VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GENERÁTOR VĚDECKÝCH WEBOVÝCH PORTÁLŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE ŠTĚPÁN RŮŽIČKA
AUTHOR

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GENERÁTOR VĚDECKÝCH WEBOVÝCH PORTÁLŮ

WEB PORTAL GENERATOR

BAKALÁŘSKÁ PRÁCE

BACHELOR´S THESIS

AUTOR PRÁCE ŠTĚPÁN RŮŽIČKA
AUTHOR

VEDOUCÍ PRÁCE doc. RNDr., Ph.D. PAVEL SMRŽ
SUPERVISOR

BRNO 2007

Abstrakt

V tomto dokumentu se zabývám dynamickou webovou aplikací, kterou vytvářím za pomocí technologií Ruby on Rails, XML, MySQL a AJAX.

V úvodu vysvětluji, proč jsem si vybral právě toto téma. A popisuji, co lze v této publikaci najít.

Dále podrobněji rozebírám spolupráci Ruby on Rails s XML. Popisuji, co je v možnostech knihovny REXML a uvádím nějaké příklady pro snazší pochopení.

Následuje popis knihoven, které poskytují dynamické animační, zvukové a jiné efekty. Jedná se o Prototype a script.aculo.us. Opět uvádím nějaké příklady.

V předposlední kapitole se nejprve zabývám obecně projektem "Generátor vědeckých webových portálů" (zasazení do kontextu) a dále pak následují jednotlivé etapy při programování.

V poslední kapitole se na aplikaci snažím nahlížet z hlediska uživatelské přívětivosti. Tedy popisuji, jakým způsobem jsem se snažil udělat aplikaci přívětivou. Zde jsem také připojil reakce třech osob, které měly možnost si mnou vytvořenou aplikaci vyzkoušet.

V závěru shrnuji získané zkušenosti.

Klíčová slova

Ruby on Rails, REXML, parsování, XPath, DTD, iterace, bílé znaky, strom, streamované parsování, jmenné prostory, automatické zpracování, UTF řetězce, SAX2, AJAX, JavaScript, helpery, Prototype, script.aculo.us, framework, dynamické efekty, DOM, API, animované efekty, motivace, jádrové efekty, kombinované efekty, drag & drop, in-place editory, posuvníky, akce, buildery, zvuk, XML, HTML, CSS, webové rozhraní, ER diagram,

Abstract

In this document I am talking about dynamic web application, which I am making via Ruby on Rails, XML, MySQL and AJAX technology.

Introduce describes, why I have chosen this job, and what you can find in this document.

Next part is about Ruby on Rails and XML, where I am describing how to use REXML library and what it offers to you. There are some examples to better understanding that.

Continuing with describing libraries, which can offer dynamic, sound and some other effects. Prototype and script.aculo.us are described there and some examples as well.

Semifinal chapter is about Web portal generator in general at first and next about single steps when I was working on my application.

In last one chapter I am trying to look on my application from user's point of view. I am describing the way, how I tried to make application better for usual user. I joined some reactions from persons who had the opportunity to work with my application.

Document terminates by resume of gained experiences.

Keywords

Ruby on Rails, REXML, parsing, XPath, DTD, iterativ, white spaces, tree, stream parsing, name spaces, automatic processing, UTF strings, SAX2, AJAX, JavaScript, helpem, Prototype, script.aculo.us, framework, dynamic effects, DOM, API, annimation effects, motivation, core effects, combined effects, drag & drop, in-place editors, scrollers, actions, builders, sound, XML, HTML, CSS, web interface, ER diagram

Citace

Štěpán Růžička: Generátor vědeckých webových portálů. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Generátor vědeckých webových portálů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením...
Další informace mi poskytli...
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Chtěl bych poděkovat vedoucímu projektu, doc. RNDr., Ph.D. Pavlu Smržovi za jeho ochotu a trpělivost při konzultacích. Také je obdivuhodné, že i přesto, jak je zaměstnaný si dokázal najít čas a tím mi pomohl dotáhnout bakalářský projekt do zdárného konce.

© Štěpán Růžička, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů..

Obsah

Obsah 1
Úvod 2
1 Ruby On Rails a XML 4
1.1 REXML (Ruby Electric XML) 4
1.1.1 Stromové parsování a přístup k elementům 4
1.1.2 Streamované parsování 9
2 Ruby On Rails a AJAXové knihovny 13
2.1 Prototype a script.aculo.us 13
2.1.1 Prototype JavaScript Framework 13
2.1.2 Script.aculo.us 14
3 Vytvoření aplikace 27
3.1 Formulace cíle 27
3.2 Současný stav 28
3.3 Analýza problému a princip jeho řešení 28
3.3.1 Zadání 28
3.3.2 Nejasnosti 28
3.3.3 Možná řešení 29
3.4 Návrh řešení 29
3.5 Popis řešení 30
3.5.1 Použité technologie 30
3.5.2 Postup 30
3.5.3 Vlastní implementace 31
4 Aplikace z hlediska uživatelské přívětivosti 39
Literatura 42
Seznam příloh 43

Úvod

Pokud se ohlédneme do minulosti, určitě si vzpomeneme na okamžiky vniku významných vynálezů, bez kterých si dnes nedovedeme představit běžný den. Tehdy těmto milníkům nejspíš nikdo nepřipisoval takový význam, jakého nabyly v dnešní době. Avšak v průběhu dalších let (ať už desítek, či stovek) se ve vývoji pokračovalo a díky tomu dnes můžeme tyto okamžiky oslavovat a užívat výdobytky moderního světa, které nám odkázaly předchozí generace.

Čím se od těchto liší počítač a s ním spojený internet? Možná tím, že jsou to jedni z mála fenoménů, které se proslavili už nyní (pár let po jejich vzniku). O jejich významu nejspíše nikdo nikdy nepochyboval a to je také důvodem, proč je jejich vývoj tak rychlý, a proč naše společnost vynakládá takové úsilí pro jejich zdokonalování.

Doslova ze dne na den se rutinní činnosti jako například pošta, noviny, knihy, platební transakce, nákupy či telefonie přesunuly na internet. Myslím, že by bylo hodně těžké vrátit se v čase a obejít se bez něj. Možná bychom se také mohli stát obětí honu na čarodějnice, pokud bychom o těchto fenoménech v minulosti vyprávěli.

Internet se zkrátka stal součástí našeho života a všichni jej využíváme ať už jako zdroj informací, multimédií nebo komunikační prostředek.

I já vidím v internetu budoucnost a proto jsem si zvolil jako téma bakalářské práce "Generátor vědeckých webových portálů".

Jak už bylo řečeno výše, internet je "studnou" informací. Bohužel tato studna je často dosti hluboká a potápění může být pracné a únavné. Generátor má tuto operaci usnadnit a má za úkol žádaná data shromáždit a poskytnout jako ucelený obsah.

Konkrétně se zabývá shromažďováním dat o vědcích, o kterých vyhledá nejrůznější informace, ať už je to rok narození, počet sourozenců, či vydané publikace (které stáhne a rovněž zahrne do výstupu).

Protože je aplikace rozsáhlá, mým úkolem bylo vytvořit pouze její část.

"Úprava výsledků automatického rozpoznávání sémantických informací ze značkovaného textu". Takto zní můj úkol. Ve stručnosti lze říct, že se ve své práci zabývám úpravou dat, které se mají brát v úvahu při samotném stahování a shromažďování dat. Generátor nejprve vyhledá dostupné informace o daném vědci a vrátí XML soubor v dané struktuře. Protože je aplikace pouze program, jednotlivé entity nemusí být správně rozpoznány (například jeho prostřední jméno může být zaměněno za jméno publikace apod.). V tomto okamžiku přichází na řadu můj program, který uživateli tuto datovou strukturu poskytne v čitelné formě a nabídne mu její úpravu.

První kapitola nese název Ruby on Rails a XML. Jak již název napovídá, zabývám se zde frameworkem RoR[1], který jsem si pro svou implementaci vybral, a jeho vztahem k XML. Především zde popisuji knihovnu REXML, která operace s XML poskytuje. Tato knihovna nemá dlouhou historii, ani není nikterak složitá, proto jsem se zaměřil na její metody a jejich praktické využití.

Další kapitola se jmenuje Ruby on Rails a Ajax a zabývá se dynamickým obsahem. Především potom frameworkem/knihovnou "Prototype" a knihovnou "script.aculo.us". Jejich použití umožňuje "rozpohybovat" webové stránky. Opět zde uvádím něco málo příkladů, aby si čtenář mohl udělat představu, jak se s těmito nástroji pracuje.

V kapitole předposlední popisuji proces vytváření vlastní aplikace. Nejprve popisuji, jak moje část souvisí s celým projektem. Něco málo o použitých technologiích a dále popisuji jednotlivé etapy vývoje.

[1] Ruby on Rails

1 Ruby On Rails a XML

V této kapitole se zabývám knihovnou REXML, kterou ve své aplikaci hojně využívám.

1.1 REXML (Ruby Electric XML)

REXML je XML procesor určen pro jazyk Ruby. Inspirován `Electric XML` knihovnou pro jazyk Java, jehož přednostmi jsou jednoduchost, malá velikost a rychlost.

Podporuje jak stromové, tak streamované parsování. Streamované parsování je rychlejší (asi 1,5x). Avšak se streamovaným parsováním, nemůžeme využít některé nástroje (jako například jazyk Xpath).

1.1.1 Stromové parsování a přístup k elementům

1.1.1.1 Základní použití

Třetí řádek vytváří nový dokument a parsuje zadaný dokument.

```
require "rexml/document"
# používáme knihovnu REXML, tudíž ji musíme zahrnout do svého programu
file = File.new( "mydoc.xml" )
doc = REXML::Document.new file
```

Také je možné udělat toto:

```
require "rexml/document"
include REXML
# díky tomuto řádku nemusíme psát neustále prefix REXML::
string = <<EOF
  <mydoc>
    <someelement attribute="nanoo">Text, text, text</someelement>
  </mydoc>
EOF
doc = Document.new string
```

Takže parsování řetězce je stejně snadné jako parsování souboru.

1.1.1.2 Přístup k jednotlivým elementům

Lze zajistit několika způsoby:

1. Třída Element obsahuje metodu ‘each_element_with_attribute()‘

2. `Element.elements` - podtřída `elements` třídy `Element`disponuje metodou `each` a []. Obě metody mohou být použity s příkazem Xpath pro filtrování (ten tyto metody dělá opravdu výkonnými)
3. Pokud je Element podtřídou rodiče, můžeme k těmto subelementům přistoupit přímo pomocí metod `Element[]`, `Element.each`, `Element.find`, `Element.delete`. To je nejrychlejší způsob přístupu k potomkům.

Bereme v úvahu následující XML dokument:

```
<inventory title="OmniCorp Store #45x10^3">
  <section name="health">
    <item upc="123456789" stock="12">
      <name>Invisibility Cream</name>
      <price>14.50</price>
      <description>Makes you invisible</description>
    </item>
    <item upc="445322344" stock="18">
      <name>Levitation Salve</name>
      <price>23.99</price>
      <description>
        Levitate yourself for up to 3 hours per application
      </description>
    </item>
  </section>
  <section name="food">
    <item upc="485672034" stock="653">
      <name>Blork and Freen Instameal</name>
      <price>4.95</price>
      <description>
        A tasty meal in a tablet; just add water
      </description>
    </item>
    <item upc="132957764" stock="44">
      <name>Grob winglets</name>
      <price>3.56</price>
      <description>Tender winglets of Grob. Just add water</description>
    </item>
  </section>
</inventory>
```

Pak přistupujeme k elementům následovně:

```
doc = Document.new File.new("mydoc.xml")
doc.elements.each("inventory/section") { |element| puts
element.attributes["name"] }
# -> health
# -> food
doc.elements.each("*/section/item") { |element| puts
element.attributes["upc"] }
# -> 123456789
# -> 445322344
# -> 485672034
# -> 132957764
root = doc.root
puts root.attributes["title"]
# -> OmniCorp Store #45x10^3
puts root.elements["section/item[@stock='44']"].attributes["upc"]
# -> 132957764
puts root.elements["section"].attributes["name"]
# -> health (vrací první nalezený správný výsledek)
puts root.elements[1].attributes["name"]
# -> health (vrací prvního potomka)
root.detect {|node| node.kind_of? Element and node.attributes["name"] ==
"food" }
```

Potomci jsou v REXML indexovány od 1 (ne od 0) a to kvůli Xpath, který čísluje elementy právě takto. Např. root.elements['*[1]'] = = root.elements[1]. Poslední řádek kódu hledá prvního potomka se jménem `food`.

