Slovenská technická univerzita
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Ilkovičova 3, 842 16 Bratislava 4

Tvorba obaľovačov na získavanie informácií z webu

Softvérový systém

Tím číslo 5: *Ľubomír Chamraz , Ivan Kišac , Ján Krausko , Michal Kurťák , Marián Šimko , Michal Šimún*
Vedúci tímu: *Mgr. György Frivolt*
Študijný odbor / program: Softvérové inžinierstvo / Softvérové inžinierstvo
Ročník, typ štúdia: 1, inžinierske štúdium
Dátum odovzdania: máj 2007

Obsah

1 Úvod

Predložený dokument opisuje riešenie zadania „Tvorba obaľovačov na získavanie informácií z webu". Ide o dokumentáciu softvérového systému, v jednotlivých kapitolách sa postupne venujeme všetkým etapám jeho výboja.

Kapitola 2 sa zaoberá analýzou existujúcich riešení na pôde predprogramovaných obaľovačov. Približujeme v súčasnosti najpoužívanejšie riešenia, ako aj aplikačný rámec, ktorý bol vytvorený v tímovom projekte počas minulého roka. Na základe výsledkov analýzy definujeme v kapitole 3 ciele projektu a požiadavky, ktoré sú softvérový systém kladené. Kapitola 4 sa venuje špecifikácii obaľovača z hľadiska prípadov jeho použitia. V kapitole 5 je uvedený stručný opis nových technológií, ktoré je pri návrhu požadovanej funkcionality obaľovača potrebné poznať a ktoré budú súčasťou nášho riešenia. Kapitola 6 je zameraná na hrubý návrh systému a na základe uvedenej architektúry rozoberá jednotlivé moduly navrhovaného riešenia.

1.1 Slovník pojmov

Pojmy týkajúce sa procesnej funkcionality obaľovača:

Obaľovač (angl. wrapper)	Nástroj na dolovanie dát z neštruktúrovaného vstupu
Extrakcia	Proces dolovania semištruktúrovaných dát do štruktúrovaného výstupu
Navigácia	Pohyb po dokumentoch s účelom dostať sa k cieľovému zdroju
Kontext	Dátová štruktúra, ktorá obsahuje aktuálny stav obaľovania patriaci k akcii. Obsahuje vstupné premenné, vzor, aktuálny subdokument, lokálny výstupný objekt
Vstupný parameter	Podmienka obaľovania, zvyčajne zadaná používateľom
Akcia	Operácia nad kontextom. Extrakčná, navigačná
Vzor	Výber subdokumentu. Obsahuje filtre ako kritérium na tento výber. Aplikovaním vzoru na subdokument vznikajú nové subdokumenty
Filter	Kritérium na výber subdokumentu; v prípade XPath výrazu – DOM, v prípade regulárneho výrazu – String
Subdokument	Časť dokumentu vybraná filtrom
Výstupný objekt	Objekt obsahujúci výsledky procesu obaľovania

Pojmy týkajúce sa učenia obaľovača:

Učenie	Proces generalizácie a konkretizácie vzoru na základe interakcie s používateľom
Príklad	Časť dokumentu, ktorý špecifikoval používateľ. Pozitívny alebo negatívny
Generalizácia	Zovšeobecnenie príkladov také, že vytvorený vzor obsahuje všetky pozitívne a žiaden negatívny príklad

Ostatné pojmy:

cookie	Krátka textová informácie o internetovom spojení týkajúcom sa relácie so vzdialeným serverom. Je uložená na strane klienta a obsahuje informácie o používateľovi týkajúce sa napr. autentifikácie, preferovaných vlastností, atď.
JavaBeans	Znovupoužiteľné softvérové komponenty napísané v programovacom jazyku Java.

1.2 Použité skratky

CSS	Cascade Sheet Styling	Kaskádové štýly
DOM	Document Object Model	Objektový model dokumentu
HTML	HyperText Markup Language	Hypertextový značkovací jazyk
HTTP	HyperText Transfer Protocol	Hypertextový transportný protokol
IDL	Interface Decription Language	Jazyk opisu rozhraní
URL	Uniform Resource Locator	Jednotný identifikátor zdroja
XML	eXtensible Markup Language	Rozšíriteľný značkovací jazyk
XPCOM	Cross Platform Component Object Model	Multiplatformový komponentový objektový model
XPConnect	Cross Platform Connect	Multiplatformové prepojenie
XPIDL	Cross Platform Interface Description Language	Multiplatformový jazyk opisu rozhraní
XUL	XML User Interface Language	Jazyk na opis používateľského rozhrania založený na XML

1.3 Použitá notácia

`ForEachTag`	neproporcionálnym písmom budeme označovať úryvky z programového kódu, názvy tried a pod.
Lixto Wrapper Designer	tučné písmo zvýrazňuje a zdôrazňuje obzvlášť dôležité kľúčové slová
Actions	• kurzívou budú označené názvy XML elementov, resp. ich atribútov • taktiež budú zvýraznené menej dôležité kľúčové slová

2 Existujúce nástroje na tvorbu obaľovačov

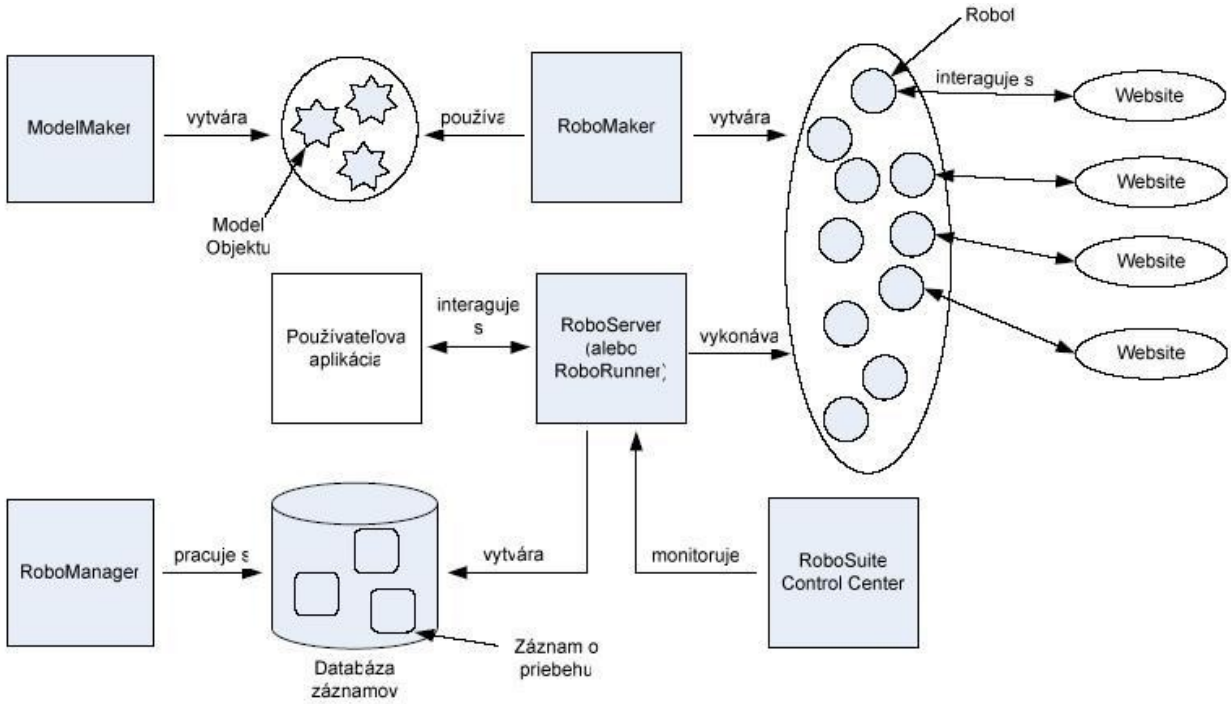
V tejto kapitole sa postupne pozrieme na najpoužívanejšie nástroje na tvorbu obaľovačov (časti 2.1 a 2.2) a pokúsime sa analyzovať aplikačný rámec Wrapper Suite, ktorý je výsledkom tímového projektu minulého roka (časť 2.3).

2.1 Kapow RoboSuite

RoboSuite je nástroj, ktorý napomáha transformovať informácie a funkcionalitu webových systémov do presne definovaného formátu [1]. Funkcionalitu interakcie, navigácie, extrakcie a integrácie dát obsiahnutých v informačnom priestore zabezpečuje robot (obaľovač).

2.1.1 Architektúra nástroja

RoboSuite pozostáva z viacerých aplikačných častí. Obr. 2-1 zobrazuje vzťahy medzi definovanými časťami nástroja.



Obr. 2-1: Architektúra nástroja RoboSuite [1]

Prvou etapou používania nástroja RoboSuite je modelovanie charakteristík robota pomocou aplikačných častí ModelMaker a RoboMaker. Program vytvoreného robota možno vykonávať viacerými spôsobmi:

- dávková úloha pomocou aplikačnej časti RoboRunner, ktorý umožňuje spustenie robota z príkazového riadku
- spustenie robota zo vzdialenej klientskej aplikácie, ktoré zabezpečuje RoboServer; vzdialený monitoring časti RoboServers a ich robotov umožňuje aplikačná časť RoboSuite Control

Center

RoboManager predstavuje nástroj na riadenia a udržiavanie viacerých robotov, umožňuje tiež prezeranie stavových a chybových hlásení vygenerovaných robotmi.

V nasledujúcej časti opisu nástroja Kapow RoboSuite sa budeme venovať aplikačným častiam na tvorbu obaľovača (ModelMaker a RoboMaker), keďže tie predstavujú najdôležitejšie časti nástroja vzhľadom na ciele nášho projektu.

2.1.2 ModelMaker

ModelMaker predstavuje aplikáciu pre tvorbu objektov, s ktorými pracuje robot [2]. Objekty modelujú reálne časti informačného priestoru a možno ich rozdeliť na nasledovné časti:

- vstupné objekty – objekty, ktoré slúži ako vstup robota pre vykonávanie jeho práce. Medzi vstupné objekty možno zaradiť napríklad dáta, ktoré bude robot vkladať do formulárov
- výstupné objekty – objekty extrahované robotom, stavové alebo chybové objekty
- databázové výstupné objekty – objekty, ktoré sú extrahované a následne uložené do špecifikovanej databázy

Objekty sa skladajú z atribútov modelujúcich určitú charakteristiku objektov a predstavujú miesto v objekte, kde sú dáta uložené (napr. názov pracovnej ponuky). Vytvorené objekty sú združené do doménového modelu, ktorý predstavuje informačný priestor robota.

2.1.3 RoboMaker

RoboMaker predstavuje aplikačnú časť nástroja slúžiacu na tvorbu robota. Môže byť dvoch typov:

- robot na zbieranie dát (data collection robot)
- robot na získavanie funkcionality web systému alebo jeho časti (clipping robot)

Ďalej sa budeme zaoberať iba prvým typom robota, keďže cieľom nášho projektu je tvorba nástroja na extrakciu dát.

Robot je realizovaný ako stavový automat, ktorý sa počas vykonávania svojej činnosti nachádza v určitom stave [3], ktorý je definovaný nasledovnými komponentmi:

- *Windows* – predstavuje práve otvorenú web stránku, s ktorou robot práve pracuje
- *Objects* – aktuálne hodnoty objektov
- *Cookies* – HTTP cookies získané počas komunikácie s web serverom
- *Autentifications* – autentifikačné údaje získané počas komunikácie s web serverom

Stavový automat robota pozostáva z jednotlivých krokov, ktoré určujú elementárnu činnosť robota. Krok prijíma ako vstup stav robota a produkuje v závislosti od typu kroku nula alebo viac stavov robota. Pri tvorbe kroku je možno využiť jednu z definovaných typov akcií:

- navigačná akcia – získanie definovanej web stránky, nasledovanie odkazu
- extrakčné akcie – výber lokalizovaných dát z dokumentu, ich konverzia do definovaného

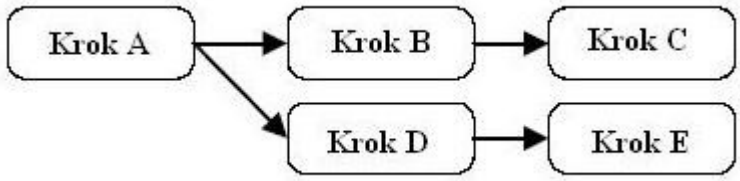
formátu z textového formátu a uloženie do atribútov objektu. Získané dáta môžu byť pred uložením do objektov spracované pomocou regulárnych výrazov

- iteračné akcie – akcie, ktoré sa vykonávajú pre všetky lokalizované časti dokumentu (napr. `ForEachTag`)
- vyplňovanie formulárov

Krok pozostáva z nasledujúcich častí:

- názov kroku
- tag finders – lokalizácia časti dokumentu (jeden alebo viac elementov)
- step action – akcie, ktoré vykonávajú zmenu stavu robota v závislosti od typu akcie. Napríklad extrakčná akcia mení objekty robota alebo navigačné akcie menia *Windows*, *Cookies* a *Autentifications*

Posledným krokom v tvorbe robota je vytvorenie spojení medzi špecifikovanými krokmi, ktoré definujú poradie vykonávania jednotlivých akcií. Výstupný stav kroku robota predstavuje vstupný stav robota nasledujúceho kroku vo vytvorenej postupnosti krokov. Špeciálnou akciou je iteračná akcia, ktorá produkuje viacero výstupných stavov (pre každý krok iterácie jeden stav). Krok môže byť spojený s viacerými nasledujúcim krokmi, takýto typ spojenia predstavuje vetvenie. Príklad postupnosti krokov je zobrazený na Obr. 2-2.



Obr. 2-2: Príklad postupnosti krokov robota [3]

Vykonanie akcií zobrazených na obrázku záleží na móde vetvenia. V prostredí tvorby RoboMaker je možné zadefinovať dva módy vetvenia:

- vykonanie všetkých vetiev – paralelne sa vykonajú všetky vetvy
- vykonaj vetvy pokiaľ sa nenájde úspešná vetva – postupne sa vykonávajú všetky vetvy akcií v definovanom poradí, pokiaľ nejaká vetva neskončí úspešnou akciou, ktorá nemá žiadny chybový stav

Vyššie uvedený príklad vykoná najprv akcie v kroku A a výstupný stav tohto kroku bude predstavovať vstupný stav kroku B a tiež kroku D. Za predpokladu, že je nastavený krok vetvenia vykonania všetkých vetiev a zároveň sa nevyskytnú žiadne chybové stavy, bude po ukončení akcie v kroku B vykonaný krok C a po ukončení akcie v kroku D aj krok E.

2.1.4 Lokalizácia častí dokumentu (tag finders)

Tag finders slúžia na určenie časti dokumentu, kde bude aplikovaná daná akcia [3]. Pri určení časti dokumentu je možné využiť stromovú štruktúru HMTL dokumentu a/alebo prácu

s regulárnymi výrazmi. Tag finders pozostávajú z nasledovný častí:

- *Find where* – špecifikuje, kde sa má daná časť dokumentu vyhľadať (je možné určiť rozsah celej stránky)
- *Tag Path* – špecifikovanie cesty v stromovej štruktúre dokumentu. Tag Path predstavuje upravené XPath s obmedzenou vyjadrovacou silou a upravenou syntaxou
- *Attribute name* – možnosť špecifikovania, že element musí mať daný atribút
- *Attribute value* – možnosť špecifikovania, že určený atribút nadobúda danú hodnotu. Hodnotu atribútu možno určiť pomocou regulárnych výrazov
- *Tag pattern* – možnosť určenia vzoru elementu pomocou regulárnych výrazov (napr. .*<b>\w+<b>.*)
- *Tag depth* – špecifikuje hĺbku elementu v strome dokumentu
- *Tag number* – určuje počet elementov v prípade, že viacero elementov spĺňa definované podmienky

2.1.5 Ošetrenie chýb

Kroky robota môžu pri svojej činnosti generovať chybové stavy, napr. v prípade, že sa nemôžu lokalizovať požadované dáta v dokumente, s ktorými má akcia pracovať. Každému kroku robota možno priradiť vlastné ošetrenie chybových stavov, ktoré spĺňa jednu z nasledovných typov ošetrenia chýb [3]:

- reportuj tu – vykonávanie robota sa zastaví a vykoná sa hlásenie o chybovom ukončení činnosti robota
- pošli chybový stav predchádzajúcemu – chybový stav sa bude ošetrovať v kroku, ktorý predchádzal vykonaniu aktuálneho kroku
- ignoruj a choď na ďalší krok – chybový stav nie je ošetrený a vykonávanie robota pokračuje v ďalšom kroku
- ignoruj a ukonči vykonávanie vetvy – chybový stav nie je ošetrený, ale preskočia sa všetky kroku vo vetve, kde nastala chyba

2.1.6 Zhodnotenie nástroja

Medzi najväčšie výhody nástroja patrí jeho vizuálna podpora tvorby robota, ktorá umožňuje vytvárať postupnosti krokov a taktiež konfigurovať jednotlivé kroky iba pomocou vizuálnych možností nástroja. Prostredie tiež obsahuje vlastný integrovaný prehliadač, kde sa zobrazuje aktuálny dokument a používateľ môže vyznačiť údaje, ktoré má nástroj extrahovať. Prostredie tiež obsahuje ladiaci nástroj, kde je možné vyskúšať funkcionalitu robota a prezerať aktuálny stav robota v každom kroku. Nástroj má tiež výbornú podporu vzdialeného prístupu z iných aplikácií, umožňuje integráciu dát viacerých robotov, podporuje tiež prácu s cookies, formulármi alebo autentifikáciou.

Medzi nevýhody nástroja patrí slabá podpora výstupných objektov, kde je možné nadefinovať iba výstup v tvare XML súboru alebo zápis do určitých databáz, ktoré nástroj

podporuje. Kapow RoboSuite tiež nepodporuje učenie sa robota, kde by bolo možné na základe príkladov naučiť robota, ktoré dáta má extrahovať, čo by zvýšilo flexibilitu robota.

2.2 Lixto Visual Wrapper

Lixto je vizuálny interaktívny nástroj na vytváranie obaľovačov webových stránok pod vedením človeka. Umožňuje automatickú extrakciu informácií z neštruktúrovaných webových stránok s použitím obaľovačov a preklad získaného obsahu do štruktúrovaného XML dokumentu (produkujú sa tzv. XML Companions).

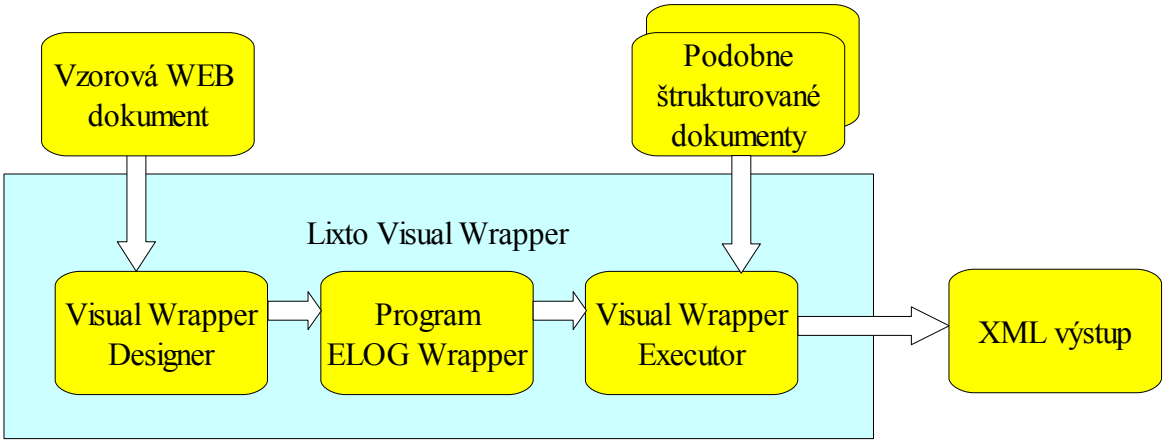
Systém pracuje na základe vizuálnej interakcie používateľa so spracovávaným dokumentom v prostredí Lixto Visual Wrapper. Používateľovi sa priebežne zobrazujú výsledky, ktoré by za súčasného stavu obaľovača získal. Tak môže podľa potreby upravovať zadané požiadavky. Tvorba obaľovača sa realizuje pomocou ovládacích prvkov, takže používateľ nemusí ovládať žiadny programovací jazyk. Pre úspešnú a efektívnu prácu so systémom by mal ovládať aspoň základy štruktúry HTML, prípadne prácu s reťazcami (regulárne výrazy a pod.).

V systéme Lixto Visual Wrapper nie je proces tvorby obaľovača limitovaný len na jednu webovú stránku alebo na webové stránky so štruktúrou rovnakého typu. Počas definície obaľovača sa môže operátor presunúť na ďalšie vzorové stránky a pokračovať v definícii obaľovača tam.

2.2.1 Architektúra nástroja

Lixto Visual Wrapper sa skladá z dvoch hlavných častí: *Lixto Wrapper Designer* a *Lixto Wrapper Executor*.

- **Lixto Wrapper Designer** je nástroj na vytváranie a uchovávanie programu pre obaľovač. Špecifikuje, ako budú programom extrahované údaje preložené do XML
- **Lixto Wrapper Executor** aplikuje program obaľovača a XML prekladovú schému na spracovávané dokumenty



Obr. 2-3: Architektúra systému Lixto Visual Wrapper [4]

Na vzorovom webovom dokumente sa pomocou Visual Wrapper Designera definuje, ktoré

údaje sa majú extrahovať a stanoví sa aj ich namapovanie do výstupného XML dokumentu. Tieto definície VW Designer zapíše v podobe programu, ktorý sa používa pri samotnej extrakcii. Program je napísaný v jazyku ELOG.

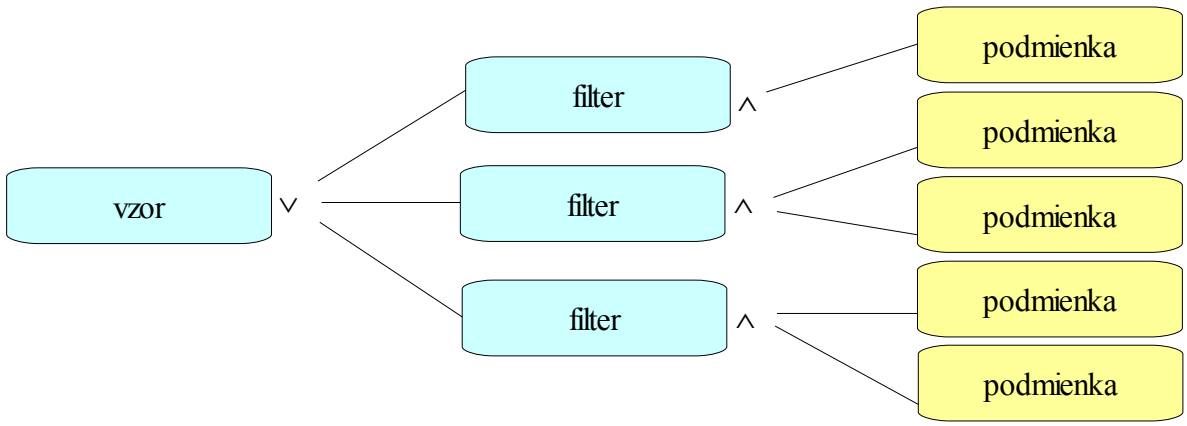
Visual Wrapper Executor potom na základe tohoto programu môže vykonávať extrakciu údajov z ďalších dokumentov. Vďaka dostatočnému množstvu metód a podmienok výberu údajov môže operátor vytvoriť robustný program, ktorý možno použiť na viaceré dokumenty vykazujúce znaky určitej štrukturálnej podoby so vzorovým dokumentom. Architektúru systému a proces tvorby obaľovača (programu pre extrakciu údajov z dokumentov) zobrazuje Obr. 2-3. Skutočná extrahovaná informácia závisí od vstupného dokumentu. Jeden program môže byť použitý na celú triedu podobne štruktúrovaných dokumentov.

2.2.2 Základné prvky – vzory, filtre, podmienky

Na tvorbu programov pre výber údajov z dokumentov sa používajú 3 základne prvky:

- vzory
- filtre
- podmienky

V architektúre Lixto *vzory* popisujú, ako extrahovať časti údajov z webových stránok. Vzor sa skladá z *filtrov*. Filtre definujú časť stránky, z ktorej sa majú získať údaje. Interpretujú sa disjunktívne, t.j. stačí, aby sa dal aplikovať jeden z filtrov. Ak filtre vyberú aj neželané údaje, na obmedzenie tohoto výberu sa používajú *podmienky*. Podmienky sa interpretujú konjunktívne, t.j. musia byť splnené všetky, aby došlo k výberu údajov, ktoré by bez aplikácie podmienok vybral filter pre daný vzor. Tieto vzťahy schématicky zobrazuje Obr. 2-4.



Obr. 2-4: Vzťahy medzi vzormi, filtrami a podmienkami [4]

Proces tvorby Lixto obaľovača pozostáva z vytvárania vzorov, následného pridelenia filtrov k týmto vzorom, testovania vybraných inštancií a prípadného nastavenia podmienok na obmedzenie výberu filtrov. Výsledný obaľovač sa interne zaznamená v jazyku *Elog*. Na podporu úprav výstupu poskytuje systém používateľovi nástroj XML Tool.

Vzory

Vzory sú základnými konštrukciami obaľovača a definujú jeho hierarchickú štruktúru. Každý vzor je postupne mapovaný na XML element a každý potomok vzoru v stromovej hierarchii vzorov je mapovaný do XML elementov vhniezdených do rodičovského.

Opisujú, ako extrahovať časti údajov z webových stránok. Každý vzor pozostáva z filtrov, ktoré rozhodujú, čo sa má extrahovať.

Vzory sa vytvárajú interaktívne: Operátor vytvorí nový vzor a označí želaný príklad na webovom dokumente. Systém zovšeobecní výber a vráti operátorovi všetky inštancie vzoru vybrané po zovšeobecnení. Odpoveď systému je čisto vizuálna – každá inštancia vzoru je zvýraznená. Takýto systém odozvy systému umožňuje predchádzať princípu: Vytvorenie a spustenie obaľovača a následná oprava chýb. Operátor môže potom zúžiť výber inštancií pomocou obmedzení (zavedenie podmienok) a následne môže pridávať ďalšie inštancie, ak aktuálny výber nezahŕňa všetky, ktoré si želal.

Kategórie vzorov

V závislosti na type požadovanej informácie si treba zvoliť jednu z troch kategórií vzorov:

- *strom* (tree) – Vzor strom slúži na extrakciu časti dokumentu v závislosti na HTML elementy alebo listy elementov.
- *reťazec* (string) – Tento vzor slúži na extrakciu textových reťazcov z viditeľných aj neviditeľných častí dokumentu. Môže to byť napríklad e-mailová adresa, poštové smerové číslo alebo nejaký atribút, napríklad meno obrázku.
- *dokument* (document) – Vzor dokument slúži na extrakciu celých webových stránok a používa sa na navigáciu na odkazmi pripojené stránky. Základom každého obaľovača je koreňový vzor (`rootPattern`). Je to dokumentový vzor, na ktorý sa môžu pripájať aj ďalšie dokumentové vzory na extrakciu informácií z ďalších prepojených stránok.

Každý vzor má svoj rodičovský vzor (*parent pattern*). Vzory a filtre obsahujú informácie o ich rodičoch.

Filtre

Logická organizácia extrakčných vzorov je nasledovná: Každý extrakčný vzor má meno a obsahuje niekoľko filtrov. Každý filter poskytuje alternatívnu definíciu údajov, ktoré sa majú extrahovať a asociovať s daným vzorom. Vzory musia mať aspoň 1 filter, aby mohli extrahovať údaje. No, môžu mať viacero ďalších vzorov ako svoje deti.

Filtre vymedzujú, aké informácie sa majú extrahovať z kontextu príslušného vzoru, t.j. ktoré časti dokumentu majú byť extrahované. Upravujú výber v stromovej štruktúre HTML dokumentu, alebo pristupujú k dokumentu ako k reťazcu a podľa toho sú aj definované podmienky výberu. Filtre môžu byť obmedzené podmienkami.

