším cieľom nebolo pokrývať životný cyklus vývoja procesu, uvedieme krátku metodológiu, ako modelovať a generovať XForms formuláre.

- Naša práca začína byť využiteľná až v neskorších fázach analýzy či designu. Vhodným nástrojom na zaznamenávanie vzťahov medzi entitami je už spomínaný XCase [19]. Analytiky môže zaznamenávať svoju prácu priamo v programe a vytvárať tak konceptuálny model (PIM).

  Výsledkom tejto časti je kompletný model predstavujúci konceptuálnu úroveň dát, ktoré budú výsledne formuláre odosielať.

- Z PIM modelov sa podľa princípov uvedených v práci [17] odvodia PSM. Stále pracujeme s nástrojom XCase. Vytvorené PSM diagramy predstavované údaje vo formulároch. Reprezentujú XML štruktúru odosielaných dát.

  Výsledkom je serializovaný XCase projekt vo forme XML súboru. Ten slúži ako vstup pre generovanie formulárov.

- Rôzny výber programovacích jazykov a prostredí spôsobuje mierne nepohodlie užívateľov, ktorí musia pre generovanie XForms formulárov použiť nástroj implementovaný ako súčasť tejto práce. Po načítaní serializovaného projektu je zobrazená stromová štruktúra všetkých koreňových elementov. Koreňové elementy potom môžu byť prevedené do podoby XForms formuláru. Formulár bude vytvorený tak, aby umožňoval zadávanie práve tých hodnôt, ktoré tvoria zvolený element. Samotný prevod je spracovaný vo forme sprievodcu.

  Výstupom je hotový XForms formulár.

- Poslednou úpravou, ktorá výrazne zlepšuje užívateľský zážitok je správna vizualizácia formulárov. Keďže cieľovým hostiteľským jazykom je v našej práci XHTML, používame rovnaké princípy. Doporučujeme teda upravovať vzhľad formulárov pomocou kaskádových štýlov CSS [6].

XForms mierne rozširujú možnosti štýlovania oproti bežnému XHTML. Napríklad vstupný prvok s neplatnou zadanou hodnotou má okamžite pridanú CSS pseudo triedu. Tým je možné označovať valídne prvky okamžite po zadaní hodnoty (alebo počas zadávania).

Kapitola 5

Implementácia XForms

V nasledujúcej kapitole si bližšie predstavíme technické riešenie prevodu konceptuálneho modelu vytvoreného projektom XCase do podoby formulára XForms. Na úvod preberieme možnosti, ktoré sme zvažovali pred implementáciou. V kapitole 5.2

5.1 Výber jazyka

Pri výbere vhodného jazyka je potrebné zvážiť niekoľko kritérií ako je dostupnosť potrebných knižníc, ich spoľahlivosť a dostupnosť, možnosť vytvárať príjemné užívateľské prostredie, výkon či náročnosť samotného jazyka.

Hľadáme teda prostriedky na vytvorenie užívateľom riadeného procesu transformácie konceptuálneho modelu na XForms. Konceptuálny model je dodaný ako rozsiahly XML dokument. Jeho štruktúru popisuje dodaná schéma v jazyku XML Schema. Model je potrebné prezentovať užívateľovi, ktorý spustí proces. Proces je riadený pomocou sprievodcu (*wizard*), ktorý zobrazuje prispôsobiteľné parametre, prípadne vyžiada od užívateľa chýbajúce dáta. Výsledný XForms kód potrebujeme na záver zobraziť užívateľovi, aby mohol formulár ešte viac prispôsobiť.

Najčastejšími jazykmi používanými pri práci s XML bývajú jazyky založené na platforme .NET (C#, C++) alebo Java. Existujú riešenia aj pre rôzne skriptovacie jazyky ako PHP, Python a iné.