Také je možné k elementům přistupovat přímo přes Xpath, pomocí Xpath třídy.

```
# Invisibility cream je první položka (<item>)
invisibility = XPath.first( doc, "//item" )
# Vypíše všechny ceny (price)
XPath.each( doc, "//price") { |element| puts element.text }
# Vrací pole všech elementů "name" nalezených v dokumentu
names = XPath.match( doc, "//name" )
```

Jiný způsob jak dostat pole s korespondujícími uzly je použít 'Element.elements.to_a()'.

```
all_elements = doc.elements.to_a
all_children = doc.to_a
all_upc_strings = doc.elements.to_a( "//item/attribute::upc" )
all_name_elements = doc.elements.to_a( "//name" )
```

REXML se snaží dělat běžné věci jednoduše, to ale znamená, že ty neběžné můžou být komplikované. To obzvláště platí, co se týká textových uzlů.

Textové uzly mají spoustu povah a v případě interní entity, výsledek nemusí odpovídat očekávání. Když REXML čte XML dokument, nejdříve rozparsuje DTD a poté vytvoří tabulku interních entit. Pokud najde nějakou z těchto entit v dokumentu, vymění jí za jejich hodnotu.

```
doc = Document.new '<!DOCTYPE foo [
<!ENTITY ent "replace">
]><a>&ent;</a>'
doc.root.text   #-> "replace"
```

Pokud se dokument opět vypisuje, REXML opět vymění hodnoty za entitní reference.

```
doc.to_s
# Generuje:
# <!DOCTYPE foo [
# <!ENTITY ent "replace">
# ]><a>&ent;</a>
```

Zde nastává problém, pokud se nějaké slovo shoduje s entitní referenční hodnotou.

```
doc = Document.new '<!DOCTYPE foo [
<!ENTITY ent "replace">
]><a>replace &ent;</a>'
doc.root.text   #-> "replace replace"
```

REXML dělá pouze věc, kterou může

```
doc.to_s
# Generuje:
# <!DOCTYPE foo [
# <!ENTITY ent "replace">
# ]><a>&ent; &ent;</a>
```

a to se pravděpodobně neshoduje s očekáváním.

1.1.1.3 Vytváření XML dokumentu

Opět existuje několik mechanismů. Ruční vkládání dokumentu je rychlejší než příslušná metoda. Avšak záleží na každém co si vybere.

```
el = someelement.add_element "myel"
# Vytvoří element pojmenovaný "myel", vloží do "someelement" a vrátí
el2 = el.add_element "another", {"id"=>"10"}
# Dělá tu samou činnost, avšak ještě nastaví atribut "id" el2 na hodnotu
"10"
el3 = Element.new "blah"
el1.elements << el3
el3.attributes["myid"] = "sean"
# Vytvoří el3 pojmenovaný "blah", ten vloží do el1, potom nastaví
atribut "myid" na "sean"
```

Pokud je zapotřebí vložit nějaký text do elementu, je zde možnost vytvořit textový objekt a vložit jej do elementu nebo za použití příslušné metody 'text ='

```
el1 = Element.new "myelement"
el1.text = "Hello world!"
# -> <myelement>Hello world!</myelement>
el1.add_text "Hello dolly"
# -> <myelement>Hello world!Hello dolly</element>
el1.add Text.new("Goodbye")
# -> <myelement>Hello world!Hello dollyGoodbye</element>
el1 << Text.new(" cruel world")
# -> <myelement>Hello world!Hello dollyGoodbye cruel world</element>
```

Avšak nutno poznamenat, že všechny takto vložené objekty jsou stále uložené odděleně. el1.text vrátí "Hello world!", el1[2] vrátí textový objekt s obsahem "Goodbye".

Také je nutné si uvědomit, že všechny texty v REXML jsou uloženy v UTF-8. Je sice možné vkládat a vracet jiné kódování (UTF-8, UTF-16, ISO-8859-1, UNILE), ale uvnitř programu by se měly vyskytovat pouze REXML UTF-8 řetězce.

REXML není vždy schopen uhádnout v jakém kódování text je, potom se předpokládá, že text je v UTF-8 (bez jakéhokoliv varovného hlášení). Pokud máme text v jiném kódování, měli bychom ho překonvertovat pomocí 'text.unpack("*C").pack("U*")'.

Pokud chceme vložit elementy mezi dva elementy, lze tak učinit pomocí "array notace" (pole) nebo "parent.insert_before()" a "parent.insert_after"

```
doc = Document.new "<a><one/><three/></a>"
doc.root[1,0] = Element.new "two"
# -> <a><one/><two/><three/></a>
three = doc.elements["a/three"]
doc.root.insert_after three, Element.new "four"
# -> <a><one/><two/><three/><four/></a>
# Dvě zmíněné metody dovolují vložit i příkaz v jazyce XPath:
doc.root.insert_after( "//one", Element.new("one-five") )
# -> <a><one/><one-five/><two/><three/><four/></a>
# Za i před element lze vkládat i jinými způsoby:
four = doc.elements["//four"]
four.previous_sibling = Element.new("three-five")
# -> <a><one/><one-five/><two/><three/><three-five/><four/></a>
```

1.1.1.4 Vypisování stromu

Kam se bude strom vypisovat jednoduše zajistíme metodou "write", která náleží objektu, který chceme vypsat. Jako parametr této metodě předáváme objekt typu <<(String). Takže například soubor, či řetězec.

```
doc.write $stdout
output = ""
doc.write output
```

Pokud požadujeme "pretty-printing"[2] výstup, stačí metodě write() předat o parametr navíc. Tento parametr by měl být větší než -1.

```
doc.write( $stdout, 0 )
```

Pokud dokument neobsahuje XML deklaraci, pak ji můžeme přidat pomocí XMLDecl.new.

```
doc = Document.new
doc.add_element 'foo'
doc.to_s   #-> <foo/>
doc << XMLDecl.new
doc.to_s   #-> <?xml version='1.0'?><foo/>
```

1.1.1.5 Iterace

Metod pro iteraci je opět celá řada. 'Element.each', která iteruje přes všechny elementy. 'Element.elements.each', iteruje jen přes potomky. 'Element.next_element' a 'Element.previous_element' slouží k získání sourozenců. Element.next_sibling a Element.previous_sibling načte sourozence bez ohledu na typ.

1.1.2 Streamované parsování

Streamované[3] parsování vyžaduje zadání třídy Listener (posluchač). Pokud REXML v dokumentu narazí na událost, upozorní třídu Listener, náležející této události. Metody je možné nastavit. Pokud však není nastavení implementováno všechno, musí být implementována metoda "method_missing".

```
list = MyListener.new
source = File.new "mydoc.xml"
REXML::Document.parse_stream source
```

Streamované parsování v REXML je velice podobné SAX, kde jsou události generovány, když na ně parser narazí. Pokud parser narazí na tag, zavolá se metoda tag_start(). Stejně tak, pokud se narazí na konec tagu, zavolá se tag_end(), či pokud narazí na text, zavolá se příslušná metoda text() apod, dokud není dosaženo konce streamu.

1.1.2.1 Bílé znaky

REXML zachovává bílé znaky. Ve spoustě aplikací tomu tak není (bílé znaky jsou zkomprimovány). Toho můžeme rovněž dosáhnout v REXML, avšak toto chování musíme explicitně nastavit.

```
doc = REXML::Document.new( source, { :compress_whitespace => %w{ tag1
tag2 tag3 } }
```

[2] Tento pojem se používá například při zformátování a zvýraznění syntaxe zdrojového (XML) souboru

[3] Kontinuální čtení

Bílé znaky pro "tag1", "tag2", "tag3" budou zkomprimovány. U ostatních tagů budou zachovány. Pomocí :raw, můžeme nastavit komprimování bílých znaků pro všechny tagy (:compress_whitespace => :all).

1.1.2.2 Automatické zpracování některých entit

REXML automaticky zpracovává některé entity. Zpracovávané entity jsou <, >, ", ', &. Pokud REXML nalezne nějaký z těchto znaků v textové nebo atributové hodnotě zpracovávaného dokumentu, automaticky se je snaží přirovnat k entitám v definici XML a případně označit za entitní reference. Naopak při opačném procesu, pokud narazí na entitní reference v definici, vymění je za znakové ekvivalenty a opět porovná. Ostatní entitní reference jsou ponechány nezpracované. Pokud existují některé nezpracované znaky <, >, ", ', & v dokumentu, REXML vygeneruje chybu parsování.

```
bad_source = "<a>Cats & dogs</a>"
good_source = "<a>Cats &amp; &#100;ogs</a>"
doc = REXML::Document.new bad_source
# Vygeneruje chybu parsování
doc = REXML::Document.new good_source
puts doc.root.text
# -> "Cats & &#100;ogs"
doc.root.write $stdout
# -> "<a>Cats &amp; &#100;ogs</a>"
doc.root.attributes["m"] = "x'y\"z"
puts doc.root.attributes["m"]
# -> "x'y\"z"
doc.root.write $stdout
# -> "<a m='x&apos;y&quot;z'>Cats &amp; &#100;ogs</a>"
```

1.1.2.3 Jmenné prostory

Jsou plně podporovány jak v REXML tak i Xpath parserem. Avšak jsou zde dvě vyjímky při použití v Xpath. Pokud není jmenný prostor zadán, je použit defaultní, podle kontextu elementu, který je použit. Toto chování má své omezení, avšak v mnoha případech je užitečné. Pokud je potřeba použít jmenoprostorové mapování, musí být použity metody each, first nebo match s příkazy jazyka XPath.

```
source = "<a xmlns:x='foo' xmlns:y='bar'><x:b id='1'/><y:b id='2'/></a>"
doc = Document.new source
doc.elements["/a/x:b"].attributes["id"]                    # -> '1'
XPath.first(doc, "/a/m:b", {"m"=>"bar"}).attributes["id"]  # -> '2'
doc.elements["//x:b"].prefix                               # -> 'x'
doc.elements["//x:b"].namespace                            # -> 'foo'
XPath.first(doc, "//m:b", {"m"=>"bar"}).prefix             # -> 'y'
```

1.1.2.4 SAX2 Streamované parsování

Originální REXML streamované parsování je velmi "minimální". Avšak na druhou stranu to znamená, že je velmi rychlé.

Pro více komplexní API[4] zahrnuje REXML také streamový parser s podporou SAX2+. Toto API se liší od SAX2 v několika věcech, jako například má více filtrů a mnohonásobné oznamovací mechanismy, ale v jádře je to SAX2.

SAX2 API obsahuje dvě třídy – SAX2Parser a SAX2Listener. Parser je možné použít pěti způsoby v závislosti na potřebě uživatele. Tři způsoby jsou užitečné při filtrování malého množství událostí obsažených v dokumentu, jako například prosté vytištění jmen všech elementů v dokumentu nebo vypsání textových obsahů elementů. Další dva způsoby lze použít pro více komplexní zpracování, kde chceme použít více událostí. První tři pracují s procesy a další dva s posluchači (listeners). Mechanismus založený na listenerech je velmi podobný původnímu REXML streamování s možností vkládání nastavení filtrování a také je rychlejší než mechanismus, který pracuje s procesy.

```
require 'rexml/sax2parser'
parser = REXML::SAX2Parser.new( File.new( 'documentation.xml' ) )
parser.listen( :characters ) {|text| puts text }
parser.parse
```

V tomto příkladě říkáme parseru aby provedl blok (puts text) pro každý textový uzel. Událost je identifikována symbolem :characters. Další možné identifikátory metod jsou např. :element_start, :end_prefix_maping atp. Jména událostí vycházejí ze SAX2Listener API.