Ak je špecifikovaný viac ako jeden filter, musí byť splnený aspoň jeden, aby bola inštancia vybraná. Typ filtra je definovaný typom rodičovského vzoru, ku ktorému filter patrí:

- Filter stromu (*tree filter*)
- Filter reťazca (*string filter*)
- Filter dokumentu (*document filter*) – slúži na pospájanie informácií z viacerých HTML stránok do jedného XML dokumentu. Používajú sa v dvoch prípadoch:
    1. Detailné informácie sa nachádzajú na stránkach, na ktoré sú na spracovávanej stránke odkazy. Tieto stránky s detailnými informáciami mávajú obyčajne inú štruktúru než pôvodná stránka a musia byť analyzované inými vzormi.
    2. Dlhé zoznamy bývajú často rozdelené na viacero stránok, na ktoré sú prepojené pomocou odkazov (*Next link*). Tieto stránky mávajú obyčajne rovnakú štruktúru ako hlavná stránka a na ich spracovanie sa môže použiť rekurzia – koreňovému vzoru sa pridá *document filter* sledujúci *Next link*.
    - Použitím filtra dokumentu môžu byť vzory zahŕňajúce informácie z ďalších stránok vytvorené veľmi ľahko. Nie je potrebné robiť žiadnu opakovanú definíciu vzoru na ďalších stránkach a nie je na to potrebné ani procedurálne programovanie. Vzory môžu byť znovu použité pomocou **rekurzie**.

Podmienky

Ak test filtra vyberie príliš veľa inštancií, dá sa tento výber obmedziť použitím podmienok. Slúžia na spresnenie výberu údajov. Lixto používa nasledovné typy podmienok:

- kontextové podmienky – špecifikujú, aký element sa musí alebo nesmie nachádzať pred alebo za týmto vzorom
    - stromová kontextová podmienka – vychádza z HTML štruktúry dokumentu
    - reťazcová kontextová podmienka – s kontextom pracuje ako s reťazcom
- interné podmienky – špecifikujú, aký element sa musí alebo nesmie nachádzať v tomto vzore (stromová, reťazcová)
- podmienka referencie na vzor – používa sa na referenciu na už existujúci súrodenecký vzor v strome vzorov
- podmienka existencie n-tého dieťaťa – používa sa na výber dieťaťa z detí HTML elementu alebo regiónu. Špecifikuje, že jedno z detí musí existovať
- podmienky rozsahu – špecifikovanie rozsahu je jednou z možností pre obmedzenie filtra. Definuje sa, ktoré inštancie v rozsahu filtra majú byť extrahované.

Všetky podmienky poskytujú možnosť použiť regulárne výrazy (používa sa typ výrazov ako v jazyku Perl verzia 5[1]).

Navyše môže operátor vyžadovať zhodu inštancie s výrazom z preddefinovanej ontologickej triedy (uloženej v nejakej databáze) a porovnanie inštancií vzoru (napríklad: inštancia musí byť číslo menšie ako 7).

Je na operátorovi, ktoré podmienky si zvolí na výber želaných údajov. Obyčajne sám vie, ktoré podmienky sú z hľadiska správneho výberu danej inštancie najlepšie. Systém asistuje

1 Perl, výnimočne dynamický programovací jazyk, http://www.perl.org/

operátorovi pri výbere týchto podmienok.

2.2.3 Elog

Na zadefinovanie výberu sa používa extrakčný jazyk Elog. Tento jazyk bol vyvinutý špeciálne na extrakciu údajov z webu. Je to pružný, intuitívny a ľahko rozšíriteľný jazyk podobný jazyku Datalog[2]. Má jasne definovanú sémantiku. Extrahuje informácie na základe okolitých značiek, samotného obsahu údajov, HTML atribútov, poradí výskytu, sémantických a syntaktických konceptov.

V extrakčnom programe Elog sú vzory hierarchicky organizované. Pri vytváraní ďalších vzorov extrakcie sa postupuje výhradne v kontexte aktuálneho vzoru. Táto stromová metóda vytvárania vzorov napomáha vzniku robustných programov obaľovačov, ktoré potom môžu správne pracovať aj nad dokumentami podobnými vzorovému. Hoci sú vzory hierarchicky organizované, nemusia tvoriť stromovú štruktúru a umožňujú tak operátorovi nahradiť implicitnú štruktúru prítomnú vo webovej stránke.

Dôležité je, že návrhár obaľovača (operátor) sa nepotrebuje jazyk Elog učiť, dokonca ho nemusí ani poznať. Všetky vlastnosti sú dostupné cez vizuálne rozhranie.

Sémantický koncept

Pri tvorbe obaľovačov môže návrhár použiť aj sémantické koncepty. Tieto koncepty sú založené na ontológiách. Na ich načítanie a prácu s nimi slúži editor – Semantic Concept Editor. Samotné sémantické koncepty sú uložené v SQL databáze.

Syntaktický koncept

Lixto Visual Wrapper umožňuje použitie syntaktických konceptov na výber určitej syntaxe z webových stránok. Napríklad dátumy, čas alebo čísla sú zapísané v špeciálnom formáte v závislosti od regiónu. Tento formát môže byť uložený ako syntaktický koncept a použitý vo filtri na výber príslušných inštancií z webovej stránky.

2.2.4 XML Tool

Dôležitou časťou systému Lixto Visual Wrapper je nástroj XML Tool. Tento nástroj poskytuje viaceré možnosti práce s úpravou obsahu výstupu do výsledného XML dokumentu. Medzi hlavné funkcie, ktoré nástroj poskytuje, patrí:

- **Transformácia extrahovaných vzorov do výsledného XML dokumentu**
  - Používateľ (operátor) môže určiť, ktoré z extrahovaných vzorov budú zaznamenané vo výslednom XML dokumente. Ďalej môže určiť poradie, v akom sa budú jednotlivé elementy v dokumente vyskytovať a tiež aj namapovať vybrané vzory s nastaveným menom na XML tagy s iným menom.
- **Prezeranie štruktúry obsahu výstupného dokumentu**

2 Datalog, dopytovací jazyk používaný v deduktívnych databázach, v podstate podmnožina jazyka Prolog

- Nástroj umožňuje prezeranie XML štruktúry výsledného dokumentu pre aktuálny obaľovač.

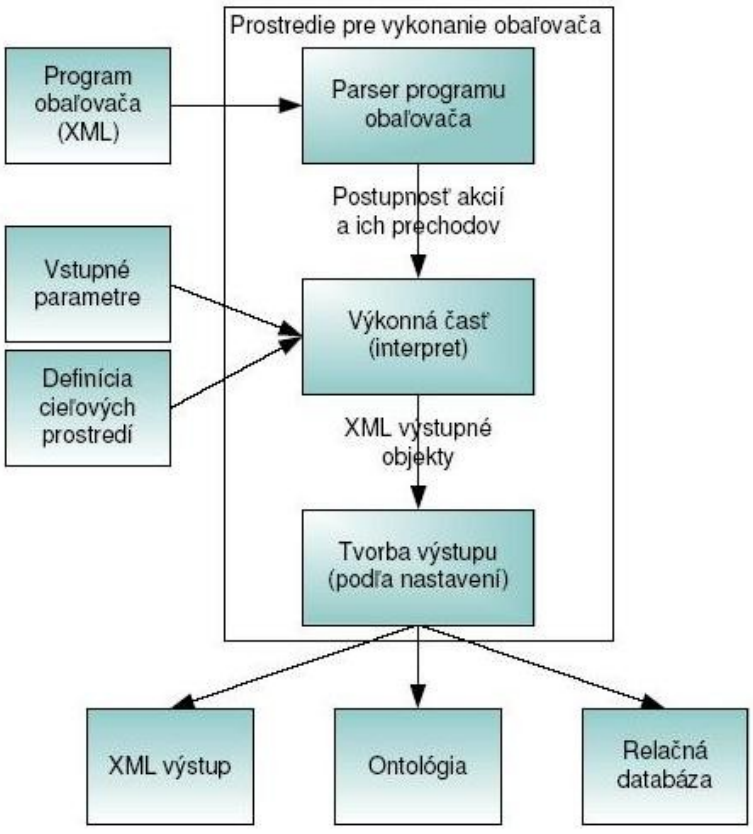
- **Verifikácia**
  - XML Tool poskytuje množstvo ďalších funkcií na stanovenie štruktúry aj obsahu cieľového dokumentu. Patria sem napríklad aj mnohé kontrolné mechanizmy, napr. stanovenie minimálneho alebo maximálneho počtu výskytov daného elementu, hlásenie v prípade zmeny ich počtu a pod.

2.3 Aplikačný rámec Wrapper Suite

Aplikačný rámec Wrapper Suite predstavuje prostredie na tvorbu predprogramovaných obaľovačov. Bol vytvorený ako tímový projekt v akademickom roku 2005/2006. Náš nástroj si kladie za cieľ jeho rozšírenie.

2.3.1 Architektúra

Na Obr. 2-5 je znázornená základná architektúra systému.



Obr. 2-5: Základná architektúra Wrapper Suite [5]

Definícia obaľovača je obsiahnutá v programe obaľovača, kde sú opísané jednotlivé časti obaľovača. Úlohou parsera programu obaľovača je transformovať program obaľovača vo forme XML do vnútornej reprezentácie. Súčasťou vnútornej reprezentácie obaľovača je vytvorenie akcií,

ich zreťazenie pomocou definovaných prechodov, registrovanie štartovacej akcie a tiež aj určenie HTTP klienta pre komunikáciu so serverom a HTML parsera pre získanie stromovej štruktúry (DOM) určeného dokumentu. Interpret zabezpečuje vykonávanie definovaných akcií obaľovača v poradí ako je dané zoznamom prechodov (v programe obaľovača). Vstupné premenné predstavujú informácie potrebné na vykonanie akcií obaľovača (napr. prihlasovacie údaje) a sú uložené v kontextových premenných. Definícia cieľových prostredí určuje špecifikáciu výstupných objektov obaľovača. Modul tvorby výstupu vykonáva zápis extrahovaných dát do výstupných objektov.

2.3.2 Program obaľovača

Program obaľovača sa skladá z:

- identifikácie akcií – definuje typ akcie (navigačná, extrakčná, iteračná, pomocná) a jej atribúty
- definovania prechodov medzi akciami
- premenných – vstupné premenné obaľovača

Príklad programu obaľovača zapísaného vo forme XML sa nachádza na Obr. 2-6. Akcie sú definované v elemente *Actions*, kde pre každú akciu je definované jedinečné meno (*name*) a typ (*type*) akcie. Každá akcia je špecifikovaná definovaním atribútov akcie v elemente *Attribute*, každý atribút obsahuje jedinečné meno (*name*), definovaný typ (*type*) a hodnotu atribútu. Prechody medzi akciami sú definované v elemente *Transitions*, kde pre každý prechod (*Transition*) je určená zdrojová (*src*) a cieľová (*dst*) akcia. Premenné sú definované v elemente *Variables*. Pre každú premennú (*Property*) je určený jedinečný názov (*name*), definovaný type (*type*) a hodnota (*value*) premennej.

```
<Wrapper>
	<Actions>
		<Action name="" id="" type="">
			<Attribute name="" type="">value</Attribute>
			. . .
			<Attribute name="" type="">value</Attribute>
		</Action>
	. . .
	</Actions>
	<Transitions>
		<Transition src="" dst=""/>
		. . .
		<Transition src="" dst=""/>
	</Transitions>
	<Variables>
		<Property name="" type="">value</Property>
		. . .
		<Property name="" type="">value</Property>
	</Variables>
</Wrapper>
```

Obr. 2-6: Príklad programu obaľovača [5]

Akcie obaľovača pracujú nad globálnym kontextom obaľovača (menia ho, využívajú jeho

údaje). Globálny kontext pozostáva z:

- dokumentov – mapa stromovej štruktúry dokumentov (DOM), nad ktorými daná akcia pracuje
- výstupných objektov – štruktúra objektov, ktoré sú získané procesom extrakcie dát
- premenných – parametrizácia akcií počas behu programu. Premenné môžu v sebe obsahovať aj iné premenné
- cookies – zoznam cookies získaných pri navigácií
- autentifikačných dát

Akcie, ktoré pracujú s dátami v dokumente obsahujú lokalizáciu týchto dát v stromovej štruktúre. Na lokalizáciu elementov v stromovej štruktúre dokumentu je určený `TagLocator`, ktorý sa skladá z 3 častí:

- `inDocument` – určuje dokument v kontexte, v ktorom sa bude hľadať,
- XPath výraz – stanovenie cesty v stromovej štruktúre dokumentu,
- regulárny výraz – časť dokumentu získanú aplikovaním XPath výrazu možno filtrovať pomocou regulárneho výrazu.

Navigačné akcie

Akcie, ktoré zabezpečujú prechod na určenú stránku a jej uloženie do globálneho kontextu. Medzi navigačné akcie patrí:

- `LoadPage` – načítanie stránky (určenej pomocou url adresy) do dokumentu
- `FollowLink` – nájdenie odkazu na stránke a prechod na dokument, kde odkaz smeruje. Element odkazu je v dokumente určený pomocou lokátora (`TagLocator`)
- `SubmitForm` – odoslanie formulára určenou metódou (GET, POST), do formulára sú odoslané hodnoty určených vstupných parametrov

Extrakčné akcie

Akcie, ktorých cieľom je získanie požadovaných dát zo stránky:

- `SelectSubDocument` – výber a uloženie do kontextu nového poddokumentu,
- `ExtractData` – extrahovanie a konverzia dát z elementu (určeného lokátorom) do objektu alebo premennej v kontexte,
- `WriteObject` – výstupný objekt z kontextu identifikovaný svojim názvom sa pošle modulu tvorba výstupu.

Iteračné akcie

Akcie, ktoré zabezpečujú opakované vykonávanie určitých operácií:

- `ForEachTag` – pre každý nájdený podstrom (určený lokátorom) sa opakovane vykonajú nasledujúce akcie

- `DoWhileLinkExists` – opakovanie nasledujúcich akcií, pokiaľ existuje určený odkaz

Pomocné akcie

Pomocné akcie sú určené na prácu so zásobníkom kontextov a na vykonávanie všetkých vetiev akcií obaľovača.

2.3.3 Výkonná časť obaľovača

Obaľovač je určený jednotlivými špecifikovanými akciami a ich zreťazením vykonávania. Vnútorná reprezentácia obaľovača sa vytvorí špecifikovaním akcií obaľovača, odkazom na prvú akciu, vytvorením kontextu obaľovača, zoznamu zapisovačov do cieľových prostredí, HTML parsera dokumentu a tiež HTTP klienta pre komunikáciu so serverom. Cieľom výkonnej časti obaľovača je spustenie obaľovača vytvoreného vo forme vnútornej reprezentácie. Spustenie programu obaľovača je zabezpečené spustením štartovacej akcie, ktorá po ukončení svojej činnosti spustí akciu, ktorá nasleduje za danou akciou.

Spracovanie chýb

V akciách obaľovača možno ošetrovať chybové stavy (napr. nenašli sa dáta v dokumente) nasledovným spôsobom:

- `StopThrowErrorHandler` – v prípade vzniku chyby sa ukončí vykonávanie obaľovača
- `ReturnBackErrorHandler` – v prípade vzniku chyby sa nepokračuje vykonávaním nasledovníkov, ale chybový stav je ošetrený v predchodcovi
- `IgnoreContinueErrorHandler` – chybový stav sa ignoruje a pokračuje sa ďalej
- `ExecuteCommandErrorHandler` – v prípade vzniku chyby sa vykoná definovaný príkaz (notifikácia mailom o vzniknutej chybe)

2.3.4 Načítanie obaľovača

Trieda `WrapperLoader` slúži na načítanie programu obaľovača z XML reprezentácie do vnútornej formy skladajúcej sa z navzájom prepojených akcií a kontextu. Na základe typu jednotlivých akcií uvedeného v atribúte *type* elementu *Action* sa volá metóda príslušnej triedy na napĺňanie jej parametrov. Počas vytvárania a napĺňania akcií dochádza zároveň k registrácií akcií k programu obaľovača, k inštancii triedy `WrapperProgram`. Ako prvá sa teda vytvorí trieda `LoadPage` a následne sa volajú jej metódy `setUri` a `setAsDocument` na nastavenie jej základných parametrov.

Vytváranie nových akcií a volanie ich metód je realizované pomocou reflexie, čo zabezpečuje flexibilnosť v prípade, že budeme potrebovať implementovať nové akcie alebo meniť už existujúce. Triedu `WrapperLoader` v týchto prípadoch nie je vôbec potrebné meniť. Potrebná je zmena metódy na napĺňanie údajov a je tiež nutné tieto zmeny zohľadniť aj v XML schéme, ktorou je validovaná vstupná XML reprezentácia. Na jednoduchší prístup k funkciám Java

Reflections API slúži trieda `Reflections`, ktorá poskytuje metódy na vytváranie nových objektov a na volanie funkcií zadaním ich názvu a parametrov.

Vstupný XML súbor obsahuje časť nazvanú *Transitions*, v ktorej sú definované prechody medzi akciami. Na ich základe dochádza k zreťazeniu akcii vo vektore `nextActions`, ktorý obsahuje potomkov každej akcie.

2.3.5 Pomocné triedy

Na serializáciu DOM dokumentov a elementov do XML súboru alebo textového reťazca sú implementované pomocné triedy, využívajúce `XMLSerializer`. Ide o triedu z knižnice projektu Apache XML Project Xerces[3].

- `DocumentFileWriter` - trieda implementujúca zápis DOM dokumentu do XML súboru
- `DocumentStringWriter` - trieda implementujúca zápis DOM dokumentu alebo jeho elementu do textového reťazca obsahujúceho XML kód
- `OutputObjectStringSerializer` - trieda implementujúca zápis výstupného objektu, ktorého obsah je reprezentovaný DOM elementom, do textového reťazca obsahujúceho XML kód

2.3.6 Architektúra zápisu do cieľových prostredí

Z hľadiska implementácie je fáza tvorby výstupu spojená s fázou interpretácie obaľovača. Každý program obaľovača (trieda `WrapperProgram`) obsahuje okrem iného aj zoznam zapisovačov do cieľových prostredí. Je teda možné vykonávať zápis do viacerých cieľových prostredí súčasne a zároveň to dáva priestor na implementáciu a začlenenie vlastných typov zapisovačov.

Naplnené výstupné objekty sa priebežne zapisujú do vyrovnávacej pamäte a na konci behu programu sa pre každý zapisovač asociovaný s obaľovačom vykoná fyzický zápis získaných údajov (`flush`). Aplikácia pracuje s jednotným modelom výstupného objektu a každý zapisovač musí vedieť tento výstupný objekt transformovať do jemu akceptovateľnej podoby prostredníctvom mapovacích tried (`OutputObjectToJobOfferMapper`, `OutputObjectToRDFMapper`).

Obaľovač ponúka rozhranie `OutputWriter` reprezentujúce všeobecný zapisovač získaných údajov do bližšie nešpecifikovaného cieľového prostredia. Z hľadiska údajov sa pracuje na úrovni naplnených výstupných objektov, ktoré sú predávané jeho metódam. Zapisovače do konkrétneho výstupného prostredia dedia jeho hlavné metódy rozhrania:

- `write` – vykoná priebežný zápis naplneného výstupného objektu do interného zoznamu objektov určitého typu v pamäti
- `flush` – vykoná fyzický zápis údajov do daného cieľového prostredia
- `cleanUp` – vykoná vyčistenie vnútornej vyrovnávacej pamäti po realizácii fyzického zápisu

3 Apache XML Project Xerces, http://xerces.apache.org/.

údajov

Zápis na konzolu

O zápis na konzolu sa stará trieda `ConsoleOutputWriter` rozširujúca rozhranie `OutputWriter`. Trieda má len jeden atribút `name` na jednoznačnú identifikáciu zapisovača.

- `write` – serializovaný výstupný objekt na textový reťazec zapisuje na konzolový výstup
- `flush` – výpis konzolovej vyrovnávacej pamäte na konzolu
- `cleanUp` – nerealizuje žiadnu funkcionalitu

Zápis do XML

Na zápis do XML súboru je určená trieda `XMLOutputWriter` obsahujúca nasledovné atribúty:

- `name` - identifikátor objektu zapisovača
- `objectList` – interný zoznam výstupných objektov
- `fileName` – názov výstupného XML súboru
- `rootElementName` – názov koreňového uzla v DOM dokumente

a základné metódy:

- `write` – výstupné objekty pridáva do zoznamu (`objectList`)
- `flush` – vytvorí DOM dokument s koreňovým uzlom *rootElementName*, pre všetky výstupné objekty v zozname sa do dokumentu importuje uzol s príslušnou hierarchickou štruktúrou a nastáva serializácia súboru
- `cleanUp` – maže položky zoznamu `objectList`

Zápis do ontológie

Zápis do ontológie je možné realizovať na úrovni naplnených RDF/XML reťazcov, čo je najnižšia úroveň reprezentácie údajov, závislá od štruktúry ontológie, ale nezávislá od domény, alebo na úrovni naplnených komponentov JavaBeans typu *JobOffer*[4] závislých na doméne, ale nezávislých na štruktúre ontológie.

Zapisovač na úrovni RDF/XML

Zápis do ontológie realizuje trieda `RdfTemplateOntologyOutputWriter` prostredníctvom Sesame API rozhrania ontologického úložiska Sesame[5]. Obsahuje nasledovné atribúty:

- `rdfXmlCodes` – zoznam RDF/XML kódov objektov

4 JobOffer JavaBeans boli dodané tímom číslo 8, ktorý ich vytvoril na základe ontológie projektu NAZOU, http://www2.dcs.elf.stuba.sk/TeamProject/2005/team08/.

5 Sesame, http://www.openrdf.org/index.jsp

- `outObjToRdfMapping` – „hash" mapa získaná mapovacou triedou `OutputObjectRDFMapper`, ktorá umožňuje k výstupnému objektu nájsť odpovedajúcu parametrizovanú RDF šablónu
- `baseUri` – základ URI pre ontologické úložisko
- `sesameRepository` – rozhranie umožňujúce prácu s ontologickým úložiskom
- `sesameService` – administrácia úložiska

Okrem zdedených základných metód implementuje trieda aj `connect` a `disconnect` na pripojenie a odpojenie sa od ontologického úložiska.

Zapisovač na úrovni JavaBeans typu JobOffer

Trieda `JobOfferOntologyOutputWriter` plní funkciu ukladania výstupných objektov do ontologického úložiska a to špecificky pomocou rozhrania `SesameRepositoryAcceess` dodaného tímom 8. Toto rozhranie umožňuje na základe špeciálneho XML súboru transformovať určitý komponent JavaBeans (napr. typu *JobOffer*) na zodpovedajúci Sesame graf RDF trojíc (subjekt, predikát, objekt), ktorý sa následne vkladá do ontológie. Samotná trieda má tri významné atribúty:

- `sesameRepositoryAccess` – rozhranie pre zápis do ontológie
- `graphBeanTransformerFile` – meno transformačného súboru
- `jobOffers` – zoznam vytvorených komponentov JavaBeans

Výstupný objekt sa transformuje na JavaBeans typu JobOffer mapovacou triedou `OutputObjectToJobOfferMapper`. Tak isto okrem zdedených základných metód implementuje trieda aj funkcie `connect` a `disconnect` na pripojenie a odpojenie sa od ontologického úložiska.

3 Opis riešenia

Predmetom tejto kapitoly je opis nami navrhovaného riešenia. V časti 3.1 si vytýčime ciele produktu a vlastnosti, ktorými bude produkt disponovať a časť 3.2 prinesie zoznam funkcionálnych požiadaviek, ktoré sú na systém kladené.

3.1 Ciele a vlastnosti produktu

Úlohou vytváraného softvérového systému je v prvom rade tvorba obaľovačov v prostredí webu. Obaľovanie dokumentov je pomerne komplikovaná činnosť, z čoho vyplýva aj komplikovaná tvorba týchto nástrojov. Vytvorenie jednoduchého obaľovača si vyžaduje pomerne rozsiahle znalosti štruktúry obaľovaných dokumentov, nástrojov na dopytovanie jednotlivých častí týchto dokumentov (napr. XPath v prípade XML dokumentov) a v neposlednom rade aj samotnej činnosti obaľovača (interpretácia programu obaľovača, stavové automaty). Na druhej strane je zrejmé, že obaľované dokumenty sa často menia. Malou zmenou dokumentu (napr. pridanie záhlavia) sa obaľovač stáva nepoužiteľný a vracia nesprávne výsledky. Najjednoduchším výstupom obaľovača je XML súbor, ktorý obsahuje zozbierané výsledky. Keďže však obaľovanie môže a bežne aj produkuje veľké množstvá výsledkov je potrebné, aby bolo možné výstupy generovať do iného úložiska, napríklad do databázy alebo ontológie. Na základe týchto poznatkov môžeme hlavné ciele produktu zhrnúť nasledujúcimi bodmi:

- Zameranie sa na používateľa. Interaktívna tvorba obaľovačov.
- Systém pomáha používateľovi pri tvorbe obaľovačov.
- Tvorba obaľovačov bez nutnej znalosti obaľovaných dokumentov.
- Tvorba obaľovačov bez rozsiahlych znalostí interpretácie jazyka obaľovača. Zvládnutie jazyka obaľovača bez akéhokoľvek štúdia.
- Jednoduchá zmena obaľovača, ak boli obaľované dokumenty zmenené iba preusporiadaním obaľovaných elementov, prípadne pridaním nových neobaľovaných elementov.
- Podpora rôznych výstupov obaľovačov.
- Rovnomerné zaťaženie zdrojov. Množstvo výsledkov, ktoré obaľovač produkuje musí byť zapisované do úložiska postupne, aby nedošlo k zahlteniu pamäti počítača, na ktorom bol obaľovač spustený.
- Doteraz boli spomínané iba funkcionálne ciele systému. Okrem týchto cieľov musí mať vytváraný systém aj nasledujúce nefunkcionálne vlastnosti:
  - Podpora rôznych platforiem. Systém musí byť čo najviac všeobecný.
  - Zachovanie bezpečnosti pri použití citlivých údajov. Niektoré dokumenty vyžadujú autentifikačné údaje.

Cieľom projektu je tiež dosiahnuť architektúru rozšíriteľnú o ďalšie typy reprezentácie dokumentov flexibilitu výmeny vzorov za iné typy pomocou tzv. adapcie.

3.2 Prehľad produktu

Táto kapitola ilustruje očakávané funkcionálne vlastnosti produktu uvedené v predchádzajúcej časti na zozname požiadaviek (Tab. 3-1). Každá požiadavka je definovaná unikátnym identifikátorom, kontextom a oblasťou systému, na ktorú sa vzťahuje, textom požiadavky a vysvetlením požiadavky. Takýto prehľad produktu má oproti častejšie používaným scenárom použitia jednu podstatnú výhodu. Vyznačuje sa vyššou granularitou nastavenou tak, aby sa každá požiadavka vzťahovala iba na jednu oblasť v rámci kontextu použitia. Požiadavky nám naznačujú hrubý pohľad na architektúru systému prostredníctvom kontextu a na prípady použitia prostredníctvom oblastí použitia.

