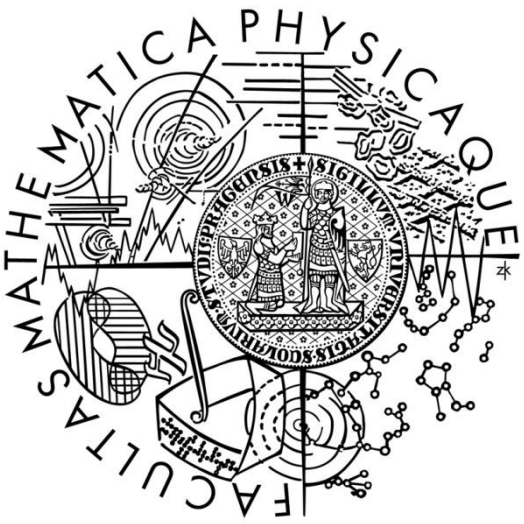
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Miloslav Beňo
Extraktor sémantických dat
Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D.
Studijní program: Informatika, Programování

2007

Děkuji panu RNDr. Filipu Zavoralovi, Ph.D. za odborné vedení mé práce, za čas, rady a doporučení, které mi věnoval.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 7.srpna 2007 Miloslav Beňo

Obsah

1 Úvod ..... - 6 -

2 Teorie ..... - 8 -

2.1 Co je sémantický web? ..... - 8 -

2.2 Potřebné technologie ..... - 9 -

2.3 Zdroje (Resources) ..... - 9 -

2.4 RDF ..... - 10 -

2.5 RDF/XML syntax ..... - 11 -

2.6 Ontologické slovníky ..... - 13 -

2.7 RDFS ..... - 15 -

2.8 Shrnutí ..... - 16 -

3 Analýza ..... - 17 -

3.1 Definice problému ..... - 17 -

3.2 Web scraping – získávaní dat z webových stránek ..... - 17 -

3.3 Analýza systému ..... - 18 -

4 Uživatelská dokumentace ..... - 22 -

4.1 Úvod do systému AgentMat ..... - 22 -

4.2 Práce se systémem ..... - 22 -

4.3 Hlavní XML elementy dokumentu ..... - 24 -

4.3.1 Agent ..... - 24 -

4.3.2 DataFeeders ..... - 24 -

4.3.3 Constants ..... - 25 -

4.3.4 DownloadMethods ..... - 26 -

4.3.5 Logická stránka - Page ..... - 27 -

4.4 Komponenty ..... - 28 -

4.5 Extractor pattern ..... - 29 -

4.6 Regulární výrazy ..... - 30 -

4.7 Hlavní komponenty ..... - 30 -

4.7.1 ElementIterator ..... - 31 -

4.7.2 DataFeeder ..... - 31 -

4.7.3 Link ..... - 32 -

4.7.4 SwitchLink ..... - 32 -

4.7.5 Spliter ..... - 32 -
4.7.6 UniqIdGenerator ..... - 33 -
4.7.7 ActualUrlGetter ..... - 33 -
4.7.8 RelativeUrlResolver ..... - 33 -
4.8 Popis rozhraní ..... - 33 -
4.9 Rozhraní IDataSaver ..... - 35 -
4.10 Logování ..... - 36 -
4.11 AgentMatTester ..... - 37 -
4.12 Získávání sémantických dat ..... - 38 -
5 Programátorská dokumentace ..... - 40 -
5.1 Rozdělení systému ..... - 40 -
5.2 Komponenty ..... - 40 -
5.3 LinkExecutor ..... - 41 -
5.4 Třídy pro komunikaci mezi komponentami ..... - 42 -
5.5 Kontextová session ..... - 44 -
5.6 DownloadThreadPool ..... - 45 -
5.7 Překlad z XML do .NET assembly ..... - 46 -
5.7.1 Převod z XML do CAbstractionTree ..... - 46 -
5.8 Příklady práce systému ..... - 48 -
5.8.1 Příklad 1 ..... - 48 -
5.8.2 Příklad 2 ..... - 50 -
5.9 Úpravy zparsovaných dat ..... - 53 -
5.10 Rozšíření o vlastní komponentu ..... - 53 -
6 Jiné aplikace ..... - 56 -
6.1 Kapow Mashup Server ..... - 56 -
6.2 WWW::Mechanize ..... - 58 -
6.3 Screen scraper ..... - 59 -
7 Závěr ..... - 61 -
Reference ..... - 62 -
Příloha 1. AgentMat XML Dokument Specifikace ..... - 64 -
Příloha 2. XSD Schéma dokumentu AgentMat Xml ..... - 68 -

Název práce: *Extraktor sémantických dat*
Autor: *Miloslav Beňo*
Katedra (ústav): *Katedra softwarového inženýrství*
Vedoucí diplomové práce: *RNDr. Filip Zavoral, Ph.D.*
e-mail vedoucího: *Filip.Zavoral@mff.cuni.cz*

Abstrakt: *Předmět této práce je implementace systému pro efektivní extrakci velkého množství dat z webových stránek. Výsledný systém je schopen extrahovat, jak z jednoduchých statických stránek, tak i ze složitých webových aplikací. Data, která ze stránek získá, mohou být nejrůznějších typů a mohou být propojeny relacemi o různých kardinalitách.*

Se systémem se pracuje přes xml dokumenty, do kterých se úloha deklarativně popisuje. Popis úlohy spočívá v použití komponent systému, jejichž propojováním se dosáhne požadované funkčnosti na obecné webové stránce.

Klíčová slova: *web scraping, extrakce dat, semantický web*

Title: *Semantic data extractor*
Author: *Miloslav Beňo*
Department: *Department of Software Engineering*
Supervisor: *RNDr. Filip Zavoral, Ph.D.*
Supervisor's e-mail address: *Filip.Zavoral@mff.cuni.cz*

Abstract: *The aim of this work is to implement a system designed to effectively extract big amounts of data from web pages. The system is able to extract data both from simple static pages as well as complicated web applications. Data extracted from the web can have all sorts of types and can be interconnected by relations having various cardinalities.*

Working with system is based on XML documents, into which the given task is described in a declarative way. The description of a task is based on using system components, which connected together are able to do desired functionality on general web page.

Keywords: *web scraping, data extraction, semantic web*

1 Úvod

Když chce dnes člověk získat nějakou informaci, většinou otevře svůj oblíbený internetový prohlížeč a začne hledat webovou stránku, která by požadovanou informaci obsahovala. Málokdy se stane, že by na Webu opravdu nenašel to, co hledá, nebo alespoň nezjistil, kde informaci najít v reálném světě.

Dnešní Web je veskrze technologií, která umožňuje lidem sdílet informace. A zde je něco co by šlo vylepšit. To něco je ve slově lidem. Vše, co web nabízí, je vytvořeno tak, aby se to líbilo uživateli za monitorem, aby se mu informace pohodlně četly, mohl si skákat pomocí hypertextových odkazů ze stránky na stránku, nakupovat, rezervovat atd. Avšak stroje bez lidského uživatele podobné věci dělat nemohou, protože Web je designován pouze pro ovládání lidmi. Zde nastupuje sémantický web, což je Web, kde jsou informace popsané tak, aby s nimi mohly pracovat stroje na podobné úrovni, jako lidský uživatel. Z celého internetu by se tak mohla stát lehce čitelná pospojovaná databáze. A stroje by tak byly schopné používat a analyzovat data z webu bez nutnosti lidského zásahu.

> *Vždy jsem si představoval vesmír informací jako něco k čemu bude mít každý okamžitý přístup a nejen k prohlížení, ale i k jeho vytváření... Stroje se stanou schopné analyzovat všechna data na Webu - obsah, odkazy a transakce mezi lidmi a stroji*
>
> *Když se Sémantický web objeví, každodenní mechanismus obchodu, byrokracie a našich rutinních životů bude obsluhován stroji mluvícím k strojům, nechávajíc lidi k poskytování inspirace a intuice.*

Tim-Berners-Lee 2000

Ačkoliv se to možná zdá jako science fiction, některé principy sémantického webu je možné vidět v praxi již dnes. Například webové služby či nejjednodušší a nejrozšířenější forma webové integrace kanály rss. Celá vize sémantického webu jde ale mnohem dál (viz kapitola 2.1 Co je sémantický web?).

Aby mohl sémantický web začít existovat v reálném světě, je třeba udělat ještě spoustu práce. Některé technologie už existují, jiné je třeba dokončit či zcela vyvinout. Jednou z nezbytných věcí je, aby mnoho lidí začalo publikovat informace za pomoci těchto někdy možná ještě nedokončených technologií. Málokterá firma ale zaplatí za vytvoření systému, který umožní publikovat jejich data skrz nějakou experimentální technologii. Důvodem může být, že je to příliš nákladné bez vidiny přímého zisku. Proto je nutné najít způsob, jakým data získat tak, aby mohly být publikovány těmito novými technologiemi. Jelikož je však největším zdrojem informací Web, bude vhodné je získávat právě odtud. Problém je, že dnešní Web je přizpůsoben pro čtení lidem a ne strojům. Je zde však metoda zvaná web scraping[19] či screen scraping[22] (jedná se o něco trošku jiného), která má za cíl data z webu vyextrahovat do nějaké již pro stroj čitelné strukturované podoby, například do XML souboru či přímo do databáze.

Tato práce se primárně zabývá extrahováním dat z webových stránek neboli web scrapingem. Jako její součást vznikl rozšiřitelný framework pro rychlou extrakci dat z webu, jenž bude v této práci popsán. Rovněž s ním vznikly dvě menší aplikace, které s tímto systémem souvisí.

Text v kapitole 2 pojednává o tom, co je to sémantický web a nahlíží na známé technologie, které k němu patří. Rozebrání web scrapingu a analýza toho, jakým způsobem je systém navržen, je uvedena v kapitole 3. V kapitole 4 se nachází uživatelská dokumentace systému a její implementační detaily jsou uvedeny v programátorské dokumentaci v kapitole 5. Přehled jiných aplikací na řešení problému extrakce dat z webových stránek je uveden v kapitole 6. Součástí práce jsou i dvě přílohy, v kterých je formální popis XML dokumentu, který se používá k deklarativnímu zápisu problému.

2 Teorie

2.1 Co je sémantický web?

Web, jak jej dnes známe, je vlastně obrovská encyklopedie. Informace v ní jsou publikovány primárně přes stránky HTML, které jsou směsicí tagů, jenž definují, jak má stránka vypadat ve webovém prohlížeči. U HTML jde pouze o vzhled, ona užitečná informace je až následně interpretovaná člověkem, který si tuto stránku přes webový prohlížeč zobrazil.

HTML nemá způsob, jak by popsalo význam věcí, které obsahuje. Maximálně dokáže přes některé tagy označit, že toto je nadpis či titulek apod. V sémantickém webu jde o obohacení stávajících webových stránek významem. Neboli jedná se o metodu, jak věci popsat způsobem, kterému by mohli „rozumět“ stroje. Samozřejmě tu nejde o pravé porozumění, ale o logiku.

Příklad:

- Jestliže osoba A je otec osoby B a B má potomka C, potom je A dědeček osoby C
- Jestliže A je dcera B, potom B je otec A.

- Řehoř je otec Svatopluka.
- Martina je dcera Svatopluka.

➢ Tedy Svatopluk je otec Martiny a potom Řehoř je jejím dědečkem.

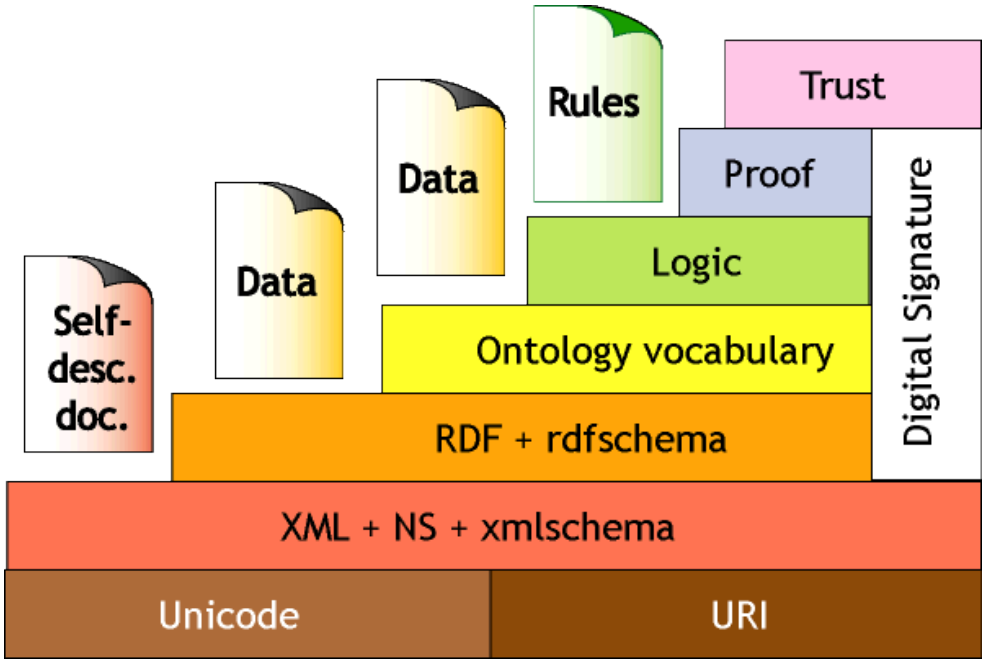
Podle předcházejícího příkladu bude mít pan Řehoř na svých stránkách informaci, že má syna Svatopluka a Martina na svých zmíní, že je dcerou Svatopluka. A někdo si nechá na sémantickém webu zjistit, kdo je dědečkem Martiny. Potom díky předchozím tvrzením v příkladu a údajům zjištěných na dvou stránkách, bude výsledkem hledání informace, že jejím dědečkem je Řehoř, ačkoliv to nikde nebylo přímo napsáno. V dnešních fulltextových vyhledávačích by výraz „dědeček Martina“ moc nepomohl, protože se nikde na stránkách nevyskytuje.

Dále bude uživatel například chtít nalézt seznam cen plasmových HDTV obrazovek s rozlišením 1080p a úhlopříčkou přes 40 palců v obchodech, které jsou v Praze 6 otevřeny v úterý minimálně do 8 večer. Taková úloha dnes vyžaduje prozkoumání několika speciálně

navržených vyhledávačů, ve světě sémantického webu by stačilo zadat tento požadavek do jednoho vyhledávače.

2.2 Potřebné technologie

Technologie potřebné pro sémantický web jsou znázorněny na Obrázek 1, občas zvaném „Semantic Web Layer Cake“. Každá technologie v nějaké vrstvě je závislá na vrstvě pod sebou, je však nezávislá na vrstvě nad ní. V tomto textu se nachází popis vrstvy RDF + RDF Schema a ontologických slovníků. Jednotlivé vrstvy jsou rozebrány v publikaci [1].



Obrázek 1- Semantic Web Layer Cake (Převzato z [11])

2.3 Zdroje (Resources)

Zdroji se nazývají všechny položky, které můžou být při publikování dat na webu zajímavé. Jsou to věci, jejichž vlastnosti nebo vzájemné vztahy mají být popsány.

Dělí se na dvě základní kategorie. Zdroje informační a neinformační. Informační zdroje jsou takové, které se dají nalézt na webu, jako třeba fotky, dokumenty, filmy apod. Je však spousta věcí, které na internetu nalézt nelze. Například se jedná o osoby, místa, předměty, vesmírné objekty, nemoci atd. Tedy o zdroje z reálného světa.

Oba druhy zdrojů musejí být jednoznačně identifikovány, na to se používají URI (Uniform Resource Identifier, specifikovaný v [12]). U sémantického webu konkrétně se

používají výhradně HTTP URI. V zásadě se tím zabezpečí jednoznačnost bez nutnosti centralizované správy a možnost přístupu k informacím o zdroji přes Web. Více o tom proč bylo zvoleno HTTP URI před jinými na [2].

Volba URI závisí na typu zdroje. Pro zdroj dosažitelný přes internet se jednoduše použije jeho adresa. Pokud se ale jedná o zdroj z reálného světa, je třeba pro něj vybrat správnou URI. Ta by měla být pod správou osoby či organizace týkající se zdroje, tím se zabezpečí jednoznačnost a rovněž schopnost vrátit příslušnou odpověď pokud se na tuto URI přes Web někdo dotáže. Příkladem dobré URI může být například http://dbpedia.org/resource/Berlin, což je odkaz na informace o Berlíně v sémantické encyklopedii http://dbpedia.org. Více o výběru správné URI a o tom co vracet při dotazu na ni je na [3].

2.4 RDF

RDF neboli Resource Description Framework pochází z W3C a byl vytvořen jako model metadat (data o datech neboli informace o nějakém obsahu), z kterého se nakonec stala obecná metoda pro modelování informací v různých syntaktických reprezentacích. Mnoho informací o RDF je možné nalézt na [4].

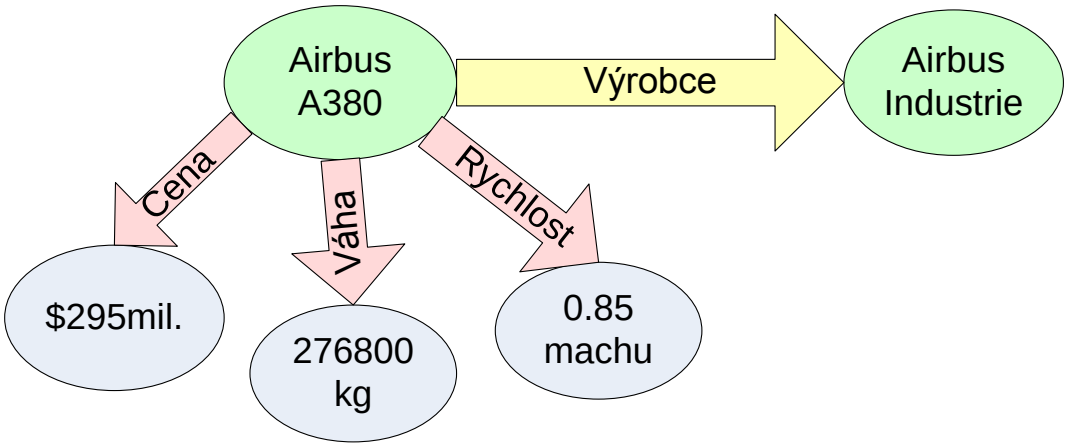
Základ tohoto modelu jsou trojice reprezentující tvrzení o zdrojích. Tyto trojice se skládají z podmětu, predikátu a předmětu. Podmět je vždy nějaký zdroj identifikovaný přes URI. Predikát vyjadřuje vztah mezi podmětem a předmětem. Například věta: „Honduras má hlavní město Tegucigalpu“ je trojice s podmětem Honduras, predikát je hlavní město a předmět je Tegucigalpa. Zde je předmětem opět další zdroj, který bude identifikován nějakým URI. Předmět může být také nějaký literál, tedy například řetězec, číslo, datum atd.