Skriptovacie jazyky ako PHP majú pomerne náročné nasadenie, nakoľko vyžadujú inštaláciu niektorého z aplikačných serverov. Výber knižníc v týchto jazykoch býva pomerne široký od open source projektov, ktorých úroveň býva rôzna až po platené knižnice. Z dôvodu komplikovaného nasadenia

sme z možných kandidátov vyradili aj možnosť implementovať projekt v Java Enterprise Edition (J2EE). Najlepšia možnosť ako v týchto jazykoch vytvoriť užívateľské rozhranie je tvorba internetových stránok a to znamená použitie ďalšej technológie.

Výber sa teda zredukoval na jazyky C#.Net s prostredím *Microsoft Visual Studio* a Java second edition *J2SE* v prostredí *Eclipse* alebo *NetBeans*.

Prostredie *Microsoft Visual Studio*, ktoré sa používa pre prácu s jazykom C#.Net je veľmi intuitívne ale aj veľmi náročné. Formulárové a dialógové okná by boli vytvárané technológiou WinForms. Jazyk C#.Net natívne zvláda mnohé oblasti, kde Java potrebuje externé knižnice. Externé knižnice doplňujúce natívnu funkčnosť sú ale takmer výlučne komerčné.

Na druhú stranu J2SE v prostredí *Eclipse* poskytuje rovnako veľmi intuitívny spôsob vytvárania užívateľských dialógov. Najväčšou výhodou J2SE je voľná dostupnosť takmer každej knižnice a jej zdrojových kódov. Knižnice bývajú na vysokej technickej úrovni s množstvom funkcii.

Pre implementáciu sme preto vybrali technológie Java second edition. Projekt bude implementovaný ako zásuvný modul (*plug-in*) pre prostredie Eclipse.

Výhodou je možnosť využiť modelovací framework Eclipse pre doplnenie napríklad WYSIWYG editora pre prirodzenejšie navrhovanie formulárov. Na Jave založená technológia XML Beans[1] od neziskovej organizácie Apache umožní z XML Schema dokumentu vygenrovať knižnicu, schopnú pracovať s dokumentmi spĺňajúcimi danú schému. Táto možnosť je pre náš projekt výhodná pre prácu s XML konceptuálneho modelu.

5.2 Komponenty

Hlavnou triedou zásuvného modulu je trieda `XFormsPlugin`, ktorá musí byť potomkom abstraktnej triedy `AbstractUIPlugin`. Spolu z malou konfiguráciou sa táto trieda postará o rozpoznanie plug-inu prostredím Eclipse. Náš modul sa tak dozvie o akciách prostredia. Doplnili sme iba niekoľko málo statických prvkov ako napríklad XML parser a pretty-printer (vytvára formátované XML vhodné pre zobrazenie užívateľom).

V konfigurácii sme doplnili vlastný *editor*. Je to trieda, ktorá zabezpečuje mnoho funkcií ako je:

- zvýrazňovanie aktuálnej časti kódu

[1] http://xmlbeans.apache.org/

- napĺňa akciami kontextové menu
- udržiava zásobník vykonaných akcií vďaka čomu je možné vykonávať *undo* spustených akcií
- inicializuje farebné zvýrazňovanie častí kódu
- obsluhuje udalosti myši

Pre zobrazenie štruktúry práve upravovaného formuláru XForms sme upravili správanie sa štandardného pohľadu *ContentOutlineView*, kde je vytvárané vlastné kontextové menu.

Pre stromové zobrazenie konceptuálneho modelu sme pridali úplne nový pohľad, ktorý po načítaní modelu dynamicky vytvára obsah jednotlivých uzlov.

Pre pohodlie užívateľov bola pridaná nová perspektíva (vlastné rozloženie spomínaných pohľadov) a konfiguračná stránka umožňujúca zmenu prednastavených namespace.