Tab. 3-1: Požiadavky na produkt

ID	Kontext	Oblasť	Text/Vysvetlenie
P1	Jadro	Akcie	V rámci navigácie sa musí vedieť obaľovač pohybovať po stránkach, pracovať s formulármi a cookies, pracovať nad protokolom HTTP a HTTPs.
			-
P2	Jadro	Akcie	Extrakčné akcie získavajú dáta z akejkoľvek časti dokumentu.
			Môže to byť celý element, iba jeho časť, obrázok alebo odkaz na iný typ súboru.
P3	Jadro	Akcie	K extrakčnej akcii je možné definovať výstupný objekt a mapovanie na extrahované dáta.
			-
P4	Jadro	Akcie	Navigačnej formulárovej akcii je možné definovať mapovanie formulárových elementov na vstupné dáta.
			-
P5	Jadro	Vzory	Akcie pracujú nad elementami dokumentu, ktoré vzniknú aplikovaním vzoru na dokument.
			Vzor je súbor filtrov nad dokumentom.
P6	Jadro	Vzory	Nový vzor je možné vytvoriť v dvoch reprezentáciách: XPath a regulárne výrazy (v budúcnosti možno aj viac).
			Nie je možné zmeniť reprezentáciu už vytvoreného vzoru.
P7	Tvorba obaľovača	-	Obaľovač pracuje nad určenou doménou.
			Doména je dôležitá z hľadiska integrácie výsledkov.
P8	Tvorba obaľovača	Učenie	Systém pomáha pri tvorbe vzorov. Používateľ zadáva pozitívne a negatívne príklady.
			Systém sa pokúša zovšeobecňovať (generalizovať) zadávané príklady. Prebieha proces učenia.
P9	Tvorba obaľovača	Učenie	Príklady v procese učenia je možné zadávať interaktívne.
			Interaktívne zadávanie: napr. grafický výber pozitívnych alebo negatívnych elementov v HTML stránke.
P10	Tvorba obaľovača	Úprava obaľovača	Vytvorené obaľovače je možné upravovať.
			Systém poskytuje možnosť upravenia stromu akcií (pridávanie, rušenie , presúvanie akcií), možnosť zmeny subdokumentu,

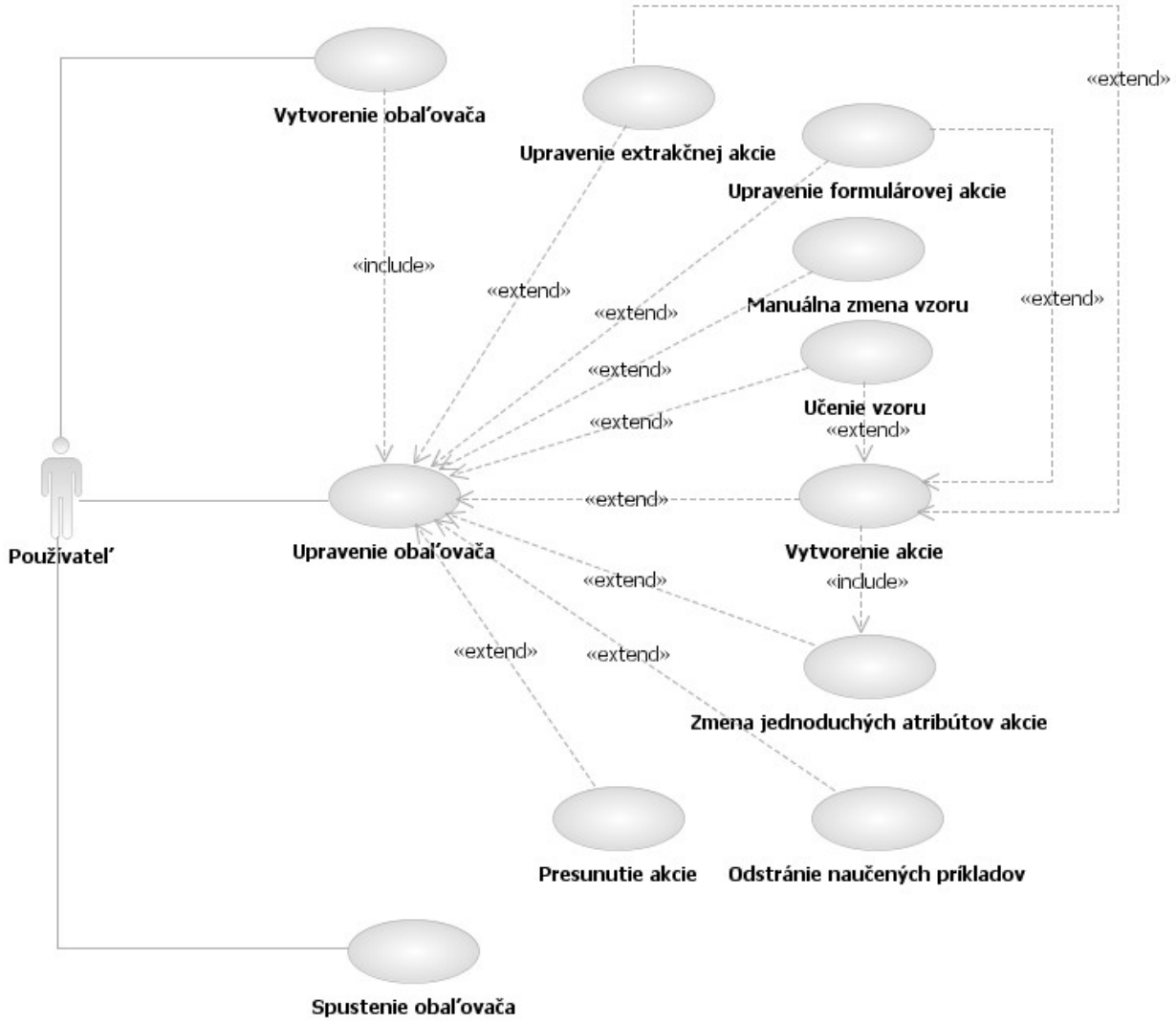
			parametrov akcie.
P11	Tvorba obaľovača	Úprava obaľovača	Pri úprave akcie musí systém upozorniť používateľa na akcie, ktoré mohli byť zmenou ovplyvnené a označí obaľovač ako konfliktný.
			Systém poskytuje používateľovi možnosť označiť obaľovač ako nekonfliktný. Používateľ si prešiel, opravil konflikty a potvrdí, že je všetko v poriadku.
P12	Tvorba obaľovača	Úprava obaľovača	Systém na tvorbu obaľovačov dovoľuje manuálnu zmenu naučeného vzoru (filtrov).
			Systém poskytuje možnosť ako naučený vzor upraviť manuálne (napr. prepísanie XPath výrazu).
P13	Tvorba obaľovača	Úprava obaľovača/ Učenie	Vzor je možné doučiť.
			Ak mal vzor manuálne zmenené filtre, sú nahradené poslednými naučenými, až potom sa začne proces doučenia.
P14	Tvorba obaľovača	Úprava obaľovača/ Učenie	Naučené príklady je možné editovať. Ak používateľ odstráni naučený príklad vzoru, filtre sa dostanú do stavu v akom boli pred naučením tohto príkladu.
			Systém si pamätá históriu učenia. Ku každej zadanej množine príkladov si systém pamätá filtre, ktoré vznikli generalizáciou týchto príkladov.
P15	Činnosť obaľovača	Spúšťanie obaľovača	Ak je obaľovač konfliktný (bola vykonaná riziková akcia P11) systém na to pred spustením obaľovača upozorní používateľa.
			-

4 Špecifikácia obaľovača

V tejto kapitole opíšeme špecifikáciu systému pomocou špecifikácie prípadov použitia, ktoré predstavujú konkrétnu realizáciu funkcionálnych požiadaviek na systém uvedených v časti 3.2. Vytváraný systém na tvorbu obaľovačov je rozšírením existujúceho systému, preto sú vytvorené len prípady použitia, ktoré pokrývajú požiadavky nerealizované existujúcim systémom.

4.1 Prípady použitia

Pri špecifikácii prípadov použitia sme podobne ako pri definovaní požiadaviek vychádzali z už existujúceho systému. V diagrame prípadov použitia (Obr. 4-1) sa nachádzajú iba nové prípady použitia systému alebo prípady použitia, ktoré sú v novom systéme prepracované na základe nových, prípadne pozmenených požiadaviek.



Obr. 4-1: Diagram prípadov použitia

4.2 Popis prípadov použitia

Každý prípad použitia je opísaný tabuľkou. Jednotlivé prípady klasifikujeme prioritou podľa Tab. 4-1.

Tab. 4-1: Priority prípadov použitia

Priorita	**Význam**
1	Vysoká
2	Stredná
3	Nízka

Každý z prípadov použitia je označený jednoznačným unikátnym identifikátorom v tvare UCXX, kde XX je číslo prípadu použitia.

Identifikátor:	UC01	**Názov**:	Vytvorenie obaľovača	**Priorita**:	1
Účel:		Vytvorenie nového obaľovača.			
Vstupné podmienky:		-			
Výstupné podmienky:		Je vytvorený obaľovač so stromom akcií.			
Pokryté požiadavky:		P7			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ zvolí meno, doménu obaľovača, názov výstupného objektu a koreňový dokument (v prípade html - koreňová stránka).		
		2.	Systém vytvorí nový obaľovač a koreňovú akciu (načítanie koreňového dokumentu). Aktuálny subdokument je koreňový dokument a vzor koreňovej akcie je prázdny (bez filtrov).		
		4.	Používateľ upravuje obaľovač (include Úprava obaľovača).		

Identifikátor:	UC02	**Názov**:	Upravenie obaľovača	**Priorita**:	1
Účel:		Úprava existujúceho stromu akcií vybraného obaľovača.			
Vstupné podmienky:		-			
Výstupné podmienky:		Obaľovač zvolený používateľom je upravený.			
Pokryté požiadavky:		P11, ostatné požiadavky úpravy pokrývajú rozširujúce UC			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ nemá zvolený žiaden aktuálny obaľovač. Systém ponúkne používateľovi zoznam obaľovačov.		
		2.	Používateľ si zvolí obaľovač, ktorý chce upravovať.		
		3.	Systém ponúkne používateľovi strom akcií obaľovača.		
		4.	Používateľ si opakovane vyberá činnosť (extension point výber činnosti).		
		5.	Obaľovač nemá konflikty. Koniec UC.		

Alternatívy:	1a.	Používateľ má zvolený aktuálny obaľovač. Choď na krok 3 v základnom toku.
	1b.	Používateľ zadá obaľovač zo súboru.
	5a.	Obaľovač má konflikty, ale používateľ jeho správnosť nepotvrdí. Koniec UC.
	5b.	Obaľovač má konflikty. Používateľ potvrdí jeho správnosť.

Identifikátor:	UC03	**Názov**:	Spustenie obaľovača	**Priorita**:	3
Účel:		Spustenie obaľovača. Účel tohto UC sa týka iba správania sa systému pri spúšťaní obaľovača s konfliktami. Ostatná funkcionalita (krokovanie, spúšťanie z príkazového riadku) je už implementovaná v existujúcom systéme.			
Vstupné podmienky:		-			
Výstupné podmienky:		Obaľovač vykonáva svoju činnosť.			
Pokryté požiadavky:		P15			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ nemá zvolený žiaden aktuálny obaľovač. Systém ponúkne používateľovi zoznam obaľovačov.		
		2.	Používateľ si zvolí obaľovač, ktorý chce upravovať.		
		3.	Systém zistil, že obaľovač nemá žiadne konflikty, uvedie obaľovač do činnosti.		
Alternatívy:		1a.	Používateľ má zvolený aktuálny obaľovač. Choď na krok 3 v základnom toku.		
		1b.	Používateľ zadá obaľovač zo súboru.		
		3a.	Systém je v interaktívnom móde. Obaľovač má nejaké konflikty. Systém to oznámi používateľovi a spustí obaľovač.		
		3b.	Systém nie je spustený z príkazového riadku. Obaľovač má nejaké konflikty. Systém to zaznamená do log súboru a spustí obaľovač.		

Identifikátor:	UC04	**Názov**:	Vytvorenie novej akcie	**Priorita**:	1
Účel:					
Vstupné podmienky:		Používateľ má vybranú akciu, ktorej chce vytvoriť novú dcérsku akciu.			
Výstupné podmienky:		Bola vytvorená akcia, ktorá má vytvorený nový lokálny kontext s novým naučeným vzorom.			
Pokryté požiadavky:		P1, P2, P5, P6			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ vybraná akcia je rozšíriteľná. Systém povolí používateľovi novú akciu.		
		2.	Používateľ si vyberie typ vzoru (napr. XPath alebo regulárne výrazy), ktorý bude používaný v rámci akcie. Nová akcia zdedí kontext, ktorého aktuálny subdokument vznikol aplikovaním vzoru rodičovskej akcie a transformáciou novým vzorom. Vzor akcie je prázdny.		
		3.	Systém ponúkne používateľovi úpravu nového vzoru (include Naučenie vzoru).		
		4.	Používateľ si vybral vytvorenie akcie, ktorá nie je extrakčná ani		

		formulárová.
	5.	Používateľ nastaví jednoduché atribúty akcie (include Zmena jednoduchých atribútov akcie)
Alternatívy:	1a.	Akcia je jednoduchá (nerozšíriteľná). Systém znemožní používateľovi zvolenie akcie. Späť na krok 1 v základnom kroku.
	3a.	Nový vzor zostane prázdny. Akcia pracuje so zdedeným subdokumentom.
	4a.	Používateľ si zvolil extrakčnú akciu.
	4a.1.	Systém vytvorí prázdnu extrakčnú akciu a mapovanie na výstupný objekt (extend Upravenie extrakčnej akcie).
	4b.	Používateľ si zvolil formulárovú akciu.
	4b.1	Systém vytvorí prázdnu formulárovú akciu a mapovanie na formulárové dáta (extend Upravenie formulárovej akcie)

Identifikátor:	UC05	**Názov**:	Učenie vzoru	**Priorita**:	1
Účel:		Naučenie vzoru pomocou pozitívnych a negatívych príkladov.			
Vstupné podmienky:		Je vybraná akcia, ktorej lokálny kontext obsahuje vzor.			
Výstupné podmienky:		Vzor má priradené/upravené filtre. Vzor si pamätá históriu učenia.			
Pokryté požiadavky:		P8, P9, P13			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ manuálne nezmenil filtre vzoru. Systém neupraví existujúce filtre.		
		2.	Systém grafický vyznačí aktuálny vzor v dokumente.		
		3.	Používateľ označí príklady, ktoré ho zaujímajú (pozitívne) a príklady, ktoré ho nezaujímajú (negatívne). Používateľ nevybral príklady , ktoré už v trénovacej množine sú.		
		4.	Systém generalizuje vzor na základe príkladov. Aktuálne filtre si systém asociuje s množinou pridaných príkladov. Tieto príklady si vloži do histórie učenia. Späť na krok 2.		
Alternatívy:		1a.	Používateľ manuálne zmenil filtre vzoru. Systém nahradí manuálne zmenené filtre filtrami z poslednej histórie učenia.		
		3a.	Medzi vyznačenými príkladmi sa nachádza príklad, ktorý sa už nachádza v trénovacej množine príkladov. Systém na to upozorní používateľa a zvolený príklad nepridá do trénovacej množiny.		
		3b.	Používateľ je spokojný so vzorom. Ukončí učenie. Ak bol vzor zmenený podstrom akcií označí systém ako konfliktný. Koniec UC.		

Identifikátor:	UC06	**Názov**:	Manuálna zmena vzoru	**Priorita**:	2
Účel:		Manuálna zmena naučených filtrov			
Vstupné podmienky:		Používateľ má zvolenú akciu a jej vzor, ktorý má priradené filtre			
Výstupné podmienky:		Filtre vzoru boli upravené. Vzor si pamätá všetky naučené filtre.			
Pokryté požiadavky:		P12			
		Krok	**Činnosť**		
Základný tok:		1.	Systém grafický vyznačí aktuálny vzor v dokumente.		

	2.	Používateľ ručne zmení naučené filtre. Späť na krok 1.
Alternatívy:	2a.	Používateľ je spokojný so vzorom. Ak bol vzor zmenený podstrom akcií označí systém ako konfliktný. Koniec UC.

Identifikátor:	UC07	**Názov**:	Úprava formulárovej akcie	**Priorita**:	2
Účel:		Vytvoriť mapovanie subdokumentu na formulárové dáta.			
Vstupné podmienky:		Používateľom zvolená akcia je formulárová.			
Výstupné podmienky:		Formulárová akcia má priradené mapovanie na formulárové dáta.			
Pokryté požiadavky:		P4			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ vytvára kontextové parametre a vkladá ich hodnoty.		
		2.	Systém postupne ponúka používateľovi vstupné formulárové elementy subdokumentu.		
		3.	Pre každý element vyberie používateľ zodpovedajúcu kontextovú premennú, ktorej hodnota sa použije pri vyplnení formulára.		
Alternatívy:		1a.	V kontexte sa nachádzajú všetky potrebné premenné. Choď na krok 5 v základnom toku.		

Identifikátor:	UC08	**Názov**:	Úprava extrakčnej akcie	**Priorita**:	1
Účel:		Namapovať subdokument (element zoznamu subdokumentov, ktorý vznikne aplikovaním vzoru na rodičovský subdokument) na výstupný objekt.			
Vstupné podmienky:		Používateľom zvolená akcia je extrakčná.			
Výstupné podmienky:		Extrakčná akcia má priradené mapovanie na výstupný objekt.			
Pokryté požiadavky:		P3			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ zadáva premenné do kontextu, ktorých hodnoty chce naplniť používateľskými dátami.		
		2.	Používateľ zadá cestu vo výstupnom objekte, do ktorej sa zapíše extrahovaný subdokument.		
Alternatívy:		2a.	Používateľ zadá premenné kontextu, do ktorých sa zapíše extrahovaný subdokument.		
		2b.	Používateľ zadá aj premenné kontextu aj cestu vo výstupnom objekte, do ktorých sa zapíše extrahovaný subdokument.		

Identifikátor:	UC09	**Názov**:	Presunutie akcie	**Priorita**:	3
Účel:		Zmena rodičovskej akcie			
Vstupné podmienky:		Používateľ má zvolenú akciu.			
Výstupné podmienky:		Presúvaná akcia je dcérskou akciou inej akcie. Presunutý je aj podstrom akcie			
Pokryté požiadavky:		P10, P11			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ si zvolí presúvanú a cieľovú akciu (systémom copy&paste alebo drag&drop)		

	2.	Cieľová akcia môže byť kompozitná. Systém presunie akciu aj s jej podstromom.
	3.	Systém označí presunutý podstrom ako konfliktný.
Alternatívy:	2a.	Cieľová akcia nemôže mať dcérske akcie. Koniec UC.

Identifikátor:	UC10	**Názov**:	Zmena jednoduchých atribútov akcie	**Priorita**:	2
Účel:		Zmena atribútov akcie, ktoré majú jednoduchú hodnotu – žiadne štruktúrované dáta.			
Vstupné podmienky:		Používateľ má zvolenú akciu.			
Výstupné podmienky:		Akcia má zmenené atribúty.			
Pokryté požiadavky:		P10			
		Krok	**Činnosť**		
Základný tok:		1.	Používateľ nastaví nové hodnoty atribútov, pridá nové atribúty alebo odstráni existujúce atribúty		
		2.	Systém zistil, že zmena atribútov môže porušiť podstrom akcií. Označí tento podstrom ako konfliktný.		
Alternatívy:		2a.	Zmena atribútov nemôže porušiť podstrom akcií. Systém neoznačí podstrom ako konfliktný		

Identifikátor:	UC11	**Názov**:	Odstránenie naučených príkladov	**Priorita**:	3
Účel:		Odstránenie naučenia vybraných príkladov			
Vstupné podmienky:		Používateľ má vybranú akciu a vzor, ktorému bude odstraňovať príklady.			
Výstupné podmienky:		Vzor je v stave(má zhodné filtre) v akom bol pred naučením najstaršieho z odstránených príkladov.			
Pokryté požiadavky:		P14			
		Krok	**Činnosť**		
Základný tok:		1.	Systém ponúkne používateľovi zoznam všetkých naučených príkladov, ktorými vznikol vzor.		
		2.	Používateľ si vyberie príklady na zmazanie.		
		3.	Systém zobrazí používateľovi príklady na zmazanie a subdokument, ktorý vyfiltruje vzor po zmazaní príkladov.		
		4.	Používateľ potvrdí zmazanie.		
		5.	Systém vymaže vybrané vzory a filtre nastaví do stavu v akom boli pred naučením najstaršieho z odstránených príkladov (informácia z histórie učenia)		
Alternatívy:		4a.	Používateľ nepotvrdí zmazanie a chce zmeniť výber príkladov. Späť na krok 2.		

5 Použité technológie

Jedným z cieľov tohto projektu je priblíženie tvorby obaľovačov rádovým používateľom. Keďže systém kladie dôraz na interaktivitu s používateľom, je nutné obaľovač zakomponovať do takého konceptu, ktorý používateľovi ponúkne jednoduché rozhranie bez nutnosti mohutných zmien (inštalácie) v systéme. Ako najvhodnejšie riešenie sa javí použitie prehliadača Mozilla Firefox a možnosti jeho rožšírenia (ide o voľné dostupný prehliadač s otvoreným programovým kódom).

S návrhom a implementáciu rozšírení úzko súvisia technológie XUL a XPCOM, ktoré sú opísané v nasledujúcich častiach.

5.1 Jazyk XUL

XUL[6] je značkovací jazyk, ktorý slúži na vytváranie používateľských rozhraní. Je dôležitou súčasťou projektu Mozilla, ktorý ho využíva v širokom spektre. Výhodou tohto jazyka je jeho prenosnosť. To znamená, že ho môžme aplikovať vo väčšine existujúcich operačných systémoch, od systému Windows až po systémy Linux. Táto technológia umožňuje tvorbu sofistikovaných aplikácií bez potreby použitia špeciálneho nástroja. Využíva pritom už existujúce ovládacie prvky (kontajnery, menu, panel nástrojov, ...).

5.1.1 Použitie jazyka XUL

Použitie jazyka XUL v systéme Mozilla sa predovšetkým zameriava na implementáciu používateľského rozhrania vytváraných prídavných modulov alebo aplikácií, ktoré uľahčujú prácu, prípadne rozširujú funkcionalitu tohto systému. Nesmieme však zabúdať aj na to, že pomocou tohto jazyka a kaskádnych štýlov CSS vieme vytvárať dizajn webových stránok, ktoré je možné zobrazovať pomocou webového prehliadača Mozilla Firefox.

Jazyk XUL podporuje aj dynamické vytváranie daného kontextu. Vďaka možnosti prepojenia so skriptovacím jazykom JavaScript sme schopní vytvoriť dynamické používateľské rozhranie. Pomocou neho dokážeme modifikovať celý vzhľad tohto rozhrania na základe interakcie používateľa.

5.1.2 Štruktúra jazyka XUL

XUL je stromovo-orientovaný štruktúrovaný jazyk, založený na metajazyku XML. Celý kontext, vytvorený v tomto jazyku, je uložený v textovom súbore (súbor s príponou .xul), čo uľahčuje prácu pri definícii používateľských rozhraní (stačí jednoduchý textový editor).

Základná štruktúra XUL súboru sa skladá z XML hlavičky a značiek, ktoré obsahujú definované atribúty a udalosti. Tieto značky môžme rozdeliť do troch skupín:

- Kmeňové elementy
- Elementy rozloženia

6 XUL, špecifikácia dostupná na http://www.mozilla.org/projects/xul/

- Vizuálne prvky

Kmeňové elementy

Kmeňové elementy tvoria základ každého používateľského rozhrania. Bez nich by ho nebolo možné vytvoriť. Kmeňové elementy sa používajú na definovanie typu zobrazenia vytváraného rozhrania (aplikačné okno, dialógové okno) a obsahujú zoznamy elementov (elementy rozloženia, kontajnery, vizuálne prvky) na nižších vrstvách v stromovej štruktúre dokumentu XUL. Medzi najpoužívanejšie kmeňové elementy patria:

- *window* - používa sa pri tvorbe hlavného okna alebo webových aplikácii
  - obsahuje atribúty, ktoré definujú pozíciu, názov okna a iné
- *dialog* - vytvorí dialógové okno, závislé od danej platformy
- *page* - slúži na vnorenie daného dokumentu do iného

Elementy uloženia

Tieto elementy slúžia na umiestenie skupiny kontajnerov a vizuálnych prvkov do riadku, stĺpca alebo do tabuľky. V XUL dokumente sa môžu vyskytovať v tele kmeňových elementov. Najpoužívanejšími elemantami uloženia sú:

- *vbox*, *hbox* - prvky budú uložené v stĺpci, resp. v riadku
- *stack* - prvky budú uložené nad sebou, všetky sa zobrazia
- *deck* - podobne ako *stack*, ale vždy sa zobrazí iba jeden prvok
- *tabbox* - manažér záložiek, obsahuje elementy *tabs* a *tabpanels*
- *grid* - umiestni prvky do tabuľky

Vizuálne prvky

Poslednou skupinou elementov sú vizuálne prvky. Sú to klasické ovládacie prvky, s ktorými sa stretávame pri tvorbe používateľského rozhrania v rôznych vývojových prostrediach. Nie je tomu inak ani pri jazyku XUL. Medzi najpoužívanejšie môžme zahrnúť:

- *label* - element na zobrazenie textu
- *textbox* - ovládací prvok na vkladanie používateľom zadaného textu
- *listbox* - prvok na zobrazenie zoznamu

5.2 Komponentový objektový model XPCOM

Multiplatformový objektový model XPCOM[7] je rámec dovoľujúci rozdeliť jednoliate a masívne softvérové projekty na malé modularizované časti. Tieto časti – komponenty – sú opäť zostavené dohromady v priebehu behu aplikácie.

7 XPCOM špecifikácia dostupná na http://www.mozilla.org/projects/xpcom/

Účelom modelu XPCOM je možnosť nezávislého vývoja jednotlivých softvérových súčastí bez nutnosti ich vzájomného spojenia. XPCOM oddeľuje implementáciu komponentu od jeho rozhrania, čím je dosiahnutá znovupoužiteľnosť v rôznych aplikáciách a jednoduchá možnosť zmeny funkcionality existujúcich aplikácií.

XPCOM je úzko spojený s projektom Mozilla, keď bol vytvorený za účelom poskytnutia prístupu k funkcionalite zobrazovacieho jadra Gecko a možnosti jeho nenáročného rozširovania. Napríklad samotný prehliadač Mozilla Firefox je vo svojej podstate tiež iba súborom komponentov. Každé tlačidlo, každé menu, jednoducho každý element je samostatný komponent, čo robí prehliadač ľahko rozšíriteľným pri zachovaní pôvodnej štruktúry. Výsledkom toho je aj množstvo tzv. rozšírení (angl. addons), ktoré prispievajú k jeho narastajúcej popularite.

Úloha komponentového objektového modelu XPCOM je zrejmá – spolu s jazykom XUL (ale nie len ním) poskytuje možnosti rozšírenia funkcionality prehliadača (a iných aplikácií projektu Mozilla) bez nutnosti zmeny jeho programového kódu.

5.2.1 Rozhrania

Pojem rozhranie je v prípade komponentového modelu kľúčový. Prostredníctvom rozhraní zabezpečíme zapuzdrenie použitých tried, čím jednoznačne definujeme metódy, ktoré môžu byť nad komponentmi použité. Každá trieda v modeli XPCOM je odvodená od elementárneho rozhrania `nsISupports` (Obr. 5-1) .

```
[scriptable, uuid(00000000-0000-0000-c000-000000000046)]
interface nsISupports {
  void QueryInterface(in nsIIDRef uuid,
                      [iid_is(uuid),retval] out nsQIResult result);
  [noscript, notxpcom] nsrefcnt AddRef();
  [noscript, notxpcom] nsrefcnt Release();
};
```

Obr. 5-1: Rozhranie `nsISupports` definované jazykom XPIDL[8]

Rozhranie `nsISupports` pomocou uvedených metód adresuje dve fundamentálne aspekty komponentového programovania – dĺžku života komponentu a tzv. „dopytovanie rozhrania", t.j. schopnosť identifikácie komponentu s podporovaným rozhraním. Obe metódy tvoria základ procesu volania komponentu napr. z kódu rozšírenia prehliadača.

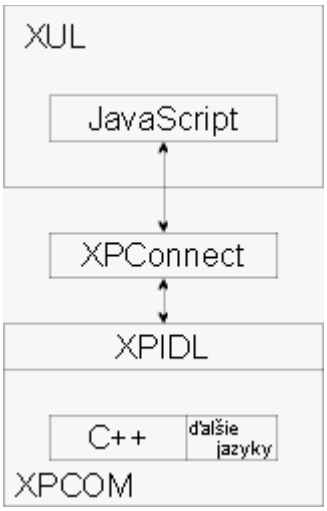
5.2.2 XPConnect

XPConnect je vrstva modelu XPCOM, ktorá slúži na prepojenie XPCOM komponentov a skriptovacieho jazyka, akým je napr. JavaScript (JavaScript je súčasťou balíkov rozšírení prehliadača Mozilla Firefox). Schématický náčrt architektúry je zobrazený na Obr. 5-2.