Predikát se stejně jako zdroje označuje pomocí URI. Tyto URI pocházejí ze slovníků, což jsou kolekce URI reprezentující informace o určité informační doméně (Například Slovník pro doménu cestování, kde budou predikáty typu kolik má hotel hvězdiček, jak daleko je od pláže atd.). Více o slovnících v kapitole 2.6 Ontologické slovníky.

Trojice se dají rozdělit na 2 hlavní druhy:

- **Literální trojice** – má za předmět literál. Vyjadřuje nějakou vlastnost podmětu. Například jakou má nějaká věc barvu, jakou má prvek atomovou hmotnost atd.
- **RDF odkaz** – trojice, která má za podmět nějaký další zdroj. Jedná se tedy o tři URI. Predikát přitom určuje typ odkazu. Dá se tak například vyjádřit vztah, že určitá osoba je zaměstnána v nějakého podniku apod.

Na RDF se dá rovněž nahlížet i jako na graf.



Obrázek 2- Příklad RDF o A380

Na Obrázek 2 jsou dva zdroje. Airbus A380 a jeho výrobce Airbus Industrie. Dále tři literální trojice a jeden rdf odkaz.

Data modelu RDF lze reprezentovat jazyky s různou syntaxí. Nejčastěji je možné se setkat s reprezentací založenou na XML s názvem RDF/XML, tento způsob je však spíše vhodný jako univerzální dorozumívací prostředek, ukládat v něm data není vhodné. Proto se jako úložiště často používají databáze trojic tzv. RDF úložiště, z nichž je možné RDF/XML dokumenty generovat. Dále je možné se setkat s jinými jazyky pro RDF, například Turtle [5].

2.5 RDF/XML syntax

RDF je model metadat a RDF/XML je pouze možnost, jak ho zobrazit.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
     <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Bill_gates">
         <dc:title>Bill Gates</dc:title>
         <dc:publisher>Wikipedia</dc:publisher>
     </rdf:Description>
</rdf:RDF>
```

Výpis 1- Příklad RDF/XML dokumentu o článku o Billu Gatesovi

V kořenovém elementu se podle pravidel XML nadefinují jmenné prostory, které budou v dokumentu použity. V příkladu na výpisu 1 se jedná o jmenný prostor pro rdf a rovněž pro dc, což je často používaný veřejný ontologický slovník (viz kapitola 2.6 Ontologické slovníky) pro popis zdrojů.

Element rdf:Description se používá při uvádění informací o nějakém zdroji, prakticky tedy pokaždé. Atribut about říká, který zdroj zde bude popsán. V tomto případě se jedná o informace o článku na Wikipedii.

Predikát title ze slovníku dc říká, že titulek článku je „Bill Gates“ a publisher určuje, kdo článek vydal. Oba tyto predikáty jsou použity jako XML elementy, můžou být také uvedeny jako atributy, výsledek by byl stejný.

Zde byly použity pouze literální trojice. V následujícím příkladu se použije i rdf odkaz. Jedná se o malou část RDF dokumentu o Bernardu Bolzanovi získanou ze sémantické encyklopedie http://dbpedia.org

```
<rdf:RDF
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:j.0="http://dbpedia.org/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:yago="http://dbpedia.org/class/yago/"
...
>

<yago:logician rdf:about="http://dbpedia.org/resource/Bernard_Bolzano">
  <rdfs:label xml:lang="en">Bernard Bolzano</rdfs:label>
  <j.0:birthplace rdf:resource="http://dbpedia.org/resource/Prague"/>
...
</rdf:RDF>
```

Výpis 2 - Kus RDF/XML dokumentu z dbpedia.org o Bernardu Bolzanovi

Prvním rozdílem oproti předchozímu příkladu je vynechání elementu rdf:Description. Ten byl nahrazen typem logician z jmenného prostoru yogo. Jedná se o tzv. typový uzel, čímž se dá rovnou určit typ pro popisovaný zdroj. Typový uzel je ekvivalentní zápisu:

```
<rdf:Description rdf:about="http://dbpedia.org/resource/Bernard_Bolzano">
    <rdf:type rdf:resource="http http://dbpedia.org/class/yago/logican"/>
...
</rdf:Description>
```

Výpis 3 - Alternativa k typovému uzlu v RDF/XML

Elementem label ze jmenného prostoru rdfs (viz 2.7 RDFS) s atributem xml:lang nastaveným na „en“ přidává tomuto zdroji pro lidi čitelné jméno v anglickém jazyce. Poslední je zde uveden predikát birthplace, který říká kde se Bolzano narodil. Toto je rdf odkaz, jenž spojuje URI Bolzana s URI Prahy.

Díky tomuto rdf odkazu je možné nějakému programu například zadat, aby zjistil kolik lidí žije ve městě, kde se narodil Bolzano. Program potom přes tento odkaz přejde na URI do Prahy, kde si příslušnou informaci zjistí.

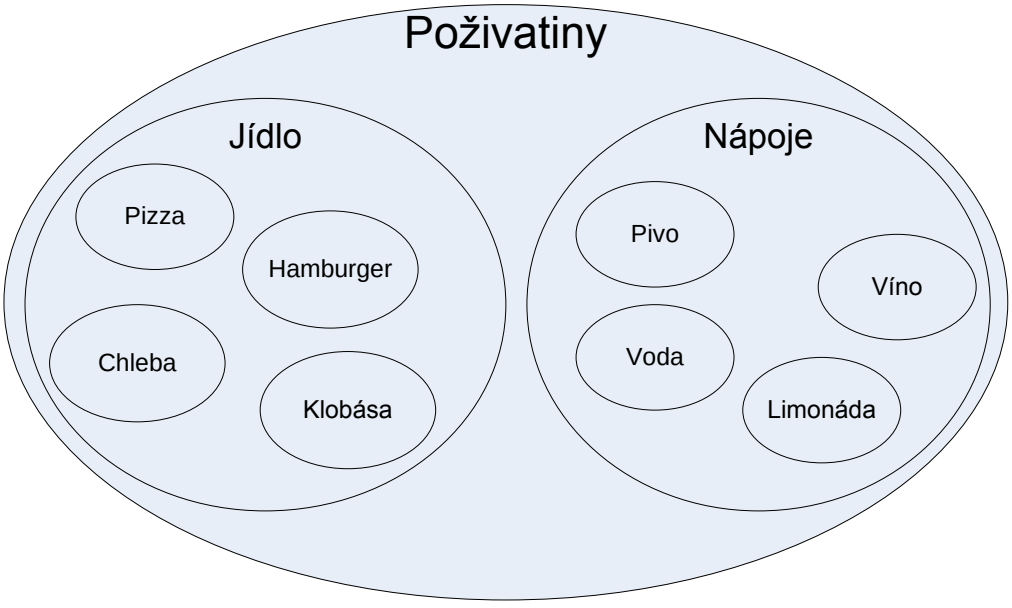
Toto bylo pár příkladů o RDF/XML pro získání představy o tomto jazyku. Jeho podrobnou dokumentaci je možné nalézt na [6].

2.6 Ontologické slovníky

Dalším základním kamenem sémantického webu jsou ontologie. Ty umožňují tvorbu metadat, neboli informací o informacích. Jedná se o explicitní popis objektů a pojmů modelující určitou informační doménu, například nějaký obor jako biologie, matematika, letectví atd.

Ontologické slovníky jsou postaveny tak, aby na ně šly aplikovat principy logiky a tyto slovníky spolu šly kombinovat.

Základem ontologií je klasifikace predikátů a zdrojů obou typů, tedy ty, co jsou přístupné z internetu, a ty, co existují pouze v reálném světě. V neformální formě zná klasifikaci každý. Jedná se o prosté rozdělení věcí do skupin, do kterých náleží.



Obrázek 3 - Klasifikace v stravovací doméně

Na obrázku 3 je znázorněno, jak by například mohla vypadat část ontologie v stravovací doméně.

Klasifikovaní při tvorbě ontologií je velmi podobné tvorbě hierarchie tříd v objektově orientovaných programovacích jazycích. Používají se zde i stejné pojmy: třída, odvozená třída, instance. Př. Pizza je podtřídou třídy Jídlo a Capricciosa je instancí třídy Pizza.

Jedna část ontologického slovníku je tedy klasifikace objektů, další částí je popis pojmů (predikátů), které o jednotlivých objektech něco vypovídají. Jedná se o stejné predikáty jako u trojic RDF, tedy o nějaký termín vyjadřující vztah mezi dvěma věcmi nebo vlastnost. Zde se definuje, mezi kterými třídami se tento pojem může vyskytovat, tedy omezení. Například pojem „má kamaráda“ by měl podmět i předmět omezen na osoby. Nemůže se však již týkat aut nebo domů.

Pro popis ontologií byly specifikovány nejrůznější jazyky. Prvním, který byl za tímto účelem pro sémantický web vytvořen, byl RDF Schema, ten je popsán v kapitole 2.7. Je velmi jednoduchý a pro spoustu věcí dostačující. Novější OWL umožňuje vše jako RDF Schema a navíc má silnější vyjadřovací schopnosti.

Pokud někdo chce vytvořit RDF data, bude potřebovat takovýto slovník. Ten si buď může sám vytvořit anebo použít již existující. Obecně se doporučuje používat existující slovníky, jednak to dá méně práce a data budou jednodušeji srozumitelná pro okolní svět. Samozřejmě pokud žádný takový veřejný slovník nepokrývá oblast zájmů, která má být popisována, je nutné si vytvořit vlastní.

Zde je seznam nejznámějších ontologických slovníků. Pokud se potřebný pojem nachází v nich, určitě je dobré použít právě tento.

- http://www.foaf-project.org – Slovník pro popis lidí.
- http://dublincore.org/documents/dcmes-xml/ definuje obecná metadata pro popis zdrojů.
- http://sioc-project.org/ Slovník pro reprezentaci online komunit.
- http://usefulinc.com/doap/ Slovník pro popis projektů.
- http://musicontology.com/ - poskytuje pojmy pro popis umělců, alb a muziky.

Pro hledání existujících ontologií a pojmů se také používají k tomu určené vyhledávače, například Swoogle, který se nachází na http://swoogle.umbc.edu/.

2.7 RDFS

RDFS, nyní nazývaný RDF Vocabulary Description Language, byl specifikovaný konsorciem W3C k poskytování základních elementů pro popis ontologií. Je tedy určen k popisu hierarchie tříd a predikátů, které se tříd týkají.

Celý je založen na modelu RDF. Vše se tedy vyjadřuje pomocí trojic podmět, predikát, předmět. Jakékoliv tvrzení RDFS je rovněž platným tvrzením RDF.

RDFS a RDF definuje 13 základních tříd a 16 predikátů, jejichž specifikace je uvedena na [7]. Následuje seznam těch nejčastěji používaných.

- rdfs:Resource – všechny věci popisované v RDF jsou zdroje (Resources) a jsou instancí této třídy.
- rdfs:Class – Třída pro zdroje, které jsou RDF třídami.
- rdfs:Property –Třída pro RDF predikáty.
- rdf:type – Predikát, který přidružuje třídu nějakému zdroji.
- rdf:label – Predikát, který nastavuje pro lidi čitelné jméno pro podmět tohoto predikátu
- rdfs:subClassOf – Určuje od jaké třídy je podmět odvozen.
- rdfs:subPropertyOf – Stejný predikát jako subClassOf, jen se týká predikátů.
- rdfs:domain – Určuje, jakých věcí se může nějaký predikát týkat. Tedy jaký může mít podmět.
- rdfs:range – Definuje, jakých hodnot může nabývat předmět tohoto predikátu.

Toto je příklad, jak takové RDFS může vypadat. Použita je syntaxe RDF/XML.

```
<rdf:RDF
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description rdf:id="DopravniProstredek">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Description>

<rdf:Description rdf:id="Auto">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="#DopravniProstredek"/>
</rdf:Description>

<rdf:Description rdf:id="Loď">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="#DopravniProstredek"/>
</rdf:Description>
<rdf:Description rdf:id="Řídí">
```