Jednotlivé elementy je možné filtrovat zadáním pole tagů naslouchací metodě. Pole tagů může obsahovat také regulární výrazy.

```
parser.listen( :characters, %w{ changelog todo } ) {|text| puts text }
parser.parse
```

V tomto příkladě bude vytištěn pouze textový obsah elementů "changelog" a "todo". Pokud není zadán symbol naslouchací metodě, bude použit defaultní symbol :element_start.

```
parser.listen( %w{ item }) do |uri,localname,qname,attributes|
  puts attributes['version']
end
parser.parse
```

Vytiskne atribut "vision" všech elementů "item".

Poslední dva mechanismy pro parsování používají SAX2Listener API. Moduly StreamListener, SAX2Listener je možné zahrnout do vlastní třídy a vytvořit tak adapter. Pro použití naslouchacího modelu je potřeba vytvořit třídu, která implementuje alespoň nějaké SAX2Listener metody nebo všechny, pokud nezahrneme SAX2Listener model. Dále jej vložíme do parseru a vložíme blok

[4] Aplication programing interface – rozhraní pro programování aplikací

příkazů. Metoda bude volána, když nastane odpovídající událost. "Posluchači" nepoužívají událostní symboly, ale umí filtrovat podle jmen elementů.

```
listener1 = MySAX2Listener.new
listener2 = MySAX2Listener.new
parser.listen( listener1 )
parser.listen( %{ changelog, todo, credits }, listener2 )
parser.parse
```

V předchozím příkladě "listener1" bude zachytávat všechny události, které nastanou a "listener2" bude zachytávat pouze události elementů "changelog", "todo" a "credits". Jak vidíme, parser může mít více posluchačů. Stejně tak můžeme vložit více bloků a posluchače i bloky spolu míchat.

2 Ruby On Rails a AJAXové knihovny

V Ruby on Rails lze vytvářet Ajaxové požadavky pomocí helperů, které vygenerují Javascriptový zápis za nás. Stačí zadat několik málo parametrů (mezi kterými nesmí chybět zpracovávací akce v controlleru) a Ajax.request je na světě. Akce odpoví například zavoláním šablony RJS[5], která prohlížeči zašle Ajax.response.

V této kapitole se dále budu zabývat knihovnami Prototype a script.aculo.us, které ve své aplikaci využívám pro vytváření dynamiky. Budu se snažit nastínit jejich možnosti a občas uvedu nějaký příklad pro snazší pochopení.

2.1 Prototype a script.aculo.us

Vytvořena Thomasem Fuchsem. Poprvé byla uvolněna veřejnosti v červnu roku 2005.

Rozšiřuje "Prototype Javascript Framework" přidáním dynamických visuálních efektů, kontroly nad elementy uživatelského rozhraní pomocí objektového modelu "Document" a utilitami. Pomalu se stává standardem pro tvorbu webových aplikací.

Je úzce spojena s Ruby On Rails, avšak používá se i ve spolupráci s jinými webovými frameworky (např. PHP, .NET, J2EE, Python, Delphi) a skriptovacími jazyky.

Poskytuje prvky orientované na uživatelské rozhraní jako například drag & drop, autokompletace, řazení elementů pomocí myši, in-place editor a v neposlední řadě, jak už bylo řečeno, nejrůznější vizuální efekty jako například zvýraznění, skrytí, znovuobjevení atp.

Prototype ve spojení se script.aculo.us v poslední době získávají na oblibě. V září roku 2006 podle průzkumu server Ajaxian.com, patří mezi dva nejvíce populární JavaScriptové frameworky. Od vzniku online referenční dokumentace, v lednu 2007, či vydání krátké knihy " Prototype quick reference ", napsané Scottem Raymondem a Sergiem Pereirou v listopadu 2006, se předpokládá ještě větší nárůst

2.1.1 Prototype JavaScript Framework

I když je Prototype relativně malá knihovna, je velice obsáhlá.

Byla navržena k vylepšení JavaScriptového prostředí prohlížečů. Rozšiřuje DOM[6] elementy o užitečné metody. Podporuje OOP (zahrnující dědičnost), události a mocné prvky AJAXu.

---

[5] Ruby JavaScript

[6] Objektový model dokumentu, objektově orientovaná reprezentace XML

Není to kompletním frameworkem pro tvorbu aplikací. Neposkytuje widgety[7] ani kompletní sadu běžných algoritmů. Stojí někde na rozhraní mezi "nízkoúrovňovým" manuálním programováním a plnohodnotnými frameworky, které oplývají nespočitatelným množstvím objektů.

Ve skutečnosti je sám využíván robustními frameworky, které na něm staví.
Ačkoliv je inspirován programovacím jazykem Ruby, není blíže spjat s nějakou serverovou technologií. Každopádně Jeho použití ve spojení s Ruby On Rails je velim jednoduché.

Je distribuován jako jednotlivý soubor "prototype.js" o velikosti asi 120 KB (před zkomprimováním). Navzdory malé velikosti disponuje rozsáhlou sadou nástrojů, které se snadno a intuitivně používají.

Oficiální webové stránky (http://prototypejs.org) jsou autoritativním zdrojem poslední zveřejněné verze knihovny Prototype a také poskytují aktuální verzi API dokumentace s mnoha praktickými ukázkami.

Rychle se vyvíjí kupředu, proto je možné, že informace zde, za několik málo dní nebudou aktuální.

2.1.2 Script.aculo.us

Je spíše vizuálně orientovaná knihovna úzce spolupracující s Prototype, které byla původně součástí. O to se zasloužil už jmenovaný Thomas Fuchs, který je se stále aktivně podílí na vývoji.

Obecně použitelné vizuální prvky, animované efekty (například vyblednutí, hýbání, změna velikosti) a drag & drop, jsou významné téměř ve všech uživatelských rozhraních, pokud mají být stylové a mají připomínat prostředí pracovní plochy operačního systému.

2.1.2.1 Použití script.aculo.us

1. Stažení

   Nejnovější verzi nalezneme na stránce http://script.aculo.us.

2. Instalace

   Uložíme soubor prototype.js, scriptaculous.js, builder.js, effects.js, dragdrop.js, slider.js a controls.js do adresáře webové aplikace (například /javascripts)

3. Odkaz na script.aculo.us

   V hlavičce dokumentudoplníme:

```
<script src="javascripts/prototype.js"
type="text/javascript"></script>
<script src="javascripts/scriptaculous.js"
type="text/javascript"></script>
```

   Scriptaculous.js si sám načte všechny potřebné javascriptové soubory (například k vytváření efektů, drag & drop, slidebary atp.).

[7] Komponenta grafického uživatelského rozhraní

Jíná možnost je, že se omezíme pouze na vybrané:

```
<script src="scriptaculous.js?load=effects,
dragdrop"type="text/javascript"></script>
```

4. Použití

Pro volání funkcí se používá HTML tagů. Nejlepší způsob jak je definovat je:

```
<script type="text/javascript" language="javascript">
   // <![CDATA[
     Effect.Appear('element_id');
   // ]]>
</script>
```

Tento způsob dovoluje použití značek jako např. "<", ">".

5. Dokumentace

http://wiki.script.aculo.us/scriptaculous/

Ještě jednodušší je použití v Ruby On Rails, kde se pracuje s AJAX helpery (pomocníky). Potřebné javascripty jsou v Ruby On Rails implicitně obsaženy, proto stačí do šablony webové aplikace vložit jeden řádek kódu (helper). Ten automaticky vloží odkazy na javascripty, obsažené v aplikaci v adresáři /javascripts.

```
<%= javascript_include_tag  :defaults %>
```

2.1.2.2 Animované efekty

Stěžejní modul, poskytuje více než 25 vizuálních efektů, 7 přechodných modů (které se mění s časem od začátku po konec efektu). Efekty se také mohou řadit do fronty (což umožňuje vytvořit pokročilejší efekty – sekvence efektů). Pokud ani to vše nestačí, je možné vytvořit efekt vlastní. Animované efekty jsou důvodem, proč spoustu lidí vůbec script.aculo.us používá. Modul je rozdělen do dvou částí.

1. Jádrové efekty
   - Effect.Highlight
   - Effect.Move
   - Effect.Opacity
   - Effect.Parallel
   - Effect.Scale
   - Effect.Tween
   - Effect.Morph

2. Kombinované efekty

Kombinované efekty obyčejně závisí na paralelním, synchronizovaném běhu ostatních efektů. Protože takové chování je možné, vytváření vlastních efektů se stává velice jednoduché.

Abychom mohli vizuální efekty používat, je potřeba načíst modul s efekty. "Úsporné"načtení by vypadalo takto:

```
<script type="text/javascript" src="prototype.js" ></script>
<script type="text/javascript" src="effects.js" ></script>
```

Protože script.aculo.us závisí na Prototype, musíme vždy načíst i prototype.js

Motivace – proč používat efekty?

- Potřebujeme upoutat uživatelovu pozornost,
- zobrazování a odstraňování prvků chceme udělat hezčím způsobem (postupné odkrývání a skrývání, přibližování, oddalování apod. vypadá lépe, než náhlé objevení a skrytí),
- chceme upozornit uživatele na přemístění prvku postupným pohybem z původní pozice na pozici novou.

Efekty toto vše jednoduše a přímo umožňují. Nabízejí široké spektrum nastavení. Každý specifický efekt potom většinou poskytuje další možnosti nastavení (například u postupného zvýraznění, lze nastavit, kterou barvou se začne a kterou skončí).

Používání efektů může dát naší aplikaci nový lesk a přispívat k ergonomii AJAXem vylepšených, webových stránek, kde se v krátkém časovém intervalu mění obsah, kterého by si uživatel jinak nemusel všimnout.

Vytvoření efektu

Správné použití s operátorem "new" (zde se často chybuje). Pokud bysme totiž chteli použít efekt jako funkci v hlavním kódu, mohlo by dojít k vícenásobným efektům místo správně, paralelně běžících.

Existují dvě možné syntaxe:

```
new Effect.EffectName(element [ , requiredArgs ] [ , options ] )
-> vrací instanci třídy Effect
extElement.visualEffect('EffectName' [ , requiredArgs ] [ , options ] )
-> vytvoří efekt pro konkrétní element extElement
```

Obě jsou technicky ekvivalentní. Je čistě na programátorovi, kterou si vybere.

Konkrétně:

```
new Effect.Scale('title',200,{scaleY:false,scaleContent:false});
$('title').visualEffect('Scale',200,{scaleY:false,scaleContent:false});
```

Zde je vidět, jak spolu obě syntaxe souvisí.

Jádrové efekty

Effect.Highlight (zvýraznění)

Mezi jeho specifické nastavení patří "startcolor", "endcolor" a "revertcolor". Poslední dva parametry jsou defaultně nastaveny na barvu pozadí. Pokud z nějaké příčiny tuto barvu nelze přiřadit, je nastavena barva bílá. Začínající barva (startcolor) je defaultně nastavena na hodnotu #ffff99, která odpovídá světle žluté. Zvýraznění bude začínat u začínající barvy, pozvolna se změní na barvu konečnou a poté se resetuje na barvu návratovou (revertcolor).

Obrázek na pozadí (pokud je nějaký nastavený), bude defaultně během průběhu efektu odstraněn. Pokud ho chceme zachovat, nastavíme hodnotu "kepBackgroundImage" na "true".