Prístup ku komponentom pomocou XPConnect by sa dal zhrnúť do troch krokov [6]:

8 Jazyk XPIDL je mutáciou jazyka IDL, slúži na opi rozhraní softvérových komponentov; referencie dostupné na: http://developer.mozilla.org/en/docs/XPIDL

1. získanie objektu komponentu
2. prístup k rozhraniu
3. volanie požadovanej metódy



Obr. 5-2: Architektúra XPCOM [7]

Po vykonaní prvých dvoch krokov môžeme ten tretí opakovať koľkokrát to bude nutné. Uvedenú postupnosť demonštruje Obr. 5-2, ktorý uvádza volanie Mozilla-zabudovaného komponentu `nsILocalFile`.

```
var aFile =
  Components.classes["@mozilla.org/file/local;1"].createInstance();
if (aFile instanceof Components.interfaces.nsILocalFile){
  aFile.initWithPath("/mozilla/testfile.txt");
  aFile.remove(false);
}
```

Obr. 5-3: Stručný príklad prístupu ku komponentu pomocou XPConnect

V uvedenom príklade je ukážka vymazania súboru pomocou JavaScript kódu. Na začiatku dôjde k inicializácii premennej `aFile`, do ktorej sa uloží inštancia komponentu súboru. Na ďalšom riadku sa realizuje prístup k rozhraniu `nsILocalFile`. Zvyšná časť kódu sú metódy, ktoré zabezpečia samotnú funkcionalitu nad súborom uvedeným ako argument.

5.2.3 JavaXPCOM

Zatiaľ nebolo spomenuté, že XPCOM implicitne podporuje jazyk C++. Pre tvorbu nepôvodných Mozilla komponentov v inom programovacom jazyku je potrebná nadstavba v podobe externého modulu, v našom prípade JavaXPCOM. JavaXPCOM slúži ako most medzi

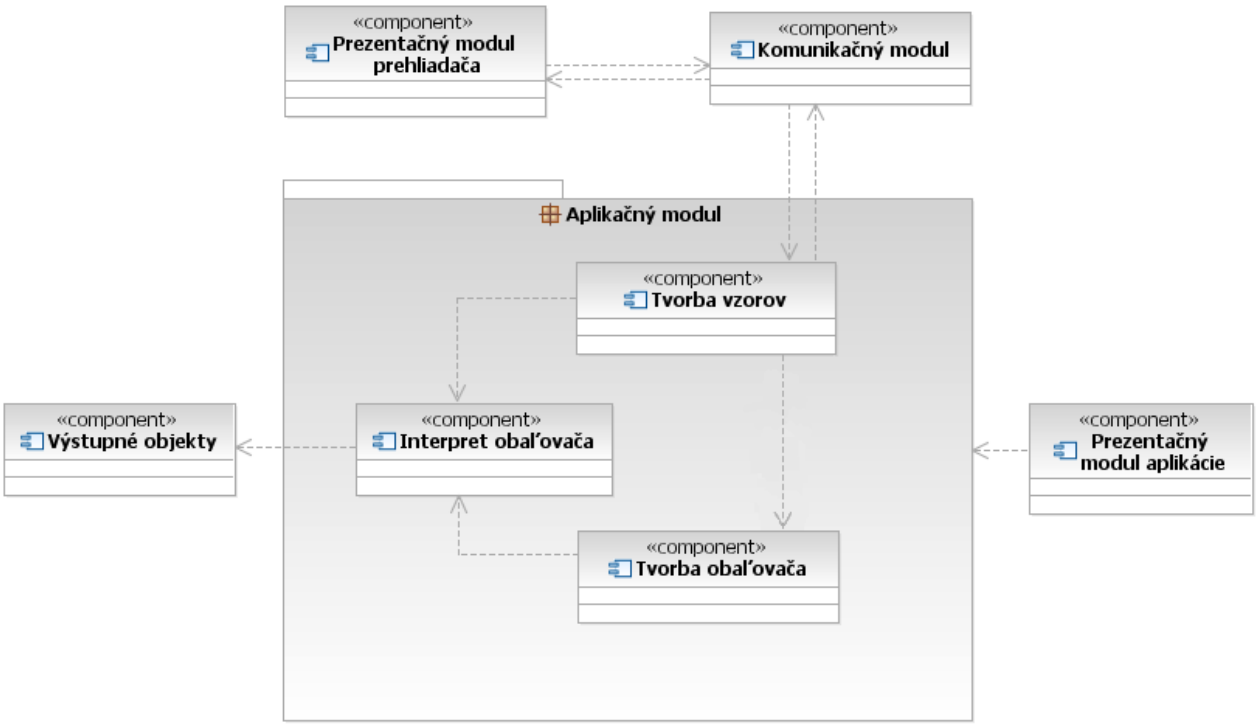
modelom XPCOM a s ním súvisiacou funkcionalitou na jednej strane a programovým kódom v jazyku Java na strane druhej.

6 Hrubý návrh systému

Kapitola 6 prezentuje hrubý návrh systému. Na základe analýzy existujúcich riešení a využiteľnosti nových technológii navrhujeme architektúru systému, ktorá zodpovedá špecifikácii požiadaviek na systém (časť 6.1). Na základe architektúry postupne v ďalších siedmich častiach opisujeme hrubý náčrt jednotlivých komponentov – modulov systému. Kapitolu uzatvára časť 6.9, v ktorej uvádzame použité implementačné prostriedky a vývojové nástroje.

6.1 Architektúra systému

Základná architektúra systému sa nachádza na Obr. 6-1.



Obr. 6-1: Architektúra obaľovača

Opis návrhu každého z modulov je predmetov ďalších častí tejto kapitoly.

6.2 Prezentačný modul aplikácie

Prezentačný modul aplikácie slúži ako rozhranie medzi používateľom a obaľovačom. Umožňuje tak riadenie obaľovača na základe interakcie používateľa. Hlavnou úlohou je distribúcia získaných údajov pre modul tvorby vzorov, ale aj prezentácia spracovaných údajov z obaľovača. Používateľ bude môcť zasahovať do nasledujúcich činností:

- vytvorenie obaľovača

- upravenie obaľovača
- spustenie obaľovača

6.2.1 Proces tvorby obaľovača

Na začiatku tvorby obaľovača je potrebné, aby používateľ vykonal jeho inicializáciu. V tomto prípade používateľ vyplní formulár, ktorý zahŕňa nasledujúce informácie:

- názov obaľovača
- doména obaľovača
- názov výstupného objektu
- odkaz na koreňový dokument

V prípade, že koreňovým dokumentom je webová stránka, je potrebné zadať URL adresu. Túto adresu je možné zadať ručne, alebo pomocou interakcie používateľa v prehliadači Mozilla Firefox (toto spravuje prezentačný modul prehliadača a komunikačný modul).

Po zadaní inicializačných údajov môže používateľ daný obaľovač rôzne konfigurovať. Vykonáva to cez používateľské rozhranie, kde si môže vytvárať a modifikovať celý strom akcií (formulár bude tento strom zobrazovať a používateľ ho môže modifikovať – tvorba, rušenie a presun akcií), ktoré sa majú vykonať. Každá akcia obsahuje určité parametre, ktoré môže používateľ podľa potreby meniť (zobrazenie príslušného formulára nastavení po vybratí danej akcie v strome akcií). V prípade extrakčných akcií je potrebné zadefinovať výstupný objekt, do ktorého sa budú ukladať vydolované údaje (výberom objektu zo zoznamu). Ak pracujeme s navigačnými formulárovými akciami, musíme zabezpečiť namapovanie formulárových elementov na vstupné údaje (rozhranie, kde používateľ priradí ku každému typu vstupného údaja práve jeden element vo formulári). Niektoré akcie si vyžadujú zásah používateľa. Jednou z nich je aj akcia učenia vzorov, kde používateľ určí príklady toho, čo chce, prípadne nechce získať (pozitívnych, resp. negatívnych príklad). Toto určovanie sa vykonáva pomocou prezentačného modulu prehliadača, ktorý je popísaný v kapitole 6.3.

Medzi ďalšie funkcie prezentačného modulu aplikácie patrí aj odstránenie naučených príkladov a manuálna zmena vzorov. V prípade manuálnej zmeny vzorov si používateľ vyberie daný vzor (XPath výraz alebo regulárny výraz) zo zoznamu a tento sa mu zobrazí. Ak ho potrebuje zmeniť, môže to vykonať. V prípade odstránenia naučených príkladov si používateľ vyberie určitú akciu a vzor (zo stromu akcií), ktorému chce odstrániť príklady, a potvrdí zrušenie.

6.2.2 Proces úpravy obaľovača

Úprava obaľovača obsahuje rovnakú podskupinu činností, ktoré sa vyskytujú aj počas tvorby obaľovača. Nie je preto dôležité opisovať tieto činnosti a návrh rozhrania ešte raz. Tieto informácie sú dostupné v predchádzajúcej časti. Jediným rozdielom je začiatok vykonávania týchto činností. Pri tvorbe obaľovača musel používateľ uviesť inicializačné údaje o obaľovači. V prípade úpravy už nemusí zadávať počiatočné informácie, ale vyberie si už vytvorený obaľovač a ten

modifikuje. Systém mu ponúkne zoznam existujúcich obaľovačov a používateľ postupuje rovnako ako v prípade jeho tvorby (tvorba, rušenie a presun akcií).

6.2.3 Spustenie obaľovača

Ak chce používateľ dolovať údaje, musí spustiť obaľovač. Tu mu systém ponúkne, ktorý obaľovač chce použiť (zoznam všetkých definovaných obaľovačov). Používateľ si jeden vyberie a môže spustiť režim behu alebo ladenia. Počas spracovania sa mu vo formuláre zobrazujú výsledky dolovania, prípadne dodatočné informácie pri režime ladenia.

6.3 Prezentačný modul prehliadača

Prezentačný modul prehliadača sa využíva na uľahčenie práce pri tvorbe obaľovača. Slúži ako nadstavba prezentačného modulu aplikácie a využívajú sa tu funkcie systému Mozilla Firefox. Prezentačný modul prehliadača pracuje na princípe rozšírenia tohto systému.

Jeho prvou úlohou pri tvorbe obaľovača je získanie adries koreňových dokumentov (URL) a zaslanie tejto informácie prezentačnému modulu aplikácie. Túto adresu zadá používateľ do okna prehliadača, prezentačný modul prehliadača (rozšírenie systému Mozilla Firefox) ju vhodne zakóduje a vyšle do komunikačného modulu. Ďalšou úlohou prezentačného modulu je získavanie pozitívnych, prípadne negatívnych príkladov. Tie slúžia pri procese učenia vzorov. Používateľ si vyberie určitý element dokumentu a prezentačný modul túto operáciu zaznamená. Zistí, o ktorý element sa jedná a cestu k nemu pošle prezentačnému modulu aplikácie (cez komunikačný modul), ktorý ho spracuje. Táto cesta sa reprezentuje pomocou XPath alebo regulárnych výrazov. Poslednými úlohami tohto modulu je zobrazenie určitej časti dokumentu (subdokument), ktorá bližšie špecifikuje oblasť požadovaných (dolovaných) údajov, a grafické označenie aktuálneho vzoru v dokumente.

6.4 Komunikačný modul

Návrh komunikačného modulu vychádza z analýzy komponentového objektového modelu XPCOM v časti 5.2. Keďže je nutné zabezpečiť komunikáciu medzi prezentačným modulom prehliadača a samotnou aplikáciou obaľovača (založenou na platforme Java), bude využitá aj nadstavba JavaXPCOM.

Hlavnou úlohou komponentového modelu XPCOM bude priniesť rozšíreniu do prehliadača Mozilla Firefox (t.j. prezentačnému modulu prehliadača) takú funkcionalitu, na ktorú JavaScript a XUL nestačia. Pôjde o využitie funkcionality obaľovača, ktorý slúži na dolovanie dát z webu.

Komunikačný modul kopíruje schému uvedenú na Obr. 5-2. Na jednej strane bude definované rozhranie jednotlivých Java tried aplikačného rámca WrapperSuite, na strane druhej Mozilla Firefox rozšírenie tvorené jazykom XUL a JavaScript-om. Komunikácia bude prebiehať v dvoch smeroch:

- prezentačný modul prehliadača -> aplikačný modul

- aplikačný modul -> prezentačný modul prehliadača

V prvom prípade prezentačný modul prehliadača Mozilla Firefox, ktorý zabezpečuje interaktivitu nad HTML stránkami, bude aplikačnej časti poskytovať informácie o tom, aké pozitívne, resp. aké negatívne príklady boli vybrané. Tie budú reprezentované XPath výrazmi, resp. regulárnymi výrazmi. Taktiež budú prenášané informácie o navigačných akciách, tj. o pohybe v priestore webu.

Druhý prípad bude prenášať informácie z aplikačného modulu smerom do rozšírenie. V tejto časti bude nutné zabezpečiť prispôsobovanie sa obsahu prehliadača požiadavkám, ktoré definuje používateľ prostredníctvom obaľovača. V prevažnej miere pôjde o zmeny nad DOM modelom zobrazovaných stránok. Predpokladáme využitie W3C DOM Connectora [9]. W3C Connector je v podstate balík jazyka Java, ktorý sa používa na prístup k DOM stromu prehliadača Mozilla Firefox z platformy Java, pretože implementuje štandardné rozhrania org.w3c.*. Znamená to, že bude možné efektívne dopytovanie nad DOM stromom prostredníctvom robustných XPath nástrojov, akými sú napr. Xerces alebo Saxon.

6.5 Aplikačný modul – Interpreter

Spúšťanie a vykonávanie akcií používateľom definovaného obaľovača zabezpečuje Interpereter. Vykonávanie začína spustením operácií štartovacej akcie, ktorá predstavuje načítanie požadovanej web stránky a inicializácia kontextu obaľovača. Po ukončení aktuálnej akcie Interpreter automaticky spusti vykonanie všetkých ďalších používateľom naplánovaných akcií (NextAction). Interpreter pre nasledujúce akcie inicializuje lokálny kontext vzhľadom na predchádzajúcu akciu.

Z hľadiska architektúry systému (časť 6.1) Interpreter vykonáva akcie vytvorené v module Tvorba obaľovača a v lokálnych kontextoch akcií používa vzory z modulu Tvorba vzorov. Výsledky svojej práce, extrahované dáta, odovzdáva Interpreter modulu Výstupné objekty (všetky uvedené moduly budú popísané neskôr).

Interpreter je spustený cez API Dialog a umožňuje okrem normálneho spôsobu spustenia dolovania aj proces dolovania v ladiacom režime, kedy je možné beh akcií krokovať.

6.5.1 Kontext

Všetky akcie súčasnej implementácie Wrapper Suite pracujú s globálnym kontextom. Našim cieľom je prejsť na systém tzv. lokálneho kontextu, teda každá ďalšia nasledovná akcia v strome akcii bude pracovať už len so svojim lokálnym kontextom, nezávislým od kontextu akcie z inej vetvy stromu akcií. To zvýši odolnosť vytvorených obaľovačov proti prípadným čiastočným zmenám štruktúry na obaľovanej stránke. Pod kontextom rozumieme dátovú štruktúru, ktorá obsahuje aktuálny stav obaľovania. Obsahuje vstupné premenné, vzor, aktuálny subdokument, lokálny výstupný objekt, cookies, prípadne autentifikačné údaje. Aktuálny subdokument sa vytvorí prostredníctvom rozhrania na dokument, teda v závislosti od vstupného dokumentu to môže byť

časť DOM stromu prípadne reťazec String.

6.5.2 Akcia

Pod akciou rozumieme operácie nad lokálnym kontextom. Každá akcia vo svojej reprezentácii obsahuje atribút `NextActions`, ktorý určuje, ktoré akcie majú v programe po nej bezprostredne nasledovať (t.j. aké vetvy z nej vychádzajú). Každá ďalšia akcia pracuje nad svojim lokálnym kontextom.

Rozoznávame typy akcií:

- *Navigačné akcie* - zabezpečujú pohyb po web priestore, prechod na želanú stránku a jej samotné načítanie do interného formátu využívaného obaľovačom, ale aj odoslanie formulára na server. Napríklad akcia LoadPage, ktorá načíta stránku zo zadanej URL.
- *Extrakčné akcie* - zabezpečujú výber a dolovanie požadovaných dát prostredníctvom vzorov.

Detailnejší popis akcii a ich atribútov je v časti 6.7 - Tvorba obaľovača.

6.5.3 Vzor

Ide o nový termín, s ktorým doterajšia implementácia obaľovača nepracuje. Akcia pomocou filtrov identifikuje vzor. Vzor teda obsahuje filtre ako kritérium na výber subdokumentu. Aplikovaním vzoru na subdokument vznikajú nové subdokumenty. Vzory nám umožnia do tvorby obaľovača zaviesť prvok učenia sa. Obaľovač sa na základe pozitívnych a negatívnych príkladov, získaných interakciou s používateľom, naučí identifikované vzory. Do budúcnosti počítame s prítomnosťou rôznych typov vzorov, preto samotný vzor bude okrem filtrov obsahovať aj svoj typ a spôsob učenia. Ak teda pracujeme s HTML dokumentom, aktuálnym subdokumentom je DOM strom, vzor je typu HTML a filtre sú typu XPath. Ak by bol potrebný napr. vzor typu Text, filtre by boli regulárne výrazy a aktuálny subdokument by bol znakový reťazec (string).

6.5.4 Cookies

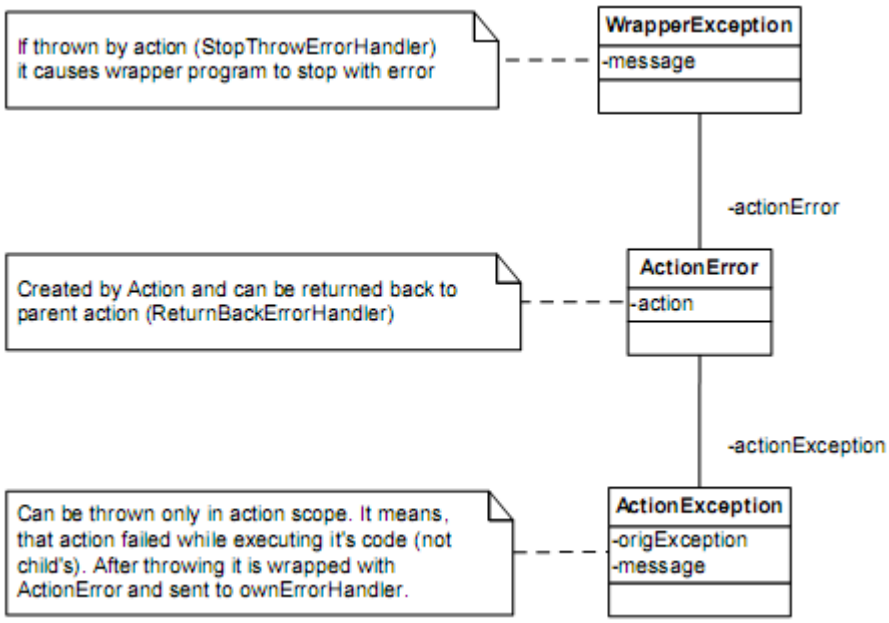
Pri odosielaní požiadavky na web server sa serveru posiela aj zoznam všetkých doteraz pri navigácii na tomto serveri získaných cookies.

6.5.5 Autentifikačné údaje

Sú web serverom vyžadované údaje potrebné na pre http autentifikáciu, ktoré sa musia posielať po prihlásení na server v každej ďalšej požiadavke.

6.5.6 Ošetrenie výnimiek

Model ošetrenia výnimiek vychádza už z existujúceho aplikačného rámca obaľovača a nepredpokladáme potrebu jeho výraznej zmeny.



Obr. 6-2: Model výnimiek [5]

`WrapperException` je výnimka, ktorá nie je odchytávaná akciami a dostane ju až interpret, ktorý spustil obaľovač a má za dôsledok ukončenie vykonávania programu obaľovača. Generuje sa akciami v prípade vzniku chyby a zároveň je použitý `StopThrowErrorHandler`. Táto výnimka potom obaľuje `ActionError`, ktorý v akcii vznikol. V prípade, že vykonávanie kódu akcie skončí chybou (napr. na stránke nie sú požadované dáta) je vygenerovaná `ActionException`, obalená do `ActionError` a poslaná na vstup `ownErrorHandler`, ktorý ma na starosti spracovanie vlastných chýb. V prípade, že ide o `ReturnBackErrorHandler`, chyba sa vráti akcii, ktorá volala aktuálnu akciu.

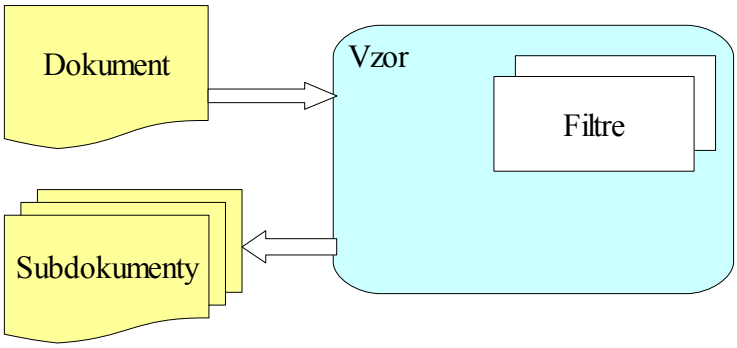
6.6 Aplikačný modul – Tvorba vzorov

6.6.1 Vzor

Pri práci s dokumentom treba pristupovať postupne k jeho jednotlivým častiam, aby sa nad nimi mohli vykonávať konkrétne akcie. Tieto akcie spracovávajú určitý kontext, ktorý dostanú na vstupe. Na zmenu a získavanie nového kontextu slúžia vzory.

Vzor je objektom systému, ktorý sa aplikuje v rámci určitého kontextu obsahujúceho dokument (alebo časť dokumentu) na získanie žiadanej časti tohoto dokumentu (subdokument).

Za účelom výberu časti dokumentu obsahuje každý vzor jeden alebo viac filtrov, ktoré definujú tento výber. Výsledkom výberu môže byť súvislý subdokument, alebo viacero samostatných subdokumentov. Objekt vzor musí preto poskytovať vo svojom rozhraní metódy na získavanie jednotlivých inštancií vybraných subdokumentov. Schému aplikovania vzoru zobrazuje Obr. 6-3.

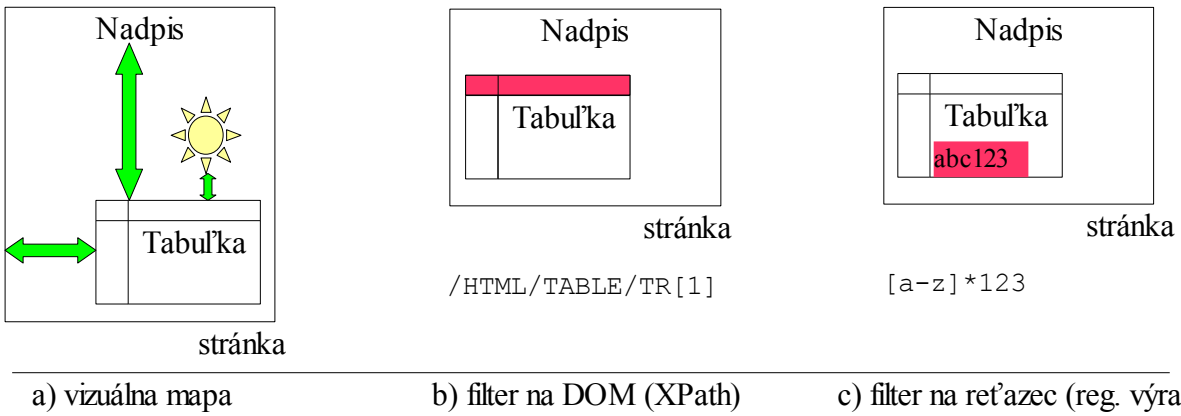


Obr. 6-3: Schéma aplikácie vzoru

6.6.2 Typy filtrov

V súčasnosti existuje viacero typov filtrov na výber informácií z webovej stránky. Jednotlivé typy sú zatiaľ rozpracované do rôznych úrovní. Pri výbere typov filtrov, ktoré má výsledný systém obsahovať, do úvahy pripadali najmä nasledovné typy:

- *vizuálna mapa* – Filter definuje požadovanú časť dokumentu na základe jej polohy na zobrazovanej stránke a tiež jej vzájomnej polohe na zobrazenej stránke vzhľadom k ostatným prvkom. Príklad - Obr. 6-4a).
- *filter na DOM* - Tento filter vychádza z hierarchickej stromovej štruktúry HTML kódu stránky. K jednotlivým častiam pri výbere pristupuje ako k elementom tohoto stromu alebo k podstromom. Pričom aj jeden element je vlastne podstromom stromu. Príklad - Obr. 6-4b).
- *filter na reťazce* – Filter pristupuje k dokumentu ako k reťazcu, pričom naň aplikuje kritériá vyhľadávania v reťazci. Špecifikácia vyberanej časti dokumentu môže byť realizovaná napríklad pomocou regulárnych výrazov. Príklad - Obr. 6-4c).



Obr. 6-4: Príklady typov filtrov

Vzhľadom na požiadavky kladené na projekt boli do návrhu systému zaradené nasledovné dva typy filtrov – *filter na DOM* a *filter na reťazec*.

Adapcia vzorov

Vzory pracujú s viacerými druhmi filtrov. Tieto filtre pristupujú k spracovávanému dokumentu rôznymi spôsobmi, a preto musia vzory vstupný dokument upraviť do tvaru vhodného na spracovanie daným filtrom. Takéto zobrazenie dokumentu medzi tvarmi vhodnými pre rôzne filtre a tým schopnosť spracovávať rôzne tvary dokumentu musia zabezpečiť metódy adapcie vzorov.

6.6.3 Učenie

Každý vzor musí vyberať zo vstupného dokumentu práve tie prvky, ktoré si používateľ žiada. Spôsobom, akým sa vzoru odovzdajú požiadavky používateľa na výber elementov, bude učenie. Použijeme *učenie s učiteľom* (angl. supervised learning) – učenie sa pomocou pozitívnych a negatívnych príkladov.

Príkladom je časť dokumentu, ktorú špecifikoval používateľ. Vzor obdrží príklad od komunikačného modulu. Tento príklad môže byť:

- *pozitívny príklad* – predstavuje elementy, ktoré používateľ chce mať vo výstupnom dokumente, a teda sa musia vo výbere nachádzať
- *negatívny príklad* – predstavuje elementy, ktoré používateľ nechce mať vo výstupnom dokumente, a teda sa vo výbere nesmú nachádzať

Po odoslaní prvej sady pozitívnych a negatívnych príkladov systém začne generalizovať výber, čiže sa snaží získať čo najvšeobecnejšie kritérium výberu (t.j. riešenie, ktoré vo výbere zahŕňa čo najviac inštancií), pričom musia byť dodržané požiadavky kladené množinou pozitívnych a negatívnych príkladov.

Systém má byť pružný aj z hľadiska spôsobu učenia, aby bolo možné na učenie použiť rôzne metódy. Dosiahnutie splnenia tejto požiadavky bude zabezpečovať *stratégia učenia*, ktorú bude možné určiť / nastaviť pomocou rozhrania vzoru.

Spravovanie príkladov

Systém má umožňovať prezeranie zadaných príkladov, ich editovanie, prípadne odstránenie nesprávnych príkladov alebo príkladov zadaných do zlej skupiny (pozitívne, negatívne). V dôsledku toho si musí každý vzor pamätať všetky príklady, ktoré boli zadané pre jeho filtre. Ďalej musí uchovávať aj priradenie jednotlivých príkladov k filtrom, ktoré boli pomocou nich odvodené, aby bolo možné jednoduchšie vykonávať editáciu filtrov a ďalšie narábanie s nimi.

Odstraňovanie príkladov

Pri odstránení príkladu sa musia vykonať zmeny vo všetkých filtroch, ktoré boli týmto príkladom ovplyvnené. V prístupe k tejto situácií sa naskytujú 2 možnosti riešenia:

- *Roll-back* – t.j. vrátenie vzoru do stavu, v akom sa nachádzal pred vložením odstraňovaného príkladu. Táto možnosť by zahŕňala odstránenie všetkých príkladov zadaných po aktuálne odstraňovanom príklade a tiež odstránenie všetkých filtrov, ktoré boli vytvorené na základe

týchto príkladov.

- *Delete* – dôjde len k odstráneniu požadovaného príkladu. Po odstránení tohoto príkladu dôjde k opätovnému učeniu všetkých filtrov, aby sa vzor dostal do takého stavu, v akom by bol, ak by sa odstránený príklad nebol vôbec zadal.