```
    <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:range rdf:resource="#DopravniProstredek"/>
</rdf:Description>
</rdf:RDF>
```

Výpis 4 - Příklad RDFS

Na výpisu 4 je velmi jednoduché RDF Schema definující třídu DopravniProstredek, od které jsou následně odvozeny třídy Auto a Loď. Dále je zde predikát „Řídí“, jenž je pomocí rdfs:domain omezen na zdroje typu http://xmlns.com/foaf/0.1/Person, což je třída definující nějakou osobu. Dále je tento predikát omezen pomocí rdfs:range pouze na instance typu DopravniProstredek.

RDF dokument, který bude používat toto schéma, může obsahovat například tyto trojice:

- Audi, rdf:type, nas_namespace:DopravniProstredek
- Jirka, rdf:type, http://xmlns.com/foaf/0.1/Person
- Jirka, nas_namespace:Řídí, Audi

2.8 Shrnutí

Pro sémantický web jsou položeny základní technologické stavební bloky. Nyní je důležité, aby se co nejvíce dat publikovalo pomocí těchto technologií. Již existují velké veřejné projekty, které se o toto snaží, například http://dbpedia.org . V tom je potřeba pokračovat a data tímto způsobem zveřejňovat. Jedním ze způsobů, jak data získávat je extrahovat je z webu, o čemž pojednává zbytek práce.

3 Analýza

3.1 Definice problému

Informace, s kterými chceme pracovat, jsou roztroušeny po webových stránkách. Ty jsou navrženy tak, aby šly pohodlně číst přes webové prohlížeče ovládané lidmi. Každá taková stránka informace prezentuje nějakým svým vlastním způsobem. Používají nejrůznější rozvržení, grafiku, atd. Není jednotný způsob, jakým lze programově přistupovat k webovým stránkám a získat z nich požadované informace. Pro naše potřeby je však nutné mít požadované informace z webových stránek dostupné ve formě, která bude jednoduše a rychle zpracovatelná strojem.

3.2 Web scraping – získávaní dat z webových stránek

Web scraping [18] je jako metoda získávání dat sice řešení neelegantní, ale pokud není jiné možnosti, jak informace získat, je nevyhnutelná. V dokonalém světě, potažmo světě sémantického webu, by nebylo potřeba, ale nyní je nutností.

Jsou tři základní fáze, ze kterých se Web scraping skládá:

- Navigace
- Extrakce dat
- Složení výsledku do požadovaného formátu

Navigace spočívá v prolézání webových stránek za cílem získání konkrétních stránek html, z kterých se data budou získávat. Z jedné stránky se často získá pouze část nějaké informace. Všechny takovéto fragmenty výsledku je dále nutné poskládat do požadovaného formátu.

Vhodné je si celý web scraping rozdělit ještě podle techniky, jenž se bude aplikovat při datové extrakci. Jsou tři základní kategorie, každá z nich má své pro a proti a každá se hodí na jiný scénář.

První a pravděpodobně nejčastější metodou, jak data získávat, je vytvořit si nějaký prográmek či skript, který využije sílu regulárních výrazů. Tyto výrazy se ukázaly být pro extrahování dat velmi vhodnými z důvodu jejich obecnosti, vyjadřovací síly a přítomnosti jejich implementaci ve většině programovacích jazyků. Pokud je například potřeba z nějaké

stránky, která obsahuje seznam novinek, vytvořit rss soubor a přístup k její databázi není k dispozici. Vytvoří se regulární výraz, který se aplikuje na každou takovouto novinku, a získá z ní titulek, odkaz na článek, anotaci, datum, atd. Pro úlohu tohoto typu se těžko hledá jednodušší a lepší způsob. Jestliže je však potřeba získávat data z většího množství stránek, bude nutné zápolit i s dalšími body web scrapingu - navigací a skládáním výsledku. Takto mohou vzniknout nepřehledné, často velmi pomalé programy/skripty specializované právě na tento jeden konkrétní web. Údržba většího množství takových programů může byt nepříjemná.

Další mnohem vyspělejší metoda využívá sofistikovanějších technik jako je umělá inteligence, ontologie či hierarchické slovníky. Takovéto řešení může inteligentně analyzovat obsah html stránek a vybírat zajímavé informace. Často bude také neprůstřelné vůči změnám na webové stránce a půjde použít na velkou množinu případů. Implementace takovéto metody není jednoduchá a i pokud by byla zakoupena od nějaké společnosti, bude potřeba ji s vysokými nároky na znalosti přizpůsobit. Dalším problémem je, že pravděpodobně bude nutné se opět vypořádat s dalšími fázemi web scrapingu, totiž navigací a manipulací s výsledkem.

Posledním z možných přístupů je pořízení si již hotové aplikace, která je specializovaná právě na web scraping. Takovéto aplikace umožňují celý proces vytvoření a následného používání nějakého extraktoru dat velmi urychlit. Rovněž je k jejich ovládání možné vyškolit i trochu zkušenější uživatele pc. Na trhu je těchto aplikací celá řada (viz kapitola 6 Jiné aplikace), každá s jiným přístupem, ale díky svojí univerzálnosti se na velkou množinu případů hodí. Pro větší projekty můžou být opravdu dobrou volbou. Samozřejmostí je, že pořízení takovéto aplikace může být finančně nákladné a je nutné počítat i s časem, který bude potřeba na získání znalostí potřebných k jejich obsluze, případně k tomu jak je zaintegrovat do stávajícího systému.

3.3 Analýza systému

Před samotným návrhem systému byly napsány aplikace, které přímo bez jakýchkoliv dalších nástrojů získávaly data z nějaké konkrétní webové stránky. Po provedení analýzy těchto aplikací se ukázalo, že se v kódu objevovala velká množina úkolů, které se neustále opakují. Z tohoto důvodu bylo usouzeno, že dobrým přístupem k řešení problému bude

vytvoření komponent, které budou tyto opakující se úkoly vykonávat. Propojením těchto komponent tak vznikne celkový popis úlohy, která má být na webové stránce provedena.

Komponentou se zde rozumí objekt, který na vstupu získá nějaká data (HTML dokument, kolekci dat,…), něco s ním provede a výsledek pošle na výstup. Bude zde omezení, aby takováto komponenta mohla mít pouze jeden vstup a více výstupů, protože vstupní data budou často jedna. Naopak na výstup by mělo jít navázat více komponent a rovněž by zde měla být možnost odlišit různé druhy výstupů z komponenty. Příkladem může být komponenta, která na vstupu získá HTML stránku se seznamem tříd a jmény žáků pod číslem třídy. Na jeden výstup pak bude zasílat čísla tříd a na druhý jména žáků. Pro různé druhy výstupů se v textu bude používat slovo delegát (důvodem je jejich implementace přes .NET delegáty, více na[13]).

Popsat úlohu by mělo být snadné a její zápis přehledný a dobře upravovatelný, jelikož se stránky mění a požadavky, co z nich získávat také. Dále je vhodné, aby uživatelova práce byla z velké části omezena na deklarativní popisování úlohy, tedy na propojování jednotlivých komponent. Jako vhodný kandidát pro takovýto zápis je XML, díky své široké podpoře a možnosti nad ním vybudovat editační nástroje. Vnoření XML elementů, které zastupují komponenty, bude vyjadřovat, že komponenta zastoupená vnořeným elementem získává vstup od komponenty rodičovské.

Může se stát, že komponenta bude umět z velké části to, co je potřeba, ale bude po ní vyžadováno trošku jiné chování. V takovém případě je zbytečné vytvářet novou komponentu podobné té existující, která by měla téměř identický kód, ale pouze určité části by byly odlišné. Kvůli tomu bude možné do komponent na tyto místa, která budou pravděpodobně často měněna, přidat události a na ně programově reagovat v přiloženém zdrojovém souboru. Ten bude součástí popisu úlohy, jeden XML dokument s deklarativním popisem a případný druhý dokument s reakcemi na tyto události. Pokud je předem známá i množina změn, které může uživatel vyžadovat, je možné chování komponenty ovlivňovat vlastnostmi. Jak vlastnosti, tak události budou v XML zápisu reprezentovány atributy elementu, který zastupuje tuto komponentu.

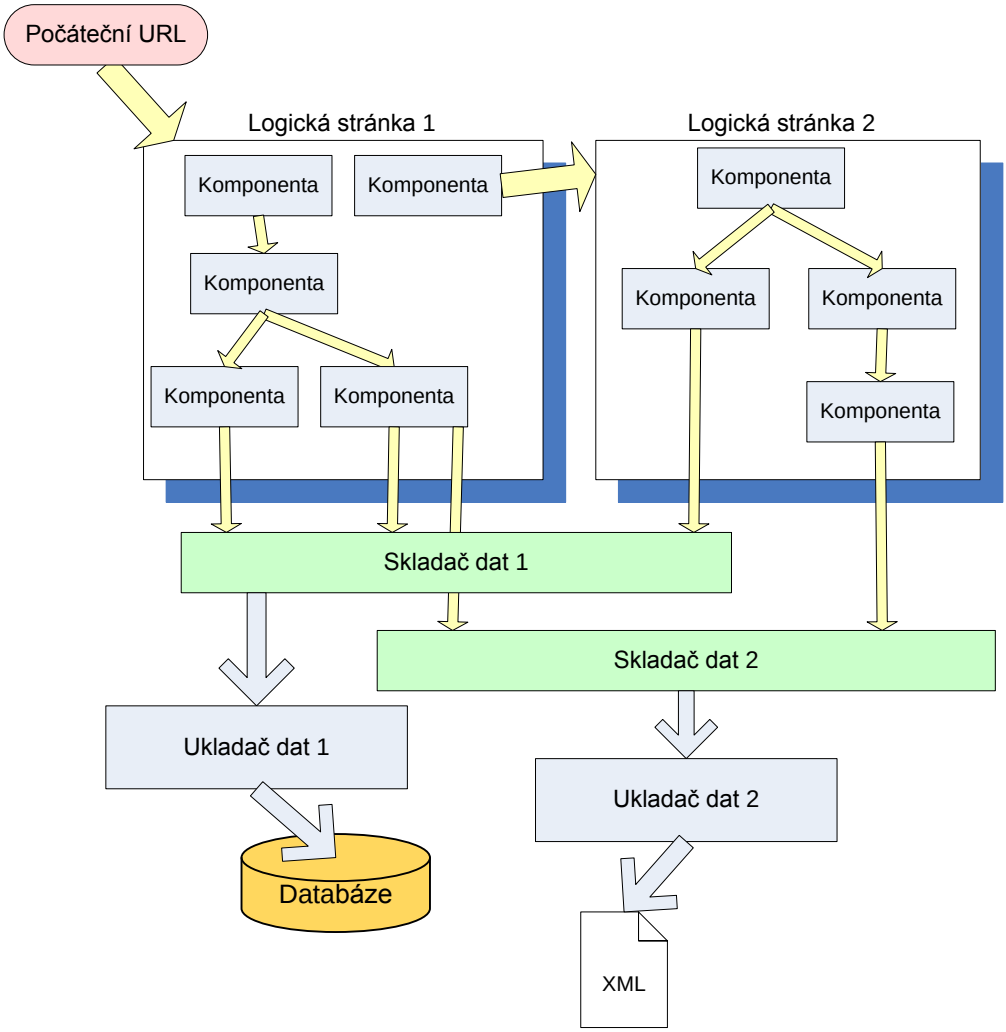
V systému budou zabudovány standardní komponenty, které budou využívat pro zpracovávání HTML stránky výrazy. Užitečné se opět ukázaly regulární výrazy kvůli jejich obecnosti a vyjadřovací síle. Na některé úlohy jsou však až zbytečně příliš mocné a hlavně nepřehledné. Proto do systému bude implementován i jiný druh výrazu zvaný extractor pattern (viz 4.5 Extractor pattern), jelikož na většinu úloh stačí jednoduché vyjadřovací schopnosti a popis toho, co je nutné ze stránky získat.

Celkový popis úlohy by mohl být popsán jedním velikým propojením komponent, to by však bylo nepřehledné. Proto bude tento popis rozdělen na tzv. logické stránky, které budou sdružením komponent, jenž se budou aplikovat na určitou HTML stránku. Mezi těmito logickými stránkami budou vést odkazy. Z obsahu stránky se pomocí komponent získá odkaz na jinou HTML stránku a zavolá se na něj nějaká logická stránka. Tím se zároveň řeší problém navigace, tedy způsob jak projít strukturu nějaké webové stránky. Proto zde jediná přímo zadaná URL bude ta počáteční. Dále bude navigace spočívat v následování odkazů získaných z obsahu a jejich zpracováváním logickými stránkami.

Struktura webových stránek (to jak jsou jednotlivé HTML stránky propojeny) může být často podobná, jen se odlišují prvky na stránce. Z pozorování se například ukázalo, že je mnoho stránek, které vypadají úplně jinak, ale mají data zobrazeny v nějakém seznamu, v kterém se listuje pomocí tlačítek Předchozí a Další. Proto bude vhodné popis úlohy oddělit od výrazů. Jednak kvůli přehlednosti výsledného XML souboru a dalším důvodem je, že pro novou stránku, z které bude nutné získat data, často půjde zkopírovat tento popis a změnit pouze výrazy.

Dále je potřeba vyřešit složení z komponent získaných dat, jelikož informace o nějaké věci může být roztroušena po více stránkách. Takto roztroušenou informaci je nutné složit, aby šla uložit jako jeden záznam, ať už řádek tabulky či cokoliv jiného. Na to bude speciální komponenta (viz 4.3.2 DataFeeders), která bude používána jako poslední v řetězci volání komponent. Data si bude průběžně ukládat do nějaké cache a až bude mít vše potřebné, záznam zašle k uložení.

Data, která již byla poskládána, je nutné někam uložit. K tomuto účelu bude v systému rozhraní (viz 4.9 IDataSaver), které si může uživatel systému zahrnout do nějaké své třídy nebo využít existující implementace, která bude ukládat data do paměti jako tabulky. S těmi bude možné po dokončení stahování pracovat a případně si z nich data uložit na libovolné místo. Je možné, že bude potřeba data rovnou v průběhu stahování ukládat do finálního datového úložiště, například z důvodu nedostatku paměti pro velikou úlohu. V takovém případě si uživatel vytvoří vlastní implementaci třídy pro ukládání dat.



Obrázek 4- Koncepce rozvržení systému v příkladu

Dalším požadavkem na systém je rychlost, které se nejlépe dosáhne využitím paralelismu. K tomu je potřeba celkovou úlohu rozdělit na menší podúlohy, které mohou být vykonávány paralelně. Nejpomalejším procesem v systému je komunikace s HTTP serverem, proto je nejlepší úlohy dělit právě zde. To už ale bylo učiněno rozdělením propojení komponent do logických stránek, které se v tomto kontextu dají brát jako úlohy, jež budou spouštěny na různých vláknech.

4 Uživatelská dokumentace

4.1 Úvod do systému AgentMat

Systém je navržen pro efektivní extrakci velkého množství dat z webových stránek. Umožňuje extrahovat, jak z jednoduchých statických stránek, tak i ze složitých webových aplikací. Data, které ze stránek získá, můžou být nejrůznějších typů a mohou být propojeny relacemi o různých kardinalitách. Celý systém je napsaný v jazyce C# na platformě .NET 2.0.

AgentMat je vytvořen pro spouštění agentů, což jsou programy, které se specializují na konkrétní webovou stránku (aplikaci), kterou umí projít a získat z ní požadovaná data. V systému AgentMat se práce agenta definuje popisem úlohy do XML souboru. Tento popis v první řadě spočívá v zápisu propojení komponent. Celý systém staví právě na těchto komponentách, kterými se zde rozumí objekt, jenž na vstupu získá nějaká data (HTML dokument, kolekci dat,…), něco s ním provede a výsledek pošle na výstup.

Propojováním příslušných komponent se dosáhne požadované funkčnosti na obecné webové stránce (viz kapitola 3.3 Analýza systému). Data extrahovaná systémem z webových stránek se standardně ukládají do tabulek v paměti, ale je možné je ukládat do čehokoliv jiného. Například do databází, XML souborů… Systém má několik základních komponent, které slouží například pro iteraci skrz elementy získané regulárním výrazem, převod relativní adresy na absolutní, rozdělení HTML dokumentu podle separátoru atd. Uživatel není omezen na množinu těchto zabudovaných komponent a má možnost vytvořit si vlastní komponenty, které mohou vstup zpracovávat jinak. Například přes XPath či jiné druhy výrazů. Jeden takový výraz se ukázal být velmi praktický a je společně s regulárními výrazy podporován některými ze standardních komponent systému. Více o něm v kapitole 4.5 Extractor pattern.

Systém je navržen tak, aby úlohy zpracovával, co nejefektivněji a vhodně je paralelizoval bez toho, aby se o to uživatel musel jakkoliv zajímat.

4.2 Práce se systémem

Systémem AgentMat pracuje s agenty zapsanými v XML souborech. Uživatel vytvoří tento soubor a předá ho systému, který ho přeloží, spustí a následně bude uživateli vracet získaná data. Rovněž je možné agenta uložit jako třídu do DLL knihovny nebo případně přímo vygenerovat zdrojový kód této třídy v některém .NET jazyku.

```
<?xml version="1.0" encoding="utf-8"?>
<Agent Name="FiroTourDownloader" Language="C#" CodeFile="Agent.cs"
Namespace="Agents">
  <DataFeeders>
    <DataFeeder Id="dfTerminy" AutoGenerateColumns="true" Table="Terminy">
...
    </DataFeeder>
    <DataFeeder Id="dfCeny" AutoGenerateColumns="true" />
  </DataFeeders>

  <Constants>
    <Item id="HotelInfo">
      <!—Regularni ci jiny vyraz -->
    </Item>
  </Constants>

  <DownloadMethods>
    <DownloadMethod Name="Stahuj" EntryPage="MainPage" Url="www.neco.cz">
...
    </DownloadMethod>
    <DownloadMethod Name="SmallUpdate">
...
    </DownloadMethod>
  </DownloadMethods>

</Agent>
```

Výpis 5- Hlavní kostra XML dokumentu pro popis agenta

Na Výpis 5 je zobrazen příklad XML dokumentu, který bude popisovat agenta pro extrakci dat z nějaké webové stránky. Jsou zde zobrazeny hlavní XML elementy, které jsou rozebrány dále v textu. Formální specifikace tohoto dokumentu a jeho XSD schéma je přiloženo na konci textu.

4.3 Hlavní XML elementy dokumentu

4.3.1 Agent

Agent je kořenový element dokumentu, kde se specifikují věci jako je jméno agenta, popis k čemu slouží atd. Podle jména uvedeného v atributu Name se v případě vygenerování agenta do třídy v DLL knihovně nebo zdrojového souboru, pojmenuje výsledná třída. Atribut Language udává jazyk, v kterém bude výsledný soubor vygenerován. Aktuálně jsou povoleny tyto jazyky:

- CSharp - C#
- Visual Basic - VB

V atributu CodeFile se zadá jméno souboru, kde je možné reagovat na události z hlavního XML souboru. Tento soubor musí být ve stejném jazyku, jaký je zapsán v atributu Language. Jedná se o stejný princip jako je v ASP.NET, kde je soubor aspx s HTML kódem a komponentami a potom CodeBeside (příp. CodeBehind) soubor, který reaguje na události z hlavního aspx souboru.

V atributu Namespace se uvádí název jmenného prostoru, do kterého bude třída agenta v případě generování do DLL knihovny či zdrojového souboru vygenerována.

4.3.2 DataFeeders

DataFeeder je speciální komponenta systému, která se stará o skládání dat do požadovaného formátu. Data získává od komponent, jak je naznačeno na Obrázek 4, kde je označována jako skladač dat. Elementy DataFeeder v XML souboru jsou určeny k popisu dat, které je potřeba extrahováním získávat, tak aby je tato komponenta mohla správně složit a předat dále k uložení do třídy implementující rozhraní IDataSaver (Na Obrázek 4 je znázorněno jako ukladač dat. Popsán je v kapitole 4.9 Rozhraní IDataSaver, prozatím si ho stačí představovat jako paměťovou tabulku). Každá komponenta DataFeeder musí mít přiřazenou jedna třídu IDataSaver se jménem, který je uvedený v atributu Name, aby jí data mohla posílána k uložení.

Všechny XML elementy DataFeeder musí mít v rámci souboru unikátní id, podle kterého se na ně bude odkazovat.

Dále je zde atribut AutoGenerateColumns. Ten říká, že se schéma instance třídy IDataSaver může měnit podle toho, jaká data jsou získána. Tedy, že sloupečky mohou být přidávány dle potřeby. Pokud je zde false, což je defaultní nastavení, ukládají se data pouze do sloupečků, jenž jsou vyjmenovány prostřednictvím elementů Column (viz dále).

DataFeeder obsahuje i atributy pro události ParseCell a ParseRow. ParseCell se volá pro každý jednotlivý prvek, který se bude ukládat a ParseRow až když jsou všechny prvky připraveny na uložení. Definice funkcí, které mají na události reagovat, je nutné zapsat do souboru uvedeného v atributu CodeFile v elementu Agent a do těchto atributů napsat jejich jméno.

Element DataFeeder může být rovněž rodičovským elementem pro element Columns, v kterém prvky Column definují schéma pro IDataSaver.

```
<DataFeeder Id="dfTerminy" AutoGenerateColumns="true"
ParseCell="JmenoMetody" Table="Terminy">
  <Columns>
    <Column name="Name" type="System.String" expression=""/>
    <Column name="Date" type="System.DateTime" expression=""/>
    <Column name="Price" type="System.Int32" expression=""/>
    <Column name="AgencyID" type="System.Int32" expression="21"/>
  </Columns>
</DataFeeder>
```

Výpis 6 - Příklad definice schématu pro komponentu DataFeeder

Do instance IDataSaver spojené s komponentou DataFeeder budou přidány sloupečky s názvy zapsanými v atributech name, které budou typu type. Atribut expression určuje nějaký výraz, jaký závisí na implementaci IDataSaver. Pro standardní implementaci IDataSaver, která vše ukládá do paměťové tabulky DataTable z ADO.NET je popis tohoto výrazu na [14].

4.3.3 Constants

Element Constants je místo, kam se zapisují všechny výrazy či jiné dlouhé řetězce, aby byly odděleny od samotného popisu úlohy.

Elementy Item nadřazeného elementu Constants definují řetězcové konstanty (regulární výrazy, Xpath,…), které bude možné přiřadit atributům jednotlivých komponent. Hlavním důvodem pro oddělení je přehlednost, jelikož můžou být některé výrazy či jiné řetězce dlouhé. Je zde také možnost, že budou použity vícekrát.

Každý takový element musí být jednoznačně identifikován unikátním jménem zapsaným v atributu Id.

```
<Constants>
  <Item Id="SwitureSignature">
   <![CDATA[<strong\\sclass=\"nazev-
zajezdu\"><a\\shref=\"(?<Link>[^\"]*)\">]]>
  </ Item >
</Constants>
```

Výpis 7- Příklad bloku Constants

V tomto příkladu jsou znaková data regulárního výrazu oddělena značkami <![CDATA[a]]>, to není nutné pokud by regulární výraz neobsahoval nepovolené symboly, či by je nahradil odpovídajícími odkazy na entity v xml, které jsou vloženy mezi symboly & ;. Zde je seznam příslušných symbolů a odpovídajících entit.

<	lt
>	gt
&	amp
‘	apos
“	quot

Tabulka 1- Symboly a odpovídající entity

To na přehlednosti regulárního výrazu však nepřidá, proto je doporučeno používat raději CDATA.

K použití u komponent stačí do atributů místo hodnoty zapsat @jméno_konstanty (viz příklad v kapitole 4.3.5 Logická stránka - Page)

4.3.4 DownloadMethods

Těmito elementy se definují metody, které bude agent poskytovat. Každý agent může mít nula až neomezeně metod. Tyto metody jsou určeny k rozlišení různých stahovacích úloh. Agent může například poskytovat metodu FullDownload, která získá všechny data z webové stránky a metodu Update, která získá pouze nejnovější data. Uživatel si potom při spouštění agenta vybere, jaká metoda má být použita.

V elementu DownloadMethod může být jeden až neomezeně mnoho elementů Page, které reprezentují logické stránky. Tedy kolekci propojených komponent, které budou

aplikovány na dokument získaný z nějaké URL. V atributu EntryPage elementu DownloadMethod se určuje, která logická stránka bude aplikována na počáteční URL zadané ve stejnojmenném atributu elementu DownloadMethod. Dalším atributem je Name, podle kterého se výsledná metoda pojmenuje.

4.3.5 Logická stránka - Page

Logické stránky jsou v systému proto, aby zaobalovaly komponenty, které mají být aplikovány na jednu HMTL stránku, tj. nějaké URL. Logické stránce se předá URL a ona komponentám v ní obsažených zašle k zpracování HTML dokument. URL se těmto stránkám dá zaslat buď počáteční při spouštění nějaké stahovací metody DownloadMethod nebo direktivou Link (viz 4.7.3 Link), kde se URL získá z obsahu HTML stránky zpracovaném nějakou komponentou. V XML dokumentu zastupuje logickou stránku element Page. Více o důvodu zavedení logických stránek je popsáno v kapitole 3.3 Analýza systému.