Protože je toto často používaný efekt, byla zavedena zkrácená forma vytváření:

```
extElement.highlight([ options ] )
```

Takže například takto:

```
$('resultMessage' ).highlight();
```

Effect.Move (pohyb)

Přenese element na novou pozici (vyjádřeno buď absolutními, nebo relativními souřadnicemi). Přenášet lze jak v horizontálním, tak vertikálním směru. Specifické parametry jsou "x" a "y", oba defaultně nastaveny na původní umístění elementu (absolutně nebo relativně).

Effect.Opacity (průhlednost)

Pozvolna mění průhlednost elementu až na nastavenou hodnotu. Závisí na metodách z Prototype, které se průhledností zabývají. Efekt začíná na aktuální hodnotě průhlednosti elementu a končí na definované, či defaultní, která je nastavena na 1.0.

Effect.Parallel (paralelní)

Mechanismus, který umožňuje běh efektů v synchronizovaném paralelním módu. Pokud nepotřebujeme synchronizaci (efekt začíná a končí ve stejném čase), jednoduše spustíme efekty nezávisle.

Pokud chceme spustit efekty paralelně, předáme konstruktoru pole efektů, které chceme synchronizovat. Tyto efekty musí být vytvořeny s parametrem "sync", nastaveným na "true".

```
new Effect.Parallel([
  new Effect.Opacity('notice' , { sync: true }),
  new Effect.Scale('notice' , 100, { sync: true, scaleFrom: 50 })
], { duration: 2 });
```

Nutno poznamenat, že zadané efekty nemusí nutně náležet stejnému elementu. Nicméně je možné nastavit pouze jednu hodnotu trvání (duration). To znamná, že synchronizované efekty budou probíhat společně.

Effect.Scale (změna rozměru)

Postupně zvětší, nebo zmenší element, avšak pouze v jednom nebo druhém směru (horizontálně nebo vertikálně).

Lze nastavit "scaleX", "scaleY" defaultně nastavené na "true". Určuje, zda změna rozměru v daném směru nastane.

Parametr "scaleContent" defaultně nastaven na "true". Aktivuje, změnu rozměru samotného obsahu elementu.

Parametr "scaleFromCenter" je defaultně nastaven na "false". Velikost se nebude měnit od levého horního rohu, ale od středu (tzn. že se bude zvětšovat vně a zmenšovat dovnitř).

Poslední "scaleFrom" umožňuje měnit velikost elementu, již zvětšeného respektive zmenšeného (na určitou procentuelní hodnotu). Defaultně nastaveno na "100".

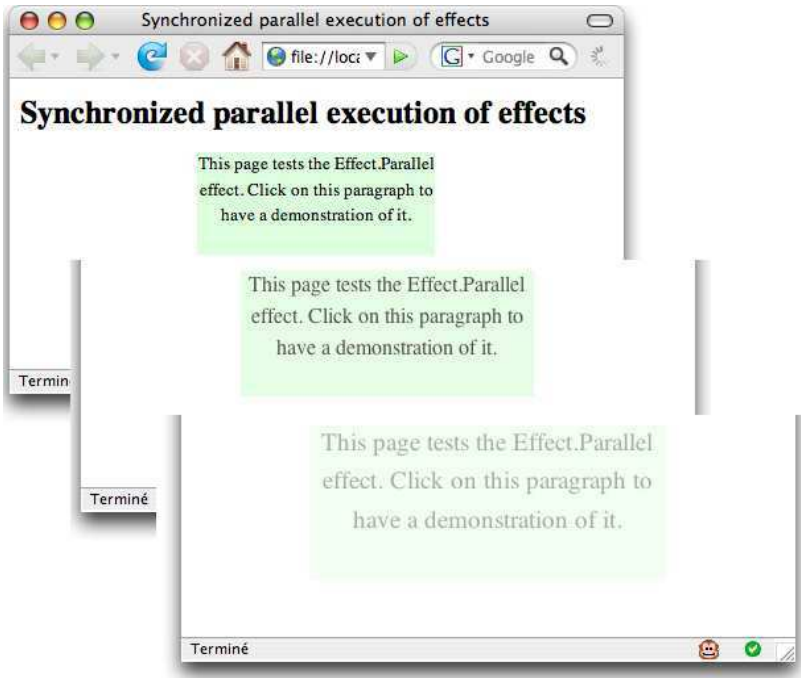
Volání může vypadat například takto:

```
new Effect.Scale('greeter' , 200, { scaleFromCenter: true });
```

Příklad kombinující Effect.Opacity a Effect.Scale pomocí Effect.Parallel

```
new Effect.Parallel([
  new Effect.Opacity('demo' , { sync: true, from: 1, to: 0.33 }),
  new Effect.Scale('demo' , 150, { sync: true,
  scaleFromCenter: true })
], { duration: 2 });
```

Výsledek popisuje následující obrázek



Obrázek 1- Převzato z knihy Pragmatic Prototype and Script.aculo.us

Effect.Tween

Je obsažen od verze 1.8. Tento efekt zapouzdřuje pokročilejší série nastavení. Zapíšou se nějaké vlastnosti a poté se zavolá nějaká metoda nebo dokonce například námi nadefinovaná funkce "odezvy" (callback). Tento mechanismus bude více používaný ve verzi 2.0.

Syntaxe:

```
new Effect.Tween(obj, from, to[ , options ] , propertyOrMethodName)
new Effect.Tween(obj, from, to[ , options ] , callback)
```

V opakovaných intervalech (pro každý "frame" efektu), použije pokročilejší hodnoty (někde mezi "from" a "to") v sekvenci, pro zapsání vlastnosti, volání metody nebo volání naší "callback" funkce.

Pro pochopení bude lepší uvést konkrétní příklad:

```
new Effect.Tween('edtGain' , 1, 20, { fps: 10, duration: 2 }, 'value' );
```

U elementu s id "edtGain" (což je políčko formuláře) se bude postupně měnit obsah hodnotami (value) od 1 do 20. Efekt bude trvat 2 sekundy a za 1 sekundu se políčko překreslí 10x (10 různých hodnot.

```
var notice = $('notice' ), color = notice.getStyle('backgroundColor' );
var blue = parseInt((color.match(/\d+/) || '' )[2], 10) || 0;
var rgb;
new Effect.Tween(notice, 0, 255, function(rg) {
  rgb = [rg, rg, blue].inject('#' , function(acc, comp) {
    return acc + comp.toColorPart();
  });
  notice.setStyle({ backgroundColor: rgb });
});
```

Poslední uvedený příklad postupně žlutě natónuje pozadí elementu (z žádného na plné).

Jak je vidět, efekt "Tween" je velmi obecný a přizpůsobivý.

Effect.Morph

Zavedena ve verzi 1.7. Vezme původní nastavení CSS elementu a plynule přetransformuje na nové (tzn. plynule se změní vzhled elementu).

Cílovou stylovou definici můžeme vyjádřit třemi způsoby:

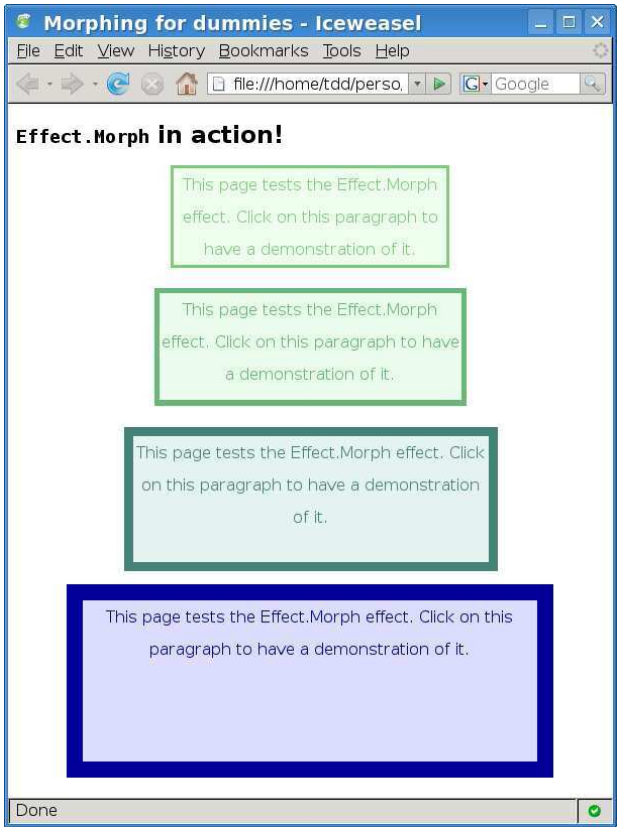
- Jméno třídy CSS. Styl elementu se bude měnit na styl této třídy.
- Stylová specifikace na jednom řádku (styl = hodnoty atributů)
- Množina vlastností CSS.

Prvním argumentem metody je styl, druhým pak možné nastavení:

```
extElement.morph('erroneous' ); // jmeno CSS třídy
extElement.morph(
  'color: maroon; background: #fdd; border-color: maroon;' );
extElement.morph({
```

```
    width: '50ex' , height: '10em' ,
    backgroundColor: '#ddf' , color: '#009' ,
    borderWidth: '1em' , borderColor: '#009' ,
    opacity: '1'
  }, { duration: 2 });
```

Chování posledního volání můžeme vidět na následujícím obrázku:



Obrázek 2 - Převzato z knihy Pragmatic Prototype and Script.aculo.us

Tato metoda funguje nejlépe pro parametr "color" (barva) a pro parametry dány rozměrem, avšak můžeme ji použít téměř na všechno. V případě barev musíme používat rgb(r, g, b), nelze používat jejich jména (white, black atp.). Vlastnosti, které nemohou být plynule modifikovány (například font-style, text-decoration) budou nastaveny až na konci efektu. Správná funkčnost také závisí na prohlížeči. Proto je nutné efekt vždy otestovat.

Kombinované efekty

Jak již bylo řečeno, jsou založeny na jádrových efektech.

Na rozdíl od jádrových efektů, operátor "new" je zde zcela zbytečný. Funkce pro kombinované efekty nejsou konstruktory. Na druhou stranu, lze využít zkrácené syntaxe, díky které s nimi můžeme pracovat jako s metodami elementů.

Dvě možné syntaxe:

```
Effect.CombinedEffectName(element [ , options ] )
extElement.combinedEffectName([ options ])
```

Ke kombinovaným efektům již dále nebudu uvádět ukázky použití, protože není účelné, aby se v této práci vyskytovalo více zdrojového kódu, než samotného textu.

Effect.Fade/Effect.Appear

Objevení nebo skrytí elementu. Skrytí elementu začíná na defaultní hodnotě (na původní hodnotě průhlednosti – "opacity"). Když efekt skončí, element je skrytý a jeho původní hodnota průhlednosti je obnovena. Efekt objevení nejdříve zkontroluje, zda je element zobrazený a poté začíná na hodnotě průhlednosti (pokud byl zobrazený).

Tyto efekty zřetelně závisí na jádrových.

Effect.BlindUp/Effect.BlindDown

Tyto efekty srolují nebo rozrolují plochu, na které je element zobrazen. Obsah elementu je během efektu ponechán beze změny (nehýbe se).

Efekty nebudou fungovat, pokud použijeme BlindDown (srolování) a element již srolován je. Podobně pro BlindUp.

Effect.SlideUp/SlideDown

Velice podobný předchozím, avšak na rozdíl od nich roluje obsah. Efekt vyžaduje, aby byl obsah v elementu . Není možné vložit obsah do elementu ovlivněného jiným efektem (například BlindDown).

Effect.DropOut

Element se začne pohybovat směrem dolů a zároveň mizí (dokud nezmizí).

Effect.Grow/Effect.Shrink

Efekt expanze respektive smrsknutí. Element se zvětšuje z nulové velikosti nebo se do nulové velikosti smrskne. Tyto efekty závisí na jádrových efektech "Scale" a "Move".

Zde nalezneme specifické nastavení "direction", které určuje, jakým způsobem bude probíhat expanze či kontrakce. Defaultně je nastavené na "center" (z/do prostředka), ale může to být klidně jeden z rohů: "top-left" (horní-levý), "top-right" (horní-pravý), "bottom-right" (dolní-pravý) a "bottom-left" (dolní-levý).