Stratégie učenia

V prístupe k učeniu existujú viaceré stratégie. V navrhovanom systéme budú použité minimálne dva:

1. Prvou stratégiou je jednoduchá stratégia učenia určená na otestovanie funkčnosti prototypu. Bude založená na zovšeobecňovaní XPath výrazu, ku ktorému sa bude pristupovať ako k reťazcu. Napríklad, majme HTML zápis tabuľky:

```
<HTML>
<BODY>
<H1> ... </H1>
<TABLE>
<TR>
<TD>Cislo</TD>
<TD>Meno</TD>
</TR>
<TR>
<TD>1</TD>
<TD>Jano</TD>
</TR>
...
```

   Ak používateľ vyberie 2 pozitívne príklady, ktorými budú elementy tabuľky „Cislo“ a „1“, vzor obdrží XPath výrazy:

```
/HTML/BODY/TABLE/TR[1]/TD[1]
/HTML/BODY/TABLE/TR[2]/TD[1]
```

   Po zovšeobecnení by mal vzniknúť výraz:

```
/HTML/BODY/TABLE/TR/TD[1]
```

   čím by vznikol výber 1. stĺpca tabuľky.

2. Ďalšou stratégiou môže byť priamo implementácia, prípadne upravená implementácia algoritmu učenia použitím zložených filtrov [8].

   Tento algoritmus charakterizuje elementy pomocou referencií na iné elementy. Referencie sú reprezentované pomocou *cesty* od referenčného uzla k referencovanému (používa sa cesta v HTML strome). Cesta sa skladá z časti smerujúcej ku koreňu (po spoločný uzol oboch uzlov) a časti smerujúcej od tohoto uzla k referencovanému.

   Algoritmus pri definícii filtra využíva *testy* (na prítomnosť atribútu, na jeho hodnotu, test na koreňový element, ...).

   Z jednoduchých filtrov sa vytvárajú zložené ich spájaním pomocou logických operátorov $\wedge$ (konjunkcia), $\vee$ (disjunkcia), $\neg$ (negácia). Napríklad pre filtre C, C_1, C_2 a C_3: $C = C_1 \wedge C_2 \wedge (\neg C_3)$.

Algoritmus:

- Každý príklad sa popíše pomocou jednoduchého filtra. Popisujú sa aj pozitívne aj negatívne príklady, pričom sa vyznačí, či ide o pozitívny alebo negatívny.
- Na základe toho, ako jednotlivé filtre pokrývajú dokument sa urobí redukcia počtu filtrov, ak niektorý pokrýva aj prípady pokryté iným, alebo, ak sú filtre ekvivalentné.
- Zo vzniknutej skupiny filtrov sa vytvorí zložený filter, ktorého tvorba sa dá zredukovať na učenie disjunktívnej normálnej formy. Takto dostaneme výsledný naučený filter.

6.7 Aplikačný modul – Tvorba obaľovača

Na základe vstupov od používateľa získaných z prezentačných modulov (prehliadača a aplikácie) modul na tvorbu obaľovača vytvorí požadovanú inštanciu obaľovača, ktorý pozostáva z:

- vykonateľných akcií
- relácií medzi akciami
- vstupných premenných
- štartovacej akcie
- HTTP klienta a parser dokumentu

Definície akcií, relácií medzi akciami a vstupných premenných sa využijú na tvorbu XML dokumentu, ktorý opisuje obaľovač a následne sa využije rozhranie parsera programu obaľovača nástroja Wrapper Suite (kapitola 2.3). Rozhranie parsera programu poskytuje možnosť vytvorenia inštancie obaľovača na základe jeho opisu vo forme XML súboru. Posledným krokom v procese tvorby obaľovača je nastavenie zvyšných charakteristík obaľovača (výber štartovacej akcie a určenie HTTP klienta a parsera dokumentu).

6.7.1 Akcie obaľovača

Akcie predstavujú elementárne operácie s presne definovanou činnosťou, ktoré pracujú nad lokálnym kontextom (časť 6.5). Akcie pozostávajú z 2 častí:

- typ akcie – navigačné, extrakčné akcie alebo odoslanie formulára
- atribúty akcie – vlastnosti akcie

Navigačné akcie

Navigačné akcie zabezpečujú prechod na stanovený dokument a jeho načítanie do lokálneho kontextu. Medzi navigačné akcie patrí:

- `LoadPage`
- `FollowLink`

LoadPage

Načítanie požadovanej web stránky. Akciu charakterizujú nasledovné atribúty:

- *url* – adresa web stránky, ktorá sa má načítať
- *header* – hlavička HTTP požiadavky, ktorá je zaslaná na server s danou adresou

Akcia odošle HTTP požiadavku (s definovanou hlavičkou a URL adresou), pričom pri odosielaní požiadavky sa použijú cookies a autentifikačné údaje z kontextu. Následne sa vykoná transformácia dokumentu na validný XHTML dokument (ak je to možné) a uloží sa do dokumentu lokálneho kontextu ako objekt stromovej štruktúry (DOM), do cookies v kontexte sa uložia prijaté cookies.

FollowLink

Zmena aktuálneho dokumentu na základe nájdeného odkazu na stránke. Akcia je určená atribútmi:

- *link* – určenie lokalizácie odkazu v dokumente pomocou identifikácie vzoru, ktorý vytvoril používateľ (modifikácia vzoru v lokálnom kontexte)
- *header* – hlavička http požiadavky, ktorá je zaslaná na server
- *repeat* – ohraničenie počtu nasledovaní odkazov

V aktuálnom dokumente sa lokalizuje požadovaný odkaz (určený atribútom *link*) a extrahuje sa z neho hodnota odkazu (adresa dokumentu). Ak atribút *link* nie je zadaný, akcia predpokladá, že v aktuálnom lokálnom kontexte sa bude nachádzať dokument, ktorého súčasťou je len požadovaný odkaz. Zo získanej hodnoty odkazu a adresy aktuálnej stránky sa vytvorí URL adresa a ďalej sa pokračuje rovnakým spôsobom ako pri akcii `LoadPage`. Používateľ má možnosť nastaviť počet opakovaní vykonávania nasledovania daného odkazu v dokumente prostredníctvom atribútu *repeat* (napr. pre prípad, že extrahované údaje sa nachádzajú v dokumentoch spojených odkazom „nasledujúci"). V takomto prípade sa zapamätá lokálny kontext obaľovača po ukončení každej akcie `FollowLink`, ktorý bude predstavovať vstupný lokálny kontext nasledujúcej akcie `FollowLink` (pre ďalší počet nasledovania odkazu). Používateľ má tiež možnosť nastaviť neohraničený počet nasledovania odkazu, v takom prípade sa bude nasledovať odkaz, pokiaľ existuje.

Extrakčné akcie

Extrakčné akcie zabezpečujú proces dolovania dát do štruktúrovaného výstupu. Prvým krokom extrakcie dát je aplikovanie filtra aktuálneho vzoru uloženého v lokálnom kontexte. Vzory uložené v lokálnom kontexte môže používateľ manuálne modifikovať, v tomto prípade sa do lokálneho kontextu akcie vloží vzor vytvorený používateľom. Pre všetky dáta, ktoré spĺňajú podmienky filtra (napr. všetky riadky tabuľky) sú vykonané nasledovné operácie:

1. získané dáta sú skonvertované do požadovaného formátu použitím regulárnych výrazov
2. uloženie dát do definovaného výstupného objektu, ak je to požadované

3. uloženie dát do definovaných premenných v kontexte, ak je to požadované
4. zmena aktuálneho dokumentu v lokálnom kontexte obaľovača (dokument v lokálnom kontexte bude predstavovať extrahované dáta – strom s extrahovanými dátami)

Extrakčné akcie sú opísané nasledovnými atribútmi:

- *outputObject* – výstupný objekt (kontextový), ktorý určuje, kde sa majú extrahované dáta uložiť. Výstupný objekt je určený relatívnou cestou vo výstupnom strome pre daný obaľovač vzhľadom na svojho predchodcu. Vo výstupnom objekte tiež možno určiť, či budú dáta uložené ako element alebo atribút
- *pattern* – identifikovanie vzoru v prípade, že používateľ vytvoril nový vzor pre danú akciu (modifikoval vzor uložený v lokálnom kontexte)
- *variable* – určenie premenných v kontexte, kde budú extrahované dáta uložené
- *format* – formát, do ktorého sa majú extrahované dáta transformovať, určený pomocou regulárnych výrazov

Odoslanie formulára

Jedinou definovanou akciu, ktorá slúži na odoslanie formulára je akcia `SubmitForm`.

SubmitForm

Odoslanie formulára, ktoré je špecifikované nasledovnými atribútmi:

- *form* – určenie formulára v dokumente pomocou identifikácie vzoru, ktorý vytvoril používateľ (modifikácia vzoru v lokálnom kontexte)
- *inputs* – parametre, ktorých hodnoty sú odoslané vo formulári. Parametre predstavujú premenné v kontexte
- *header* – hlavička HTTP požiadavky, ktorá je zaslaná na server

V aktuálnom dokumente sa lokalizuje požadovaný formulár (určený atribútom *form*) a z atribútu nájdeného formulára sa extrahuje odkaz a použitá metóda odoslania požiadavky (GET, POST). Ak atribút *form* nie je zadaný, akcia predpokladá, že v aktuálnom lokálnom kontexte sa bude nachádzať dokument, ktorého súčasťou je len požadovaný formulár. Z hodnoty získaného odkazu spracovania formulára a adresy aktuálnej stránky sa vytvorí URL adresa, vytvorí sa HTTP požiadavka so získanou metódou (GET, POST) a do vytvorenej požiadavky sa vloží hlavička (*header*) a obsah formulára (*inputs*). Vytvorená požiadavka sa následne odošle na server.

6.7.2 Ostatné charakteristiky obaľovača

Po vytvorení akcií obaľovača je potrebné vytvoriť relácie medzi akciami (stromovú štruktúru akcií), kde jedinou možnou reláciou je určenie nasledujúcich akcií, ktoré sa majú vykonať po skončení aktuálnej akcie. Týmto spôsobom sa vytvorí poradie vykonávania akcií. Zároveň je potrebné určiť štartovaciu akciu (koreň stromu akcií).

Na získanie dokumentov z webových systémov je potrebná komunikácia s webovými servermi prostredníctvom protokolu HTTP, ktorá je zabezpečená prostredníctvom existujúcich

knižníc prístupných pomocou rozhrania HTTP klienta v obaľovači.

Úlohou parsera dokumentu je transformovať získaný HTML dokument DOM reprezentácie dokumentu.

6.8 Výstupné objekty

V súčasnej verzii aplikačného rámca predstavuje tvorba výstupných objektov samostatnú akciu. Pre zefektívnenie aplikácie sme sa rozhodli pre integráciu tohto procesu do akcie extrahovania dát. Počas vykonávania extrakčnej akcie (počas dolovania dát) sa bude priebežne vytvárať výstupný strom zo získaných údajov, pričom koreňový element je špecifikovaný v doménovej oblasti. Vetvy stromu sú určené relatívnou cestou vzhľadom na predchodcu. Získané dáta podľa svojho charakteru a potreby môžu byť ukladané nie len ako elementy, ale aj ako atribúty elementu.

Výstupné objekty budú ukladané do vyrovnávacej pamäte a odovzdávané jednotlivým zapisovačom, ktoré ich fyzicky zapíšu do konkrétneho cieľového prostredia (viac informácií v časti 2.3.6).

6.9 Implementačné prostriedky a vývojové nástroje

Vzhľadom na to, že systém na tvorbu obaľovačov staviame na existujúcom systéme, máme množstvo výhod, ale aj obmedzení. Medzi obmedzenia patrí najmä výber implementačných prostriedkov a vývojových nástrojov. Na druhej strane však vieme, že tieto prostriedky boli analýzou zvolené za vhodné a môžeme sa sústrediť na nové nástroje, ktoré sú potrebné na splnenie nových cieľov produktu. Prostriedky a nástroje existujúceho systému, ktoré použijeme:

- systém na správu verzií SVN
- platforma JDK 1.5, na ktorej bol systém vyvíjaný
- vývojové prostredie Eclipse, systém na správu verzií udržiava projekt pre toto vývojové prostredie vždy aktuálnym
- Jakarta commons HTTP klient - nie je potrebné modifikovať spôsob práce systému s protokolom HTTP
- HTML parser NekoHTML - použijeme existujúci modul na kontrolu a automatickú opravu chýb HTML dokumentov
- XML parser Xerces - štruktúru dokumentov nie je potrebné meniť
- XML spracovávanie dokumentov pomocou nástroja Jaxon
- balík na podporu Unit testovania v prostredí Java - naviažeme na vytvárané testy
- systém Maven2 na automatizovanú distribúciu, tvorbu balíkov, dokumentácie, štatistík.
- k týmto existujúcim prostriedkom sme pridali iba niekoľko rozšírení vo vývojovom prostredí, ktoré pomôžu členom tímu, ktorí sa s niektorými nástrojmi ešte nestretli:
  - Subclipse a JavaSVN – rozšírenia nástroja Eclipse, ktoré podporujú prácu s verzionovacím systémom SVN

- Maven 2.0 integration – rozšírenie integrujúce podporu systému Maven2 do prostredia Eclipse

7 Prototyp riešenia

V tejto časti dokumentácie je opísaný prototyp navrhovaného riešenia aplikácie tvorby obaľovačov. V jednotlivých častiach sú rozobraté ciele, ktoré sme prototypom sledovali (časť 7.1), implementované zmeny v jadre (časť 7.2), implementované stratégie učenia (časť 7.3) a zmeny týkajúce sa používateľského rozhrania (časť 7.4).

7.1 Ciele

Náš produkt nadväzuje na prácu minuloročného tímu, ktorý vytvoril aplikačný rámec na tvorbu obaľovačov. Cieľom nášho prototypu je rozšírenie existujúceho systému v nasledovných bodoch:

- úprava jadra modulu *Tvorba vzorov*, konkrétne:
  - overenie návrhu učenia sa obaľovača
  - návrh a implementácia vybraných tried modulu
  - tvorba rozhraní
- implementácia jednoduchých stratégií učenia sa
- úpravy v *Prezentačnom* module aplikácie – doplnenie potrebných súčastí

Hlavným cieľom, ktorý sme si stanovili v prototype, je overenie návrhu učenia sa obaľovača. V súčasnosti sú dáta zo stránky získané pomocou filtra (určený XPath výrazom), ktorý musel používateľ presne zadefinovať. Našim cieľom je pridanie učenia pomocou pozitívnych príkladov. Pozitívne príklady od používateľa sú generalizované do filtra, ktorého podmienkou je, že aplikovaním na dokument musí vrátiť všetky zadané príklady (pozitívne). Táto požiadavka je realizovaná v module *Tvorba vzorov*, kde sa pri vytvorení novej akcie, ktorá pri svojej činnosti vyžaduje lokalizovať dáta na stránke, vytvorí vzor s priradenou stratégiou učenia sa (o stratégií rozhodne používateľ).

V module *Tvorba vzorov* bolo ďalej potrebné vytvoriť triedy, ktoré budú zabezpečovať požadovanú funkcionalitu:

- filter – v súčasnosti XPath filter
- príklady – pozitívne a negatívne príklady potrebné pri vytváraní filtra
- subdokument – výsledok filtrovania aktuálneho dokumentu, v súčasnosti XML subdokument
- vzor – zahŕňa filter a stratégiu učenia, ktorá generuje filter
- stratégia učenia – v prototype implementované 2 stratégie učenia:
  - filter priamo vytvorený používateľom
  - filter naučený z pozitívnych príkladov

Keďže v požiadavke na systém je umožnenie viacerých stratégií učenia a tiež viacerých

typov filtrov, triedy pre filter a stratégiu učenia sú reprezentované ako rozhrania. V prototype sú vytvorené triedy, ktoré implementujú tieto rozhrania (XPath filter implementuje filter a stratégia učenia je implementovaná dvoma triedami, ktoré realizujú stanovené stratégie učenia). Keďže do konečnej aplikácie implementujeme aj filter regulárnych výrazov nad reťazcami, je potrebné transformovať dokument v XML formáte na reťazec. V prototype sme sa preto zaoberali aj možnosťami existujúcich riešení transformácie dokumentov (adaptačné rámce).

Vzhľadom na vytvorené zmeny v existujúcom systéme je potrebné aj rozšírenie prezentačného modulu:

- umožnenie voľby stratégie učenia pri vytváraní akcie,
- rozhranie na pridávanie príkladov pri učení pomocou pozitívnych príkladov.

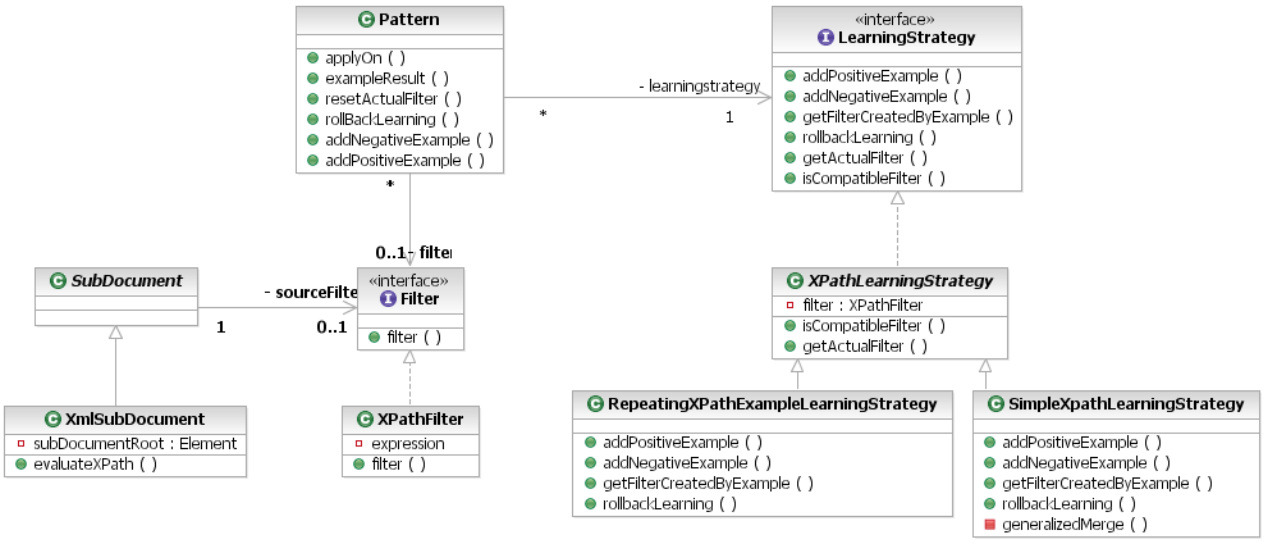
Umožnenie voľby stratégie učenia sa je implementované v existujúcom rozhraní vytvárania a modifikovaní akcií obaľovača. Na základe voľby používateľa sa danej akcii priradí do vzoru zodpovedajúca stratégia učenia. Rozhranie na pridávanie príkladov je riadené kontrolórom, ktorý na základe požiadavky používateľa na pridanie príkladu vytvorí dialóg a po zadaní príkladu pridáva do vzoru akcie vytvorený príklad. Vo finálnej verzii budú príklady v kontrolóri získavané cez komunikačný modul, a preto kontrolór predstavuje rozhranie, ktoré je v prototype implementované triedou na riadenie pridávanie príkladov z dialógu *Prezentačného* modulu aplikácie.

7.2 Zmeny v jadre

Keďže projekt vychádza z existujúceho rámca na tvorbu obaľovačov, prototyp riešenia integrujeme priamo do tohto prostredia. Cieľom prototypu je ukážka alebo naznačenie prístupu, ktorý chceme použiť pri realizácii učenia vzorov, konkrétne vzorov typu XPath. Implementácia prototypu si preto vyžaduje niektoré nevyhnutné úpravy jadra existujúceho rámca na tvorbu obaľovačov.

7.2.1 Návrh modulu „Tvorba vzorov“

Učenie vzorov realizuje modul *Tvorba vzorov* (pozri časť 6.1). Keďže v neskorších fázach projektu je potrebné kompletne zmeniť existujúce jadro systému, modul *Tvorba vzorov* musí byť navrhnutý tak, aby bolo možné tento modul použiť čo najjednoduchším spôsobom v existujúcom systéme (ukážka prototypu) a rovnako aj v novom systéme. Logický diagram tohto modulu sa nachádza na Obr. 7-1.



Obr. 7-1: Diagram tried modulu „Tvorba vzorov" (neobsahuje signatúry metód)

Opis jednotlivých entít diagramu:

- **Pattern** – fasáda nad filtrom a stratégiou učenia. Vzor deleguje príklad na stratégiu učenia, ktorá na základe príkladu vytvorí nový filter.
- **Filter** – rozhranie, ktoré reprezentuje komunikáciu s filtrom. Obsahuje metódu filter.
- **XPathFilter** – konkrétny filter. Filtrovanie realizuje aplikovaním XPath výrazu na uzol v DOM strome.
- **LearningStrategy** – rozhranie, ktoré reprezentuje stratégiu učenia. Stratégia učenia umožňuje pridávať pozitívne a negatívne príklady, udržuje históriu učenia (vie vrátiť filter, ktorý bol naučený vstupným príkladom) a vie zistiť, či je filter kompatibilný so stratégiou učenia.
- **XPathLearningStrategy** – abstraktná stratégia učenia, ktorá sa učí filtre typu XPathFilter.
- **RepeatingXPathLearningStrategy** – stratégia učenia, obsahuje učenie sa filtra typu XPath tak, že naučený filter je iba kópiou vstupného pozitívneho príkladu. Stratégia si neudržuje históriu učenia a nevie sa učiť pomocou negatívnych príkladov.
- **SimpleXPathLearningStrategy** – stratégia učenia, obsahuje učenie sa filtra typu XPath tak, že zadaný príklad sa generalizuje s doteraz naučeným filtrom. Stratégia spracováva len pozitívne filtre. Udržuje si históriu príkladov a aj z nich vytvorených filtrov.
- **SubDocument** – abstraktná trieda, ktorá reprezentuje subdokument. Subdokument obsahuje filter, ktorým bol vytvorený.
- **XmlSubDocument** – subdokument, ktorý reprezentuje uzol v DOM strome.

7.2.2 Integrácia modulu „Tvorba vzorov“ do jadra systému

Úlohou modulu *Tvorba vzorov* je vytváranie nových vzorov pomocou príkladov, a teda vytváranie nových filtrov a filtrácia subdokumentov pomocou týchto filtrov. Podobná zodpovednosť prislúcha v existujúcom systéme triede `TagFinder`. Trieda `TagFinder` si udržuje XPath výraz vo forme reťazca (filter) a filtruje prichádzajúce DOM dokumenty, takže vracia DOM uzly, ktoré vzniknú aplikovaním XPath výrazu na vstupný dokument.

Trieda `TagFinder` je silne zviazaná s existujúcim jadrom systému, preto jej nahradenie nie je vhodné a vykoná sa až v ďalšej fáze projektu. Existujúci `TagFinder` bol upravený tak, že jeho XPath výraz bol nahradený vzorom, pričom filtrácia je delegovaná na vzor.

7.3 Stratégie učenia

Hlavnou úlohou stratégie učenia v rámci modulu „Tvorba vzorov“ je vytvoriť filter pre daný vzor. Filter sa vytvára postupne na základe zadávaných príkladov. Musí byť vytvorený tak, aby subdokument (prípadne subdokumenty), ktoré sa pomocou neho získajú z pôvodného dokumentu, pokrývali všetky pozitívne a žiadne negatívne príklady.

K úlohám stratégie učenia sme pridali aj možnosť získať filter, ktorý bol vytvorený na základe zadaného príkladu a možnosť vrátiť učenie do stavu, v akom bolo pred zadaním určitého príkladu.

Stratégia učenia charakterizuje, akým spôsobom sa bude filter vytvárať, čiže na základe tých istých príkladov môžu dve rôzne stratégie vytvoriť dva rôzne filtre. Pre výsledný produkt sme špecifikovali požiadavku, aby podporoval viaceré stratégie učenia. Prototyp obsahuje 2 stratégie: opakujúcu stratégiu učenia sa XPath príkladov a jednoduchú stratégiu učenia sa XPath príkladov. Obe zatiaľ pracujú iba s pozitívnymi príkladmi.

7.3.1 Opakujúca stratégia učenia XPath príkladov (RepeatingXPathExampleLearningStrategy)

Ide o stratégiu použiteľnú na spracovanie príkladov zadávaných pomocou XPath výrazu. Je schopná pamätať si len jeden zadaný príklad, ktorý potom poskytuje ako vytvorený filter. Jej cieľom je poskytovať filter, ktorý vyberá z dokumentu práve tie elementy, ktoré vybral používateľ. Uplatní sa v ďalšej etape projektu, konkrétne v časti vytvárania prepojenia medzi internetovým prehliadačom a aplikáciou. Tu poslúži na overenie správneho odovzdávania príkladov z prehliadača a ich správneho zobrazovania naspäť v prehliadači.

Opis činnosti:

- ***pridanie pozitívneho príkladu*** – stratégia zadaný príklad nijak nespracováva, iba si ho zapamätá a tento príklad (XPath výraz) poskytuje ako filter
- ***pridanie negatívneho príkladu*** – stratégia zadanie príkladu ignoruje, t.j. dodaný príklad si nikam nezaznamená a naďalej poskytuje doterajší filter

- ***vrátenie filtra vytvoreného pomocou zadaného príkladu*** – ak sa príklad zhoduje s filtrom, vráti ten istý filter (ktorý je v tomto prípade totožný s príkladom), inak vráti null – čiže, na základe zadaného príkladu nebol vytvorený filter
- ***návrat v učení*** – ak je zadaný príklad zhodný s filtrom, odstráni sa filter. Inak sa používa aj naďalej doterajší

7.3.2 Jednoduchá stratégia učenia XPath príkladov (SimpleXPathExampleLearningStrategy)

Stratégia predstavuje jednoduchý spôsob učenia. Zo zadaných príkladov sa usiluje vytvoriť filter, ktorý ich bude všetky zahŕňať (generalizovať). Keďže spracováva iba pozitívne príklady, stačí pri zadaní každého príkladu iba zovšeobecňovať doterajší filter.

Opis činnosti:

- ***pridanie pozitívneho príkladu*** – stratégia zovšeobecní doterajší filter pomocou zadaného príkladu, čím dostane nový filter, ktorý bude ďalej poskytovať. Ak sa pridáva príklad, ktorý sa už predtým pridal, ignoruje sa, pretože by aj tak výsledný filter neovplyvnil.
- ***pridanie negatívneho príkladu*** – pridávaný negatívny príklad sa ignoruje (podobne ako u predchádzajúcej stratégie)
- ***vrátenie filtra vytvoreného pomocou zadaného príkladu*** – ak bol zadaný príklad už pri učení použitý, vráti sa filter, ktorý vtedy pomocou neho vznikol. Ak nebol použitý, vráti sa null.
- ***návrat v učení*** – keďže stratégia použije každý príklad práve raz, dá sa jednoznačne na základe príkladu určiť filter, ktorý mala stratégia pred použitím tohoto príkladu. A pretože stratégia udržuje históriu použitých príkladov a filtrov, ktoré pomocou nich vznikli v časovom poradí, môže sa učenie vrátiť do stavu pred naučením zadaného príkladu.

Princíp zovšeobecňovania

Do zovšeobecňovania vstupujú dva XPath výrazy. Stratégia s nimi pracuje ako s reťazcami, pričom sa berie do úvahy, že ide o XPath výrazy. Východiskom je rozdelenie oboch reťazcov na elementy podľa oddeľovača. Hlavnou myšlienkou je na základe týchto dvoch výrazov vytvoriť nový výraz, ktorý pokrýva (aj) všetky elementy, ktoré pokrývali tieto výrazy. Zároveň sa snažíme neurobiť výsledný výraz príliš všeobecný.

Základným princípom je prechádzanie elementov oboch reťazcov od ich začiatku a nájdenie najdlhšej spoločnej časti, pričom jednotkou pri tomto porovnávaní je práve element. Nasledujúca ukážka predstavuje zovšeobecnenie dvoch výrazov do tretieho (pod čiarou):

/HTML/BODY/TABLE[1]/TR[1]/TD[2]/P
/HTML/BODY/TABLE[1]/TR[1]/TD[2]/IMG

/HTML/BODY/TABLE[1]/TR[1]/TD[2]

Táto základná myšlienka je v tejto stratégii doplnená o ďalšie zovšeobecnenie, a to

odstránením predikátu. Aby sa obmedzilo nadbytočné zovšeobecnenie v tomto smere, odstraňujú sa iba rozdielne predikáty.