```
<Page id="MainPage">
  <ElementIterator id="eiFirstPage" RegExp="@FirstPage">
    <Link To="ListPage"/>
  </ElementIterator>
</Page>
```

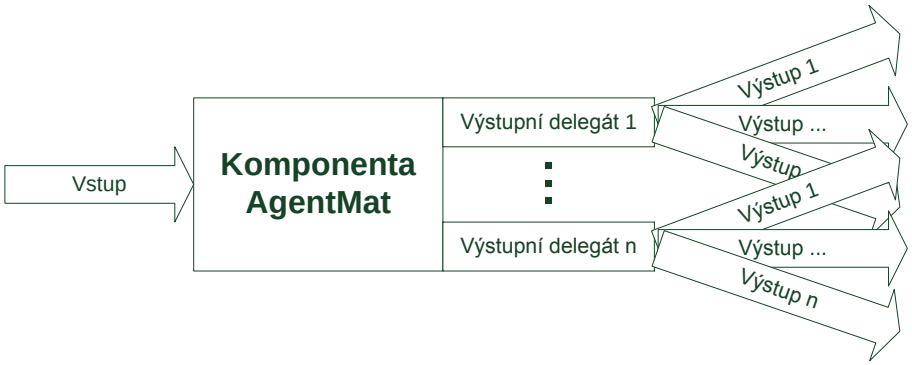
Výpis 8 - Příklad logické stránky

Kus kódu z Výpis 8 ukazuje, jak může taková logická stránka v XML dokumentu vypadat. Tato logická stránka posílá data pouze komponentě ElementIterator (viz 4.7.1 ElementIterator), která na stránku aplikuje nějaký (například regulární) výraz a každý element, který vyhovuje, bude zpracován (vyberou se z něj zajímavá data, to jsou v regulárním výrazu kusy označené jako pojmenované skupiny) a data z něj budou poslány na výstup. V tomto případě budou data poslána direktivě Link, která si zkusí v kolekci výsledků od komponenty ElementIterator najít nějakou proměnnou s názvem Link a její obsah použije jako URL, kterou nechá otevřít logickou stránkou s identifikátorem ListPage. Nyní tedy musí být ve stejné DownloadMethod definována i Page s id=”ListPage”.

4.4 Komponenty

Základem systému AgentMat jsou komponenty. Komponenta je objekt, který na vstupu dostává nějaká data (html dokument, kolekci dat), s těmi něco provede a výsledek pošle na výstup. Propojováním těchto komponent se popisuje řešení úlohy. Tyto komponenty můžou mít rovněž různé vlastnosti, které se v xml zapisují pomocí atributů. Nebo mohou definovat události, na které se dá programově reagovat v souboru, jehož jméno je zapsané v atributu CodeFile elementu Agent.

Každá komponenta může mít různé druhy výstupů, které se označují jako delegáti či výstupní delegáti. Na jeden takovýto výstupní delegát může být připojeno libovolné množství jiných komponent. Naproti tomu musí komponenta získávat vstup právě od jedné komponenty nebo být v nějaké logické stránce (důvod je rozebrán v kapitole 3.3 Analýza systému). Příkladem komponenty s více druhy výstupů je systémová komponenta Spilter, která rozděluje vstup podle nějakého výrazu. Tato komponenta má dva druhy výstupů, jeden delegát je SeparatorParsed, na který se posílají všechny vyhovující oddělovače, a Splited, na který jsou zaslány všechny oddělené kusy web stránky.



Obrázek 5 - Schématické znázornění AgentMat komponenty

V XML zápisu se komponenty vnořují do elementu Page, který vyjadřuje logickou stránku. Další vnořování elementů, které zastupují komponenty, vyjadřuje, že komponenta zastoupená vnořeným elementem získává vstup od komponenty nadřazené. Tato nadřazená komponenta posílá data přes nějaký výstupní delegát. Nejčastěji má komponenta pouze jeden výstupní delegát, proto se implicitně použije tento. Pokud má komponenta více výstupních delegátů, které chce uživatel využít, je nutné je rozlišit explicitě, což se dělá použitím XML

elementu Delegate. Element Delegate se vnořuje hned do elementu komponenty a do něj se následně vnořují komponenty, které mají získat vstup skrz tento delegát.

```
<Spliter id="Spliter">
  <Delegate Name="SeparatorParsed">
    ...
  </Delegate>
  <Delegate Name="Splited">
    ...
  </Delegate>
</Spliter>
```

Výpis 9- Příklad explicitního rozlišení delegátů

Tento explicitní způsob se dá použít i pokud má komponenta jeden delegát, nicméně to není nutné. Pokud má více delegátů, rovněž není nutností explicitně určit, který má být použit, protože jeden z nich je nastaven jako defaultní a ten by byl v takovém případě použit. V tomto případě je v rámci přehlednosti doporučeno tyto delegáty rozlišit a raději je explicitně určit.

4.5 Extractor pattern

Některé AgentMat komponenty umějí mimo regulárních výrazů pracovat i s tzv. extractor patterny. Jde o výrazy, které jsou na rozdíl od regulárních velmi jednoduché k používání, ale nemají tolik možností.

Takovýto výraz se skládá z kusu textu (nejčastěji kusu nějaké HTML stránky) a speciálních tokenů oddělených od ostatního textu značkami ~@ a @~. Těmito tokeny se označují místa, jejichž obsah má být extrahován. Ostatní text ve výrazu je zde proto, aby šlo tato místa zájmu najít.

Vše bude srozumitelnější na příkladu. Na HTML stránce se vykytuje takovýto text:

```
<strong class="nazev-zajezdu">Cesta kolem světa za 80 dní</strong>
```

Výpis 10- kus html kódu

Pomocí extractor patternu je potřeba z této HTML stránky získat jméno zájezdu. Výraz bude vypadat následovně.

```
<strong class="nazev-zajezdu">~@Nazev_zajezdu@~</strong>
```

Výpis 11 - Příklad extractor patternu

Výsledkem extrakce je, že se v proměnné Nazev_zajezdu nachází hodnota, která byla na HTML stránce.

Je zde jeden speciální token ~@IGNORE@~, kterým se určí části textu, jenž nejsou zajímavé a mohou se měnit. Například pokud má být ignorována barva fontu apod.

Pro tvorbu těchto výrazů byla jako součást systému vytvořena aplikace, kde je možné si výraz vyzkoušet.

4.6 Regulární výrazy

Při psaní regulárního výrazu, který má být použit komponentami systému AgentMat, je nutné označit místa, které budou z textu extrahována, pomocí skupin. S těmi pak systém pracuje a kolekce jejich hodnot posílá jako výstup z komponent, například z ElementIterator (viz kapitola 4.7.1 ElementIterator).

Pokud by bylo potřeba extractor pattern z Výpis 11 zapsat jako regulární výraz, vypadal by následovně.

```
<strong\sclass=\"nazev-zajezdu\">(?<Nazev_zajezdu>[^<]*)</strong>
```

Výpis 12- Příklad regulárního výrazu

4.7 Hlavní komponenty

V této kapitole jsou ve stručnosti popsány některé základní komponenty systému AgentMat. Jsou to ty, s kterými se uživatel setká nejčastěji. Podrobný popis těchto a ostatních komponent včetně jejich členů je možné najít v CHM dokumentaci systému, která se nachází na přiloženém CD.

4.7.1 ElementIterator

Komponenta ElementIterator se používá k iterování přes všechny položky, které vyhovují příslušnému regulárnímu výrazu či extractor patternu. Jako výstup posílá dále získané hodnoty skupin regulárního výrazu nebo hodnoty proměnných, jde-li o extractor pattern.

string Pattern	Vlastnost určující, jaký výraz bude komponenta na svůj vstup aplikovat.
bool UseExtPattern	Nastavuje jestli má být výraz brán jako regulární výraz či jako extractor pattern.
int MaxElements	Maximální počet elementů, které je možné poslat na výstup.

Tabulka 2-Vlastnosti komponenty ElementIterator

4.7.2 DataFeeder

Element DataFeeder byl popsán již dříve v kapitole 4.3.2 DataFeeders, kde se používal k definování schématu dat, která chceme získávat z běhu agenta. Použití elementu DataFeeder jako komponenty se chápe jako odkaz na v dokumentu dříve deklarované komponenty DataFeeder, které se takto posílají data k uložení. Používá se vždy až jako poslední v řetězci volání komponent.

string RefId	Identifikátor komponenty DataFeeder, do kterého mají být data z výstupu uložena. Ten musí být deklarován v elementu DataFeeders.
string Method	Jaká metoda se má použít pro ukládání. Povoleny jsou následující dvě hodnoty: • **SetStaticData** Tato metoda si uloží předaná data pouze do tzv. kontextové session a až volání metody SetData uloží najednou tyto data spolu s těmi, co obdržela, do IDataSaver. Důvodem je, že často nelze získat požadovaná data najednou. • **SetData** Volání této metody sloučí data z předchozích volání SetStaticData a data, která tato metoda získala. Výsledná data se uloží najednou do

	IDataSaver, jehož jméno instance bylo nastaveno atributem Name dříve při deklaraci DataFeederu.

Tabulka 3- Vlastnosti komponenty DataFeeder

4.7.3 Link

Tato direktiva vezme ze vstupu hodnotu proměnné s názvem „Link“ (case-sensitiv) jako URL (může být relativní i absolutní), otevře na ní umístěnou HTML stránku a její kód předá logické stránce, jejíž id je v atributu To.

```
<Link To="HotelInfoPage"/>
```

Výpis 13- Direktiva Link

4.7.4 SwitchLink

Tato direktiva je podobná direktivě Link, oproti ní má schopnost se rozhodnout, jaké logické stránce získanou HTML stránku předá. Rozhodnutí činí podle toho, zda tato stránka vyhovuje regulárnímu výrazu určenému v atributu Pattern či nikoliv. Pokud vyhovuje, předá se stránce v atributu Then, pokud ne tak stránce v atributu Else.

```
<SwitchLink Pattern="@HotelList" Then="HotelListPage"
Else="HotelInfoPage"/>
```

Výpis 14 - Direktiva SwitchLink

4.7.5 Spliter

Rozdělí vstup regulárním výrazem či extractor patternem. Pro části stránky vyhovující výrazu, které se vezmou jako separátory, se zavolá delegát SeparatorParsed. Pro zbytky, oddělené separátory se použije delegát Splited. Volání delegátů pro jednotlivé elementy je v tomto pořadí: Splited, SeparatorParsed, Splited, ...

string SeparatorPattern	Výraz, kterým se vstup rozdělí na části.
bool UseExtPattern	Určuje, jestli se SeparatorPattern má brát jako regulární výraz nebo extractor pattern.

Tabulka 4- Vlastnosti komponenty Spliter

4.7.6 UniqIdGenerator

Komponenta generující unikátní id, která na výstup zašle jedinečnou id v proměnné „Id“. To se dá použít nemáme-li jinou možnost, jak data co se ukládají dvěma implementacemi IDataSaver propojit.

4.7.7 ActualUrlGetter

Na výstup pošle aktuální URL adresu v proměnné specifikované v atributu UrlVarName. Pokud není nastaven, bude se proměnná jmenovat ActualUrl.

4.7.8 RelativeUrlResolver

Komponenta, která z relativní URL udělá absolutní. Ze vstupu vezme proměnou se jménem v atributu UrlVariableName (default: Url), která se vezme jako relativní URL, jenž se má převést na absolutní. Jestliže je GetBaseURLFromActualURL rovno TRUE, vezme se jako základní URL, ke které se má relativní vztahovat, adresa aktuálně zpracovávané stránky. Je-li nastavena na FALSE, použije se proměnná ze vstupu se jménem v atributu BaseUrlVariableName (default: BaseUrl).

4.8 Popis rozhraní

Platný dokument AgentMat je potřeba i spustit, aby vykonával činnost, kvůli které byl napsán. Je třeba ho předat systému a naopak ze systému si vybrat data, která agent popsaný v tomto XML z webu získá. Na následujících řádcích je popis, jak toho lze docílit.

Nejdříve je potřeba přidat příslušné jmenné prostory.

```
using AgentMat;
using AgentMat.Xml;
```

Výpis 15 - Jmenné prostory

Následně je nutné vytvořit instanci generátora agentů. Ten umí XML soubor agenta přeložit a spustit metodou GenerateAgent. Případně uložit jako DLL (metoda CompileAgentToDll) nebo vygenerovat pouze zdrojový kód agenta v nějakém z .NET jazyků (metoda GetAgentSourceCode). Každá z těchto metod má ještě analogickou metodu, která se jmenuje stejně, jen za slovem Agent následuje FromFile. Rozdíl je v tom, že první druh bere jako argument obsah souboru XML a druhý cestu k němu.

```
AgentGenerator gen = AgentGenerator.GetGenerator();
DownloadAgentBase agent = gen.GenerateAgentFromFile(pathToXml);
```

Výpis 16 - Příklad použití generátora agentů

Pokud se při kompilaci XML stala nějaká chyba, je možné ji najít v kolekci ErrorCollection. Pokud chyba vedla k tomu, že agent nemohl být přeložen, bude vlastnost gen.Errors.hasError nastavena na TRUE. V opačném případě vše proběhlo bez problémů nebo kolekce obsahuje pouze varování a doporučení. Jestli tedy vše proběhlo v pořádku, nic nebrání v puštění agenta. Aby se spustit extrakce dat, je nutné určit, která DownloadMethod (viz kapitola DownloadMethod) bude použita. Programově je k seznamu přístup přes vlastnost agent.Modes.

Aby bylo možné stahování zahájit, je potřeba vytvořit stahovací kontext DownloadContext, který bude prováděnou úlohu reprezentovat a přes který je možné stahování ovlivňovat. K tomu bude potřeba vytvořit třídu ResultSet, která je kolekcí tříd implementující rozhraní IDataSaver. Pokud nebude naplňana, udělá to systém podle schémat v elementu DataFeeders a jako třídy se použijí MemoryDataSaver, což jsou v podstatě tabulky v paměti. To může být plně postačující, pokud se neextrahují obrovská kvanta dat a postačuje pamět. Pokud nestačí, je nutné si třídu implementující rozhraní IDataSaver udělat, což je popsáno v následující kapitole.

Za předpokladu, že data stačí ukládat do paměti, se vytvoří objekt ResultSet a DownloadContext.

```
result = new ResultSet();

context = AgentManager.CreateDownloadContext( agent,
                                              result,
                                              agent.Modes[“Stahuj”]);
agent.Download(context);
```

Výpis 17 - Příklad vytvoření ResultSet, DownloadContext a spuštění stahování agenta

Poslední řádek z Výpis 17 spustí stahování. Data budou ukládána do tabulek MemoryDataSaver. Z nich je možné je následně vybrat přes vlastnost MemoryDataSaver.DataTable, která vrácí instanci třídy DataTable, což je jedna ze základních tříd ADO.NET [23].

4.9 Rozhraní IDataSaver

Třídy, které implementují toto rozhraní, jsou používány pro ukládání vyextrahovaných dat. Pro každou komponentu DataFeeder (viz kapitola 4.3.2 DataFeeders) se použije jedna třída IDataSaver. Komponenta DataFeeder data složí a pošle je k uložení přes toto rozhraní.

V systému je k dispozici standardní implementace tohoto rozhraní MemoryDataSaver. Je to třída, která veškeré výsledky ukládá do paměťové tabulky ADO.NET DataTable. Pokud je z nějakého důvodu nepostačující, ať už nestačí paměť nebo je se získanými daty nutné pracovat nějakým jiným způsobem, je možné si naimplementovat vlastní třídu.

```
public interface IDataSaver
{
    string Name {get;set;}
    void AddRow(ResultRow row);
    void AddColumn(string columnName,Type type,string expression);
    void AddColumn(string columnName);
    bool ContainsColumn(string columnName);
}
```

Výpis 18 - Rozhraní IDataSaver

První položkou je jméno typu string. Podle shody s ní a atributem Name komponenty DataFeeder, si tato vybere instanci IDataSaver z kolekce ResultSet a bude jí posílat vyextrahovaná data. Name je identifikátorem v kolekci ResultSet, musí být v rámci ní jednoznačný.

Pokud měla komponenta DataFeeder definováno schéma, které má používat, zavolá se na začátku stahování metoda AddColumn s třemi argumenty pro každý definovaný sloupeček. Jestli je nastaveno AutoGenerateColumns, bude se metoda AddColumn volat kdykoliv bude potřeba. Definované schéma a volba AutoGenerateColumns se vzájemně nevylučují.

Aby mohl DataFeeder s novou třídou IDataSaver pracovat musí rovněž implementovat metodu ContainsColumn, která při dotázání odpoví, zda-li je příslušný sloupeček v schématu.

Rovněž je třeba pamatovat na to, že metody jsou volány asynchronně a musí se v nich zabezpečit kritické sekce!

4.10 Logování

Při běhu agentů dochází k různým nepředvídatelným chybám, kdy například server neodpoví do příslušného časového limitu, nebo data nemohly být z nějakého důvodu uloženy apod. Kvůli tomu je v systému zabudována logika logování chyb, aby šlo zjišťovat, co a kdy se stalo.

Vše funguje bez nutnosti cokoliv nastavovat apod. V standardním nastavení používá systém AgentMat logovací knihovnu log4net [9], tu je možné ovlivňovat přidáním konfiguračního souboru log4net.xml. Pokud zde tento soubor nebude systém bude ukládat záznamy do souboru „AgentMatLogFile.txt“, což je pro mnoho situací postačující.

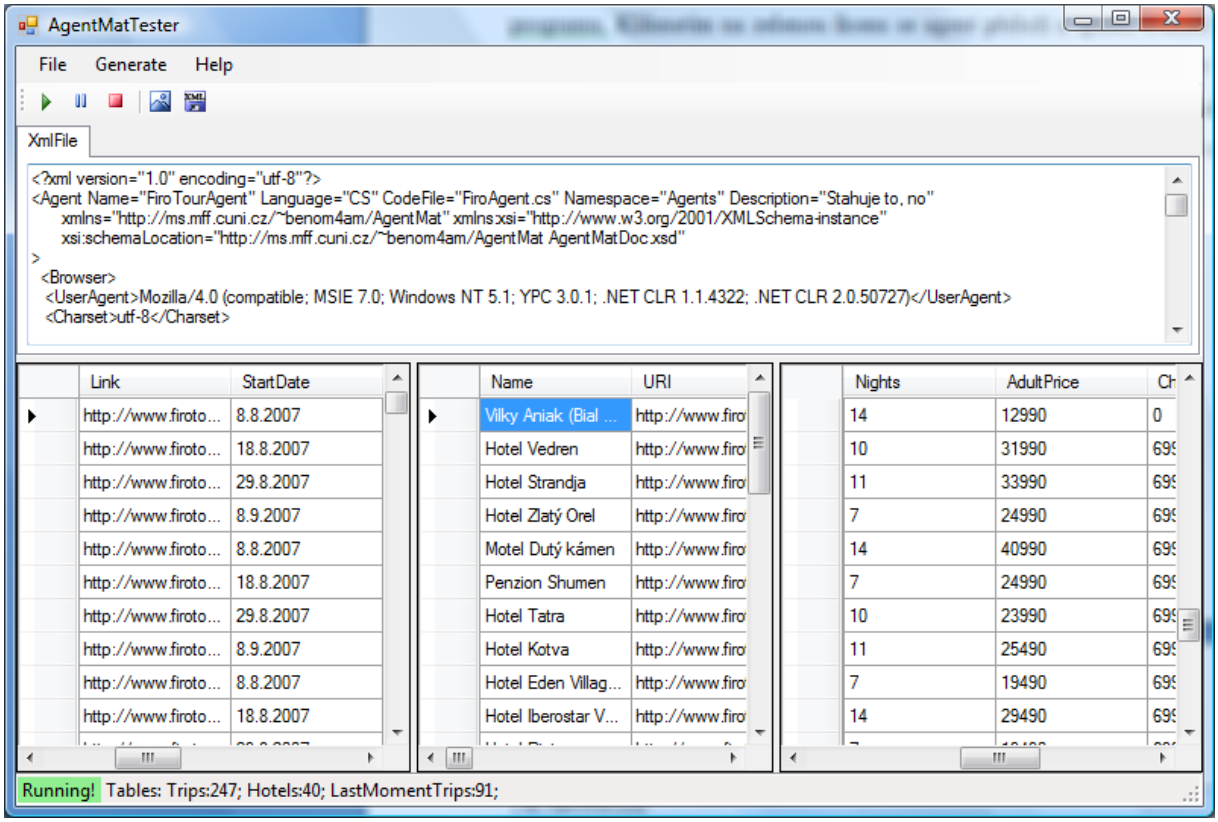
Jestli je potřeba jinou logovací logiku než může knihovna log4net poskytnout, je možné naimplementovat si vlastní. Stačí použít rozhraní v jmenném prostoru AgentMat.Logging.