Effect.Puff

"Pomocí magické hůlky necháme zmizet element v mraku kouře.". Element expanduje na dvojnásobnou velikost a zároveň mizí (dokud nezmizí).

Effect.SwitchOff

"Efekt vypnutí televize". Element nejprve lehce zprůsvitní a poté se smrskne směrem do středu, až zmizí.

Effect.Squish

Smrsknutí do směrem do určitého, referenčního bodu. Pro nejvíce elementů je to levý-horní roh.

Effect.Fold

Dvoufázové smrsknutí. Nejprve se zmenší vertikálně, dokud není výška 5 pixelů a poté horizontálně. Obsah (ani jeho pozice) se během efektů nemění.

Effect.Pulsate

"Je naživu!". Element několikrát plynule zmizí a poté se opět plynule objeví. Počet opakování je defaultně nastaven na 5x za 2 sekundy.

Effect.Shake

Zatřese elementem doprava a doleva 3x za půl sekundy.

Je to například dobrý způsob jak zvýraznit formulářové okénko, které nebylo zadáno.

Tento efekt je obsažen od verze 1.8. Nyní umožňuje nastavit délku (časově) i vzdálenost.

2.1.2.3 Drag & Drop

Modul 'dragdrop' umožňuje pohybovat s elementy a pokládat je do zóny k tomu určené. Umožňuje také vytvořit seznamy elementů, které se dají pomocí drag & drop (táhni a pusť) řadit.

2.1.2.4 Autokompletace (automatické doplňování)

Tento modul poskytuje nástroj pro automatické doplňování textových políček. Doplňovaný text pochází ze serveru (pomocí AJAXu) nebo z přednačtených možností.

2.1.2.5 In-place editor

Umožňuje upravovat text nebo vybrané položky po pouhém kliknutí, kdy se objeví speciální rozhraní a upozorní server, pokud je editace dokončena, aby mohl upravená data uložit.

Co je přesně in-place editor?

Vezme elementy, které nelze upravovat, jako například <p>, <h1>, , a dovolí uživatelovi upravit jejich obsah pouhým kliknutím na něj. To způsobí, že se ze statického elementu stane editační oblast (buď jedna, nebo více řádek) a zároveň se objeví tlačítka potvrzení a zavření (submit, cancel), aby mohl uživatel buď uložit, nebo vrátit změny. Poté se změny na serveru synchronizují a element se stane opět needitovatelný.

Úprava pomocí in-place editoru je jeden z charakteristických znaků webu verze 2.0. Který přispívá k větší interakci webových stránek.

Script.aculo.us obsahuje dvě třídy na úpravu pomocí in-place. Ajax.InPlaceEditor a Ajax.InPlaceCollectionEditor.

První je velice známá a hojně využívaná. Poskytuje textové pole, do kterého se dá svobodně psát a tak upravovat obsah.

Druhá limituje uživatele seznamem hodnot, ze kterých se dá vybírat.

Jak jejich jména prozrazují, obě používají Ajax k synchronizaci nového obsahu se serverem. Obě disponují bohatými možnostmi nastavení, co se týká vzhledu, či charakteru.

In-place editory krok za krokem

1. Vytvoříme editovatelný in-place element. Vše zařídí volání jedné metody.
2. Potom nastavíme prvky, které označí element za editovatelný, pokud na něj najedeme myší. Defaultně:
    - Při najetí se element zvýrazní (highlight)
    - Zobrazí se titulek s defaultní hodnotou "Click to edit" (Klikni pro úpravu)
    - Při najetí myší mimo element se vše vrátí do původního stavu
3. Kliknutí na element spustí upravovací mód. Vytvoří se plovoucí formulář (například jeden řádek, nabídka možných položek, potvrzovací a zavírací tlačítko apod.). Ačkoliv se to málo používá, editovací mód lze upravit. Jako možné výběrové položky lze načíst (na straně serveru) XHTML, který se vhodně zobrazí.
4. Pro ovládání editovacího módu lze použít klávesy. Stisknutím klávesy enter potvrdíme editaci, klávesou Esc zavřeme editaci bez uložení.
5. Pokud editovací mód ukončíme, obnoví se původní obsah. Pokud editaci potvrdíme, editovací pole zmizí a objeví se zpráva, která uživatelovi oznamuje, že se ukládá modifikace. Poté se objeví uložený obsah.
6. Element se na chvíli zvýrazní, aby oznámil změnu.

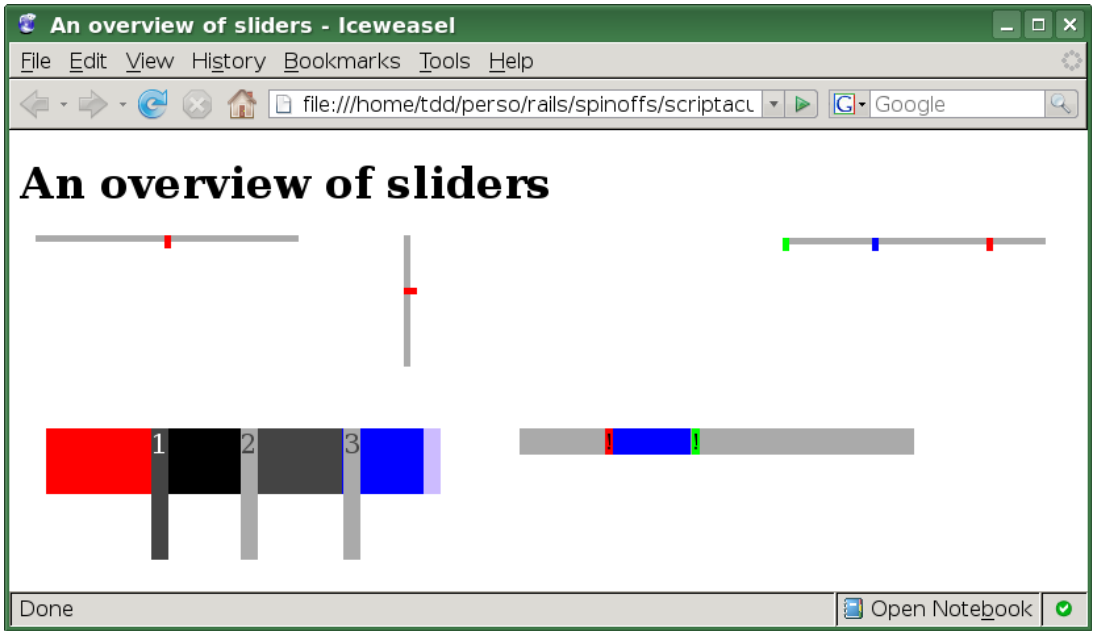
Jednoduchý příklad in-place editoru

```
...
<body>
  <p id="freeZone" ><%= editableHTML %></p>
</body>
...
document.observe('dom:loaded' , function() {
  new Ajax.InPlaceEditor('freeZone' , '/update' );
});
```

2.1.2.6 Posuvníky

Posuvník je reprezentován posuvným prvkem ("šoupátko"), a dráhou, po které se tento prvek posunuje. Poloha tohoto prvku určuje numerickou hodnotu. Script.aculo.us dovoluje vytvářet posuvníky, které lze velice dobře nastavit (lze definovat hranice, omezení na možné hodnoty, které odpovídají specifickým hodnotám, použít mnohonásobné posuvné prvky atd.). Na druhou stranu je možné přizpůsobovat posuvníky pouze do určité míry, která je dána komplexností elementů uživatelského rozhraní, vytvářeného v XHTML.

Na obrázku níže je vidět, jak takové posuvníky můžou vypadat. V kódu jsou reprezentovány elementy a , které si nadefinujeme podle svého, takže vzhled je zcela na nás. Posuvník lze jednoduše zasadit do jakéhokoliv designu stránek, který zrovna používáme.



Obrázek 3 - Převzato z knihy Pragmatic Prototype and Script.aculo.us

Vytvoření jednoduchého posuvníku

V dokumentu vytvoříme dva nové elementy – jeden pro "šoupátko" a druhý pro dráhu.

```
new Control.Slider(handle, track [ , options ] )
```

Element pro dráhu je obvykle a posuvný prvek je nebo . Oba můžou být metodě zadány jak jejich "id", tak jejich DOM referencí. Kompletní základní posuvník by mohl vypadat takto:

```
<div id="track1" class="track" style="width: 20em;" >
  <div id="handle1" class="handle" style="width: 0.5em;" ></div>
</div>
new Control.Slider('handle1' , 'track1' );
```

Základní nastavení

axis

Toto nastavení nám dovolí definovat, zda bude posuvník položen horizontálně (defaultně) nebo vertikálně. Toto nastavení omezí pohyb pohyblivého prvku pouze na jednu osu.

sliderValue

Tohle určí inicializační pozici posuvného prvku (defaultně je nastavena na začátek rozsahu). Protože je defaultní rozsah nastaven na [0;1], začátek je v nule. Pokud bychom chtěli dát "šoupátko" do půlky, bude rovna 0,5.

Akce

Obě akce mají dva argumenty – hodnotu posuvníku (vyjádřenou jako hodnotu z rozsahu, takže defaultně mezi 0 a 1) a objekt posuvníku samotného (instance Control.Slider).

onSlide

Spustí se, pokud se pohne s posuvným prvkem.

onChange

Spustí se pokud se tlačítko myši uvolní (konec posuvu "šoupátka")

Omezení rozsahu, povolené hodnoty

Pravděpodobně nejčastěji potřebujeme pracovat s jiným rozsahem hodnot, který odpovídá nějakým záležitostem v kontextu s našimi stránkami. Ať už je příčina jakákoliv, rozsah hodnot lze přizpůsobit našim potřebám.

range

Výjádřený jako ObjectRange objekt

Posuvník s několika posuvnými prvky

Posuvníky nabízí zajímavou možnost – použít více posuvných prvků na jedné dráze. Což uživatelovi umožňuje nastavit několik hodnot na jednom posuvníku.

2.1.2.7 DOM builder (stavitel objektového modelu dokumentu)

Vývojový nástroj, který podstatně ulehčuje vytváření DOM. W3C rozhraní, není vždy nejvhodnější. Často se setkáme s tím, že je nekorektní, nemotorné a kupodivu ani podpora prohlížeči není bezchybná. Narážíme zde na problém se <select> tagy, jejich obsahy, s elementy závislými na tabulkách, s atributy name= u některých tagů, s atributy, jejichž jméno je "JavaScript" apod.

2.1.2.8 Zvuk

Verze 1.7.1 zavedla systém, který umožňuje jednoduše přehrát zvuk, pouštět několik smyček za sebou, paralelně atd. Toto nevyžaduje Flash a jiné prvky závislé na zvukových možnostech prohlížeče a jeho pluginu.
Efektivní použití krátkých zvukových efektů, které mohou přilákat uživatelovu pozornost, i když stránka není viditelná (schovaná za jiným oknem, přemístěná na straně, minimalizovaná, či jenom neaktivní)

3 Vytvoření aplikace

Cílem implementace bylo vytvořit část rozsáhlé aplikace "Generátor vědeckých webových portálů".

Tento program má za úkol zautomatizovat vyhledávání informací, konkrétně informací o vědcích.

Je složen z těchto částí:

- Řídící systém s webovým rozhraním (Horák Jan)
- Systém pro automatické rozpoznávání vyjmenovaných entit (Križanský Jan)
- Úprava výsledků automatického rozpoznávání sémantických informací ze značkovaného textu (Růžička Štěpán)
- Vyhledání osobních stránek podle zadaných entit (Kundrát Miloš, Bc.)
- Indexace PDF (???)
- Portál, který bude publikovat výsledky automatického generátoru portálů (???)