/HTML/BODY/TABLE[1]/TR[1]/TD[2]/
/HTML/BODY/TABLE[1]/TR[2]/TD[2]/

/HTML/BODY/TABLE[1]/TR/TD[2]

7.4 Používateľské rozhranie

Ako používateľské rozhranie sa v našom prototype použilo rozhranie systému, ktoré vytvorili členovia minuloročného tímu. Toto rozhranie bolo modifikované a doplnené o ďalšie ovládacie prvky, ktoré používateľovi sprístupňujú funkcionalitu stratégie učenia a tvorby vzorov.

Prvá zmena používateľského rozhrania sa týka zmeny dialógových okien akcií `FollowLink`, `ExtractData`, `ForEachTag` a `DoWhileNextLink`. Tieto dialógové okná umožňovali jednotlivým akciám vkladať regulárne výrazy, ktoré slúžili pri hľadaní značiek v dokumente. Funkciu regulárnych výrazov sme v prototype nepoužili, a preto sme odstránili ovládacie prvky, ktoré tieto výrazy umožnili vkladať. Naopak v týchto dialógoch pribudla položka, ktorá zobrazuje zoznam všetkých dostupných stratégií učenia vzorov a umožňuje výber jednej z nich. Vybraná stratégia sa potom použije pri tvorbe vzoru (RepeatingLearningStrategy – filter je zadaný priamo používateľom a SimpleLearningStrategy – jednoduché učenie generalizáciou pozitívnych príkladov).

Z funkcionality tvorby vzorov pri jednoduchom učení sa obaľovača vznikla aj požiadavka na vkladanie pozitívnych a negatívnych príkladov vo forme XPath výrazov (v prototype sa používajú len pozitívne príklady). Bolo potrebné vytvoriť dialógové okno, ktoré používateľovi umožňuje vkladať tieto príklady a rozlišovať medzi nimi. Na vkladanie príkladu sa použil ovládací prvok textové pole. Do tohto prvku používateľ môže zapísať požadovaný príklad. Musí taktiež určiť, o aký typ príkladu sa jedná (pozitívny alebo negatívny). Na to slúžia dva prepínače, z ktorých je vždy aktívny práve jeden (podľa typu príkladu, ktorý si používateľ vyberie). V našom prototype využívame na učenie obaľovača iba pozitívne príklady (vloženie negatívnych príkladov nemá žiadny dopad na učenie sa obaľovača). Dialóg je ovládaný kontrolórom, ktorý na žiadosť používateľa otvára nový dialóg, získava údaje od používateľa a zabezpečuje vytvorenie nových príkladov, ktoré sú vložené do pozitívnych príkladov vzoru.

Dialógové okno vkladania príkladov sa používa len pri akciách `FollowLink`, `ExtractData`, `ForEachTag` a `DoWhileNextLink`, kde je potrebné zadávať príklady na tvorbu vzorov. Aktiváciu tohto okna je možné vykonať z menu aplikácie alebo z rolovacieho menu danej akcie (v položke menu Add Example).

Viac o používateľskom rozhraní je možné nájsť v prílohe A – Používateľská príručka prototypu.

8 Návrh konečného riešenia

8.1 Zmena špecifikácie a priority obaľovača

Hlavnou prioritou, ktorú sme si stanovili pri tvorbe obaľovača, bolo vytvorenie nového prístupu k tvorbe obaľovača pozostávajúci z nasledovných charakteristík:

- dekompozícia práce obaľovača,
- učenie akcií,
- uľahčenie vytvorenia obaľovača.

Dekompozícia práce obaľovača predstavuje ohraničenia kontextu, v ktorom pracujú jednotlivé akcie, rodičom akcií (aplikovaný predávaním lokálneho kontextu). Tento prístup bol využitý v extrahovaní dokumentu, ktorý je jednotlivými akciami v strome dekomponovaný, a ďalšie akcie pracujú už len nad relatívnou časťou získaného dokumentu. Dekompozícia bola využitá aj pri vytváraní výstupného dokumentu, kde si jednotlivé akcie uchovávajú len relatívnu pozíciu vo výstupnom objekte vzhľadom na svojho rodiča. Opísaný princíp zvyšuje flexibilitu obaľovača k zmenám v extrahovanom dokumente – v mnohých prípadoch stačí zmeniť lokalizáciu údajov len v jednej akcii a charakteristiky ostatných akcií ostanú zachované (napr. pri pridaní reklamy na webovej stránke).

Ďalšou prioritou pri implementácií obaľovača bolo vytvorenie nového prístupu k lokalizácií údajov v extrahovanom dokumente – učenie akcií. Používateľ zadaním pozitívnych a negatívnych príkladov naučí obaľovač, ktoré údaje sa majú extrahovať. Spôsob naučenia lokalizácie (filtra) je vytvorený na základe priradenia stratégie učenia. Týmto spôsobom možno jednoducho rozšíriť obaľovač o novú stratégiu učenia.

Veľmi dôležitou charakteristikou obaľovača z používateľského hľadiska je uľahčenie spôsobu vytvárania obaľovača – najmä zadávanie príkladov a zobrazovanie extrahovaných častí dokumentu. Opísané uľahčenie vytvárania obaľovača bolo dosiahnuté integráciou webového prehliadača do obaľovača a jeho interakciou s ostatnými časťami aplikácie. Pri integrácií prehliadača sa vyskytli viaceré ohraničenia a problémy (opísané v časti 9.5), ktoré vyplývali z používania existujúceho nástroja.

Stanovením opísaných prioritných cieľov aplikácie sme museli upustiť od niektorých požiadaviek na obaľovač určených v minulom semestri (najmä v časti prezentačného modulu aplikácie). V Tab. 8-1 sa nachádza zhrnutie zmien v požiadavkách na prípady použitia opísané v kapitole 4.

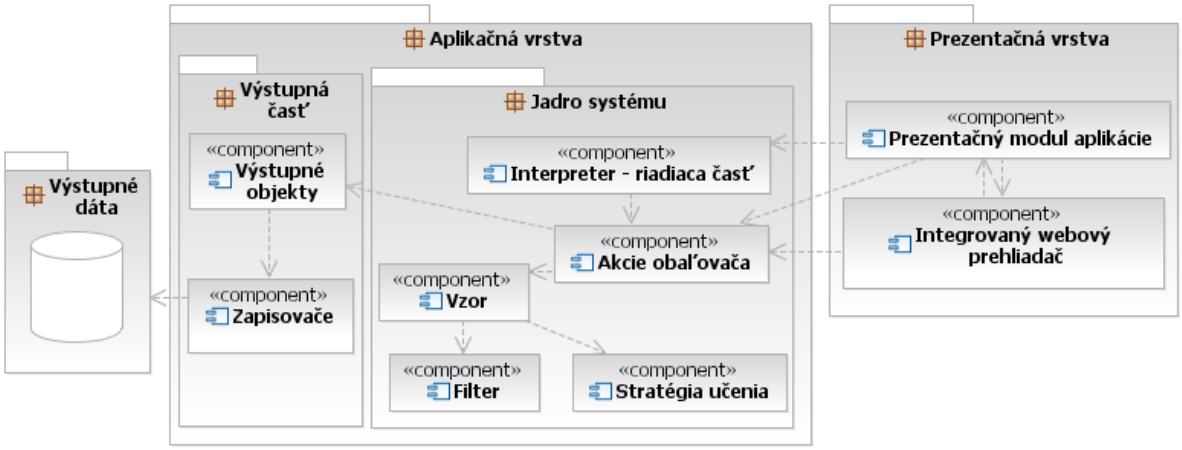
Tab. 8-1: Zmeny v prípadoch použitia

Prípad použitia	Realizácia prípadu použitia
UC01	Pri vytvorení nového obaľovača sa vytvorí prázdny obaľovač, používatelia

	špecifikujú koreňovú akciu jej editáciou.
UC02	Implementácia hlásenia konfliktov bola naplánovaná s nižšou prioritou.
UC03	Spúšťanie obaľovača bolo implementovaná bez zisťovaní konfliktov.
UC04	Charakteristika akcia bolo rozšírená o výber stratégie učenia a jej špecifikovanie. Typ vzoru je určený automaticky na základe výberu stratégie.
UC05	Používateľovi je umožnené aj opakovaný výber rovnakého príkladu, v tomto prípade je daný príklad preklasifikovaný z pozitívneho na negatívny (a naopak).
UC06	Grafické vyznačenie vzorov a manuálna zmena filtrov boli implementované v oddelených dialógoch (integrovaný prehliadač a prezentačná vrstva aplikácie).
UC07	Úprava formulárovej akcie bola naplánovaná s nižšou prioritou. Hodnota formulárových komponentov nie je špecifikovaná premennými.
UC08	Extrakcia dát je vykonávaná len do výstupných objektov.
UC09	Presúvanie akcií v strome nebolo implementované.
UC10	Označovanie konfliktných stromov nebolo implementované.
UC11	Prípad použitia plne implementovaný podľa požiadaviek.

8.2 Architektúra obaľovača

Architektúra konečného produktu sa nachádza na Obr. 8-1.



Obr. 8-1: Architektúra obaľovača

Obaľovač možno rozdeliť na 3 logické časti:

- *prezentačná vrstva* – zabezpečuje komunikáciu s používateľom počas vytvárania, zmeny a používania obaľovača,
- *aplikačná vrstva* – zapuzdruje logiku celého obaľovača,
- *výstupné dáta* – predstavujú údaje získané interpretáciou obaľovača.

Prezentačná vrstva pozostáva z častí prezentačného modulu aplikácie a integrovaného

webového prehliadača. Používateľ vytvára nový obaľovač (alebo načítava existujúci) v prezentačnom module aplikácie, kde pri vytváraní špecifikuje všetky charakteristiky obaľovača. Na výber častí, ktoré používateľ chce získať – pozitívne príklady (a tiež častí, ktoré získať nechce – negatívne príklady) zo špecifikovanej webovej stránky, slúži integrovaný webový prehliadač. Integrovaný webový prehliadač je otváraný priamo z prezentačnej časti aplikácie a okrem vyznačenia časti stránky slúži aj na vizualizáciu častí, ktoré sa budú obaľovať. Použitie integrovaného webového prehliadača má za cieľ uľahčiť prácu používateľa a odbremeniť používateľa od vnútornej reprezentácie častí webovej stránky (XPath výraz).

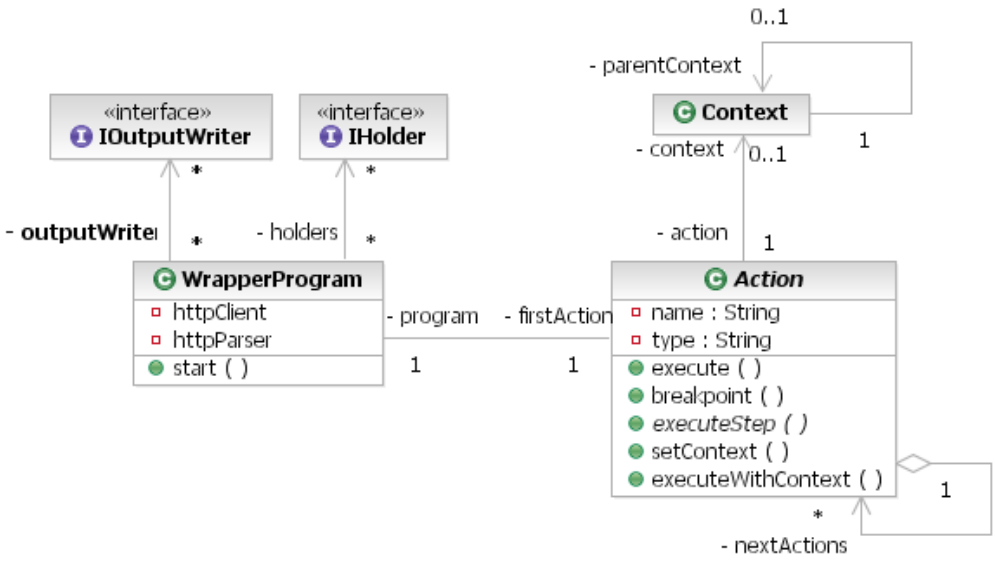
Údaje zadané používateľom v prezentačnej vrstve sú spracované v aplikačnej vrstve obaľovača. Vytvorený obaľovač pozostáva zo stromu akcií rôzneho typu. Pri vytváraní nového obaľovača sa vytvorí interpretačná – riadiaca časť (označovaná ako *WrapperProgram*). Riadiaca časť udržiava globálne charakteristiky obaľovača (napr. HTTP klient), ktoré využívajú viaceré akcie pri svojej činnosti a taktiež udržiava strom akcií. Pri žiadosti o vykonanie obaľovača interpretačná časť získa koreň stromu akcií, nastaví kontext danej akcie a zavolá jej vykonanie. Každá akcia po ukončení vykonávania svojej činnosti zmení získaný kontext (na základe logiky akcie), získa svojich potomkov, ktorým nastaví zmenený kontext a vykoná ich činnosť. Jednotlivé akcie možno špecifikovať a modifikovať v prezentačnom module aplikácie. Ak akcia získava údaje z webových stránok (extrakčné akcie), lokalizácia dát je identifikovaná používateľom pomocou integrovaného prehliadača. Extrakčné akcie majú priradené vzory, ktoré spracujú lokalizovanú časť dokumentu vo vnútornej reprezentácií (XPath výraz, klasifikátor elementov) a zapuzdrujú filter a stratégiu učenia. Filter predstavuje lokalizátor častí webových stránok, ktoré sa majú získať a jeho zmenu zabezpečuje stratégia učenia. Stratégia učenia určuje spôsob, akým sa príklad zadaný používateľom (pozitívny alebo negatívny) premietne do zmeny filtra. Použitím vzoru stratégií je možné jednoduché rozšírenie obaľovača pridaním novej stratégie učenia. Opísané časti aplikačnej vrstvy tvoria jadro systému. Medzi základné zodpovedností jadra systému patrí:

- manažment obaľovača – vytvorenie stromu akcií, načítanie (uloženie) a modifikácia existujúceho obaľovača na základe požiadaviek používateľa,
- špecifikácia akcií – určenie charakteristík akcií,
- vytváranie a zmena vzorov – naučenie (doučenie) vzoru priradeného akcii, na základe ktorého je vykonávaná lokalizácia extrahovaných častí,
- vykonanie obaľovača – vykonanie stromu akcií, získanie a spracovanie extrahovaných dát.

Údaje extrahované akciami sú zapísané vo výstupných objektoch. Výstupné objekty sú reprezentované podobne ako akcie v strome, ktorého listy predstavujú extrahované hodnoty. Štruktúra stromu výstupných objektov je vytvorená na základe stromovej štruktúry akcií, kde používateľ pri jednotlivých akciách identifikuje výstupný objekt a zadefinuje názov vetvy vzťahujúcej sa k danej akcii. Vytvorená stromová štruktúra sa zapisuje do výstupných dát pomocou zapisovačov definovaných používateľom.

8.3 Jadro systému

V tejto kapitole sa sústredíme na opis návrhu aplikačnej vrstvy produktu. Podrobný návrh s presným opisom identifikovaných entít sa nachádza v časti 9.2, kde je opísané overenie riešenia. V tejto kapitole sa sústredíme aj na zmeny návrhu jadra oproti pôvodnej špecifikácii a hrubému návrhu. Jadro systému tvorí niekoľko základných častí, ktoré sú znázornené na Obr. 8-2. Uvedený diagram nie je kompletný a zobrazuje iba základnú schému vzťahov medzi hlavnými triedami modulu `jadro`.



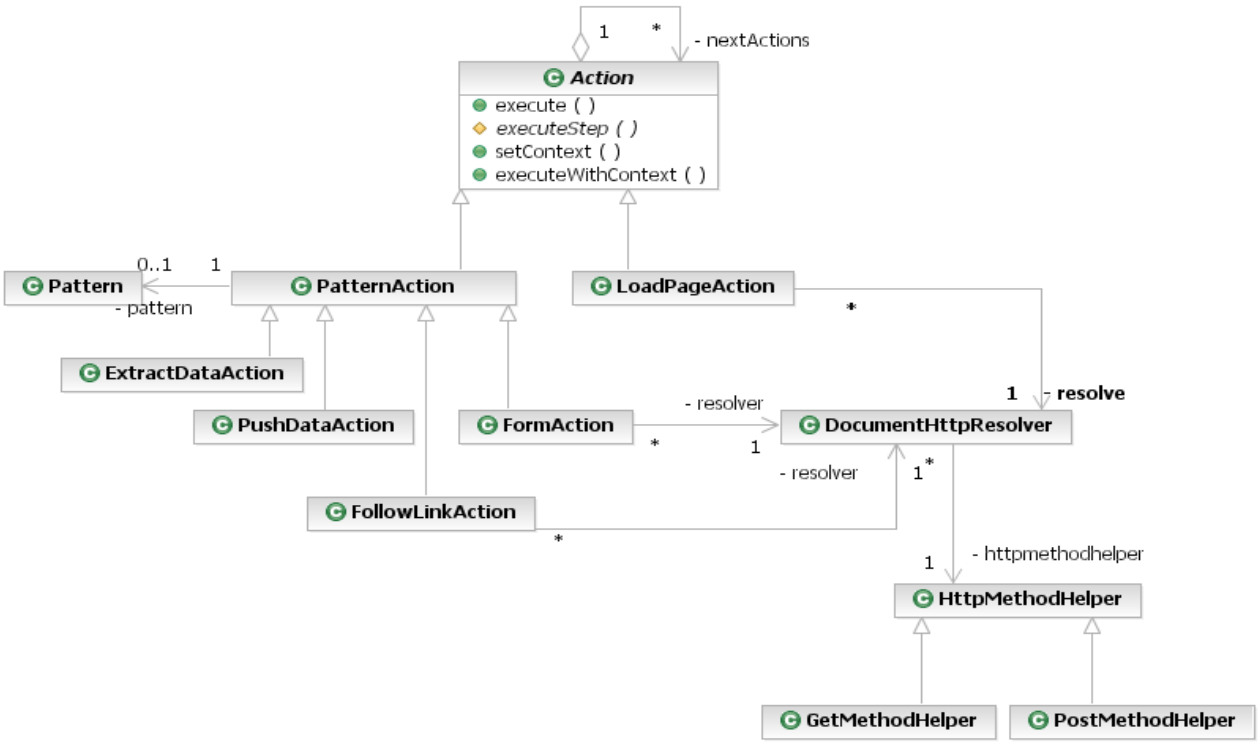
Obr. 8-2: Základné triedy modulu `jadro`

Popis jednotlivých tried:

- `WrapperProgram` - úlohou tejto triedy je poskytovať rozhranie na interpretáciu akcií a učenie akcií. Obsahuje strom akcií, dáta spoločné pre celý tento strom (HTTP klient a HTML parser, pomocou ktorých akcie získavajú dokumenty), napojenie na modul výstupov a zoznam naučených kontextov akcií (bližšie vysvetlenie v ďalšej časti dokumentu),
- `IOutputWriter` – rozhranie do modulu výstupov,
- `Action` – bázová trieda akcií obaľovača. Akcia obsahuje zoznam nasledujúcich akcií a referenciu na svoj kontext, ktorý získa počas vykonania. Akciám sa bližšie venuje nasledujúca časť dokumentu,
- `IHolder` - Rozhranie, ktoré slúži na získanie informácií z akcie pred jej spustením. Toto rozhranie bolo použité v pôvodnej verzii obaľovača v implementácii krokovača obaľovacieho program,
- `Context` – obsahuje kontextové informácie akcie, pričom každá akcia má vlastný kontext. Návrh tejto triedy je taktiež podrobnejšie opísaný v nasledujúcej časti,

8.3.1 Akcie obaľovača

Návrh akcií vychádza z dvoch základných požiadaviek na akcie, a to *učenie* (akcie identifikujú časti dokumentov na základe zadávania pozitívnych a negatívnych príkladov) a *relatívne kontexty* akcií. Prvú požiadavku sa nám podarilo overiť v prototype riešenia a návrh rámca pre učenie akcií bol prevzatý zo spomínaného prototypu, preto sa budeme tomuto rámcu venovať minimálne. Popis a návrh konkrétnych stratégií učenia je uvedený v časti 8.3.5. Typy akcií, ktoré boli navrhnuté a implementované, vychádzajú z pôvodnej špecifikácie, pričom sme museli aplikovať určité zmeny, ktoré sa ukázali potrebné až pri návrhu alebo implementácii systému. Model akcií sa nachádza na Obr. 8-3.



Obr. 8-3: Model akcií

Popis jednotlivých akcií:

- `Action` – bázová trieda akcií. Obsahuje dcérske akcie a kontext (definícia kontextu a spôsob práce s kontextom je uvedený v nasledujúcej časti). Akcia umožňuje dva druhy vykonania:

  1. štandardné vykonanie (metóda `execute()`) – akcia si prečíta informácie z kontextu, vykoná svoje telo a vytvorí kontexty pre svoje dcérske akcie a postupne ich všetky vykoná znova metódou `execute()`,

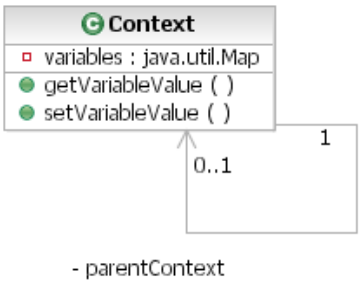
  2. vykonanie za účelom vytvorenia obaľovača (metóda `executeWithContext()`) – pri vytvorení novej akcie obaľovača je potrebné získať kontext, ktorý vytvorí jej rodičovská akcia. Na to, aby bolo možné tento kontext získať, slúži práve toto vykonanie akcie. Akcia si zo vstupného kontextu prečíta informácie, vykoná svoje telo, vytvorí iba

kontext pre jednu dcérsku akciu a tento kontext vráti.

- `PatternAction` – reprezentuje akciu, ktorá pracuje so vzormi. Rámec vzorov, filtrov a vytvárania týchto vzorov pomocou učenia bol navrhnutý a popísaný už v prototype aplikácie (viac v kapitole 7 - Prototyp riešenia), preto sa mu nebudeme venovať hlbšie.
- `ExtractDataAction` – Ak spracovávame štruktúrované dáta, napr. údaje o osobe, potrebujeme mechanizmus, ktorým vytvoríme relatívne cesty k údajom o osobe. `ExtractDataAction` je akcia, ktorá pri vykonaní odfiltruje dokument tak, aby pre dcérske akcie mohli používať relatívne cesty v tomto dokumente. Naviac, keďže výstupné objekty sú implementované pomocou DOM stromov, akcia vytvorí vo výstupnom objekte novú vetvu, do ktorej zapisujú jej dcérske akcie. Cieľom tejto akcie je teda relativizácia dokumentu a výstupných objektov.
- `PushDataAction` – akcia, ktorá zapisuje získané dáta do výstupných objektov. Táto akcia nemôže obsahovať dcérske akcie.
- `LoadPageAction` – akcia, ktorej úlohou je načítať HTML stránku prostredníctvom protokolu HTTP. Adresu stránky dokumentu musí nastaviť do akcie používateľ. Akcia používa triedu `HttpDocumentResolver` a metódu HTTP/1.1 GET (metódu vytvára objekt triedy `GetMethodHelper`). Načítaný dokument vloží do kontextov, ktoré vytvorí pre dcérske akcie.
- `FollowLinkAction` – akcia, ktorá si získa v odfiltrovanom dokumente odkaz (element *<a>*) a tento odkaz nasleduje rovnako ako metóda `LoadPageAction`. Akcia `FollowLinkAction` umožňuje naviac nastaviť mód opakovania, čo znamená, že po vykonaní všetkých dcérskych akcií sa akcia znova vykoná nad získaným dokumentom. Takéto nastavenie `FollowLinkAction` sa dá použiť v prípade, že používateľ chce získavať dáta zo stránky pokiaľ existuje „next“ odkaz.
- `FormAction` – akcia, ktorá získa v odfiltrovanom dokumente formulár, skontroluje nastavené parametre s parametrami formulára, získa metódu, ktorou sa odosiela formulár a odošle formulár prostredníctvom objektu triedy `HttpDocumentResolver`. Načítanú odpoveď vloží do kontextov svojich dcérskych akcií.

8.3.2 Kontext akcií

Kontext akcie je miesto, z ktorého akcie získavajú informácie potrebné pre svoje vykonanie. Tieto informácie sú do kontextu zapisované rodičovskými akciami počas ich vykonania, teda majú dynamickú povahu. Medzi takéto informácie patria aktuálny dokument, „cookies“, aktuálne výstupné objekty. Kontext je reprezentovaný pomocou triedy `Context` (Obr. 8-4).



Obr. 8-4: Trieda `Context`

Účelom tejto triedy je poskytovať zapisovanie dynamických informácii a získavanie dynamických informácií, ktoré boli zapísané akciou, ktorá sa nachádza vyššie v strome. Riešenie tejto požiadavky spočíva v mape premenných, do ktorej si môže akcia zaregistrovať informáciu pod zvoleným identifikátorom, a v princípe rodičovských kontextov. V prípade, že akcia vykonáva dcérske akcie, nastavuje im nový kontext, ktorého rodičom je kontext tejto akcie. Ak si potom dcérska informácia získava informácie zo svojho kontextu a požadovaná informácia sa v kontexte nenachádza, získa si túto informáciu z rodičovského kontextu atď.

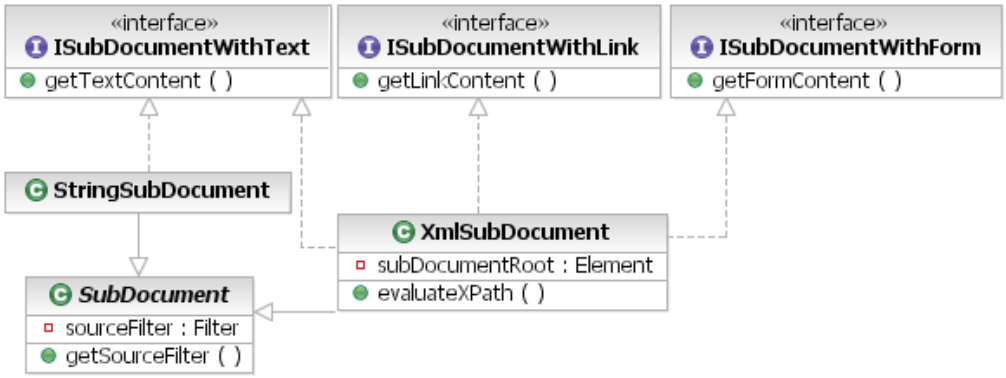
Počas učenia nemôže byť kontext zapisovaný do akcie, avšak pri vytváraní novej akcie je potrebné nastaviť jej kontext, aby sa mohla akcia napríklad naučiť vzor. Túto situáciu riešime tak, že trieda `WrapperProgram` si udržuje zoznam naučených kontextov. Po vytvorení novej akcie sa táto akcia registruje do `WrapperProgramu` tak, že sa spustí s nastaveným kontextom a kontext, ktorý vznikne takýmto učením, uloží do asociovaného objektu typu `WrapperProgram`..

8.3.3 Dokument – základ obaľovania

Kľúčovým pojmom domény obaľovačov je dokument, keďže účelom obaľovača je získať štruktúrované informácie z neštruktúrovaných dokumentov. Vo fáze prototypovania aplikácie sme navrhli a implementovali model dokumentu, ktorý sme ešte rozšírili v návrhu finálnej aplikácie. Zistili sme, že nad dokumentom vykonávame štyri druhy operácií:

1. filtrovanie – úspešne implementované vo fáze prototypovania pomocou vzorov a filtrov (do finálnej aplikácie bola zapracované rozšírenia uvedené v časti 8.3.5),
2. získanie obsahu vo forme reťazca,
3. získanie odkazu vo vhodnej reprezentácii – momentálne pracujeme nad HTML stránkami, ale odkazy existujú aj v iných typoch dokumentoch (napr. PDF dokumenty),
4. získanie formulára vo vhodnej reprezentácii.

Dokument reprezentujeme triedou `SubDocument` (návrh tejto triedy a realizácia operácii s dokumentami sú znázornené na Obr. 8-5).



Obr. 8-5: Model dokumentov

- `SubDocument` – bázová trieda dokumentov, asociuje filter, ktorým bol dokument vytvorený,
- `ISubDocumentWithText` – rozhranie, ktoré nesie informáciu o tom, že subdokument môže obsahovať text,
- `ISubDocumentWithLink` – rozhranie, ktoré nesie informáciu o tom, že dokument môže obsahovať odkaz,
- `ISubDocumentWithFormulár` – rozhranie, ktoré nesie informáciu o tom, že dokument môže obsahovať formulár,
- `XmlSubDocument` – dokument, ktorým reprezentujeme HTML stránku. HTML stránka môže obsahovať text, odkazy aj formuláre. Vnútorne je `XmlSubDocument` reprezentovaný ako DOM strom, pričom obsahuje element, ktorý znamená vrchol dokumentu.