```
public interface ILoggingProvider
{
        void Write(object message, Exception exception, LoggingMode mode);
        void Write(object message, LoggingMode mode);
}
```

Výpis 19 - Rozhraní ILoggingProvider

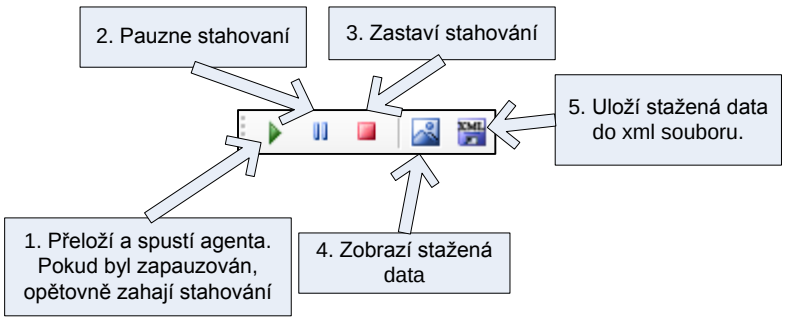
Třídu, která toto rozhraní implementuje, je možné předat vlastnosti LoggingProvider v statické třídě AgentManager. Následně budou všechny záznamy posílány sem.

4.11 AgentMatTester



Obrázek 6 - okno programu AgentMatTester

Součástí práce je aplikace AgentMatTester určená k testování agentů zapsaných v XML. S touto aplikací je možné si vyzkoušet agenta, ještě před jeho nasazením nebo získaná data přímo uložit do XML souboru.



Obrázek 7 - Panel nástrojů programu

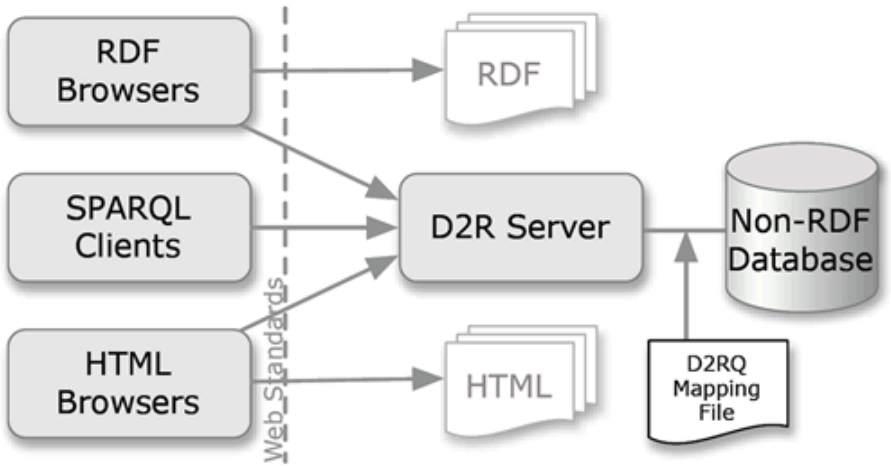
XML soubor s popisem agenta se dá otevřít přes File>Open. Následně se obsah souboru objeví v okně programu. Potom je možné agenta přeložit a spustit. Pokud jej přeložit nešlo nebo se během překladu vyskytla nějaká varování, zobrazí se dialog, kde budou vyjmenovány. Jestliže vše proběhlo v pořádku, je zobrazen dialog s možností výběru stahovacích metod, které agent poskytuje. Po výběru se tato metoda spustí a agent začne vykonávat svou práci.

V průběhu extrakce je možné data zobrazit v tabulce pod obsahem XML souboru, jak je vidět na Obrázek 6. Stahování je možné rovněž pozastavit a následně opět spustit, či úplně zastavit.

V nabídce Generate se dá vygenerovat zdrojový kód v jazyce, jenž je v XML souboru nastavený nebo přeložit agenta do DLL knihovny.

4.12 Získávání sémantických dat

Při použití systému pro extrahování dat, z kterých je potřeba udělat data sémantická, je možné postupovat dvěma způsoby. Prvním je data systémem vyextrahovat a uložit do relační databáze jako běžné tabulky. Následně publikovat pouze pohled, který data bude poskytovat jako sémantická. K tomuto je možné využít nástroj D2R Server, který umožňuje deklarativní mapování mezi schématem databáze a cílovými RDF pojmy. Na základě tohoto mapování vytvoří D2RServer pohled na relační data a rozhraní pro jazyk SPARQL [20].



Obrázek 8 - D2R Server (převzato z [15])

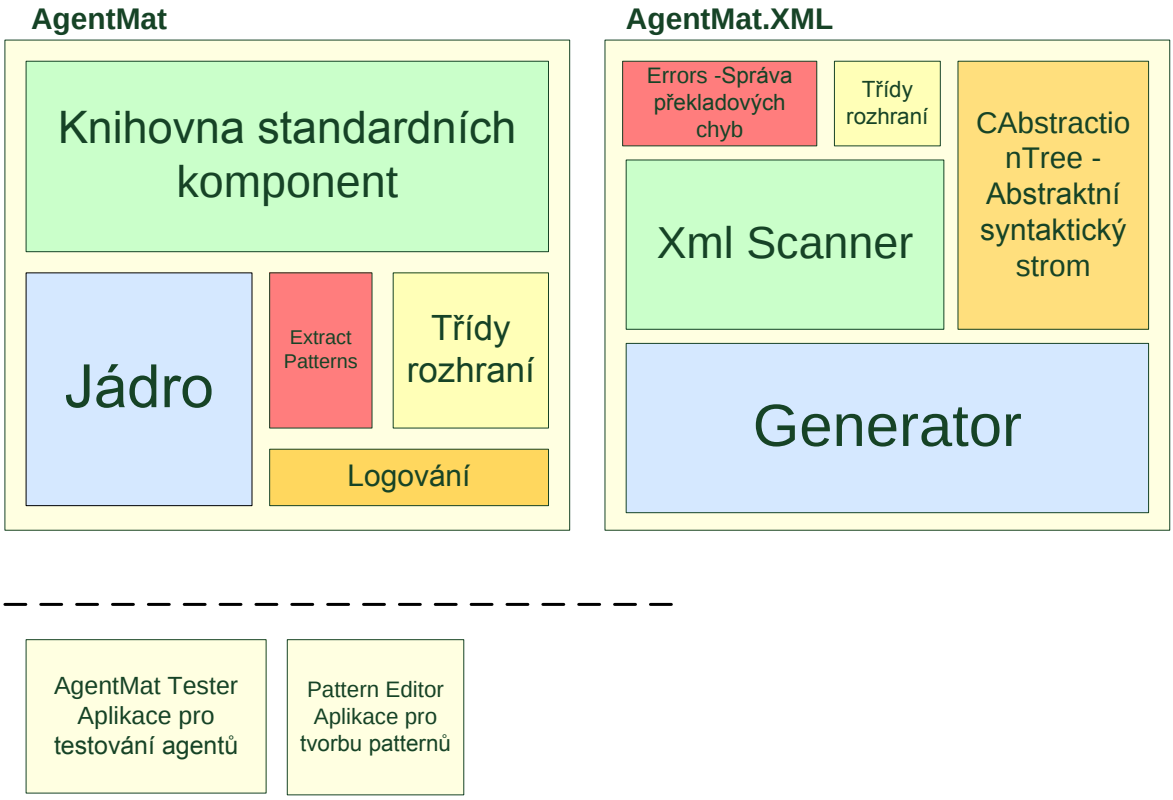
Druhým způsobem je data přímo při stahování ukládat jako sémantická, například do nějakého RDF úložiště (databáze trojic). Zde je možné buď nechat data stáhnout do paměťové

tabulky MemoryDataSaver a nebo si vytvořit vlastní třídu implementující IDataSaver. V obou případech budeme ukládat RDF trojice (podmět, predikát, předmět). Predikáty pro tento účel je nejlepší opět vzít z veřejně známého ontologického slovníku, pokud však žádný vyhovující není k dispozici, bude nutné si vytvořit vlastní (viz 2.6 Ontologické slovníky a 2.7 RDFS).

5 Programátorská dokumentace

5.1 Rozdělení systému

Systém se skládá z dvou hlavních subsystémů, které jsou dále rozděleny na další části. Mimo stojící jsou dvě aplikace, které se systémem souvisí. AgentMatTester pro zkoušení agentů (popsaný v kapitole 4.11 AgentMatTester) a editor výrazů.



Obrázek 9 - schéma architektury systému AgentMat

Části systému, které by uživatel měl znát, byly popsány v uživatelské dokumentaci. V této kapitole budou popsány interní třídy, které jsou pro systém důležité, avšak běžný uživatel systému se o ně nemusí starat.

5.2 Komponenty

Doposud pojem komponenta popisoval takový objekt, který získá vstup, s ním něco provede a výsledek pošle na výstup. Komponenty jsou ve skutečnosti třídami, které vstup

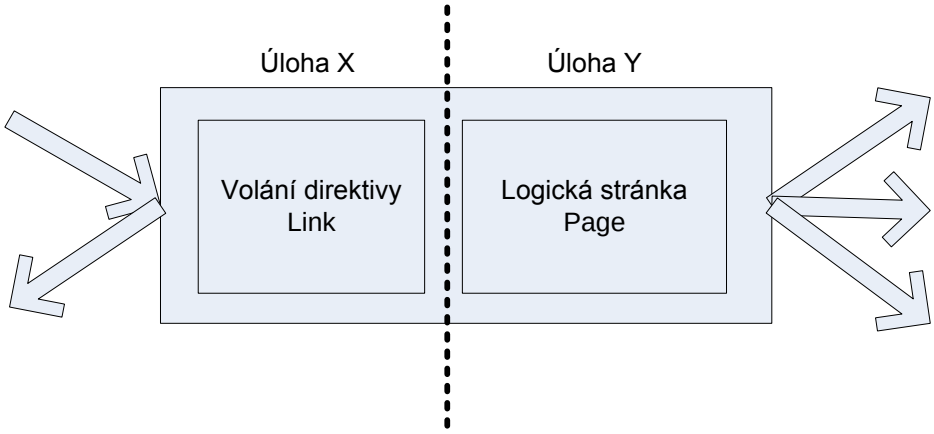
získávají přes jednu svou speciální metodu, která je označena tzv. custom atributem [16]. V této metodě se vstup zpracuje a výstup se posílá připojeným komponentám přes .NET delegáty [13], jak název výstupů v systému napovídá. Na .NET delegát je možné navázat libovolně mnoho metod se stejnými argumenty. Bližší pohled na to, jak komponenty mezi sebou komunikují, je v kapitole 5.4 Třídy pro komunikaci mezi komponentami. Jak konkrétně vypadá třída AgentMat komponenty je vidět v kapitole 5.10 Rozšíření o vlastní komponentu.

5.3 LinkExecutor

Jednou ze základních komponent je třída LinkExecutor. V XML zápisu je však skryta z důvodu zvýšení úrovně abstrakce. Používá se k otevírání URL odkazů na webové stránky. Tam, kde se v XML zápisu objeví direktiva Link a logická stránka Page, je ve skutečnosti komponenta LinkExecutor.

Kvůli efektivnosti se systém snaží zpracovávání úlohy co nejvíce paralelizovat. To se děje separováním celkové úlohy na menší podúlohy, které mohou být vykonávány souběžně. Místem pro tuto separaci je právě komponenta LinkExecutor.

Při volání komponenty LinkExecutor se vytvoří objekt DownloadTask, který si uschová veškeré aktuální kontextové informace a předá se objektu DownloadThreadPool. Běh se ihned z komponenty LinkExecutor vrací. Až objekt DownloadThredPool usoudí, že je vhodné tuto úlohu spustit na nějakém svém vlákně, obnoví kontextové informace z DownloadTask a předá je komponentě LinkExecutor, která otevře požadovanou stránku a pošle její data připojeným komponentám.



Obrázek 10- Schéma práce komponenty LinkExecutor

Jak je vidět z Obrázek 10 práce komponenty LinkExecutor se dělí na 2 části, které běží jako různé úlohy. Jedna část je volána nějakou komponentou při použití direktivy Link. Druhá část je zavolána až na ni dojde řada v objektu DownloadThreadPool a následně volá všechny komponenty, které jsou připojeny k logické stránce, kterou zastupuje.

LinkExecutor je na paralelizaci nejvhodnější, protože většinu času musí systém čekat na odpověď ze serveru a její zpracování ve skutečnosti zabírá jen minimum procesorového času. To nemusí být pravda, pokud je použit nějaký opravdu složitý regulární výraz, který se bude zpracovávat dlouho, ale ten je většinou spíše chybou jeho autora nikoliv úmyslem. Takovou situaci je možné jednoduše identifikovat 100% vytížením procesoru.

Komponenta LinkExecutor se v systému používá ve dvou nastaveních, lišících se od sebe hodnotou ve vlastnosti WaitUntilDown. Ta, jež ji má nastavenou na TRUE, je volána jen na první stahovací úlohu, tedy URL zadanou jako počáteční a běh se z ní navrací až po dokončení všech ostatních podúloh. Druhá, která byla v této kapitole rozebrána, získává URL ze vstupu z proměnné s názvem „Link“ a vytváří úlohu DownloadTask, která je zařazena na zpracování do třídy DownloadThreadPool. Běh se zde vrací ihned.

5.4 Třídy pro komunikaci mezi komponentami

Komponenty mezi sebou komunikují skrz delegáty, kteří volají jednotlivé vstupní metody komponent. Delegáti jsou v systému dvojího typu a liší se dle místa použití. První se používá při otevření URL stránky, kdy ještě nebyl vstup nijak zpracováván. Tedy pro komponenty přímo umístěné v elementu Page (v komponentě LinkExecutor).

```
public delegate void LinkOpenedHandler(IContext context);
```

Výpis 20 - První vstupní delegát

A rozhraní, které se používá po zpracování výstupu.

```
public delegate void ComponentOutputHandler(
                                    IContext Context,
                                    IResultCollection subelements,
                                    string[] GroupNames);
```

Výpis 21 - Delegát určený pro výstup dat z komponent

Při komunikaci mezi komponentami je nejdůležitější rozhraní IContext, jehož implementace Context se používá pro předávání kontextových informací mezi jednotlivými komponentami. Více o problematice kontextu je uvedeno v kapitole 5.5 Kontextová session. Na Výpis 22 je zdrojový kód tohoto rozhraní.