Vzhledem k tomu, že celý generátor vědeckých webových portálů se skládá z několika částí, je nutné tyto části propojit a řídit. K tomu slouží Řídící systém generátoru. Má na starost spouštění jednotlivých částí, předávání vstupů, přebírání výstupů atd. Protože některé úkony musí vykonávat uživatel sám, má řídící systém webové rozhraní. Pomocí tohoto rozhraní může jednotlivým úlohám spravovat priority, sledovat jejich průběh atd.

Dále přichází na řadu systém, který dostane pokyn vyhledat a shromáždit informace (například podle jména vyhledat rok narození, úmrtí, vydané články apod.). Ke své práci využívá API existujících vyhledávačů (google, yahoo apod.). Protože tyto operace vyžadují ke své práci do jisté míry rozhodovací smysl a protože tento smysl programy nevlastní, výsledek nemusí být vždycky optimální a potřebují zásah člověka. Výstupem je soubor XML.

Zde přichází na řadu část aplikace, která "obyčejnému" uživateli umožní XML soubor ve stylu user friendly[8] upravit.

Po úpravě souboru může začít "opravdové" shromažďování a stahování informací, které se týkají dané osobnosti a které se následně publikují veřejnosti. To je úkolem zbývajících částí.

3.1 Formulace cíle

Mým úkolem bylo vytvořit třetí bod osnovy – "Úprava výsledků automatického rozpoznávání sémantických informací ze značkovaného textu".

[8] Uživatelsky přívětivé

Požadavkem bylo, aby aplikace byla přehledná, intuitivní a aby uživatele do jisté míry omezovala v úpravách značkovaného textu (nemůže zadat informace, které generátor neumí zpracovat).

3.2 Současný stav

Do jisté míry se mi podařilo splnit všechny výše uvedené požadavky, ačkoliv vkládání nových entit (správcem generátoru či programátorem – tyto entity může uživatel vložit do stromu značkovaného textu) do programu se neděje automaticky, jak jsem původně chtěl udělat (v případě XML se nebere v úvahu DTD ani XSD). I přesto považuji svůj výtvor za dobrý.

3.3 Analýza problému a princip jeho řešení

Protože je projekt komplexnějšího rázu, při kterém bude nutné zkombinovat několik technologií dohromady a zajistit, aby spolu vzájemně komunikovali, podívám se na něj v této kapitole podrobněji.

Dále se v této kapitole zaměřím na různé možnosti, které se pro řešení daného problému nabízí.

3.3.1 Zadání

Cílem tohoto projektu je vytvoření programu za pomoci libovolných nástrojů, který uživateli umožní upravovat výsledky automatického rozpoznávání vyjmenovaných entit. Uvažujte, že program musí umět ovládat běžný uživatel, neznalý této problematiky. Výsledkem bude upravený vstup, formát musí být zachován.

3.3.2 Nejasnosti

Je potřeba se zamyslet, jakým způsobem se budou předávat rozpoznané entity. Určitě by to měla být struktura dobře generovatelná a zpětně dobře zpracovatelná. Také bychom měli vzít v úvahu velký objem předávaných dat, a zda neexistuje již něco vhodné pro naše účely, co by nám práci usnadnilo.

Dále si musíme ujasnit, jestli je vhodné pracovat přímo s touto strukturou, či využít nějaký mezičlánek, který by uchoval data během úprav.

Měli bychom si zvolit nástroj, kterým uživatelovi zprostředkujeme strukturu pro úpravy. Tento nástroj by měl nějakým způsobem umožnit s danou strukturou komunikovat.

3.3.3 Možná řešení

Jako první řešení mě napadlo použití nějakého značkovacího jazyka z rodiny SGML. V dnešní době je již standardem XML, což je vlastně zjednodušený SGML. Tento jazyk odpovídá všem našim požadavkům a má podporu u většiny pokročilejších nástrojů.

Dále by bylo možné pracovat přímo s databází. Zde se fantazii meze nekladou. Je jich velké množství a podpora je také slušná. Mezi možné můžeme zařadit například SQL, Gupta, MySQL, SQLite apod.

Jako nástroj pro reprezentaci dat bychom mohli zvolit klidně grafické API nějakého programovacího jazyka. Například Java, nebo C++. Ale protože se jedná o internetovou aplikaci, měli bychom si zvolit spíše nějaký framework, který generuje HTML (XHTML). Mezi možné frameworky můžeme zařadit například PHP nebo Ruby on Rails, ale opět je jich spousta a záleží pouze na podpoře serveru, na kterém aplikace poběží.

3.4 Návrh řešení

Po analýze problému jsem se rozhodl využít datový formát XML pro výměnu informací. Ruby on Rails pro reprezentaci XML ve spojení s AJAXem a jako mezičlánek pro uložení dat, databáze MySQL.

Ruby on Rails podporuje zpracování XML (knihovna REXML), proto by načítání a ukládání dat něměl být problém. Jak jsme požadovali, Ruby on Rails generuje kód HTML a podporuje i práci s databází.

Pomocí nástrojů z knihovny REXML nejprve vytvořím DOM, který poté budu procházet a jednotlivé elementy budu ukládat do databáze. Strukturu dokumentu zohledním v databázi tak, že elementy provážu pomocí jejich ID. Elementy budu ukládat do jedné tabulky. Tím zajistím, že jejich počet i zanoření může být libovolné. Do další tabulky budu ukládat jejich obsah, případně atributy, pokud je budu uvažovat. XML budu zobrazovat pomocí HTML. Ruby on Rails sice nabízí nástroje, které umí generovat přímo XML, avšak takový dokument je statický a nelze jej interaktivně upravovat.

Do HTML lze snadno zakomponovat AJAX. Tím docílím toho, že XML v HTML bude možné upravovat například pomocí myši a za pomoci formulářů.

Pro uložení dokumentu vytvořím vlastní, nejspíš rekurzivní funkci. Ani zde nepůjde použít RXML šablony, kterou RoR nabízí.

3.5 Popis řešení

V této kapitole popíšu, jak jsem postupoval při řešení daného problému.

3.5.1 Použité technologie

3.5.1.1 Ruby on Rails

Protože moje část vyžaduje interakci s uživatelem, a protože veškeré dění probíhá na webu, bylo žádoucí si pro realizaci aplikace zvolit nějaký webový framework[9]. A protože jsem se v poslední době stále více a více dozvídal o tomto dynamicky se rozvíjejícím, některými opěvovaným, jinými zavrhovaným, i přesto čím dál více populárním frameworku, rozhodl jsem se, udělat si vlastní názor a zkusit svou aplikaci udělat právě v něm.

3.5.1.2 XML

Výstupem systému pro automatické rozpoznání je XML[10]. Je to osvědčený značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Mimo jiné je určen především pro výměnu dat mezi aplikacemi.

3.5.1.3 MySQL

Jak jsem se později dozvěděl, použití databáze nebylo nutné (v Ruby on Rails lze s XML dokumentem pracovat jako s DOM[11]). I když řešení pomocí databáze spoustu věcí značně zkomplikovalo, ukázalo se, že oproti DOM mělo i své výhody (viz. níže).

3.5.1.4 Prototype a script.aculo.us

Pro úpravu XML jsem se rozhodl použít dynamické efekty (drag & drop, in-place editor apod.). A co jiného by bylo vhodnější použít pro tyto účely, než technologii, která vznikla vedle výše zmíněného RoR[12].

3.5.2 Postup

1. Rozparsovat XML a vhodným způsobem uložit do databáze.

---

[9] Framework je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovnu API, návrhové vzory nebo doporučené postupy při vývoji.

[10] eXtensible Markup Language, česky rozšiřitelný značkovací jazyk

[11] Document Object Model - objektově orientovaná reprezentace XML nebo HTML dokumentu

[12] Ruby on Rails

2. HTML reprezentace rozparsovaného XML v databázi.
3. Použití Ajaxu pro úpravu XML, podle změn obsahu na webové stránce se bude měnit obsah databáze.
4. Export zpět do XML, vymazání databáze (návrat do původního stavu).

3.5.3 Vlastní implementace

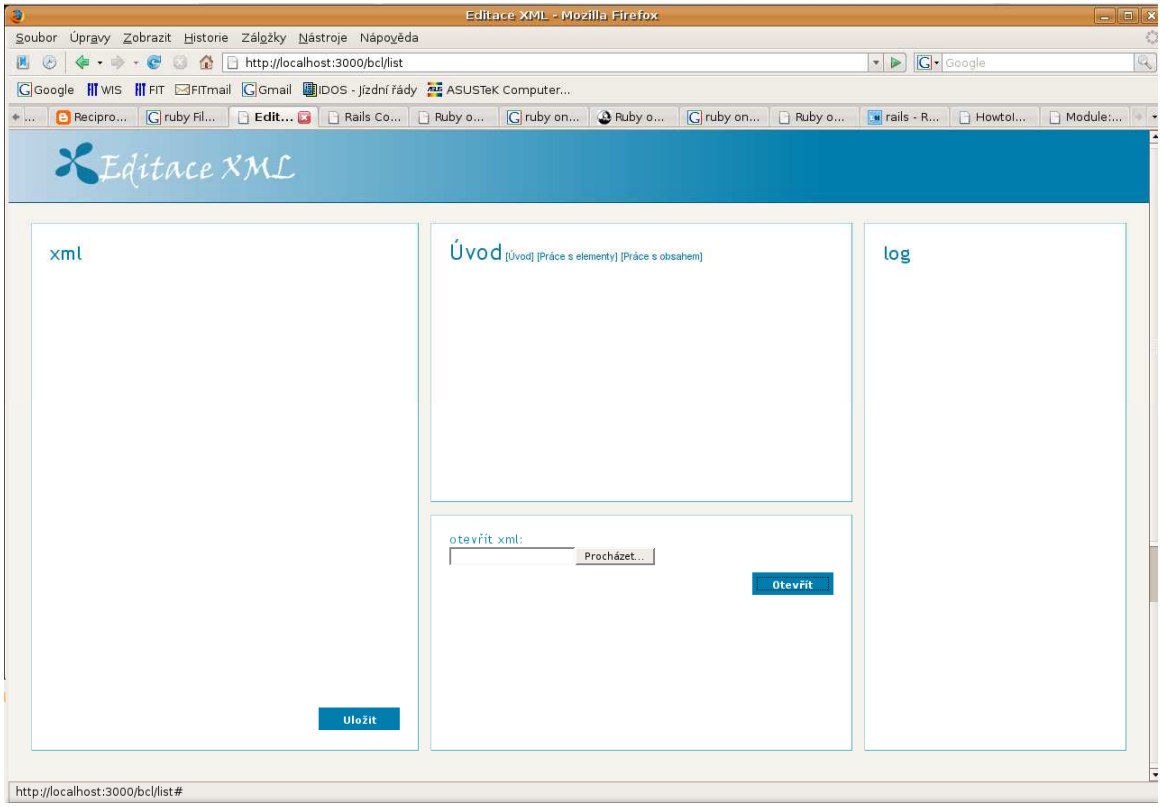
Svůj program jsem vytvářel na operačním systému Ubuntu 7.04 Feisty Fawn. Předpokladem bylo mít nainstalovaný již výše zmíněný framework RoR a MySQL.

3.5.3.1 Webové rozhraní

V terminálu operačního systému jsem pomocí příkazu 'rails jmeno_aplikace' vygeneroval typickou strukturu adresárů a souborů pro aplikace RoR. Mimo jiné se zde objevují adresáře, které odpovídají RoR filosofii – tedy models, views a controllers.

Vytvořil jsem si kontrolér, který zatím obsahuje pouze jednu nadefinovanou metodu 'index', která se provádí při spuštění aplikace. Této akci bylo nutné vytvořit odpovídající zobrazovací šablonu (view). A také layout, který do šablony automaticky vkládá opakující se elementy HTML dokumentu (html, head, body apod.). Kde jsem také vložil tagy pro načtení javascriptových knihoven a pro načtení CSS.

Šablonu index jsem upravil, aby odpovídala vzhledu na obrázku níže za pomoci obyčejného html v kombinaci s CSS.