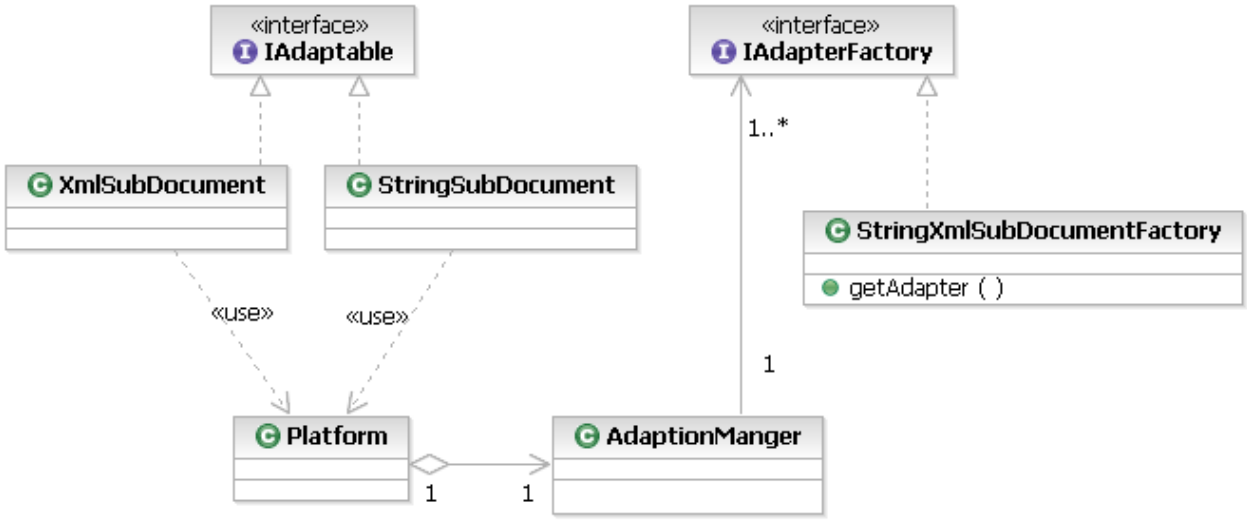
Keďže cieľom projektu je poskytovať rámec aj pre iné projekty, bolo potrebné navrhnúť dostatočne flexibilný model dokumentov. Príkladom je implementácia textového dokumentu, ktorý nepozná ani odkazy ani formuláre (trieda `StringSubDocument` – táto trieda je súčasťou jednej z diplomových prác na FIIT STU).

8.3.4 Adaptačný rámec

Jedným z cieľov projektu (časť 3.1) bolo tiež vytvoriť jednoducho rozšíriteľnú architektúru systému. Pre potrebu použitia viacerých stratégií učenia bolo nutné navrhnúť univerzálne riešenie transformácie medzi jednotlivými reprezentáciami dokumentov. Takýmto riešením je adaptačný rámec.

Návrh adaptačného rámca vychádza z návrhového vzoru Adaptér [10] a riešenia adaptácie v IDE platforme Eclipse[9]. Základom riešenia je rozhranie `IAdaptable` (Obr. 8-6).

9 Pre viac informácií pozri dokumentačný projekt Eclipse dostupný na: http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/org/eclipse/core/runtime/IAdaptable.html



Obr. 8-6: Diagram vybraných tried adaptačného rámca

Rozhranie `IAdaptable` musí byť implementované každým (sub)dokumentom, ktorý je predmetom adaptácie. Dokumenty, ktoré implementujú toto rozhranie, delegujú volanie adaptácie na objekt `AdaptionManager`, ktorý je súčasťou statickej `Platform` (odkaz na 8.3.6). Trieda `AdaptionManager` je miestom, kde prebieha manažment medzi jednotlivými typmi dokumentov. Obsahuje zoznam tovární (*factories*), ktoré majú implementovaný samotný mechanizmus transformácie z jedného typu na druhý.

Univerzálnosť riešenia spočíva v možnosti využitia adaptačného rámca nielen na adaptáciu medzi dokumentami, ale na akékoľvek transformačné účely. Rámec poskytuje možnosť návrhu a implementácie ľubovoľného počtu tovární, ktoré budú vykonávať transformácie medzi jednotlivými typmi dát. Implementácia tovární je sústredená na jedno logické miesto a preto je toto riešenie prehľadné a ľahko používateľné.

8.3.5 Stratégie učenia

Stratégie učenia prestavujú spôsob, akým chceme používateľovi uľahčiť spôsob zadávania jednotlivých častí dokumentu vyberaných na ďalšie spracovanie v etape tvorby obaľovača a aj samotného procesu obaľovania zadaného dokumentu. Keďže obaľovač je primárne určený na obaľovanie webových stránok, brali sme túto vlastnosť spracovávaných dokumentov do úvahy a s ohľadom na tento fakt boli vytvárané stratégie aj navrhnuté.

Základné prístupy v stratégiách učenia

S ohľadom na požiadavky zadania a z nich vytvorenej špecifikácie požiadaviek na produkt boli implementované viaceré stratégie učenia. V prvej etape vývoja projektu boli navrhnuté a implementované dva základné druhy stratégií učenia. Tieto druhy sa rozlišujú na základe spôsobu, akým pristupujú k spracovávaniu dokumentu, t.j. akým spôsobom prebieha výber časti dokumentu, ktorú sa obaľovač učí na základe používateľských vstupov. Základnými prístupmi, ktoré sme

definovali, boli:

- výber na základe XPath výrazu,
- prístup k časti dokumentu ako k reťazcu.

Ako vyplýva z rozdelenia, učenie sa zameriava v prvom prípade na prístup k dokumentu v zmysle prístupu k DOM stromovej architektúre webovej stránky ako *well-formed* XML dokumentu.

Druhá skupina navrhovaných stratégií pristupuje k spracovávanej časti dokumentu ako k reťazcu, čiže sa predpokladá, že bude prebiehať určitá analýza postupnosti znakov reťazca, jej ďalšie vyhodnotenie a spracovanie na základe poznatkov získaných v rámci tohto vyhodnotenia.

Etapa prototypovania

V etape prototypovania boli vytvorené len stratégie orientujúce sa na učenie sa na základe XPath výrazov. Do prototypu boli takto navrhnuté a implementované stratégie učenia:

- *opakujúca stratégia učenia* – stratégia učenia určená najmä na účely prototypovania, slúžila najmä na overenie funkčnosti jednotlivých častí prototypu.
- *jednoduchá stratégia učenia* – táto stratégia predstavovala prvé kroky v procese učenia sa obaľovača. Realizovala jednoduchý princíp zjednodušovania pozitívnych príkladov.

Obe stratégie sa zaoberali iba spracovaním pozitívnych príkladov zadaných používateľom. Tieto stratégie sú bližšie popísané v dokumentácii prototypu (pre viac informácií pozri časť 7.3) a ďalej sa nimi v tejto kapitole nebudeme zaoberať.

Návrh konkrétnych stratégií učenia

V druhej etape sa návrh stratégií týka nasledujúcich aspektov učenia:

- rozšírenie stratégií navrhnutých a implementovaných v etape prototypovania
  - použitie reťazcovej reprezentácie pri učení
  - interaktívne zadávanie príkladov
  - tvorba negatívnych príkladov
- návrh novej stratégie učenia na základe atribútov

Tieto body sú postupne opísané v nasledujúcich častiach.

Použitie reťazcovej reprezentácie pri učení

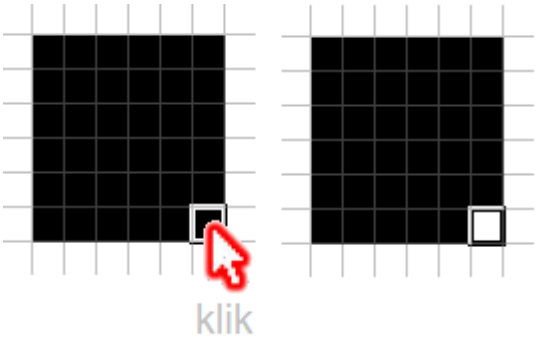
V oblasti prístupu k spracovávanému dokumentu ako k reťazcu neboli v rámci nášho projektu ďalej došpecifikované konkrétne stratégie. Na základe priebežne vyhodnocovaných cieľov projektu sme po konzultácii so zadávateľom projektu vytvorili iba určitý základný podporný rámec (framework) pre podporu takéhoto spôsobu učenia. Príprava rámca zahŕňala vytvorenie na to vhodného typu subdokumentu a rozhrania umožňujúceho ďalšiu integráciu týchto stratégií. Konkrétnymi integračnými prvkami pre použitie iných stratégií sa zaoberajú časti dokumentu popisujúce jadro systému (časť 8.3).

Keďže týmto prístupom k selekcii v rámci dokumentu sa zaoberali iné projekty prebiehajúce na fakulte nesúvisiace s našim tímom, vytvorené stratégie a ich opis nie sú súčasťou nášho projektu a nebudú v tomto dokumente bližšie popisované (projekty sa napr. zaoberali analýzou dokumentu pomocou Markovovských automatov, a pod).

Zadávanie príkladov

Zmeny sa netýkali len samotného učenia ako takého. V druhej etape projektu bola významným priblížením sa k potrebám používateľa, k jeho pohodliu a jeho ľahšej interakcii so systémom zmena spôsobu zadávania príkladov pri výučbe obaľovača. Od spôsobu zadávania jednotlivých príkladov ručne do poľa dialógového okna vo forme XPath výrazu, kde si používateľ vyberal, či ide o alebo negatívny príklad, sme sa presunuli k pohodlnejšiemu zadávaniu príkladov pomocou myši. Používateľovi sa zobrazuje obaľovaná časť dokumentu, v ktorej si kliknutím myši volí pozitívne príklady. Ak sa rozhodne pre zadanie negatívneho príkladu, jednoducho pred kliknutím stlačí a počas samotného kliknutia drží stlačenú kláves *Control*.

Algoritmus rozhodovania medzi tým, či ide o pozitívny alebo negatívny príklad, je v svojej podstate jednoduchý a vychádza z toho, že používateľ zadáva príklady interaktívne. Vychádzame teda z toho, že po zadaní jedného príkladu a pred zadaním ďalšieho príkladu má používateľ možnosť náhľadu na priebežný výsledok učenia, teda vie čo je aktuálne vybranou časťou dokumentu. Ak teda teraz zadá príklad, ktorý sa nachádza vo vybranej časti, t.j. akoby znovu vyberal už vybranú časť dokumentu, môžeme predpokladať, že chce túto časť, práve naopak, odstrániť, a teda ide o negatívny príklad (viď Obr. 8-7).



Obr. 8-7: Zadanie negatívneho príkladu

Doplnenie učenia o negatívne príklady

V úsilí o rozvíjanie spôsobov učenia sa obaľovača z používateľských vstupov sme sa zamerali na spôsoby, ktoré k spracovávanému dokumentu pristupujú ako k DOM-stromu, čiže vychádzajú z toho, že dokumentom je webová stránka so stromovou štruktúrou HTML značiek (tagov). Dôvody, prečo sme sa sústredili na túto oblasť, sú uvedené v predošlej podkapitole.

Aj ciele v tejto oblasti boli ešte priebežne usmerňované a aj pozmenené. Prvým cieľom bolo rozšírenie jednoduchej stratégie učenia z etapy prototypovania o spracovanie negatívnych používateľských príkladov. Princíp spočíval v tom, že akcia, ako doposiaľ, generalizovala pozitívne

príklady, ale potom dostala negatívny príklad z tej oblasti dokumentu, ktorá bola výsledkom filtrácie dokumentu stratégiou vytvoreným filtrom. Tento negatívny príklad mala spracovať tak, aby sa potom vo vyfiltrovanom dokumente daný negatívny prvok neobjavil. Túto situáciu zobrazuje vyššie uvedený obrázok (Obr. 8-7). Algoritmus činnosti jednoduchej stratégie učenia bol vysvetlený v dokumentácii k prototypu.

Ako bolo v úvode k rozširovaniu učenia o nové stratégie spomenuté, rozvoj stratégií si vyžiadal určité rozšírenia systému. Za účelom implementácie tejto stratégie bol vyvinutý nový typ filtra – *CompoundFilter*. Ide o zložený filter so schopnosťou filtrovať dokument na základe viacerých XPath filtrov. Tento filter bol navrhnutý s cieľom splniť požiadavku, aby mohol byť dokument filtrovaný viac ako len jedným XPath filtrom a aby boli výsledky filtrovania viacerými XPath filtrami súčasťou jedného procesu filtrácie, t.j. výsledkom jedného požiadavky na filtráciu kladenej na daný filter.

Keďže sa naďalej zaoberáme filtráciou dokumentu na základe jeho DOM štruktúry a na výber časti dokumentu používame XPath výrazy, ide o rozšírenie pôvodného XPathFiltra na filter uchovávajúci viaceré XPath výrazy, ktoré sa potom všetky postupne použijú pri filtrovaní filtru predloženého dokumentu.

Učenie na základe atribútov

Ďalším prístupom v stratégii učenia je reprezentácia zadaných príkladov pomocou atribútov. Jednotlivé atribúty budú sledovať vybrané charakteristiky príkladov, kde medzi základné charakteristiky príkladov sme si zvolili vizuálnu charakteristiku vybranej časti dokumentu (štýly, fonty, veľkosť písma a iné), typ elementu (napr. bunka v tabuľke), hĺbku v DOM strome dokumentu a pri tabuľkách určujeme tiež hĺbku elementu v tabuľke (označovane ako index – napr. 2. riadok tabuľky). Pre jednotlivé príklady používateľ vyberie triedu, do ktorej daný príklad – pozitívny alebo negatívny – patrí.. Následne bude vytvorený klasifikátor, ktorý sa bude učiť pravidlá klasifikácie príkladov (pozitívne, negatívne) na základe používateľom zadaných príkladov. Pravidlá klasifikátora budú určovať, aké hodnoty atribútov predstavujú pozitívne a negatívne príklady. Opísaným spôsobom bude môcť klasifikátor určiť triedu elementov, na ktoré nebol natrénovaný. Na reprezentáciu a učenie klasifikátorov sme využili existujúce prístupy zo strojového učenia (napr. rozhodovací strom alebo Bayesov klasifikátor), kde trénovacie dáta predstavujú príklady zadané používateľom.

Použité atribúty

Základnou črtou učenia na základe atribútov je, že o označených elementoch, ktoré sú používateľom zadanými príkladmi, sa zisťuje pokiaľ možno čo najviac ich vlastností. Medzi dôležité atribúty určujúce triedu príkladu možno považovať vlastnosti, ktoré sa prejavujú vo vizuálnej stránke, čiže pri zobrazovaní elementu. Týmito vlastnosťami sú atribúty zobrazovaných elementov, konkrétne teda ide o atribúty ovplyvňujúce zobrazenie (veľkosť písma a podobne).

V súčasnom webe hrajú významnú úlohu pri realizácii návrhu vzhľadu webovej stránky

použité štýly zobrazenia a práve štýly môžu byť v mnohých prípadoch tou vlastnosťou elementov, ktorá bude určovať triedu daného príkladu (napríklad oddelenie stĺpcov na základe farby pozadia). Vzhľadom na tento fakt sme chceli dosiahnuť získanie informácie okrem iného práve aj o štýle zadaného príkladu. Informácie o štýle príkladu sú získané z integrovaného prehliadača.

Medzi ďalšie použité atribúty, ktoré sme sa rozhodli aplikovať pri určovaní triedy príkladu, patrí:

- typ elementu – určuje jeden z definovaných typov elementov (tagov v prípade webovej stránky, napr. bunka tabuľky),
- trieda použitého štýlu – pri vytváraní elementov v štandarde XHTML je možné elementom zadefinovať triedu štýlu, ktorej vlastnosti sú zadefinované v kaskádových štýloch,
- hĺbka elementu v strome dokumentu,
- index príkladu – pri elementoch vnorených (vizuálne) vrámci iného elementu, je možné určovať index daného elementu vrámci iného elementu, napr. pri stĺpcoch tabuľky je možné určiť index vrámci danej tabuľky (napr. piaty stĺpec).

Princíp učenia

Učenie na základe atribútov prebieha v princípe tak, že sa vytvorený klasifikátor naučí klasifikovať predložené príklady do jednej z tried. V našom prípade budeme mať 2 triedy: jednu triedu tvoria elementy, ktoré budú vo vytvorenom subdokumente po tom, čo sa pôvodný dokument predloží na filtráciu, t.j. ide o elementy, ktoré sa stratégia naučila vybrať. Druhú triedu tvoria elementy, ktoré sa do výberu na základe aktuálneho naučenia nedostanú. Preto aj triedy nazveme „pozitívny“ a „negatívny“ príklad.

Klasifikátor na základe predložených príkladov (trénovacia vzorka) spolu s ich zaradením do tried uskutoční učenie, v ktorom sa snaží odvodiť vzťahy, na základe ktorých boli príklady priradené do jednotlivých tried. Potom, po naučení, môže klasifikátor vyhodnocovať predložené príklady, ktoré už neobsahujú svoje zaradenie do tried. Toto zaradenie do tried vykoná klasifikátor. Pri opísanej klasifikácii sme použili existujúce prístupy v oblasti strojového učenia (*machine learning*) – rôzne druhy klasifikátorov (zameniteľných) spolu s ich princípmi učenia sa – rozhodovacie stromy, Bayesov klasifikátor, asociačné pravidlá.

Nasledujúca tabuľka (Tab. 8-2) demonštruje jednoduchý príklad učenia a následného vyhodnocovania príkladov klasifikátorom. Je vidno, že klasifikátor sa naučil do triedy „pozitívny“, zaraďovať ľubovoľné modré objekty.

Tab. 8-2: Príklad učenia a vyhodnocovania jednoduchého klasifikátora

Učenie				Vyhodnocovanie		
Názov	Farba	Trieda		Názov	Farba	Trieda
auto	modrá	pozitívny		brána	modrá	pozitívny
auto	červená	negatívny		brána	červená	negatívny

bicykel	modrá	pozitívny		auto	žltá	negatívny
bicykel	červená	negatívny		váza	modrá	pozitívny

Na podobnom princípe funguje aj táto stratégia učenia sa vzorov. Klasifikátor sa naučí, aké hodnoty musia mať jednotlivé atribúty elementov, aby boli do výberu klasifikované ako vhodné.

Požadované prípravy

Takýto spôsob učenia má určité podmienky, ktoré musia byť splnené ešte pred tým, ako sa začne učenie vzoru, t.j. v tomto prípade podmienok, ktoré musia spĺňať atribúty elementu, aby bol klasifikovaný do výberu, ako vybraná časť filtrovaného dokumentu. Táto podmienka implicitne vyplýva aj z uvedenej jednoduchej tabuľky učenia a klasifikovania príkladov. Ide o to, aby mali všetky príklady všetky uvažované atribúty. Medzi všetky uvažované atribúty patrí každý taký atribút, ktorý sa nachádza aspoň u jedného zo zadávaných učiacich alebo hodnotených príkladov.

Pre tvorbu klasifikátora a riešenie učenia jednotlivých príkladov návrh využíva funkcionalitu poskytovanú projektom Weka[10], ktorá implementuje požadované klasifikátory. Aj tento systém pre učenie vyžaduje, aby mali príklady jednotnú sadu atribútov.

Cieľom však nie je používanie len obmedzeného počtu konkrétnych atribútov, ktoré navyše nemusia byť všetky prítomné v každom elemente, ktorý bude ponúknutý na klasifikáciu. Cieľom je vytvoriť podporu pre klasifikovania na základe rôznych a rôzne dostupných atribútov.

Na riešenie tohto problému slúži príprava predkladaných príkladov pre klasifikátor. Keď sa ide klasifikátor učiť, vezme sa množina všetkých príkladov, ktoré sa bude učiť. Táto množina je potom normalizovaná, t.j. každý príklad bude mať rovnaké atribúty ako všetky ostatné príklady. Postup normalizácie pozostáva z dvoch prechodov množinou príkladov:

1. Pri prvom prechode sa zostaví množina atribútov, čiže sa prejdú postupne všetky príklady a u každého príkladu sa skontroluje, či všetky jeho atribúty sú v zostavovanej množine atribútov. Ak niektorý atribút v množine nie je, pridá sa.
2. Pri druhom prechode sa kontroluje každý príklad, či obsahuje všetky atribúty z množiny atribútov. Ak príklad niektorý z atribútov neobsahuje, atribút sa mu dodá a hodnota tohto atribútu sa nastaví na neexistujúcu, resp. chýbajúcu (*!missing*).

10 http://www.cs.waikato.ac.nz/~ml/weka/index.html

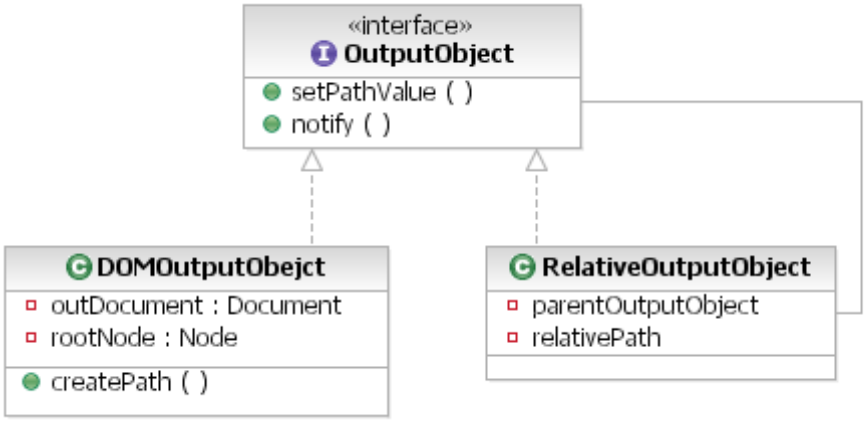
8.3.6 Platforma

`Platform` je statická trieda vytvorená na účely registrácie rozličných druhov informácií. Do `Platform`-u sa môžu registrovať objekty do určitých skupín. Potom v rámci každej skupiny je objekt reprezentovaný menom svojej triedy a vlastní registračnú položku v `Platform`-e. Táto položka môže obsahovať ľubovoľný počet dvojíc kľúč, hodnota, v ktorých je možné ukladať informácie o registrovanej triede.

V systéme sa `Platform` využíva na registráciu stratégií učenia. Umožňuje tiež, aby fungoval ako konfiguračný objekt nesúci základné nastavenia systému – nad touto možnosťou sa v rámci vývoja (aj budúceho) uvažovalo.

8.4 Výstupné objekty

Výstupné objekty (*OutputObject*) udržiavajú extrahované dáta v stromovej štruktúre a sú následne pomocou dostupných zapisovačov transformované do výstupného prostredia. Časť modelu výstupných objektov sa nachádza na Obr. 8-8.



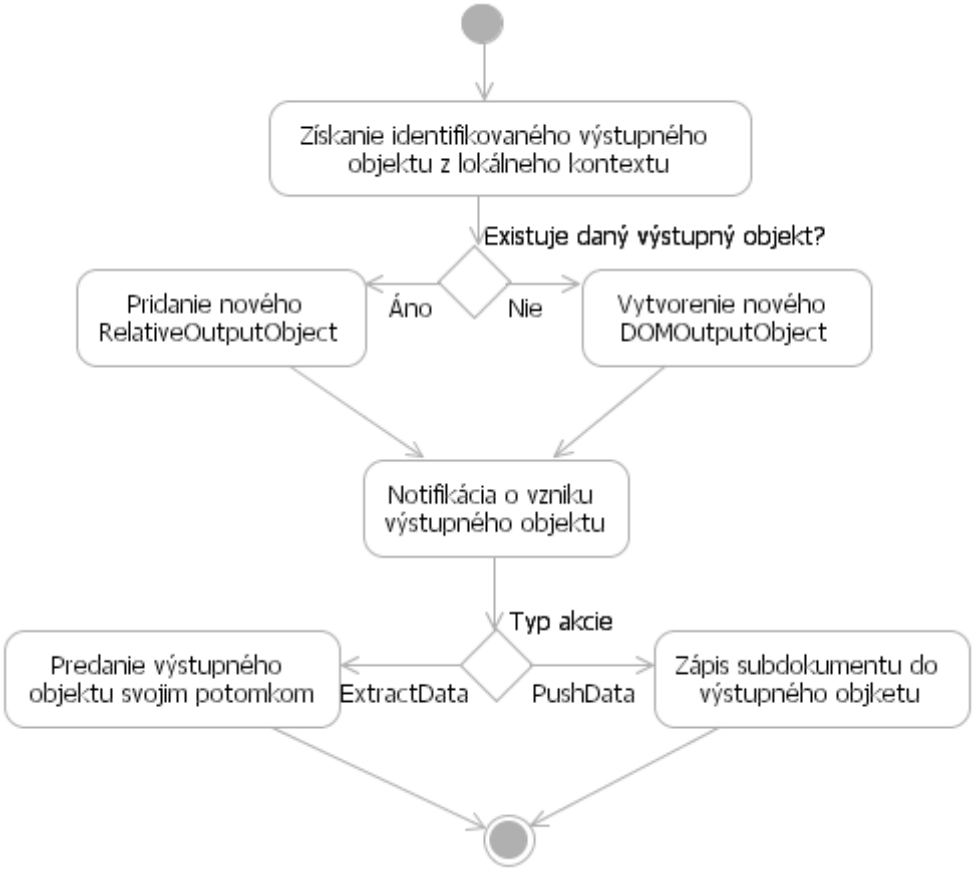
Obr. 8-8: Model výstupných objektov

Výstupné objekty sú reprezentované v podobe DOM dokumentov (Document Object Model), ktoré sa spolu s koreňovým uzlom dokumentu udržiavajú v triede *DOMOutputObject*. Táto trieda zodpovedá za vytvorenie nového dokumentu a pridávanie uzlov spolu s ich hodnotami (extrahované dáta) do existujúceho dokumentu. Lokalizácia časti výstupného objektu potrebná pri zápise extrahovaných hodnôt je určená absolútnou cestou od koreňa stromu po list stromu, kde bude daná hodnota zapísaná.

Vo vytvorenom obaľovači bol využitý princíp postupnej dekompozície extrahovaného dokumentu pomocou predávania si lokálneho kontextu (s extrahovaným dokumentom) medzi akciami v strome. Akcie z daného dokumentu vyberú určitú časť, ktorú predajú svojim potomkom a potomkovia danej akcie budú pracovať už len nad touto časťou dokumentu. Opísaný princíp možno aplikovať aj na výstupné objekty. Akcie budú buď vytvárať nový výstupný objekt, ak požadovaný ešte neexistuje, alebo bude pokračovať v zápise v časti, kde skončil jeho rodič, čiže

daná akcia bude pracovať už len nad dekomponovanou časťou celého výstupného dokumentu – celý výstupný dokument sa relativizuje k rodičovi danej akcie. Ak daná akcia je typu *ExtractData*, do dokumentu vo výstupnom objekte pridáva novú vetvu určenú relatívnou cestou v strome výstupného objektu vzhľadom na svojho rodiča. Ak je akcia typu *PushData*, reťazcovú hodnotu aktuálneho dokumentu zapíše do vytvoreného uzla výstupného dokumentu. Z tohto dôvodu nie je potrebné udržiavať celý výstupný dokument (v *DOMOutputObject*) pre akcie, ktoré zapisujú do existujúceho výstupného objektu, ale stačí uchovávať relatívnu cestu a odkaz na výstupný objekt svojho rodiča. Opísaným spôsob sa vytvorí nový typ výstupných objektov – *RelativeOutputObject*. Pri pokuse o zápis (metóda *setPathValue()*) extrahovaných dát do *RelativeOutputObject* sa vykoná delegácia zápisu na rodičovský výstupný objekt s predaním relatívnej cesty zápisu. Ak je rodičovský výstupný objekt *DOMOutputObject*, vykoná sa zápis pod koreňom dokumentu do uzla lokalizovaného pomocou predanej relatívnej cesty. Ak je rodičovským výstupným objektom opäť *RelativeOutputObject,* k získanej relatívnej ceste na začiatok pridá svoju relatívnu cestu a zápis deleguje na rodičovský uzol. Opísaným spôsobom sa postupne získava absolútna cesta zápisu dát vo výstupnom stromu. Extrahované dáta sa zapisujú do vytvorenej absolútnej cesty v *DOMOutputObject* metódou *setPathValue()*, ktorá vytvorí nový uzol v strome (*createPath()*) a novému uzlu priradí extrahovanú hodnotu.