5.2.1 Zobrazovanie kódu

Spomínaná komponenta *Editor* je potomkom triedy *TextEditor* a slúži primárne na zobrazovanie XForms kódu. Priame editovanie nie je povolené. Všetky úpravy budú prebiehať ako akcie na uzloch XML stromu zobrazenom v *ContentOutlineView* pohľade.

Text XForms kódu je farebne zvýraznený vďaka niekoľkým spolupracujúcim triedam, ktoré zastrešuje objekt `XFormsSourceViewerConfiguration`. Inštancie tried:

- `TagScanner`
- `CommentScanner`
- `PresentationReconciler`

zapuzdrené v objekte typu `PresentationReconciler` spoločne definujú pravidlá na rozpoznávanie XML značiek *tagov*, atribútov a komentárov.

Pri prechádzaní kódom sú označované príslušné uzly v stromovom zobrazení. Správny uzol sa rekurzívne hľadá za použitia informácii o začiatku a konci elementu z parsovania. Kontextové menu sa vďaka tomu môže správať identicky, akoby šlo o akcie vyvolávané na uzloch stromu.

5.2.2 Stromy

Pri rozsiahlejších formulároch a modeloch môže byť stromová reprezentácia prehľadnejšia ako textová. Prostredie Eclipse ponúka na prácu so stromovými zobrazeniami dobre prepracovaný základný model.

Základná myšlienka je, aby programátor pracoval s doménovými objektmi bez obmedzenia. Trieda implementujúca interface `ITreeContentProvider` sa potom postará o *preklad* doménových objektov na objekty vhodné k zobrazeniu v užívateľskom rozhraní. Ak je potrebné rozbaliť nejaký uzol, content provider dostane práve rozbaľovaný uzol a jeho úlohou je vrátiť potomkov. Provider túto úlohu typicky iba deleguje na aktuálny uzol. Jednotlivé uzly musia spĺňať určité podmienky. Ide napríklad o schopnosť vrátiť predka či potomkov alebo iba zodpovedať prítomnosť potomkov.

O rozlišovanie rôznych typov uzlov pomocou ikon sa stará objekt label provider. Je to jednoduchá trieda implementujúca iterface `LabelProvider`. Pri zobrazovaní nového uzlu je volaná metóda `getImage(...)`, kde parametrom je aktuálne zobrazovaný uzol. Zložitejšie modely môžu vyžadovať komplexnejšie rozhodovanie. V našom prípade je to iba výber z mnohých možností. Pre strom konceptuálneho modelu je výber založený jednoducho na type aktuálneho uzlu. Pre XForms strom je výber založený na triede uzlu. (Rozhodovanie na základe triedy bolo potrebné kvôli možnosti vkladať ikony aj do kontextového menu. Pri vytváraní kontextového menu ešte neboli jednotlivé objekty instanciované, boli dostupné iba ich triedy.)

V projekte sú implementované dva rôzne stromy, ktoré sú rôzne po stránke obsahovej aj implementačnej.

- strom XForms - ako content provider je použitá trieda `XFormsProvider`. Pracuje s uzlami typu `XFormsUzol`. Label provider pre tento strom je `XFormsLabelProvider`.
- strom konceptuálneho modelu - ako content provider je použitá trieda `SchemaProvider`. Pracuje s uzlami typu `SchemaUzol`. Label provider pre tento strom je `SchemaLabelProvider`.

Strom XForms

Strom zobrazjujúci aktuálnu štruktúru XForms formuláru je zobrazený v okne `Outline`.

Štandardné správanie tohto pohľadu dovoľuje označovanie viacerých uzlov. Preto sme niektoré časti preimplementovali a vytvorili nový strom na zobra-

zovanie s možnosťou označenia iba jedného uzlu (`SWT.SINGLE`). Označenie uzlu vyvoláva zvýraznenie príslušnej časti kódu v editore.

Základom úprav formulárov je vyvolávanie rôznych akcií. Dostupné sú základné akcie na manipuláciu s prvkami, práca so schránkou a možnosť vrátenia vykonaných akcií `undo` . Stromové zobrazenie navyše umožňuje premiestňovanie uzlov ťahaním.