```
public interface IContext
{
        IDataProvider DataProvider {get;}
        ContextSession Session {get;}
        DownloadContext DownloadContext {get;}
}

public interface IDataProvider
{
        string Url {get;}
        string Content {get;}
}

public interface IResultCollection
{
        string this[string key] {get;}
        int Count {get;}
}
```

Výpis 22 - Důležitá rozhraní

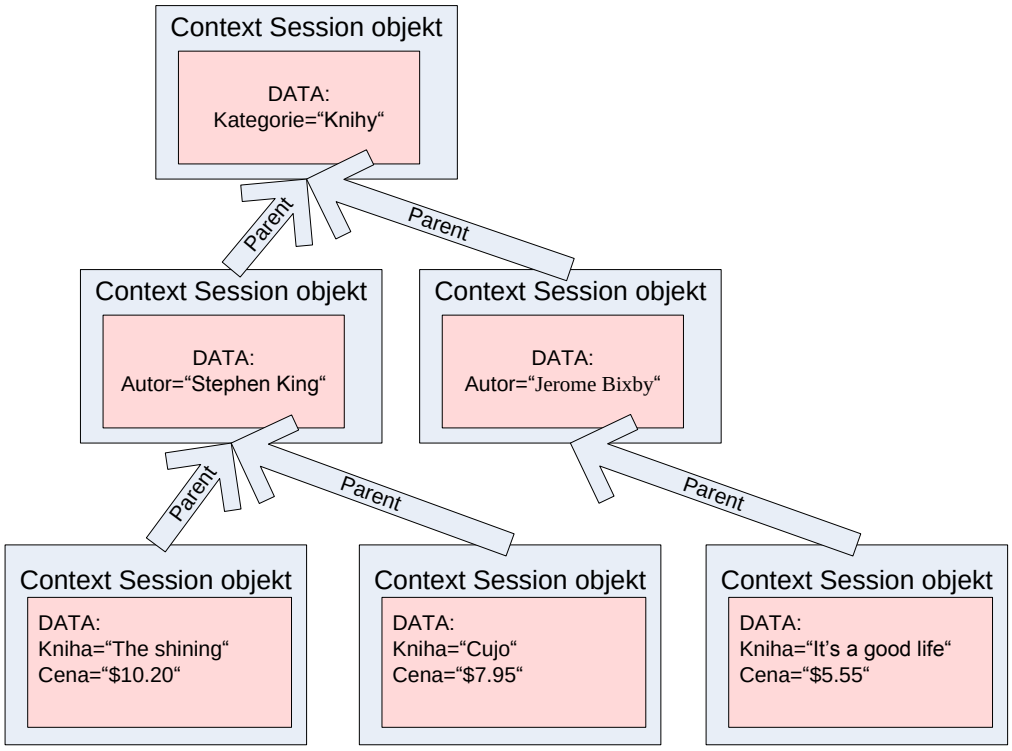
Dalšími důležitými třídami jsou třídy implementující rozhraní IDataProvider, které je určeno k poskytování dat komponentám. Nejčastěji používaná je implementace WebDataProvider, jenž poskytuje data přímo z HTTP serveru a umožňuje s ním i komunikovat metodou GET i POST. Další je třída DataProvider poskytující jakákoliv textová data.

Posledním komunikačním rozhraním je IResultCollection, které umožňuje přistupovat ke kolekci výsledků typu string klíčem typu string. Například komponenta ElementIterator určená pro iterování přes elementy vyhovující regulárnímu výrazu či extractor patternu zde ukládá jednotlivé hodnoty skupin z tohoto výrazu, respektive proměnné extractor patternu. K tomu používá třídu RegexpResultCollection. Dále je v systému k dispozici implementace CustomResultCollection, do které je možné uložit cokoliv nejen výsledek regulárního výrazu.

5.5 Kontextová session

Jelikož je AgentMat systém, který běží multivláknově, je potřeba nějakým mechanismem zajistit předávání informací mezi komponenty, tak jakoby vše bylo vykonáváno na jediném vlákně. Obrázek 11 ukazuje, jak vypadá kontextová session, která se používá k uchování dat ve stejném kontextu. Tím se míní, aby metody volané na jiných vláknech měly přístup k datům, na kterých jsou závislé. To umožňuje jakékoliv komponentě uložit si do této session různé informace.

Komponentou, která ContextSession používá je DataFeeder. DataFeeder si do ní ukládá data, která nejsou kompletní. Po získání zbylých dat si z ContextSession vyzvedne data patřící aktuálnímu kontextu a pošle je k uložení. Konkrétně to funguje následovně. Vždy když se zavolá metoda SetStaticData, uloží si do ContextSession data z této metody. Až při volání SetData, které může být na jiném vlákně, se data z volání metod SetStaticData a data ze SetData pošlou jako celek k uložení do třídy implementující IDataSaver.



Obrázek 11 - Znázornění fungování ContextSession

Obrázek 11 naznačuje, jak kontextová session pracuje. Objekt ContextSession obsahuje Dictionary<string,string>, kde si uchovává data a referenci na předchozí instanci ContextSession, ke které má přístup omezený pouze na čtení. Uvolňování místa

naalokovaného objekty ContextSession probíhá tak, že po zpracování aktuální úlohy, která si drží referenci na svůj objekt ContextSession a nemá žádné podúlohy, se reference na tento objekt ztratí a ten je postoupen k likvidaci .NET garbage collectoru[17].

Nové objekty ContextSession se tvoří při volání metody LinkExecutor.Open, jelikož je to místo, kde se vytváří nová úloha DownloadTask. Ta je následně zařazena k zpracování DownloadThreadPoolem. ContextSession, s kterou je volán LinkExecutor.Open, se zkopíruje, aby tato nová úloha měla k dispozici nezměněná data, na která navazuje. Pro tuto novou úlohu je k dispozici nový objekt ContextSession, jenž má odkaz na tento zkopírovaný ContextSession. Tak se zabezpečí, že cokoliv se v nové úloze uloží do kontextové session, nezmění objekty předchozí. Avšak pokud je potřeba data z kontextu vybrat, stačí k nim přistoupit jako k datům konkrétně tohoto objektu přes indexer, ten se postupně pokouší hledat data v řetězci objektů ContextSession.

Běh programu se v LinkExecutor.Open nezastavuje (pokud nemá vlastnost WaitUntilDone nastavenou na TRUE) a po zařazení úlohy k zpracování se hned z metody Open vrací.

5.6 DownloadThreadPool

Třída DownloadThreadPool je motorem systému, asynchronně spouští jednotlivé úlohy DownloadTask, což je třída reprezentující jeden úkol. Jedná se o zaobalující třídu pro standardní WaitCallback delegát [18].

Na začátku si DownloadThreadPool vytvoří určitý počet vláken, kterým potom úlohy přiřazuje. Implementovat vlastní threadpool bylo nutné zejména kvůli povaze kontextové session. Běžný threadpool používá pro řazení úloh frontu. Úlohy v tomto systému vznikají tak, jak se prochází struktura nějaké webové stránky, zpracovávání úloh by tedy postupovalo do šířky. Tak by kontextová session rychle rostla a její objekty by se začaly uvolňovat až při zpracovávání posledních podúloh. Normální threadpool by nebyl kvůli paměťové náročnosti přijatelný.

DownloadThreadPool používá místo fronty zásobník. Poněvadž není důležité pořadí zpracovávání úloh, ale až celý výsledek, je možné zásobník použít. Zaručí se tím to, že nejnovější úlohy budou zpracovány nejdříve. Struktura odkazů webové stránky tedy bude procházena do hloubky. Tak se může kontextová session nepřetržitě uvolňovat.

5.7 Překlad z XML do .NET assembly

O překlad z XML do .NET assembly se starají třídy z jmenného prostoru AgentMat.Xml, kde se nachází i třída AgentGenerator. Té se předává XML dokument a z ní lze získat některou z následujících reprezentací.

- DLL knihovna
- Zdrojový kód v C# či Visual Basic
- Instance třídy agenta

Popis metod pro to určených lze nalézt v CHM dokumentaci na přiloženém cd.

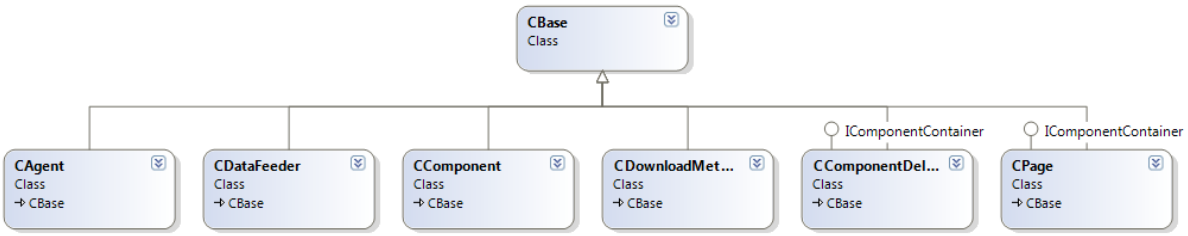
Překlad probíhá v několika fázích. Nejdříve se z XML vytvoří tzv. CAbstractionTree, ten slouží k objektové reprezentaci dokumentu agenta. Tím se zároveň odděluje samotná XML reprezentace od dalšího zpracování, což umožňuje vytvořit další reprezentace v jiných formátech než XML. Stačí vytvořit třídy, které budou převádět nový formát do CAbstractionTree.

Dále se z CAbstractionTree převádí do CodeDom [10], což je jmenný prostor obsahující třídy pro objektovou reprezentaci zdrojového kódu v .NET. Tuto objektovou reprezentaci pak .NET umí přeložit či z ní vygenerovat zdrojový kód v některém .NET jazyku.

5.7.1 Převod z XML do CAbstractionTree

O překlad XML souboru se stará subsystém AgentMat.Xml. Proces probíhá v několika fázích. Nejprve je XML soubor nutné převést do objektové hierarchie, která reprezentuje agenta. Jedná se o abstraktní syntaktický strom CAbstractionTree, s tím je možné tohoto agenta přeložit a spustit. Popis jednotlivých tříd lze nalézt v dokumentaci jmenného prostoru AgentMatXml.CATree.

CAbstractionTree je navržen tak, aby nebyl na XML souboru nijak závislý. Pokud tedy bude pro agenta vymyšlena lepší reprezentace než je XML soubor, stačí, aby se vytvořili třídy pro jeho převod do CAbstractionTree. Ostatní již zařídí třídy ve jmenném prostoru AgentMat.Xml.Generator.



Obrázek 12 - abstraktní syntaktický strom CAbstractionTree

O převod XML souboru do CAbstractionTree se starají třídy ze jmenného prostoru AgentMat.Xml.Scanner. Xml dokument se předá třídě XmlScanner, která skrz něj iteruje a volá pro jednotlivé elementy a atributy metody rozhraní IScannerCreator. Implementací tohoto rozhraní je třída CTreeCreator. Ta je určena pro tvorbu objektové hierarchii CAbstractionTree s počátkem v objektu CAgent.

S vytvořeným CAbstractionTree je možné agenta zkompilovat, spustit či získat jeho zdrojový kód. Na to jsou třídy ze jmenného prostoru AgentMat.Xml.Generator. Hlavní třídou je zde AgentBuilder, kterému se předává reprezentace dokumentu v CAbstractionTree. Ten se dále postará o převod do výše zmíněných reprezentací.

Vnitřně se nejdříve CAbstractionTree převede do CodeDom [10], který již .NET umí přeložit do .net assembly. Pokud je specifikován atribut CodeFile, je třída přeložena jako partial class a spojena s přidruženým zdrojovým souborem. Po zkompilování se získá instance agenta nebo se uloží jako DLL. Případně může .NET vygenerovat pouze zdrojový kód agenta.

Převod do CodeDom probíhá rovněž v několika fázích. V první fázi CAbstractionTree prochází instance třídy CheckVisitor, která zkontroluje, je-li CAbstractionTree v pořádku. Kontroluje vše, co je potřeba, aby se v dalších fázích dalo předpokládat, že je CAbstractionTree plně korektní. Mimo jiné využívá i technologie .NET reflection, aby se ověřilo, jestli jsou AgentMat komponenty či komponenty v jiných přiložených knihovnách správně použity. Pokud je něco v nepořádku, informace o nedostatcích se uloží do ErrorCollection. V třídě rozhraní AgentGenerator je dostupná přes vlastnost Errors. Pokud vše proběhne bez chyb, projde CAbstractionTree visitor GenerateVisitor, který již vygeneruje CodeDom strom, jehož převod do finální podoby zabezpečuje .NET.

5.8 Příklady práce systému

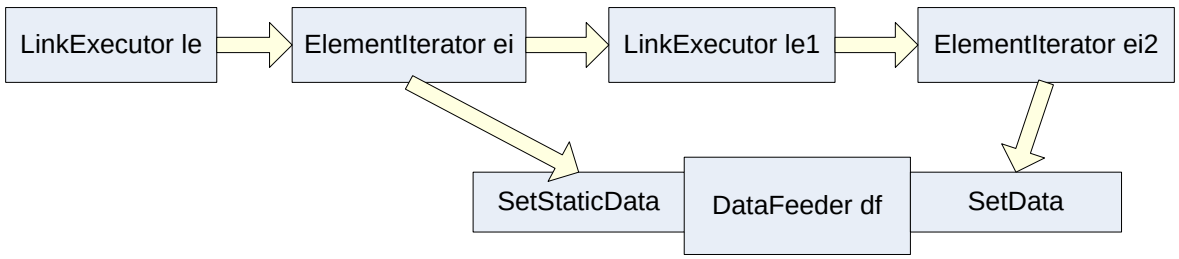
5.8.1 Příklad 1

Tento příklad demonstruje použití komponent LinkExecutor, ElementIterator a DataFeeder.

```
…
<Page id="le1">
  <ElementIterator Pattern="@regvyraz1">
    <DataFeeder RefId="df" Method="SetStaticData"/>
    <Link To="le2">
  </ElementIterator>
</Page>
<Page id="le2">
  <ElementIterator Pattern="@regvyraz2">
    <DataFeeder RefId="df" Method="SetData"/>
  </ElementIterator>
</Page>
```

Výpis 23- Příklad části xml dokumentu agenta

Následující schéma ukazuje, jak ve skutečnosti vypadá propojení jednotlivých komponent tohoto příkladu přes delegáty.



Obrázek 13 - Schéma propojení komponent

```
LinkExecutor le = new LinkExecutor(DownloadContext, true);
LinkExecutor le1 = new LinkExecutor(DownloadContext, false);
ElementIterator ei = new ElementIterator ("regularni vyraz 1");
ElementIterator ei1 = new ElementIterator("regularni vyraz 2");
DataFeeder df = new DataFeeder(dataSaver);
```

```
df.AutoGenerateColumns = true;

le.LinkOpened          +=ei.Parse;
ei.ElementParsed       +=df.SetStaticData;
ei.ElementParsed       +=le1.Open;
le1.LinkOpened         +=ei1.Parse;
ei1.ElementParsed      +=df.SetData;

le.Open("URL");
```

Výpis 24 - Schéma na Obrázek 13 zapsané v kódu

Kód pracuje následovně. Po zavolání metody le.Open s argumentem URL nějaké stránky, která má být zpracována, se vytvoří objekt Context implementující rozhraní IContext. Tento Context obsahuje vše, co je nezbytné znát k provedení úlohy (viz 5.4 Třídy pro komunikaci mezi komponentami). Zavolá se metoda (nebo metody, což není tento případ) připojené na delegát le.LinkOpened.

Objekt Contextu se předá metodě ei.Parse. Tato metoda načte data poskytnutá HTTP serverem z URL adresy nastavené v objektu Context, konkrétně z Context.DataProvider. Tato ze serveru získaná data se zparsují pomocí regulárního výrazu nastaveného konstrukturem třídy ElementIterator. Pro každý element se vezmou všechny skupiny, které regulární výraz definuje, a zavolá se delegát ElementParsed s kolekcí těchto skupin IResultCollection.

Na delegát ElementParsed jsou navázány dvě metody. První je metoda df.SetStaticData, která si data uloží do kontextové session (viz 5.5 Kontextová session). Tyto data budou uloženy do tabulky až při zavolání metody SetData v tomto kontextu.

Další metodou navázanou na delegát ElementParsed je metoda le1.Open . Tato metoda je přetíženou verzí metody Open zavolané na začátku. Na rozdíl od prvního volání, kde se přímo zadávala adresa, se této metodě předá výše zmíněná kolekce IResultCollection, v které musí být položka s klíčem “Link“. Tato položka se vezme jako URL. Pokud tam nebude, vyvolá se výjimka ArgumentException.

Dále se zavolá metoda ei1.Parse napojená na delegát le1.LinkOpened. Ta se chová úplně stejně jako u instance ei, pouze je nainicilizovaná jiným regulárním výrazem. Pro každý element tedy bude zavolána metoda df.SetData.

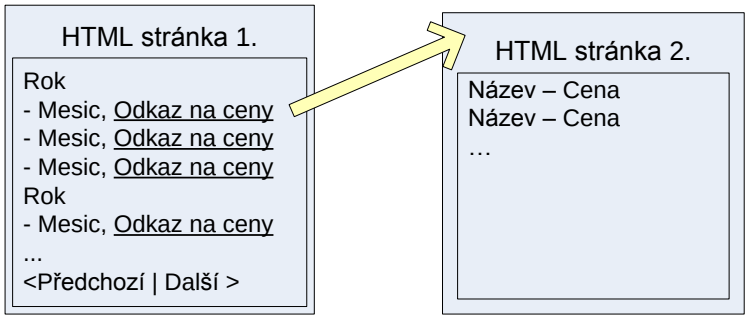
Metoda DataFeeder.SetData si vezme z kontextové sesion případná data, která do ní mohli uložit předchozí volání metody DataFeeder.SetStaticData a data, která jí byla předaná v argumentu. Toto sjednocení dat uloží do příslušných sloupečků instance dataSaver rozhraní IDataSaver. Jelikož má DataFeeder nastaveno AutoGenerateColumns na TRUE budou

všechny nezbytné sloupečky vytvořeny (V ostrém provozu je doporučno mít tento přepínač vypnutý kvůli rychlosti).

5.8.2 Příklad 2

V tomto příkladu bude ukázáno, jak napsat jednu celou DownloadMethod, určenou pro získání dat z nějaké skupiny stránek.

Na následujícím obrázku je naznačeno schéma stránek, které mají být zpracovány, tak aby jako výsledek zůstaly naplněné instance IDataSaver. Mezi daty zde bude existovat relace 1:N. Podle schématu těchto stránek je vidět, že pro každý element Rok, Mesic, čili nějaký termín, je jeden odkaz na ceny. Takže pro jeden termín může existovat (0, N) cen.



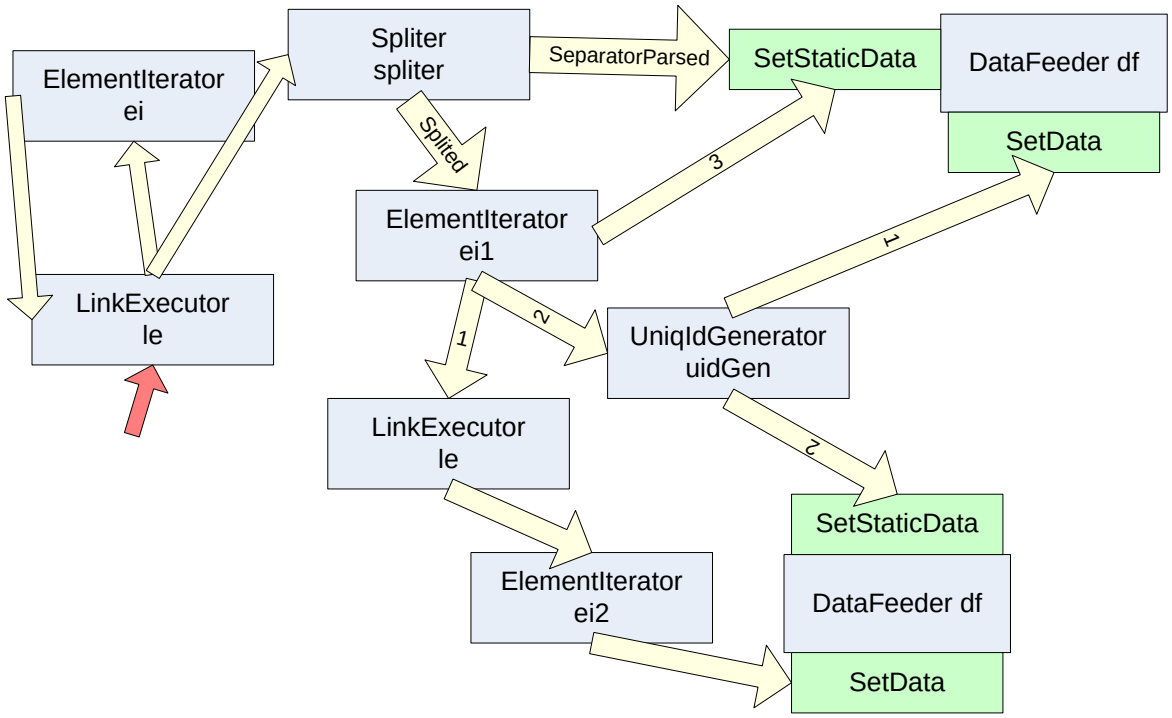
Obrázek 14 - Schéma web aplikace

Na Výpis 25 je kus XML dokumentu deklarující stahovací metodu DownloadMethod agenta, který by tuto úlohu řešil.