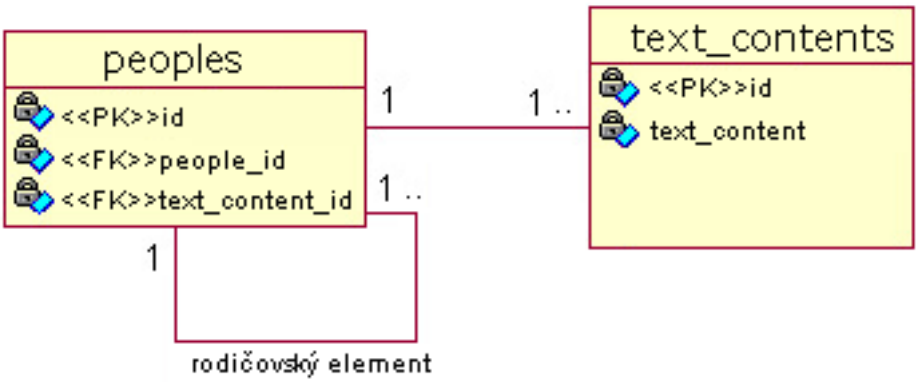
Obrázek 4 - index.rhtml

Jak je vidět, okno je členěno do několika částí. Nejlevější okénko je určeno pro zobrazování XML stromu. Vrchní okénko uprostřed má za úkol informovat uživatele, jaké funkce tato aplikace umožňuje a jak lze dosáhnout požadovaného výsledku. Spodní okénko obsahuje formulář s jedním vstupem a jedním tlačítkem. Pomocí těchto prvků může uživatel načíst libovolný xml dokument, který však musí odpovídat zásadám XML a jeho kořenový element je specifický (určen programátorem – například 'people'). Poslední, nejpravější okno je určeno k informování uživatele jaké akce proběhly a jaké probíhají. Tedy takzvané "logovací" okno.

3.5.3.2 Vytvoření databáze

Nejdříve jsem musel nastavit konfigurační soubor 'database.yml', který pro RoR zachovává parametry pro připojení k databázi. V tomto souboru je několik parametrů. Mezi ty, které by určitě neměly být opomenuty patří 'username', 'password', 'adapter', 'socket', a 'database'. Poslední jmenovaný je jméno databáze, kterou chceme použít v aplikaci.

Protože nemáme pevně daný XML soubor, není možné jej v databázi reprezentovat již klasickým způsobem, kdy pro každý element existuje jedna tabulka a parametry v tabulce odpovídají parametrům v XML. Proto jsem se rozhodl pro model odpovídající obrázku níže. Tedy dvě tabulky, kde jedna popisuje elementy a druhá jejich obsah. Protože lze atributy elementů nahradit elementy, které se v programu dají lépe zpracovat, rozhodl jsem se atributy nebrat v úvahu. Tedy tabulka s obsahem popisuje pouze obsah elementu, nikoliv jeho atributy. Toto řešení umožňuje variabilitu (v tabulce může být neomezený počet elementů).



Obrázek 5 - ER diagram

Ačkoliv bych mohl tabulky vytvořit přímo v mysql, využil jsem k této činnosti nástroj RoR 'migration'. Tedy migrační soubory, které tuto práci udělají za mě (tento nástroj je dobrý především při experimentování s databází, kdy potřebujeme tabulky opakovaně mazat, vytvářet apod.). Tabulky namigrujeme pomocí příkazu 'rake db:migrate' v terminálu. Pokud smazat obsah tabulek, stačí zadat 'rake db:migrate:reset'.

Tabulkám jsem vytvořil odpovídající modely. Podmínkou je, aby se jmenovaly stejně jako databázové tabulky s tím, že začínají velkým písmenem. Pokud je jméno tabulky víceslovné,

oddělené znakem '_', je nutné tento zápis nahradit 'velbloudím[13]'. Protože moje vytvářené třídy modelů dědí od tříd ActiveRecord, není nutné je nějakým způsobem vyplňovat. Pouze stačí zadat vztahy s ostatními tabulkami. Tedy můj model pro ukládání elementů vypadá takto:

```
class People < ActiveRecord::Base
  has_many :text_contents
  has_many :peoples
end
```

3.5.3.3 Akce

open_file

Jak název napovídá, tato akce má na starosti otevírání souboru se zadanou cestou. Formulář při akci onSubmit odešle na server Ajax požadavek, který říká, že se má vykonat právě tato akce. Požadavek odešle jako parametr cestu k uživatelem vybranému souboru.

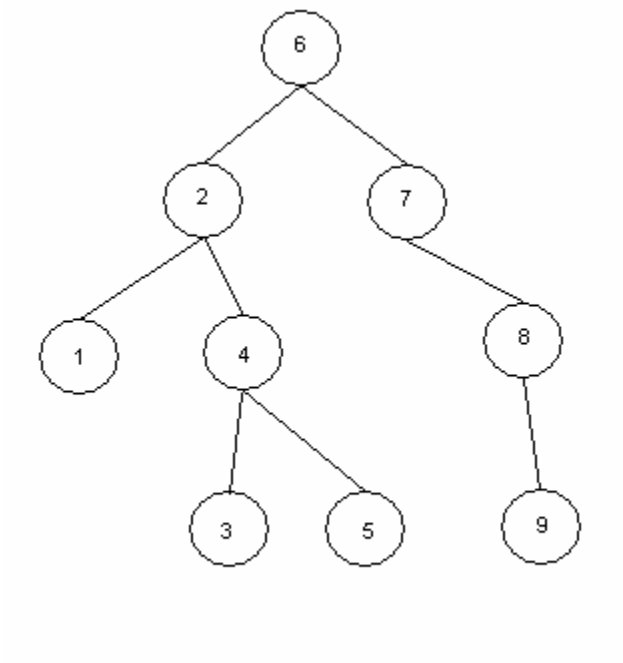
Tato akce má jako parametry 'eleclassnm' a 'contclassnm', které jsou implicitně nastavené na jména mých datových modelů. Tedy 'People' a 'TextContent'. Tyto parametry se v akci využívají pro vymazání obsahu databázových tabulek (příprava na nové načtení sobouru). Následuje volání funkce 'from_xml', které open_file předává parametr 'file_path', tedy cestu k souboru.

from_xml

Kromě cesty ke XML dokumentu, má také již zmíněné parametry 'eleclassnm' a 'contclassnm'.

Metoda využívá knihovnu REXML. Nejprve vytvoří objektový model XML dokumentu a poté ho prochází způsobem patrným z obrázku.

[13] Každé slovo, které má význam, začíná velkým písmenem. Tedy například 'MojeTabulka' (vs. moje_tabulka)



Obrázek 6 - Průchod stromem při parsování

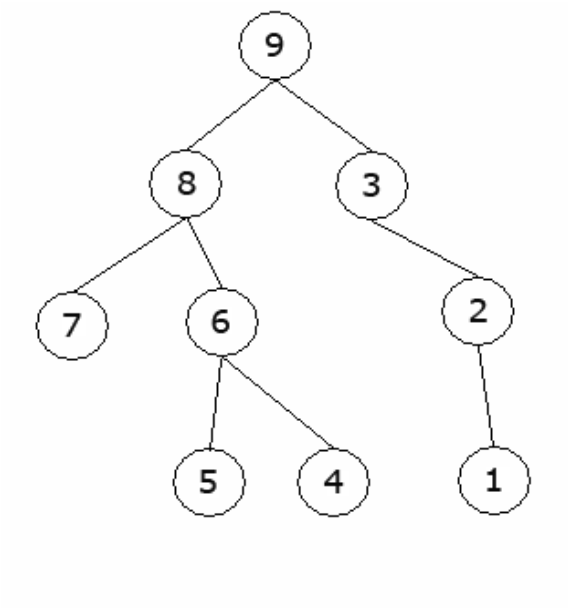
Do tabulky elementů ukládá jména elementů, id ukazující na předka (rodičovský element) a id odpovídajícího textového obsahu, který se ukládá do tabulky obsahu. Pokud aktuální prvek obsahuje subelementy, uloží svoje id na zásobník tolikrát, kolik obsahuje synovských elementů. Ty je poté ze zásobníku vytahují a ukládají si je jako id rodičovského elementu. Tím je zajištěno provázání prvků.

get_xml

Po metodě 'from_xml' volá 'open_file' metodu 'get_xml'. Ta má za úkol vytvořit HTML reprezentaci XML.

Tato metoda má opět parametry 'eleclassnm' a 'contclassnm'. Dále je to parametr 'offset', který je implicitně nastaven na hodnotu ' '. Což je odsazení XML stromu. ' ' Vloží do HTML jednu mezeru. Posledním parametrem je 'id' a určuje id kořenového element XML v databázi.

Metoda začíná u kořenového elementu. Díky provázanosti zjistí, zda má daný element synovské elementy. Pokud ano, zavolá sama sebe a jako 'id' vloží 'id' subelementů. Rekurzivita končí, pokud se narazí na element, který žádné synovské element neobsahuje.



Obrázek 7 - Princip funkce get_xml

Metoda vrací řetězec. Vrácený řetězec ve volající metodě má podobu HTML kódu popisujícího XML. Tento řetězec je pak vypsán částečnou[14] šablonou, která je volána šablonou RJS[15]. Ta odpovídá dané akci[16] a tím aplikace zodpoví dotaz Ajaxu. Šablona RJS dokáže vykreslit částečnou šablonu do konkrétního elementu html, což ve své aplikaci hojně praktikuji.

V této metodě také využívám nástrojů, které poskytuje JavaScriptový framework Prototype a script.aculo.us. Jedná se o Draggable a Droppable, které umožňují přesouvat (draggable) například kus textu ve stránce a pokládat jej na místa tomu určené (droppable). Každý text uvnitř XML element je jak draggable, tak droppable. Podmínkou je, že každý takový prvek musí mít jednoznačný identifikátor ve tvaru 'nejaky-text_nejake-cislo'. K tomu využívám 'id' z databáze. Které konkatenuji například s řetězcem 'item_' (pro případ textového obsahu). Pokud je nějaký object draggable "položen" na jiný objekt "droppable", object droppable zavolá pomocí Ajaxu příslušnou akci kontroléru, které se zároveň v poli parametrů předají i 'id' účinkujících objektů.

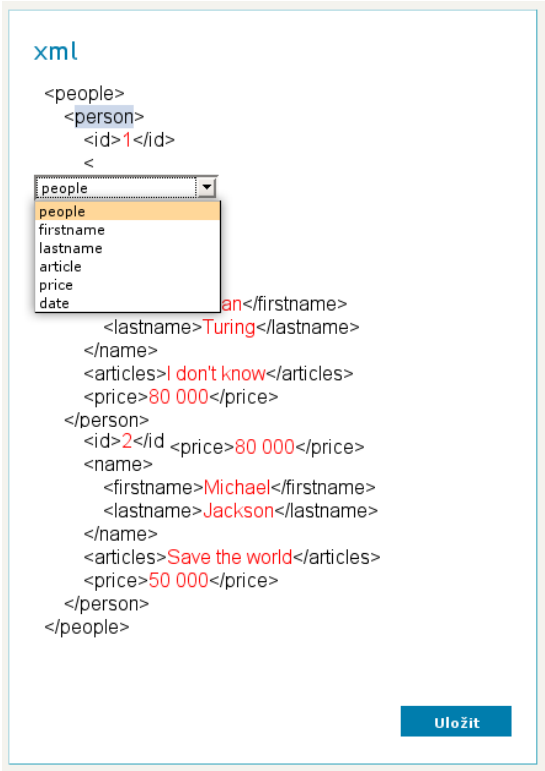
[14] Částečná šablona, neboli 'partial' – Šablona může obsahovat částečné šablony, ty jsou vhodné především, pokud chceme ve více šablonách pohledu použít stejné odkazy.