Podpora vrátenia vykonaných akcií si vyžaduje, aby každá akcia vedela vrátiť všetky ňou ovplyvnené objekty do pôvodného stavu. Ide o objekty z balíka `xForms.Operations`. Každá akcia je rozšírením triedy `AbstractOperation`.

Implementácia práce so schránkou ako aj ťahania uzlov je mierne neštandardná. Eclipse predpokladá pri týchto akciách uloženie objektov do pamäte v binárnej podobe. Vyžadovalo by si to zoserializovanie XML elementov, neskôr ich deserializáciu a postavenie nového doménového elementu (potomka triedy `XFormsUzol`). Tento prístup je upravený tak, aby sa do systémov schránky uzol neukladal, ale aby si editor sám uložil ťahaný uzol. Tým je dokonale zachovaná možnosť ťahania a kopírovania uzlov v rámci jedného okna, no nie je možné kopírovať uzly mimo aktuálny pohľad. Všetky objekty a nastavenia pre umožnenie ťahania a kopírovania uzlov sú v metóde `createControl(...)` triedy `XFormsPage`.

Akcie na uzloch by nemali porušovať pravidlá XForms. Znamená to, že pred vložením prebehne test, či je možné vložiť označený uzol na dané miesto. Pri vkladaní nového uzlu sú v menu dostupné práve tie uzly, ktoré sú v danom mieste povolené. Obe tieto úkony využívajú metódu `canAccept()`.

Vďaka tomu, že bežný formulár nie je veľmi veľký, môže byť celý XForms strom zostrojený už pri inicializácii editoru. Pri úvodnom vytváraní XForms stromu sa nedá predpokladať, aké uzly sú potomkami práve spracovaného uzlu. Hľadanie elementov z príslušného namespace vykonáva objekt `XFormsFactory`, ktorý vytvára každý XForms uzol pri volaní metódy `createChildren()` v konštruktroch XForms uzlov. Umožňuje to rýchlejšiu odozvu pri niektorých akciách. Navyše je dosť pravdepodobné, že užívateľ počas práce nechá zobraziť podstatnú časť stromu, čo by predsa len spôsobilo rovnaké nároky na pamäť.

Strom zobrazujúci konceptuálny model

Konceptuálny model je stromovo zobrazený v prídavnom pohľade.

Pre prácu so serializovaným konceptuálnym modelom sme použili knižnicu

vygenerovanú pomocou XML Beans. To nám umožnilo objektovo pracovať s modelom takmer bez nutnosti čokoľvek programovať. Keďže typy možných potomkov sú známe, nie je potrebné používať pomocný factory objekt.

Modelový strom je závislý iba na vstupnom XML, preto neumožňuje zmenu štruktúry ťahaním a kopírovaním uzlov.

Na druhej strane tento strom sa dosť podstatne zúčastňuje na vytváraní grafického sprievodcu. Využíva sa metóda `getPages(...)`, kde sa vytvára stromová štruktúra stránok a prvá verzia instance elementu.

Aby bolo možné použiť aj iné modely ako je serializovaný model projektu XCase je práca s modelom oddelená vrstvou rozhraní. Stačí teda naimplementovať tieto rozhrania pre iný typ modelu a stromy rovnako ako celý proces transformácie na XForms ostanú funkčné.

5.2.3 Grafický sprievodca prevodu

Celý prevod by mal byť užívateľovi prezentovaný v prehľadnej podobe. To je zabezpečené implementovaním triedy `SchemaWizard`.