```
<DownloadMethod Name="Neco" EntryPage="le" Url="URL">
 <Page id="le">
   <ElementIterator Pattern="@regvyraz1">
     <Link To="le"/>
   </ElementIterator>
   <Spliter Pattern="@regvyraz4">
     <Delegate Name="SeparatorParsed">
       <DataFeeder RefId="dfTerminy" Method="SetStaticData"/>
     </Delegate>
     <Delegate Name="Splited">
       <ElementIterator Pattern="@regvyraz2">
         <DataFeeder RefId="dfTerminy" Method="SetStaticData"/>
         <UniqIdGenerator>
           <DataFeeder RefId="dfTerminy" Method="SetData"/>
```

```
            <DataFeeder RefId="dfCeny" Method="SetStaticData"/>
          </UniqIdGenerator>
          <Link To="le1"/>
        </ElementIterator>
      </Delegate>
    </Spliter>
  </Page>
  <Page id="le1">
    <ElementIterator Pattern="@regvyraz3">
      <DataFeeder RefId="dfCeny" Method="SetData"/>
    </ElementIterator>
  </Page>
</DownloadMethod>
```

Výpis 25- Příklad stahovací metody



Obrázek 15 - Obrazová reprezentace propojení komponent

```
LinkExecutor le = new LinkExecutor(DownloadContext, true);
LinkExecutor le1 = new LinkExecutor(DownloadContext, false);
ElementIterator ei = new ElementIterator ("regularni vyraz 1");
ElementIterator ei1 = new ElementIterator ("regularni vyraz 2");
ElementIterator ei2 = new ElementIterator("regularni vyraz 3");
Spliter spliter = new Spliter("regularni vyraz 4");
DataFilter filter = new DataFilter("Mesic");
UniqIdGenerator uidGen = new UniqIdGenerator();
DataFeeder dfTerminy = new DataFeeder(TerminyDataSaver);
```

```
DataFeeder dfCeny = new DataFeeder(CenyDataSaver);

dfTerminy.AutoGenerateColumns = true;
dfCeny.AutoGenerateColumns = true;

le.LinkOpened                += ei.Parse;
ei.ElementParsed             += spliter.Split;
spliter.SeparatorParsed      += dfTerminy.SetStaticData;
spliter.Splited              += ei1.Parse;
ei1.ElementParsed            += filter.Filter;
filter.Filtred               += dfTerminy.SetStaticData;
filter.Filtred               += uidGen.GenerateUniqId;
uidGen.IdReceived            += dfTerminy.SetStaticData;
uidGen.IdReceived            += dfCeny.SetStaticData;
ei1.ElementParsed            += le1.Open;
le1.LinkOpened               += ei2.Parse;
ei2.Parsed                   += dfCeny.SetData;

le.Open("URL");
```

Výpis 26- Propojeni komponentv kódu ze schématu na Obrázek 15

Po zavolání le.Open se vyvolá delegát ei.Parse. Instance ei třídy ElementIterator má nastaven jako regulární výraz kód HTML odkazu, který listuje mezi stránkami. Pro každou tuto stránku se zavolá na delegát připojená metoda spliter.Split.

Metoda spliter.Split rozdělí jí předaná data (HTML stránku) podle regulárního výrazu, respektive extractor patternu. V tomto případě jsou separátorem roky. Nejdříve se zavolá delegát Splited s obsahem, který je před prvním separátorem. Podle schématu zde nic není. Dále se pro „Rok“ zavolá delegát SeparatorParsed, který spustí metodu dfTerminy.SetStaticData. Ta si uloží tento „Rok“ do kontextové session (viz 5.5 Kontextová session). Po zavolání delegátu SeparatorParsed, se volá delegát Splited s obsahem následujícím po separátoru. Dále se postupně střídají volání delegátu SeparatorParsed a Splited.

Po zavolání delegátu Splited se vyvolá metoda ei1.Parse (viz 5.8.1 Příklad 1). Pro každý zparsovaný element vytvoří tento ElementIterator kolekci IResultCollection, obsahující položky s klíči Mesic, Link a ta se předá přes delegát ei1.ElementParsed nejdříve metodě filter.Filter a poté metodě idGen.GenerateUniqId.

Kolekce IResultCollection se dále předává metodě dfTerminy.SetStaticData. Kromě toho volá delegát e1.ElementParsed metodu uidGen.GenerateUniqId. Tu IResultCollection nezajímá, jediné co udělá je, že vygeneruje unikátní id a zavolá delegát IdReceived. Ten předá tuto id (jako položku IResultCollection s klíčem “ID“) metodám dfTerminy.SetData a

dfCeny.SetStaticData. Tím se přes položku ID spojí data z objektu TerminyDataSaver a CenyDataSaver. Pro jeden záznam v TerminyDataDaver může existovat N záznamů v CenyDataSaver.

Následuje volání le1.Open a další průběh by již měl být zřejmý z příkladu číslo 1.

5.9 Úpravy zparsovaných dat

Data, která jsou získána z webu, mohou potřebovat ještě nějaké úpravy, než se uloží do IDataSaver. Jako příklad jsou data ve špatném formátu, například datum. Na to se dá využít dvojice událostí v třídě DataFeeder. Jedna je ParseCell s argumentem ParseCellEventArgs a druhá ParseRow s argumentem ParseRowEventArgs. Obsloužení těchto událostí je možné zabezpečit implementováním metod v partial class v přidruženém zdrojovém souboru, jehož cesta je zapsána v prvním elementu Agent v atributu CodeFile.

5.10 Rozšíření o vlastní komponentu

Pokud je nutné systém rozšířit o vlastní komponentu, stačí vytvořit třídu v .NET s jedním bezparametrickým veřejným konstruktorem (samozřejmě jich může být více). Dále musí být poděděna od třídy AgentMatBaseComponent z jmenného prostoru AgentMat. Rovněž je nutné, aby byly na příslušných místech použity příslušné atributy, které budou uloženy do metadat této assembly (custom attributes [16]). Podle těchto metadat generátor agenta z XML kódu pozná, co je vstupní metoda, který delegát je primární apod.

Následující komponenta má být volána na nezpracovaná data, nic s nimi však dělat nebude, pouze do výstupu přidá skupinu (budoucí sloupeček v tabulce) s názvem, který je v property Group a hodnotou, která je nastavena v property Variable. Komponenta je pouze ilustrační a jako taková by byla zbytečná. Její činnost by se dala nahradit přidáním sloupečku do DataFeeder s nějakým expression, což by samozřejmě bylo i efektivnější.

```
public class VariableSetter : AgentMatBaseComponent
{
       [AgentMat(AgentMatUsage.OutDefaultDelegate)]
       public ComponentOutputHandler Set;

       private string _group;
       private CustomResultCollection _crco;
       private string[] _groups;
       private string _variable;

       [AgentMat(AgentMatUsage.CustomProperty)]
       public string Group
       {
           get { return _group; }
           set
           {
               _group = value;
               _crco[Group] = Variable;

               _groups = new string[] { _group };
           }
       }

       [AgentMat(AgentMatUsage.CustomProperty)]
       public string Variable
       {
           get { return _variable; }
           set
           {
               _variable = value;
               _crco[Group] = Variable;
           }
       }

         public VariableSetter()
         {
            _crco = new CustomResultCollection();

         }

       [AgentMat(AgentMatUsage.InMethod)]
         public void SetVariables(IContext Context)
         {
                 if (Set != null)
                         Set(Context, _crco, _groups);
         }
}
```

Výpis 27- Příklad zdrojového kódu AgentMat komponenty

Cílem metody set v property Group a property Variable je vytvořit kolekci IResultCollection, která se používá v systému AgentMat na uložení výsledků. Důležité jsou v tomto příkladu atributy AgentMat, jejichž konstruktor bere jako parametr enum type AgentMatComponentUsage

```
enum AgentMatComponentUsage
{
// vstupní metoda (právě jedna v komponentě)
InMethod                = 0x0001,
// výstupní delegát (libovolně mnoho)
OutDelegate             = 0x0002,
// defaultní výstupní delegát (ten který nemusí být zapsán explicitně)
OutDefaultDelegate      = 0x0003,
// Property kterou bude možné nastavit jako atribut v xml
CustomProperty          = 0x0004
}
```

Výpis 28- enum AgentMatComponentUsage

Pokud je potřeba mít v komponentě i nějaké události, na které může uživatel reagovat, stačí pouze událost přidat. Není nutné ji označovat nějakým atributem, použít se budou moci všechny.

Pro použití komponenty stačí v XML dokumentu agenta referencovat assembly, v které bude tato třída a určit jmenný prostor, v němž se nachází.

```
<Agent Name="FiroTourDownloader" Language="C#" CodeFile="Agent.cs">
<References>
            <Assembly Path="…" Namespaces="…"/>
</References>
...
</Agent>
```

Výpis 29 - Příklad použití bloku pro reference

6 Jiné aplikace

Počet aplikací, které se získáváním dat a webovou integrací zabývají, se v poslední době rapidně zvýšil. Je zde široké spektrum různých řešení od komerčních velmi drahých až po ty zcela zdarma. Jelikož je nelze prozkoumat všechny, jsou zde uvedeny tři nejznámější.

6.1 Kapow Mashup Server

Kapow Mashup Server je produkt, který má aktuálně vedoucí pozici na trhu. Jedná se o velmi propracované řešení problému integrace webových aplikací na několika úrovních. Aplikace je distribuovaná ve dvou provedeních. Jedna jako komerční řešení s vlastním serverem, který se instaluje u klienta a vývojovým vizuálním skriptovacím nástrojem pro tvorbu robotů (agentů), kteří získávají data z webových stránek (i dynamických s javascriptem) či jiných zdrojů (PDF, Excel,…) a dále je distribuují dle potřeby. Cenu produktu nebylo možné v době psaní tohoto textu získat, ale vzhledem ke klientele, která tohoto řešení využívá (ministerstvo obrany US, Bank of America, cnet.com), se pravděpodobně bude jednat o vysokou částku.

Existuje i produkt Openkapow Community Editition, který je zcela zdarma. Při použití tohoto řešení si je možné stáhnout vizuální editor obdobný jako u předešlé verze. Aby však šlo výsledného robota z tohoto editoru použít, je nutné ho zaslat na server openkapow.com, kde bude k volnému použití pro všechny uživatele. Jinde než na tomto veřejně přístupném místě robot spustit nepůjde.

V této verzi je možné vytvořit následující roboty:

- **RSS/Atom feed**

  Robot, jenž získaná data vystavuje ve formě rss/atom formátu. To je užitečné pokud ho u nějaké stránky postrádáte. Zde je ho možné celkem snadno vytvořit.

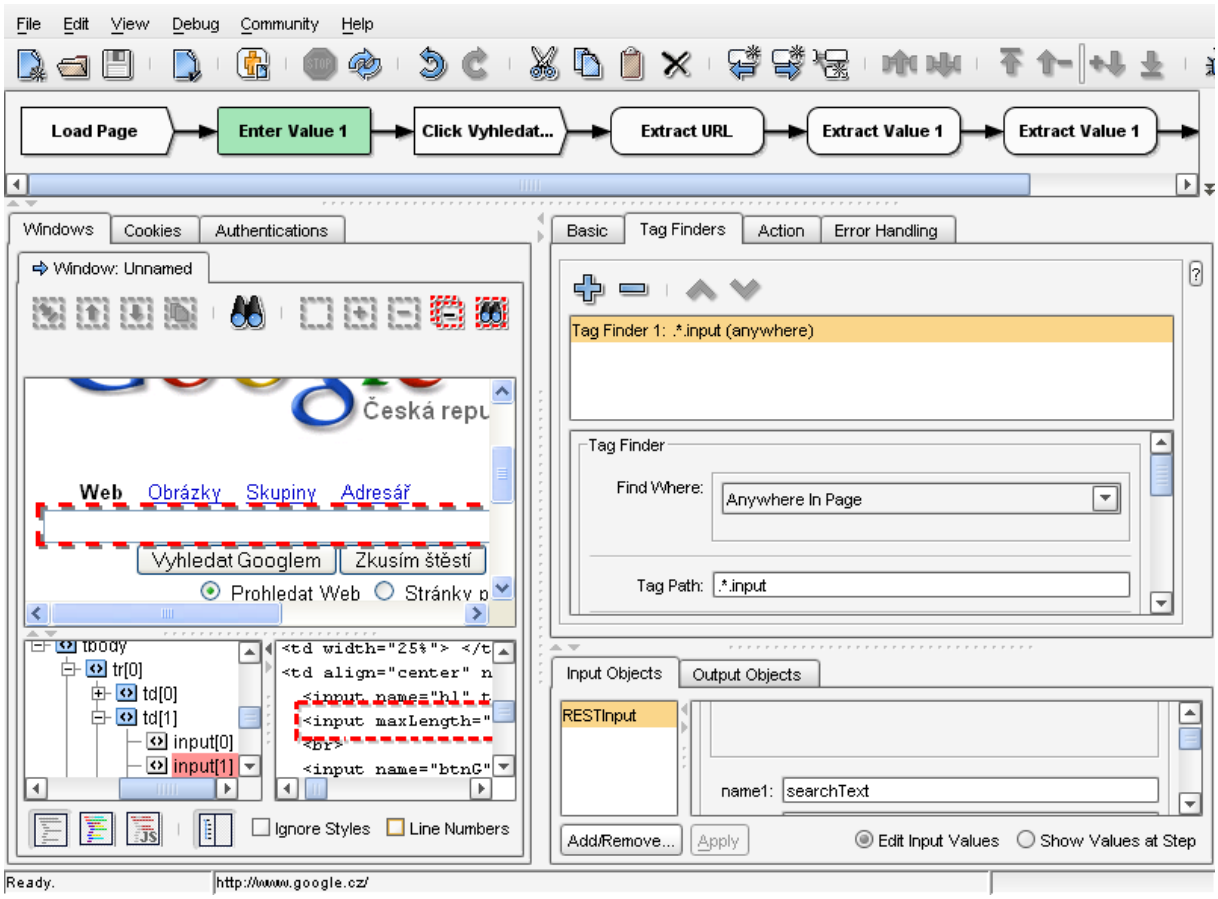
- **REST web service**

  Zde se výstup robota chová jako webová služba.

- **Web clipping**

  Tímto robotem je možné z prezentační vrstvy nějaké webové aplikace vybrat určitou část a volně ji použít v nějaké jiné webové aplikaci.

Základem úspěchu tohoto systému je propracovaný vizuální editor. Aplikace se jmenuje RoboMaker a je poměrně rozsáhlá, takže se musí počítat s tím, že chvíli potrvá, než se jí uživatel naučí ovládat. Poté je však tvorba robotů velmi příjemná. Vše lze nadefinovat klikáním na příslušné prvky webové stránky. Krok za krokem se tak vytváří postup, co bude robot se stránkou dělat. Součást prostředí je schéma kroků, kde je mezi jednotlivými kroky možné přepínat a rovnou vidět, co se v aktuálně zvoleném kroku děje. To se prezentuje přes prohlížeč, HTML kód, HTML cestu a DOM. Výsledek práce robota závisí na jeho typu (viz výše), který byl vybrán na začátku.



Obrázek 16- Kapow RoboMaker

AgentMat lze jen obtížně srovnávat s výše zmíněným robustním komerčním řešením. To by jistě šlo plně využít ve všech případech jako tato práce. Rozhodující element je zde pořizovací cena.

V případě freewarové verze již srovnání tak jednoznačné nebude. Obrovský plus pro řešení firmy kapow je vizuální editor, s kterým lze roboty vytvářet velmi jednoduše. Hlavním omezovací faktorem je zde nutnost nasazení na server openkapow.com. Tuto verzi je vhodné použít v případech, kdy nevadí, že vytvořená práce a i její výsledky budou volně přístupně a

pokud není nutné pracovat s velkým množstvím dat a databázemi, za jehož účelem však tato bakalářská práce byla navržena. Výhodu zde poskytuje hlavně editor, který by však k systému AgentMat mohl být dotačně vytvořen, jelikož se jedná o rozšiřitelný systém.

6.2 WWW::Mechanize

WWW::Mechanize je modul, který se používá v skriptovacím jazyce Perl. Umožňuje otevírat URL, prozkoumávat stránky, následovat odkazy, přijímat cookies, vyplňovat formuláře a klikat na tlačítka. Dále je s ním třeba použít modul HTML::TokeParser pro zpracovávání HTML.

Jelikož jde o poměrně nízkoúrovňové řešení, je zde potřeba psát celý skript. Nelze si jako v předešlém produktu vše naklikat ve vizuálním prostředí. Jako první se vždy používá modul Mechanize, aby byl na server poslán požadavek a výsledný HTML kód se zpracuje přes TokeParser.

```
use WWW::Mechanize;
use HTML::TokeParser;

my $email = '';
die "Must provide an e-mail address" unless $email ne '';

my $agent = WWW::Mechanize->new();
$agent->get("http://www.radiotimes.beeb.com/");
$agent->follow("My Diary");

$agent->form(2);

$agent->field("email", $email);
$agent->click();
```

Výpis 30- Příklad použití modulu WWW::Mechanize

Tento kód je celkem přímočarý. Nejdříve se přidají výše zmíněné moduly. Potom se vytvoří objekt WWW::Mechanize, který otevře příslušnou stránku a klikne na obrázkový odkaz s vlastností ALT="My Diary". Z vrácené stránky se otevře druhý formulář, v kterém se vyplní email a klikne se na jeho submit button. Je vidět, že práce s modulem je vcelku příjemná.