[15] Ruby JavaScript

[16] V Ruby on Rails implicitně každé akci, volané z pohledu (ať už ajaxem, akcí formuláře, či odkazem) , odpovídá vykreslovací šablona. Šablona s příponou .rhtml generuje HTML, šablona RJS generuje JavaScript.

Obrázek 8 - Drag & Drop

Jako další nástroj z knihovny Prototype a script.aculo.us využívám in-place editor. Ten aplikuji na jména elementů.



Obrázek 9 - In-place editor

In-place editor je obecně využíván k úpravě textového obsahu, který je součástí HTML stránky. Při kliknutí na prvek postižený in-place editingem se na jeho místě zobrazí miniformulář s

elementem 'input'[17]. Po potvrzení změny je odeslán Ajax dotaz, jehož součástí je i nový textový obsah.

Znaky '<', '>', uzavírající jméno elementu, uzavírám do řádkového elementu s onClick a onDblClick, při kterých spouštím opět Ajaxový požadavek.

delete_text_content

Metoda volaná Ajaxovým požadavkem při double-clicku na textový obsah elementu. Má za úkol tuto textovou položku smazat.

Z pole parametrů, předaného Ajaxem, získá id textové položky. Z tohoto 'id' ve formátu 'item_cislo', získá 'id' odpovídají záznamu v databázi. A textový obsah záznamu nastaví na prázdný řetězec. Pomocí funkce 'get_xml', RJS a částečné šablony znovu vykreslí XML.

form_form_change

Metoda volaná Ajaxovým požadavkem při clicku na znak '>' před textovým obsahem pomocí RJS a částečné šablony zobrazí formulář s akcí onSubmit a Ajaxovým požadavkem na změnu textového obsahu. Do textového pole formuláře vloží obsah původní položky, získané z databáze podle 'id' z pole parametrů.

change_text_content

Potvrzením změny formuláře (viz. předchozí metoda) se zavolá tato funkce, která opět podle 'id' změní položku v databázi a znovu překreslí XML.

delete_element

Metoda volaná Ajaxovým požadavkem při clicku na znak '<'. Opět podle 'id' smaže element z tabulky. Využívá k tomu pomocnou rekurzivní metodu 'delete_element_hlp'.

delete_element_hlp

Rekurzivně s výběrem pravého prochází databázi. Smaže element s jeho všemi synovskými elementy a jejich textovými obsahy.

add_subelement

Metoda volaná in-place editorem. Vytvoří synovský element elementu získaného podle 'id' z pole parametrů.

[17] Textové pole

xml_to_file

Metoda nejdříve zkontroluje, zda globální proměnná 'path', nastavená při načtení dokumentu není null. Ke své činnosti využívá funkci 'build_xml', která v řetězci vrátí obsah XML dokumentu. Tento řetězec poté zapíše do původního (načteného) XML souboru. Nakonec jako renderovací akci provede přesměrování na akci list, která pouze vykreslí šablonu jako na začátku činnosti celé aplikace (bez xml, bez logu apod.).

build_xml

Tato funkce opět za pomocí rekurzivity vytvoří proměnnou s obsahem XML dokumentu.

receive_drop

Callback funkce onDrop objektu 'Droppable' zavolá metodu 'receive_drop', pokud je na něj myší přesunut jiný objekt 'Draggable'. Ajaxový požadavek obsahuje 'id' obou objektů. Tato metoda provede výměnu textových položek odpovídajících databázových položek.

4 Aplikace z hlediska uživatelské přívětivosti

Při vytváření aplikace byl kladen důraz na jednoduchost a intuitivnost. Snažil jsem se vytěžit maximum z použitých technologií, aby byly stránky co možná nejvíce interaktivní a dynamické. A to takovým způsobem, aby i uživatel neznalý XML dokázal dokument upravovat.

K úpravě XML se využívá pouze levého tlačítka myši (click a doubleclick), při editaci textového obsahu, se pak pochopitelně musí využít i klávesnice, pomocí které se zadává nová hodnota.

Webové prostředí obsahuje mini tutoriál[18], aby se uživatel lépe orientoval a věděl, co je v jeho možnostech. Tento tutoriál obsahuje tři části. Úvodní část uživatelovi oznamuje, co aplikace nabízí. Další část popisuje, co lze s elementy provést a jakým způsobem. Část poslední popisuje operace týkající se textového obsahu uvnitř elementů.

Pokud uživatel provede krok, který by upravovaný dokument z jeho hlediska znehodnotil a následně ovlivnil další činnost programu (například smaže nesprávnou položku), není problém upravovaný dokument znovu načíst tlačítkem otevřít. Dokud neprovede příkaz uložení, úpravy se projeví pouze v databázi.

Stránka se skládá z prvků typických pro webové aplikace. Tlačítka, text, odkazy, formuláře, případně postníky, proto by pro běžného uživatele neměl být problém typicky koncipovaný program ovládat.

Logovací okno v pravé části stránky zaznamenává operace, které uživatel s dokumentem provedl. To se může hodit, například pokud by si spletl ovládáni myši a byl dezorientován (nepostřehl by, jakou úpravu provedl).

K uživatelské přívětivosti rovněž přispívá barevné rozlišení, ať už prvků XML stromu, či prvků na stránce.

Také rozvržení aplikace do okének, či kontrastní design (kombinace bílé, modré, černé a červené), přispívá k lepší orientaci a přehlednosti.

Prostředí je vytvořeno takovým způsobem, aby byla syntaxe XML generována automaticky. Uživatel je omezen v úpravě syntaxe XML.

Na stránce jsou použity také efekty zvýraznění (highlighting). Při najetí myši nad takto "velepšený" prvek se v tomto případě změní barva pozadí a písma.

Protože jsem autorem, a autor často trpí tzv. "autorskou slepotou", dovolil jsem si požádat tři osoby, aby aplikaci otestovali a zhodnotili z jejich pohledu.

[18] Stručný průvodce

Jakub Motl

Je to velice zajímavá aplikace. Líbí se mi jak vzhledově, tak provedením. Co bych ještě doplnil, tak tlačítko zpět, které by vrátilo provedené změny, ale protože nejsem programátor, nevím, jak náročné by bylo takové tlačítko vytvořit.

Jakub Černý

Co mě opravdu nadchlo, tak je drag & drop, do dneška jsem neměl tušení, že by něco takového šlo vytvořit a líbí se mi to. Stránky programuji jen málo a Ruby on Rails jsem do této chvíle neznal, zabývám se spíše grafickým designem, z tohoto pohledu se mi zdají v pořádku. Přehledné a jednoduché.

Pavel Kovář

Webové stránky se mi líbí. Podle mého názoru graficky dobře zpracované a snadné ovládání. Logovací okno je určitě dobrý nápad, ještě bych doplnil, které upravované prvky byly operací ovlivněny.

Závěr

Ačkoliv ve výsledku vypadá vše jednoduše, musím poznamenat, že celý průběh procesu vytváření bakalářského projektu, provázelo nesčetně problémů, které nebylo vůbec jednoduché vyřešit a to instalací samotného Ruby on Rails počínaje a samotnou implementací konče.

Avšak tento stav nelze nikomu vyčítat z několika důvodů. Framework Ruby on Rails je relativně novou záležitostí a pochopitelně je stále co vylepšovat a měnit. Neustále vznikají nové a nové verze, které nejsou navzájem kompatibilní a které obsahují tzv. bugy (chyby). Právě díky tomu jsou dokumentace, příklady, fóra, či newsové skupiny dostupné na internetu, zastaralé a matoucí. Není možné se spolehnout na knižní publikace, protože když taková kniha vyjde, už je hodně věcí jinak. Možná i to odradilo spoustu programátorů, kteří o něj projevili zájem a vytvořilo tak svět zarytých odpůrců a svět skalních fanoušků.

O tom, že je to relativně nový framework vypovídá i počet knižních publikací, které vyšly v českém jazyce a možná i dočasný nezájem české programátorské veřejnosti. To vše ale nemění nic na faktu, že se Ruby on Rails stává velice populární (obzvláště ve světě).

Když jsem se zabýval analýzou a návrhem aplikace snažil jsem se nalézt něco, čím bych se mohl inspirovat a zároveň si dokázat, že něco takového lze vůbec vytvořit. Ačkoliv jsem nalezl několik webových WYSIWYG xml editorů, žádný mi nepřipadal vhodný pro potřeby Generátoru vědeckých webových portálů. Proto jsem se rozhodl vytvořit aplikaci vlastní. Nicméně bych rád zmínil zajímavý projekt TinyMCE, což je propracovaný JavaScriptový WYSIWYG editor obsahu, který je možné zakomponovat do vlastních webových stránek (http://tinymce.moxiecode.com/). Nebo také Bitfluxeditor (http://bitfluxeditor.org/), či Xopus (http://xopus.com/).

Pokud bych měl projekt zhodnotit z pohledu dalšího vývoje, určitě by se měla vyřešit spolupráce aplikace s DTD, či XSD a podle specifikace dokumentu dovolit uživateli vytvářet odpovídající dokument. Dále bych určitě zmínil přesouvání samotných elementů i s obsahem, či vkládání nových elementů jako rodičů. Možná by také nebylo marné, zamyslet se nad tím, jak z projektu vynechat databázi (použití pouze DOM) a celou aplikaci tak zjednodušit.

Co se týká mého programu, jsem velice rád, že jsem se mohl zabývat právě touto problematikou a objevit tak novou ještě "neprozkoumanou" oblast.

Tak jako se kdysi přeměnil obsah internetových stránek z prostého textu na obrázky a jiná multimedia, tak se v dnešní době mění ze statických stránek na dynamické. Stránky se stávají stále více interaktivní. Jsem rád, že jsem mohl také přispět k této přeměně a doufám, že jsem vytvořil něco natolik inovativního, čím by se dalo v budoucnosti alespoň trochu inspirovat.

Literatura

[1] Thomas, D., Hansson, D.H. *Agile Web Development with Rails*. United States of America, The Pragmatic Programmers LLC., 2005

[2] Fowler, Ch.. *Rails Recipies*. United States of America, The Pragmatic Programmers LLC 2006.

[3] Portenueve, Ch. *Prototype and script.aculo.us*. United States of America, The Pragmatic Programmers LLC. 2007.

[4] Holzner, S. *Začínáme programovat v Ruby on Rails*. Brno, Computer Press, a.s. 2007.

[5] Harold, R.E., Means, S.W. *XML v kostce – Pohotová referenční příručka*. Praha, Computer Press, a.s. 2002.

[5] Vznik v rámci Wiki, S.W. *Script.aculo.us documentation.*, Dokument dostupný na URL http://script.aculo.us/docs/HomePage.html (květen 2008).

[6] Vznik v rámci Wiki, S.W. *Script.aculo.us documentation.*, rok neznámý, Dokument dostupný na URL http://script.aculo.us/docs/HomePage.html (květen 2008).

[7] Molič, J. *Seriál o vývoji webu v Ruby on Rails., 11.11.2005 – 16.03.2006*, Dokument dostupný na URL http://www.root.cz/serialy/ruby-on-rails/ (květen 2008)

[8] Prototype Core Team, *Prototype API documentation., 2006 - 2007*, Dokument dostupný na http://www.prototypejs.org/api (květen 2008)

[9] Autor neznámý, *REXML Home.*, rok neznámý, Dokument dostupný na URL www.germane-software.com/software/**rexml**/ (květen 2008)

[10] Autor neznámý, *Ruby on Rails API documentation*, prosinec 2007, Dokument dostupný na http://api.rubyonrails.org/ (květen 2008)

[11] Vznik v rámci Wiki, *Ruby on Rails API documentation*, rok neznámý, Dokument dostupný na http://www.rubyonrails.org/community (květen 2008)

[12] Skupina autorů, *Diskusní skupina na téma Ruby on Rails*, rok neznámý, Dokument dostupný na http://groups.google.com/group/noobkit/ (květen 2008)

Seznam příloh

Příloha 1. CD/DVD ...