Pred vytvorením sprievodcu sú stránky získané zoserializovaním stromového usporiadania stránok. Prechodom od koreňa každý uzol vytvorí stránku podľa vlastného typu. Použijú sa samozrejme všetky dostupné informácie. Ak mal uzol zmenenú kardinalitu vloží svoju stránku ako potomka do stránky s nastaveniami opakovaní. Vlastnú stránku potom ešte poskytne ako cieľovú stránku svojim potomkom (a atribútom). Pri tomto procese sa spolu so stránkami vytvára prvá verzia instance viď. str. 47. Každá stránka si uloží uzol instance pre ktorý bola vytvorena.

Ak je možné odvodiť hodnotu defaultneho namespace a formulár nepozná prefix pre daný namespace, je ešte pred prevodom zobrazené dialógove okno na získanie tohoto prefixu. Prevod začína vždy úvodnou stránkou (`VeryFirstPage`), kde sa nastavujú základné informácie ako je ID modelu, odosielacia akcia a iné. Ďalšie stránky už kopírujú štruktúru prevádzaného uzlu. Stránky niekedy vyžadujú, aby užívateľ zadal hodnotu. Ak nie je stránka vyplnená, nie je možné dokončiť prevod. Vždy je možné skočiť dopredu na prvú stránku, ktorá blokuje ukončenie.

Záver sprievodcu má dve časti. V prvom kroku prebehne celá transformácia, ale výsledok nie je zobrazený užívateľovi. Druhou časťou je aktualizácia textu. Vďaka tomuto oddialeniu zmeny textu je neskôr možné akciami `undo` prejsť aj do stavu pred generovaním.

Kapitola 6

Budúca práca

Teoreticky je konceptuálne modelovanie XML dát výborne pokryté pracou [17]. Navrhovaný model a prevod do XML Schema je implementovaný projektom [19]. Prevod modelu do XForms sme sa snažili pokryť po teoretickej stránke v tejto práci. Určite ale ostáva veľa priestoru na ďalšie skúmanie a vylepšenia. Niektoré z nich by mohli byť:

- v tejto práci sme model použili iba na generovanie XForms formulárov. Model však obsahuje dostatok informácii, aby bola vygenerovaná XML Schema. Ak by sme do generovania zapojili aj schému výsledky by sa mohli zlepšiť. Minimálne použitie schémy vo výslednom formulári môže zlepšiť použiteľnosť. Rovnako sme nevyužili možnosť konceptuálneho modelu popisovať akcie nad objektmi (metódy).

- výrazne rozdielny prístup k implementácii v našej práci a projektu XCase znemožnila tesnejšie spojenie procesu modelovania a prevodu do XForms. Integrácia do jednotného prostredia znamená výrazne prepracovanie minimálne jedného z projektov. Zlepšenie použitia by však bolo výrazné.

- priložená implementácia podporuje prácu s XCase modelom, ktorý je pomocou rozhraní celkom oddelený od logiky prevodu. Je preto možná implementácia aj pre iné zdrojové formáty ako napríklad DTD či XML Schema.

- predchádzajúce zlepšenie navrhuje ako zdrojový formát niektorý zo spôsobu popisu XML. Ak by sme sa tejto myšlienky ešte chvíľu držali,

mohli by sme navrhnúť iný modelovací nástroj, ktorý povedzme nemá priamu väzbu na XML a pokúsiť sa generovať XForms.

Nemožnosť priamej editácie kódu je pomerne dosť technickým obmedzením. Aj keď by bola úprava pomerne náročná zlepšenie programu by bolo výrazné Niekoľko menších technických nedostatkov.

- Akcia *Uložiť ako* umožňuje ukladanie aj do súboru, ktorý neexistuje. Bude tak vytvorený nový súbor. Tento prípad ukladania ale znefunkční farebné rozlišovanie častí.
- Tlačítko *Load ...* je dostupné, len ak je aktívna časť s kódom.
- Na spracovanie klávesových skratiek bol použitý systém, ktorý bol v najnovšom prostredí Eclipse označený ako *Deprecated*. (Samozrejme za neho bola vytvorená náhrada. Naneštastie nový systém je značne zložitejší a na prvý pohľad neprehľadnejší. Nakoľko okrem technickej dokumentácie nie sú dostupné žiadne ďalšie informácie, je použitie najnovšieho systému dosť obtiažne.)
- Otvorenie formuláru je považované za zmenu súboru a vyžaduje tak potvrdenie užívateľa o uložení zmien.