```
my $stream = HTML::TokeParser->new(\$agent->{content});
```

Výpis 31- Vytvoření objektu TokeParser

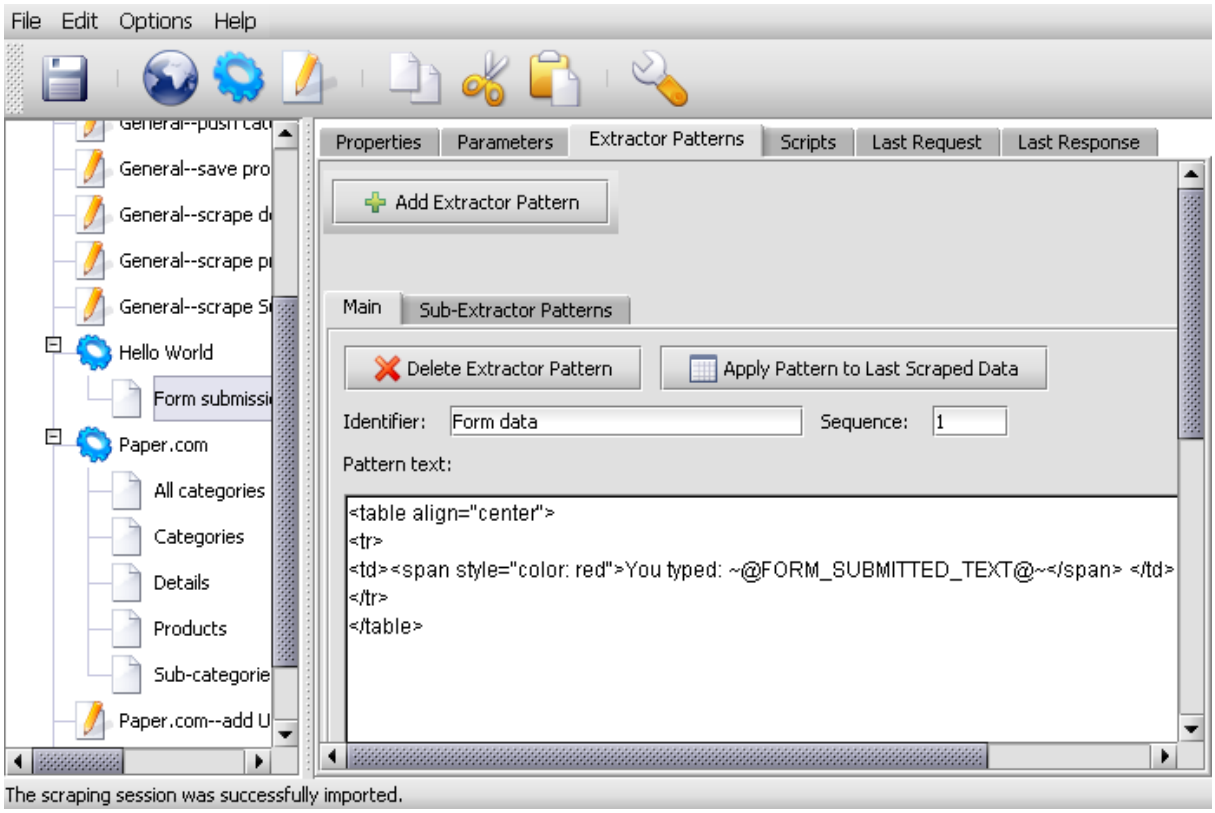
Dále se obsah vrácené stránky předá modulu HTML::TokeParser nebo případně i jinému modulu na zpracování HTML. U tohoto modulu je stěžejní metodu get_tag, která posune stream na další takový tag. Následně se jedná pouze o zpracovávání textu v jazyce Perl a používání této metody.

Modul bohužel nepodporuje javascript, na čemž se pracuje. Celkově je toto řešení pro určité úlohy šikovné a je zadarmo. Avšak pro úlohy zpracovávání velkého množství dat by bylo vhodnější použít například systém AgentMat, jelikož úlohu sám maximálně paralelizuje a tak zrychluje. Rovněž u systému AgentMat není většinou nutné psát žádný skript, pouze XML dokument, což práci zjednodušuje a vytváří prostor pro vývoj vizuálního klikacího editoru.

6.3 Screen scraper

Poslední z aplikací, která zde bude popsána je Screen scraper (www.screen-scraper.com). Je k dispozici ve dvou verzích. Základní, která je zcela zdarma, a rozšířená za 500 dolarů. Rozšířená verze má samozřejmě mnoho výhod oproti verzi, která je zadarmo.

S programem se pracuje pomocí editoru, nikoliv však vizuálního. Základ systému jsou tzv. scraping sessions, které obsahují další elementy zvané scrapable files. Ty se volají v nastavených sekvencích. Jsou to vlastně soubory, které je potřeba procházet, aby se program dostal na nějaké jiné stránky či z těchto přímo získal požadovaná data. Dají se jim nastavit různé parametry pro GET, POST nebo i autentikační tokeny.



Obrázek 17-Screen Scraper

V každém scrapable file je možné nastavit libovolné množství extractor patterns. Ty jsou základem extrahování dat v tomto programu. Tyto extract patterny jsou zjevně příjemným způsobem, jak říci, které části stránky je potřeba získat. Proto byly implementovány i do systému AgentMat.

Během procesu extrahování je možné volat skripty, které mohou vykonávat různé operace nebo volat další scrapable files. Tyto skripty můžou být v různých jazycích jako Visual Basic, JavaScript, Interpreted Java, Python atd.

Tato aplikace je na rozdíl od systému AgentMat více závislá na dodatečném psaní kódu ve skriptech, což samozřejmě dává větší možnosti. Nicméně při vývoji systému AgentMat byla snaha o zachování jednoduchosti pro specifické úlohy tak, aby nutnost přímo psát kód byla co nejmenší. Úlohu tak lze z velké části popsat pouze v deklarativním zápisu. Možnost přímo něco dopsat kódem zde samozřejmě je také, bez ní se občas obejít nedá.

7 Závěr

Cílem této bakalářské práce bylo vytvořit rozšiřitelný systém pro rychlou extrakci velkého množství dat z webových stránek tak, aby tato úloha šla z velké části popsat deklarativním způsobem do XML souboru. Vznikl systém, který zadaný cíl splňuje, a dá se aplikovat jak v komerční sféře, kde se používá, nebo v akademické, kde může sloužit například pro získávání reálných dat pro pokusy se sémantickým webem.

Hlavními přednostmi práce je deklarativní popis úlohy do XML souboru, kterým je možné popsat i velmi složité úlohy, jeho rozšiřitelnost a automatická paralelizace, která extrakci dat podstatně zrychluje.

Vylepšením do budoucna by byl vizuální editor pro tvorbu a ladění agentů, kde by bylo možné si průběh stahování odkrokovat. Další užitečnou vlastností by byla podpora javascriptu, který stávající systém neumí interpretovat.

Reference

[1] Thomas B.Passin: Explorer‘s Guide to the Semantic Web, Manning Publications Co., 2004

[2] W3C: URNs, Namespaces and Registries, http://www.w3.org/2001/tag/doc/URNsAndRegistries-50.html

[3] Chris Bizer,Richard Cyganiak,Tom Heath: How to publish Linked Data on the Web http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/

[4] W3C: Resource Description Framework, http://www.w3.org/RDF/

[5] Dave Beckett: Turtle - Terse RDF Triple Language Specification http://www.dajobe.org/2004/01/turtle/

[6] W3C: RDF/XML Syntax Specification, http://www.w3.org/TR/rdf-syntax-grammar/.

[7] W3C: RDF Schema, http://www.w3.org/TR/rdf-schema

[8] Jeffrey Friedl: Mastering Regular Expressions, O’Reilly, O'Reilly Media, Inc. 2006

[9] Log4net: Logging Services, http://logging.apache.org/log4net/

[10] MSDN: System.CodeDom Namespace, http://msdn2.microsoft.com/en-us/library/system.codedom.aspx

[11] Tim Berners-Lee: Semantic Web Slides, http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

[12] Tim Berners-Lee: Universal Resource Identifiers in WWW (RFC 1630) http://www.faqs.org/rfcs/rfc1630.html

[13] Jeffrey Richter: An Introduction to Delagates, http://msdn.microsoft.com/msdnmag/issues/01/04/net/

[14] W3C: SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/

[15] Chris Bizer, Richard Cyganiak: D2R Server - Publishing Relational Databases on the Semantic Web, http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2r-server/index.html

[16] MSDN: Extending Metadata Using Attributes, http://msdn2.microsoft.com/en-us/library/5x6cd29c(VS.71).aspx

[17] Jeffrey Richter: Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework, http://msdn.microsoft.com/msdnmag/issues/1100/gci/

[18] MSDN: WaitCallback Delegate, http://msdn2.microsoft.com/en-us/library/system.threading.waitcallback.aspx

[19] Wikipedia: Web scraping, http://en.wikipedia.org/wiki/Web_scraping

[20] W3C: SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/
[21] MSDN: DataColumn.Expression Property, http://msdn2.microsoft.com/en-us/library/system.data.datacolumn.expression(vs.71).aspx
[22] Wikipedia: Screen scraping, http://en.wikipedia.org/wiki/Screen_scraping
[23] MSDN: Overview of ADO.NET, http://msdn2.microsoft.com/en-us/library/h43ks021(VS.71).aspx

Příloha 1. AgentMat XML Dokument Specifikace

Document:=
Agent

Agent:=
<Agent Name=”IDTYPE” Language=”Cs|Vb” Namespace=”STRING” [CodeFile=”STRING”] [Description=”STRING”] >
Browser?,References?, DataFeeders, Constants, DownloadMethods
</Agent>

Root element Agenta (třídy, která získává data z webových stránek).

Name	Jméno agenta i vzniklé třídy
Language	Jazyk, v kterém bude třída vygenerována. Na výběr jsou CS – pro C# a Vb pro Visual Basic. Tento atribut má význam, pokud je potřeba získat zdrojový kód třídy v některém z těchto jazyků nebo je uveden atribut CodeFile, v kterém je jméno souboru se zdrojovým kód v tomto jazyku.
Namespace	Jmenný prostor, v kterém bude vzniklá třída agenta.
CodeFile	Cesta k souboru s definicí partial třídy, která bude součástí vzniklé třídy agenta.
Description	Popis, co agent dělá.

Browser:=
<Browser>UserAgent?,Charset?,Language?,Timeout?</Browser>

UserAgent:=
< UserAgent >STRING</ UserAgent >

Řetezcem zde zadaným se bude agent identifikovat serveru.

Charset:=
< Charset >STRING</ Charset>

Jakou znakovou sadu má při komunikaci se serverem používat.

Language:=
< Language >STRING</ Language>

Jazyk, který se bude používat.

Timeout:=

< Timeout >UNSIGNEDINT</ Timeout>

Čas, po kterém se spojení se serverem uzavře, pokud se agent nedočká odpovědi.

References:=

<References>Assembly*</References>

Assembly:=

<Assembly Path=”STRING” Namespace=”STRING”>

Assembly, z kterých chceme použít komponenty systému AgentMat.

Path	Cesta ke knihovně dll obsahující assembly.
Namespace	Seznam jmenných prostorů oddělených středníkem, které budou z assembly použity. Př.: Namespace1;Namespace2,…

DataFeeders:=

<DataFeeders>*DataFeeder**</DataFeeders>

DataFeeder:=

<DataFeeder Id=” IDTYPE” AutoGenerateColumns=”true|false” Table=”STRING” any-attributes>*Column**</DataFeeder>

Tento element deklaruje třídu, která se stará o ukládání dat získaných při běhu agenta. Pro jednu implementace IDataSaver,která ukládá data, je potřeba jeden DataFeeder. Schéma pro IDataSaver se dá specifikovat pomocí element Column.

Id	Jednoznačný identifikátor (v elementu DataFeeders)
AutoGenerateColumns	Pokud se vyskytne nějaký údaj, který patří do nedeklarovaného sloupečku, vytvoří se příslušný sloupec.
Table	Jméno tabulky

Column:=

<Column Name=” IDTYPE” Type=” STRING” [Expression=” STRING”]/>

Specifikuje sloupeček pro IDataSaver, který používá komponentou DataFeeder.

Name	Jméno sloupečku
Type	Typ sloupečku. (Př. System.Int32, System.String, System.Datetime)
Expression	Výraz. Viz [21]

7.1 Constants:=

<Constants>Item*</Constants>

Item:=

<Item Id=" IDTYPE">CDATA</Item>

Definuje konstantu. Hodnota konstanty je uvnitř elementu.

Id	Jednoznačný identifikátor konstanty

DownloadMethods:=

<DownloadMethods> *DownloadMethod**</DownloadMethods>

DownloadMethods:=

<DownloadMethod Name=" IDTYPE" EntryPage=" IDTYPE" Url="STRING"
[Description="STRING"]>*Page*{1,}</DownloadMethod>

Deklaruje stahovací metodu. Obsahuje 1 až neomezeně mnoho elementů Page pro logické stránky.

Name	Jméno stahovací metody
EntryPage	Stránka, která se aplikuje jako první
Url	Url, které bude předáno stránce v EntryPage
Desciption	Slovní popis, co tato metoda dělá.

Page:=

<Page Id=" IDTYPE" any-attributes>*Component**</Page>

Logická stránka, která sdružuje elementy komponent.

Id	Jednoznačný identifikátor stránky

Component:=

<COMPONENT-NAME any-attributes>*Delegate**|*Component**|*DataFeeder*[2]*</Component>

Element specifikující komponentu.
Atributy zastupují vlastnosti (properties) a události (events) komponenty.
Vnořeným komponentám budou předány data z aktuální komponenty přes nějakého výstupního delegáta. Delegát se dá buď specifikovat explicitně nebo se použije defaultní, pokud se podelement Delegate vynechá.

Delegate:=

<Delegate Name="STRING">*Component**|*DataFeeder**</Delegate>

Element zastupující delegáty (druhy výstupu) komponenty, předávající zpracovaný vstup dále vnořeným komponentám.

Name	Jméno delegáta komponenty (v rodičovském elementu)

DataFeeder[2]:=

<DataFeeder RefId=”IDTYPE” Method=”SetStaticData|SetData”/>

Pošle data k uložení komponentě DataFeeder.

RefId	Identifikátor DataFeederu, který chceme použít.
Method	Je nutné zvolit, jestli mají být data uložena již napevno nebo ještě budou nějaká další data do aktuálního řádku IDataSaver přidána. SetStaticData – uloží jen do kontextové cache SetData – vše uloží napevno jako řádek tabulky

Typy

IDTYPE:= [a-zA-Z0-9:_][a-zA-Z0-9:_\.\-]*

COMPONENT-NAME stejné jako jméno elementu v definici xml

Příloha 2. XSD Schéma dokumentu AgentMat Xml

V této příloze je schéma AgentMat Xml dokumentu. V elementu Page je použito xs:any, protože se jména komponent v dokumentu určují jmény elementů a dopředu není známo, jaké komponenty budou používány. To se zjišťuje až při zpracovávání xml dokumentu přes .NET reflection, kdy proběhne programová validace této části. XSD schéma komponenty by vypadalo takto:

```
<xs:element name="COMPONENT-NAME" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="COMPONENT-NAME" minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="Delegate" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="COMPONENT-NAME" minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attribute name="Name" type="xs:string" />
        </xs:complexType>
      </xs:element>
    </xs:choice>
    <xs:anyAttribute processContents="skip" />
  </xs:complexType>
</xs:element>
```

XSD schéma AgentMat Xml dokumentu:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns="http://ms.mff.cuni.cz/~benom4am/AgentMat"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:NS="http://ms.mff.cuni.cz/~benom4am/AgentMat"
targetNamespace="http://ms.mff.cuni.cz/~benom4am/AgentMat" elementFormDefault="qualified">
  <xs:simpleType name="IdType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9:_][a-zA-Z0-9:_\.\-]*" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="Agent">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Browser" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="UserAgent" type="xs:string" minOccurs="0" maxOccurs="1"
default="Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; YPC 3.0.1; .NET CLR 1.1.4322; .NET
CLR 2.0.50727)" />
              <xs:element name="Charset" type="xs:string" minOccurs="0" maxOccurs="1"
default="utf-8" />
              <xs:element name="Language" type="xs:string" minOccurs="0" maxOccurs="1"
default="en" />
              <xs:element name="Timeout" type="xs:unsignedInt" minOccurs="0" maxOccurs="1"
default="30000" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="References" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Assembly">
                <xs:complexType>
                  <xs:attribute name="Path" type="xs:string" use="required" />
                  <xs:attribute name="Namespaces" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="DataFeeders">
```

```
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DataFeeder" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Column" minOccurs="0" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:attribute name="Name" type="IdType" use="required" />
                    <xs:attribute name="Type" type="xs:string" use="required" />
                    <xs:attribute name="Expression" type="xs:string" use="optional" />
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="Id" type="IdType" use="required" />
              <xs:attribute name="AutoGenerateColumns" type="xs:string" use="required" />
              <xs:attribute name="Name" type="xs:string" use="required" />
              <xs:anyAttribute processContents="skip" />
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Constants">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Item" nillable="true" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:string">
                  <xs:attribute name="Id" type="IdType" use="required" />
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="DownloadMethods">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DownloadMethod" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Page" minOccurs="0" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="Id" type="IdType" use="required" />
                    <xs:anyAttribute processContents="skip" />
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="Name" type="IdType" use="required" />
              <xs:attribute name="EntryPage" type="IdType" use="required" />
              <xs:attribute name="Url" type="xs:string" use="required" />
              <xs:attribute name="Description" type="xs:string" use="optional" />
            </xs:complexType>
            <xs:key name="PageKey">
              <xs:selector xpath=".//NS:Page" />
              <xs:field xpath="@Id" />
            </xs:key>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="Name" type="IdType" use="required" />
  <xs:attribute name="Language" type="xs:string" use="required" />
  <xs:attribute name="CodeFile" type="xs:string" use="optional" />
  <xs:attribute name="Namespace" type="xs:string" use="required" />
  <xs:attribute name="Description" type="xs:string" use="optional" />
</xs:complexType>
<xs:key name="DataFeederKey">
  <xs:selector xpath=".//NS:DataFeeder" />
  <xs:field xpath="@Id" />
</xs:key>
```

```
    <xs:key name="DownloadMethodKey">
      <xs:selector xpath=".//NS:DownloadMethod" />
      <xs:field xpath="@Name" />
    </xs:key>
  </xs:element>
</xs:schema>
```