Kapitola 7

Záver

V tejto práci sme predstavili niekoľko rôznych spôsobov a metód prístupu k modelovaniu dát. Základnými diskutovanými prístupmi bolo ER modelovanie a metódy založené na MDA. MDA metodika bola použitá ako na modelovanie dát tak aj na modelovanie celých webových systémov.

Motivovaný najrôznejšími prístupmi sme predstavili metódu modelovania webových formulárov, ktorá je variatou čistým medzi modelovaním dát a modelovaním webového systému. Základom je modelovací jazyk XSEM, ktorý je už na konceptuálnej úrovni prispôsobený modelovaniu XML dokumentov.

V druhej časti práce pracujeme s platformne závislými modelmi jazyka XSEM a ukázali sme možný spôsob ich prevodu na webové formuláre zapísané v technológii XForms. Práca čiastočne popisuje aj fungovanie a štruktúru formulárov XForms.

Poslednou časťou práce je implementácia navrhovaného riešenia a jej popis. Výber prístupu k transformácii je založený na prácach s podobným zameraním.

Pokračovaním práce by mohlo byť rozpracovanie teoretickej časti prevodu aj na konštrukcie ako je rekurzia či opakovanie modelových skupín. S tým by súvisela zmena implementácie pre podporu iných výstupných formátov. Či experimentálne overenie úspešnosti navrhovanej metódy.

Appendix A

Obsah CD-ROM

Súčasťou tejto diplomovej práce je aj priložený CD-ROM, ktorý obsahuje text práce, zdrojové kódy projektu a niekoľko ukážkových príkladov.

```
/
|
|- /doc - užívateľská dokumentácia
|
|- /eclipse - prostredie Eclipse Ganymede
|
|- /examples - príklady formulárov a modelov
|
|- /Java - Java 5.0
|
|- /src - importovateľný projekt pre Eclipse
|
|- /text - zdrojové súbory práce
|
|- /xcase - inštalačný balík a užívateľská dokumentácia
|
|- kasarda.pdf - text práce vo formáte PDF
|
|- XForms_2.0.0.jar - jar balik so zásuvným modulom
```

Appendix B

XForms validátor

Vhodným nástrojom pri práci s ľubovoľným XML je validácia. Pravdepodobne ku každej technológii na základoch XML existuje schéma v podobe DTD alebo XML Schema zápisu. Validácia podľa schémy však neprihliada na sémantiku validovaného XML, dá sa povedať, že validovanému dokumentu *nerozumie*. Validátory, ktoré sú napísané priamo pre niektorú konkrétnu technológiu obyčajne dosahujú oveľa lepšie a presnejšie výsledky.

V súčasnosti pravdepodobne najlepším voľne dostupným validátorom XForms je projekt dostupný na `http://xformsinstitute.com/validator`.

Celá implementácia je v jazyku Python a zdrojové kódy sú voľne dostupné. Úroveň je celkovo pomerne dobrá. Projekt sa už pravdepodobne nerozvíja a preto aj detské chyby bránia lepšiemu užívateľskému pohodliu. Napríklad validácia elementu `xforms:submission` s atribútom `ref` vždy skončí chybou. Hlásenia chýb a určovanie riadkov, kde sa chyby vyskytujú potrebuje ešte výrazné zlepšenie.

Literatúra

[1] Booth D. et al.: *Web Services Architecture* W3C Working Group Note 11 February 2004. [http://www.w3.org/TR/ws-arch]

[2] Boyer J.M. et al.: *XForms 1.0 (Third Edition)* W3C Recommendation 29 October 2007. [http://www.w3.org/TR/2007/REC-xforms-20071029]

[3] Ceri S., Fraternali P., Bongio A.: *Web Modeling Language (WebML): a modeling language for designing Web sites* Dipartimento di Elettronica e Informazione, Politecnico di Milano [http://www9.org/w9cdrom/177/177.html]

[4] Clark J., DeRose S.: *XML Path Language (XPath)* [http://www.w3.org/TR/xpath]

[5] Chen P. (1976): *The entity-relationship model—toward a unified view of data*, Massachusetts Institute of Technology, Cambridge.

[6] CSS Working Group (2007): *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification* [http://www.w3.org/TR/CSS21/]

[7] Eunujung Lee, Tae-Hoon Kim: *Automatic generation of XForms code using DTD*, Kyonnggi Uversity

[8] Garvey P., French B.: *Generating user interfaces from composite schemas*

[9] Gómez J., Cachero C., Pastor O.: *On Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach* Universidad de Alicante, Universidad Politécnica de Valencia

[10] Kasarda J. (2007): *Editor XForms* Karlova univerzita, Praha

[11] Kisub Song, Kyong-Ho Lee: *An automated generation of XForms interfaces for web services* Yonsei university

[12] Koch N. (2006): *Transformation Techniques in the Model-Driven Development Process of UWE* Ludwig-Maximilians-Universität, München.

[13] Koch N., Zhang G., Escalona M. J. (2006): *Model transformations from requirements to web system design* Proceedings of the 6th international conference on Web engineering.

[14] Konzorcium W3C [http://www.w3.org/]

[15] Miller J., Mukerji J. (2003): *MDA Guide Version 1.0.1* [http://www.omg.org/docs/omg/03-06-01.pdf]

[16] Mohlanec M. (2001): *The Object-Oriented Hypermedia Design Model (OOHDM)* Katedra elektrotechnologie, České vysoké učení technické, Praha

[17] Nečaský M. (2008): *Conceptual Modeling for XML* PhD Thesis. Karlova univerzita, Praha

[18] Nečaský M. (2009): *Reverse Engineering of XML Schemas to Conceptual Diagrams* Karlova univerzita, Praha

[19] Nečaský M., Klímek J., Kopenec L., Kučerová L., Malý L. a Opočenská K.: *XCase - A Case Tool for Modeling XML Data* [http://www.necasky.net/xcase]

[20] Object Management Group (2007): *UML Infrastructure Specifcation 2.1.2* [http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/]

[21] Object Management Group (2007): *UML Superstructure SpeciŻcation 2.1.2* [http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/]

[22] Pokorny J., Halaška I.: *E-R modelování* Karlova univerzita, Praha [http://www.ksi.mff.cuni.cz/ pokorny/vyuka/srbd/er/]

[23] Schwabe D., Almeida Pontes R., Moura I. (1999): *OOHDM-Web: An environment for Implementation of Hypermedia Applications in the WWW* PUC-Rio, Brazil

[24] Schwabe D., Ricci L. A. (2006): *An authoring environment for model-driven web applications* Depto de Informática, PUC-RIO

[25] SVG Working Group: *Scalable Vector Graphics (SVG) 1.1 Specification* [Scalable Vector Graphics (SVG) 1.1 Specification]

[26] W3C HTML working group: *XHTML^{TM} 1.0 The Extensible HyperText Markup Language (Second Edition)* [http://www.w3.org/TR/xhtml1/]

[27] W3C XML working group: *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [http://www.w3.org/TR/REC-xml/]

[28] XML Schema Working Group *XML Schema Part 1: Structures Second Edition* [http://www.w3.org/TR/xmlschema-1/]

[29] http://www.x-smiles.org

[30] http://www.formfaces.com/

[31] http://www.alphaworks.ibm.com/tech/xfg/

[32] http://alphaworks.ibm.com/tech/vxd/