Celkový proces zápisu dát do výstupného objektu v akciách obaľovača je znázornený na Obr. 8-9. V prvom kroku sa z lokálneho kontextu získa výstupný objekt rodiča s rovnakým identifikátorom a vytvorí sa nový relatívny výstupný objekt s pridanou relatívnou cestou. Ak požadovaný výstupný objekt ešte nebol vytvorený, je potrebné vytvoriť nový *DOMOutputObject*, do ktorého budú zapisovať potomkovia akcie. Každú akciu v obaľovači je možné spustiť viac krát (pre každý extrahovaný dokument). Pri nasledujúcom spustení rovnakej akcie je zápis ďalších dát vykonávaný do rovnakej absolútnej cesty ako pri predošlom spustení akcie, a preto je potrebné naplnený výstupný objekt zapísať do výstupu (napr. XML súbor) pomocou zapisovačov. Opísaný prípad môže nastať napríklad pri extrahovaní pracovných ponúk z dokumentu, na ktorom sa nachádza viacero ponúk (cieľom je extrahovanie názvu a platu ponuky). Pri extrahovaní druhej ponuky je potrebné prvú ponuku zapísať na výstup – pri pokuse o zápis názvu ponuky, ktorý sa už vo výstupnom objekte nachádza (prvá ponuka). Tento zápis je vykonávaný pomocou notifikácie (metóda *notify()*) o vzniku nového výstupného objektu. Notifikácia je rovnako ako zápis delegovaná na svojho rodiča až do výstupného objektu *DOMOutputObject*, ktorý vykoná zápis súčasného výstupného objektu, ak sa v ňom nachádza uzol s notifikovanou absolútnou cestou (napr. offer/name pre názvy pracovných ponúk). Ak je daná akcia typu *ExtractData*, vytvorený výstupný objekt je predaný svojmu potomkovi rovnako ako lokálny kontext. Ak je danou akciou *PushData*, vykoná sa zápis subdokumentu (pomocou delegácie zápisu na svojho rodiča).



Obr. 8-9: Proces zápisu dát do výstupného objektu

8.5 Prezentačná vrstva

8.5.1 Prezentačný modul aplikácie

Grafická reprezentácia akcií

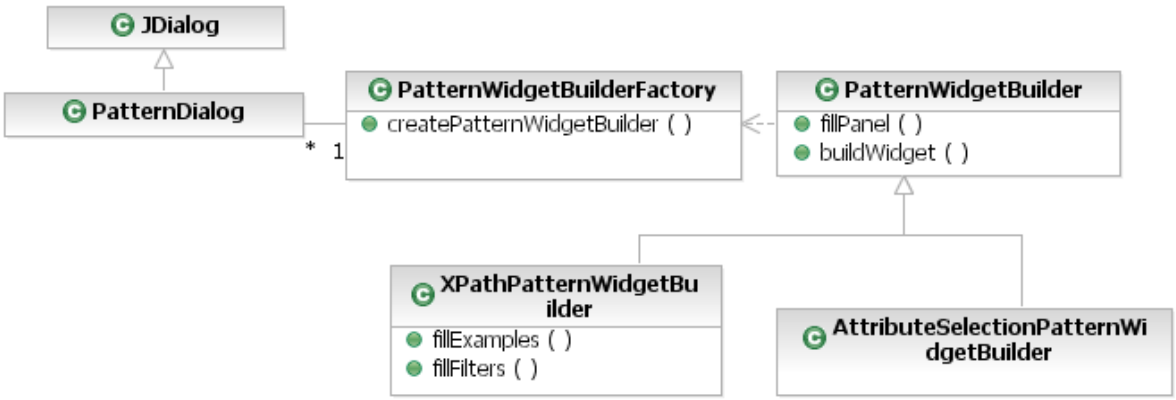
Funkciu rozhrania, prostredníctvom ktorého používateľ môže obaľovače vytvárať a upravovať, plní prezentačný modul akcií. Pri jeho koncipovaní sme vychádzali z už existujúceho rámca a z požiadavky dosiahnuť čo najlepšiu ovládateľnosť a jednoduchosť tvorby z pohľadu používateľa. Jednotlivé možnosti konfigurácie obaľovačov podáva používateľovi v prehľadnom menu. Používateľ skladá obaľovač z akcií, ktoré má rýchlo a jednoducho sprístupnené v bočnej lište. Z nich je možné vytvárať strom požadovanej funkcionality budúceho obaľovača a tiež sprístupniť integrovaný prehliadač.

Prezentačná vrstva získané údaje od používateľa distribuuje ďalej do aplikačnej vrstvy programu priamo jednotlivým akciám. Vytvára tiež akýsi medzistupeň medzi údajmi interaktívne získanými v integrovanom prehliadači a aplikačnou vrstvou, do ktorej sa údaje prenášajú prostredníctvom dialógových okien akcii. Svojou štruktúrou jednotlivé dialógy kopírujú štruktúru logiky akcií v aplikačnej vrstve a ich spôsob implementácie je bližšie popísaný v časti 9.4.1.

Celá prezentačná vrstva bola navrhovaná s ohľadom na možnosť jednoduchého rozšírenia do budúcnosti. Pridanie novej akcie, či kategórie akcií do postrannej lišty predstavuje pre vývojára veľmi ľahko riešiteľný problém.

Zobrazenie naučených vzorov – View Pattern

Používateľovi je potrebné naučený vzor prezentovať z dôvodu kontroly naučeného filtra vo vzore. Zobrazenie vzoru je závislé od vnútornej reprezentácie filtra (XPath výraz, klasifikátor) získanej na základe stratégie učenia. Z tohto dôvodu je potrebné presunúť zodpovednosť prezentovania reprezentácie vzorov z časti, kde sa zobrazuje dialóg *View Pattern* do časti, ktorá bude špecifická pre každú reprezentáciu vzoru. Základný model vykresľovania dialógov je zobrazený na Obr. 8-10. *PatternDialog* predstavuje dialóg zobrazenia vzorov s asociovaným *PatternWidgetBuilderFactory*. *PatternWidgetBuilder* predstavuje abstraktnú triedu, ktorej zodpovednosťou je vytvorenie časti dialógu zobrazenia vzorov, ktoré sú špecifické pre každú reprezentáciu vzorov. Jednotlivé komponenty dialógu sú napĺňané v metóde *fillPanel()*, ktorá je špecifická pre rôzne typy vzorov a následne je z týchto komponentov vytvorená časť dialógu (metóda *buildWidget()*). V aplikácií boli vytvorené dve špecializované triedy na tvorbu vzorov – *XPathPatternWidgetBuilder* (zobrazovanie vzorov, ktorých filter má reprezentáciu XPath výrazu) a *AttributeSelectionPatternPatternWidgetBuilder* (zobrazenie vzorov, ktorých filter je vytvorený pomocou stratégiu učenia *AttributeSelectionLearningStrategy*). Tieto špecializované triedy implementujú metódu *fillPanel()* podľa spôsobu prezentácie vzorov (napr. v triede *XPathPatternWidgetBuilder* metóda pozostáva z dvoch častí – pridanie príkladov a filtrov do dialógu). Výber špecializovanej triedy na vytvorenie prezentácie vzorov je vykonávaný v triede *PatternWidgetBuilderFactory*, ktorá na základe typu zobrazovaného vzoru vyberie jednu z tried dedených od *PatternWidgetBuilder* (metóda *createPatternWidgetBuilder()*). *PatternDialog* následne získa časť dialógu so zobrazeným vzorom (*fillPanel()*), ktorú zobrazí v dialógu. Opísaný spôsob zobrazenia vzorov bol zvolený najmä z dôvodu jednoduchej rozšíriteľnosti, keď pri vytvorení novej reprezentácie vzorov postačuje vytvoriť novú triedu dedenú od *PatternWidgetBuilder* (spolu s rozhodovaním o jeho pridelení v metóde *createPatternWidgetBuilder()*), ktorá bude implementovať metódu *fillPanel()* a triedu *PatternDialog* nie je potrebné modifikovať.



Obr. 8-10: Model tvorby zobrazenia vzorov

8.5.2 Integrovaný webový prehliadač

Integrovaný prehliadač plní funkciu rozhrania medzi používateľom a obaľovačom. Hlavným cieľom tohto rozhrania je zjednodušenie práce používateľa pri konfigurácii samotného obaľovača, resp. pri vytváraní a modifikovaní jeho extrakčných a navigačných akcií. Táto jednoduchosť spočíva vo výbere niekoľkých vzorových príkladov (pozitívnych a negatívnych), pomocou ktorých sa obaľovač naučí, s ktorými údajmi sa bude pracovať. Jednotlivé stratégie učenia sú opísané v kapitole 8.3.5.

Počas procesu vytvárania obaľovača sa v prehliadači zobrazí webová stránka, nad ktorou používateľ pracuje. Jednoduchou interakciou vyberie z webovej stránky element, ktorý požaduje, a táto voľba sa oznámi aplikačnej vrstve, konkrétne modulu učenia, vo forme XPath výrazu. Výber a spracovanie týchto elementov závisí od použitej akcie a samozrejme od stratégie učenia.

Po každom výbere požadovaného elementu sa vykoná proces učenia, ktorého výsledkom je (po splnení určitých podmienok – napr.: výber dostatočného počtu pozitívnych príkladov) zoznam všetkých elementov, ktoré majú rovnakú štruktúru ako vzorové príklady. Všetky tieto elementy sa následne zobrazia v integrovanom prehliadači. Ak tieto elementy obsahujú ešte ďalšie vnorené elementy, výber je možný vykonať v rámci dcérskych elementov a nie na vyššej úrovni.

9 Overenie návrhu konečného riešenia

9.1 Výber implementačných prostriedkov

Pri výbere implementačných prostriedkov, knižníc a prostredí sme vychádzali z nami stanovenej špecifikácie systému. Keďže vychádzame z existujúceho projektu, používame pôvodný programovací jazyk (java 1.5), existujúce nástroje (maven2, eclipse) a podporné knižnice. K týmto knižniciam sme pridali ďalšie, ktoré zodpovedajú nástrojom, ktoré používame:

- *not-yet-commons-ssl* – knižnica, ktorá slúži na prácu s bezpečnostným protokolom ssl, je potrebná kvôli HTTP autentifikácii
- *JRex* – knižnica integrovaného prehliadača; touto knižnicou sme nahradili knižnice XUL a XPCOM zo špecifikácie, ktoré mali byť použité na rovnaký účel (viac v 9.4.2)

Keďže sme pri implementácii použili nástroje a prostriedky, ktorých použitie je bežné, nie je potrebné ich podrobne opisovať. Spomenieme však minimálne nástroj maven2, ktorý sa nám veľmi osvedčil najmä pri riešení závislostí projektu. Ukážka konfigurácie tohto nástroja za účelom riešenia závislostí sa nachádza na Obr. 9-1. Kompletná konfigurácia nástroja maven2 je umiestnená v jedinom súbore `pom.xml`.

```
<dependency>
    <groupId>org.mozilla.jrex</groupId>
    <artifactId>jrex</artifactId>
    <version>1.0b1_dom3</version>
    <scope>compile</scope>
</dependency>
```

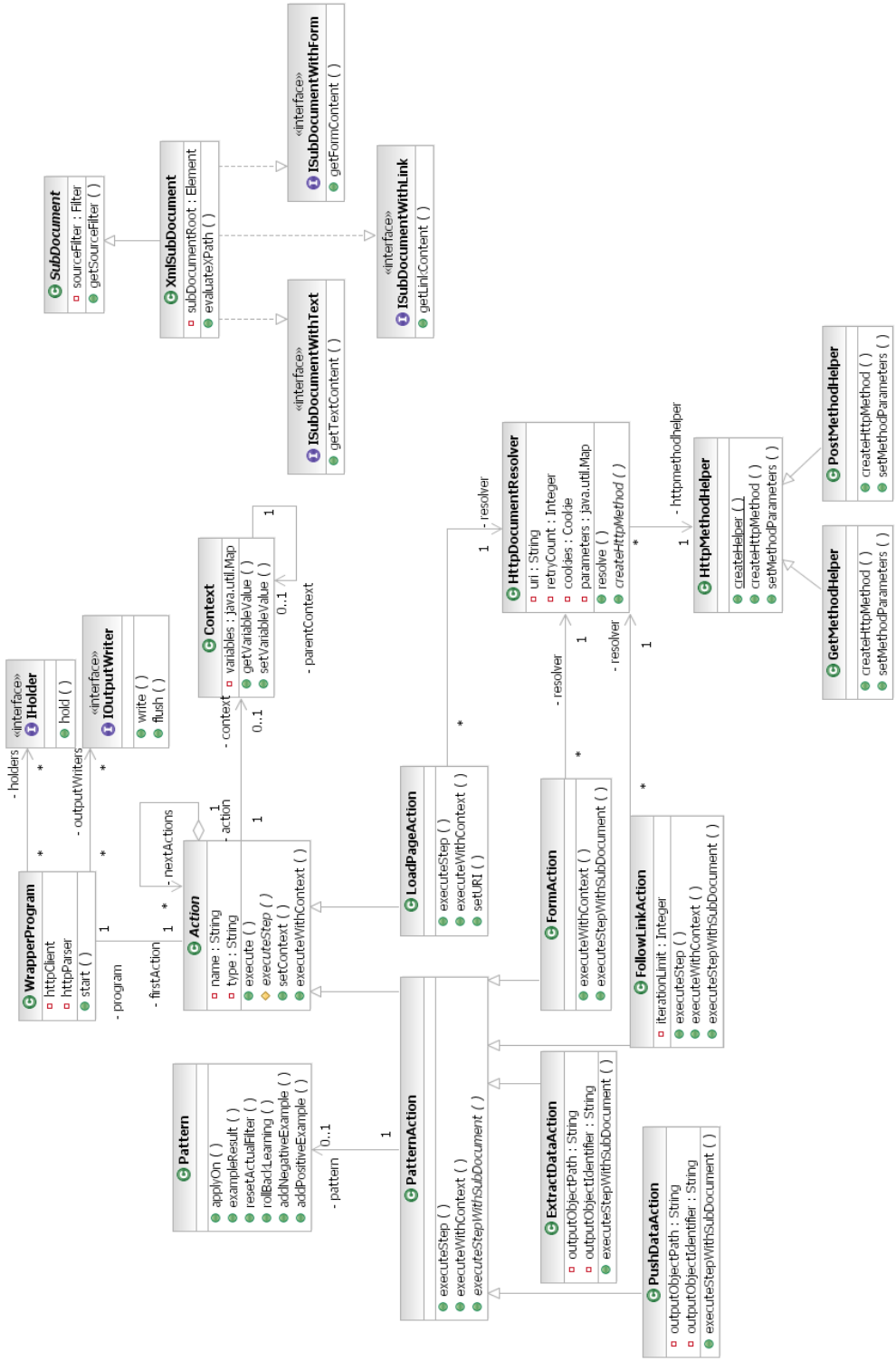
Obr. 9-1: Riešenie závislostí pomocou nástroja Maven (časť konfiguračného súboru `pom.xml`)

9.2 Implementačný model jadra systému

Kvôli prehľadnosti sme implementačný model jadra systému rozdelili na tieto časti: akcie a triedy súvisiace s akciami (`WrapperProgram`, `Context`, model dokumentu) a triedy súvisiace s učením vzorov (`Pattern`, `Filter`, stratégie učenia).

9.2.1 Akcie

Prvá časť implementačného modelu sa nachádza na Obr. 9-2.



Obr. 9-2: Implementačný model modulu jadro (časť akcie a dokumenty)

Popis jednotlivých tried atribútov a metód je uvedený v zdrojovom kóde formou Javadoc komentárov (komentáre neobsahujú diakritiku). Príklad takto okomentovanej triedy sa nachádza

na Obr. 9-3.

```
/**
 * Implementacia subdokumentu vo forme XML dokumentu. Poskytuje rozhranie na
 * manipulaciu s takouto reprezentaciou
 *
 * @author kurtak
 */
public class XmlSubDocument extends SubDocument implements IAdaptable,
ISubDocumentWithLinks, ISubDocumentWithText {

	/**
	 * Element, na ktory je naviazany aktualny subdokument. Zatial nevieme
	 * pouzivat ine nody ako elementy
	 */
	protected Element subDocumentRoot;

	/**
	 * Vyhodnoti xpath expression a vrati zoznam uzlov, ktore vyhovuju vyrazu.
	 * Dovod preco sa nevyhodnocuje priamo filter je ten ze {@link Filter}
	 * reprezentuje sposob filtrovania XPATH vyrazu)
	 *
	 * @param expression  XPath vyraz
	 * @return zoznam uzlov, ktore vyhovuju vyrazu
	 * @throws FilterException  chyba pri vyhodnocovani vyrazu
	 */
	public List<Node> evaluateXPath(String expression) throws FilterException{
		...
	}
```

Obr. 9-3: Ukážka opisu triedy a metódy

9.2.2 Učenie

Počas implementácie sme sa v súvislosti s vyvíjajúcimi sa požiadavkami venovali implementovaniu jednotlivých tried potrebných na realizáciu čiastkových cieľov. Oblasti, ktorým sme sa venovali spolu s riešeniami, ktoré sme vytvorili sú popísané v ďalších podkapitolách.

Zadávanie príkladov

Keďže sme sa rozhodli, že budeme podporovať čo najlepšie priblíženie obaľovača k používateľovi, bola implementovaná aj základná štruktúra pre podporu navrhnutého algoritmu rozlišovania pozitívnych a negatívnych príkladov. Automatizované rozhodovanie medzi pozitívnymi a negatívnymi príkladmi bolo navrhnuté, ale nebolo implementované, keďže sme sa sústredili na podstatnejšie vlastnosti obaľovača a túto rozširujúcu vlastnosť sme nechali ako možnosť na rozšírenie pre ďalšie tými, ktoré by prípadne mohli pracovať na tomto projekte v budúcnosti.

Učenie z reťazcov

Oblasť učenia reťazcov si v našom projekte vyžiadala iba vytvorenie predlohy pre dodatočnú tvorbu stratégií zaoberajúcich sa učením z reťazcov. Tejto oblasti sa venovali ľudia pracujúci mimo nášho tímu. Príkladom rozhrania pre integráciu ich prístupov k riešeniu bol navrhnutý StringSubDocument (viď Obr. 9-4).

Doplnenie učenia o negatívne príklady

K samotnej implementácii zadávania negatívnych príkladov nedošlo tak, ako to bolo v pôvodnom pláne rozšírením jednoduchej stratégie učenia, nakoľko sa úsilie presunulo do oblasti implementácie učenia na základe atribútov. Bol však implementovaný filter podporujúci filtrovanie na základe viacerých XPath výrazov, čo bola podmienka implicitne vyplývajúca z navrhovanej stratégie.

- *CompoundXPathFilter* – rozšírený XpathFilter o možnosť filtrovania pomocou viacerých XPath výrazov. Na zadaný dokument sa aplikuje filtrácia pomocou jednotlivých XPath výrazov postupne a výsledné subdokumenty sa vrátia spolu. Umiestnenie CompoundXPathFiltra medzi ostatnými filtrami znázorňuje Obr. 9-6.

Učenie na základe atribútov

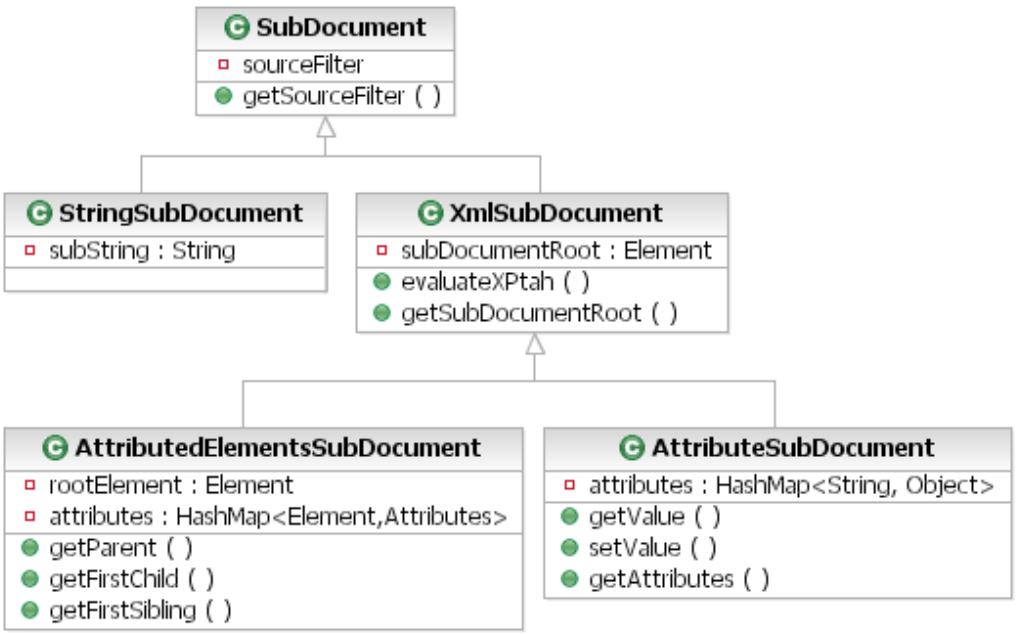
Táto stratégia učenia si vyžiadala implementáciu viacerých tried. Konkrétne sme implementovali nové typy subdokumentov, filter realizujúci filtrovanie na základe tohto typu učenia a tiež triedu reprezentujúcu samotnú stratégiu učenia. Tieto triedy sú popísané ďalej.

V oblasti získavania atribútov nastali aj isté implementačné komplikácie. Aj napriek intenzívnemu hľadaniu možností a ďalšej činnosti v tejto oblasti, vrátane kontaktovania internetových diskusných fór na stránkach nami použitého prehliadača, sa nám nepodarilo zistiť spôsob, ako z prehliadača získať informáciu o štýle (o vlastnostiach zobrazenia, ktoré definuje), ktorý sa použije pri zobrazovaní daného elementu. Problémy so zisťovaním štýlov a celkovo zobrazovacích vlastností elementov spôsobuje aj to, že platnosť nastavenej vlastnosti sa prenáša v DOM-strome z vlastnosť definujúceho uzla tohto stromu na jeho potomkov, prípadne sa môžu tieto vlastnosti nahradzovať novými hodnotami v rámci rôznych podstromov toho istého stromu. Preto sme sa rozhodli použiť v návrhu uvedenú skupinu atribútov.

Nasledujú popisy jednotlivých tried:

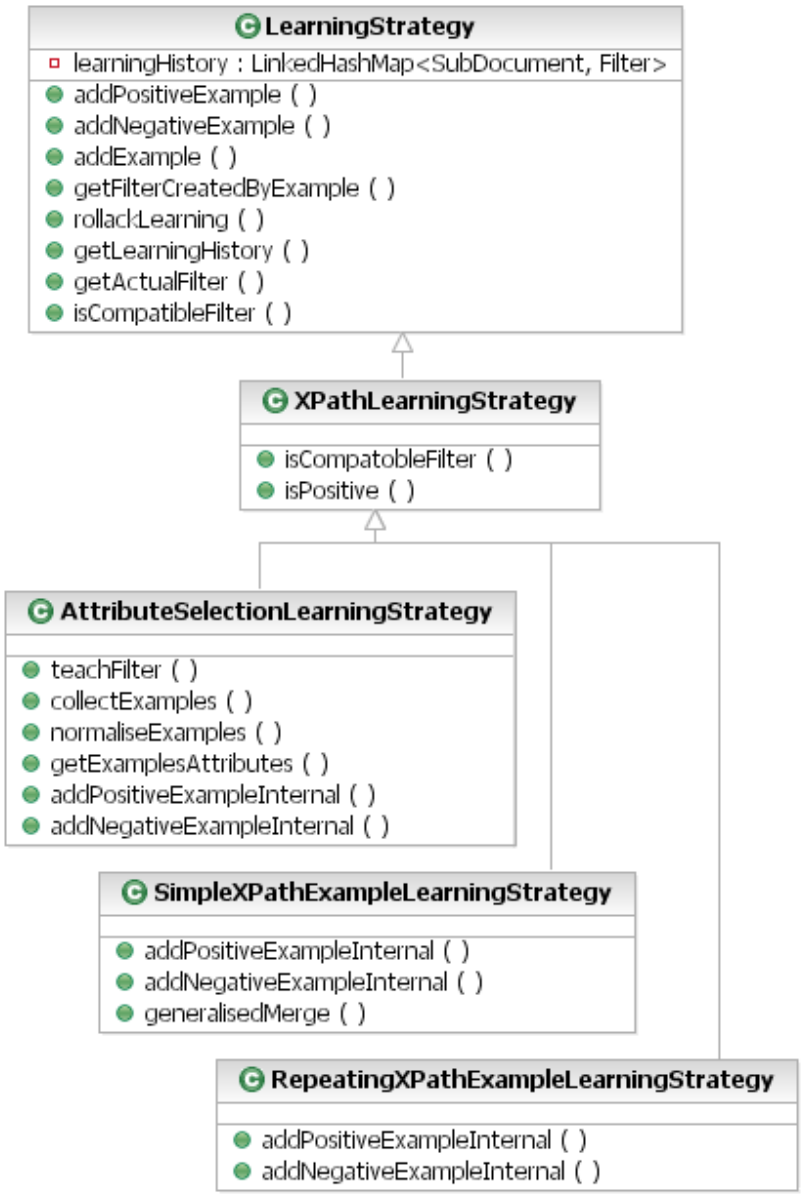
- *AttributeSubDocument* – trieda implementujúca možnosť spolu s určeným elementom prenášať aj atribúty, ktoré tento element popisujú. Ide o jeden konkrétny príklad, ktorý zadá používateľ. Tomuto príkladu sa pridajú atribúty a odovzdá sa stratégii na spracovanie.
- *AttributedElementsSubDocument* – táto trieda predstavuje celú časť dokumentu, kde každý jej element je popísaný atribútmi. Slúži na prenos informácie o atribútoch jednotlivých elementov filtrovaného dokumentu. Potreba jej zavedenia vyplynula z potreby filtra poznať tieto atribúty nie len u zadávaného príkladu ale aj u filtrovaného dokumentu. Keďže atribúty sú priamo z dokumentu prístupné iba v JRex-e.

Oba subdokumenty vo svojej podstate rozširujú XMLSubDocument o doplnenie atribútov (viď Obr. 9-4).



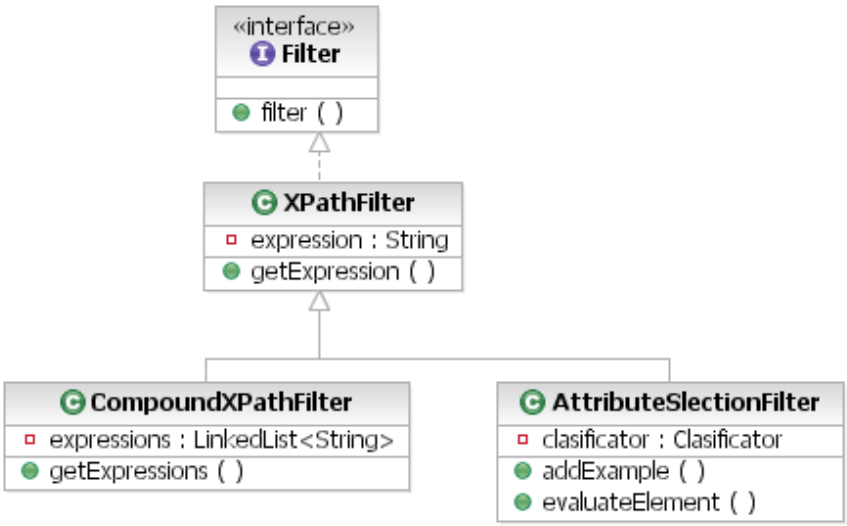
Obr. 9-4: Organizácia subdokumentov

- *AttributeSelectionLearningStrategy* – trieda reprezentujúca v systéme samotnú stratégiu učenia. Má implementovanú tvorbu príslušného potrebného filtra. Taktiež má implementovaný spôsob, akým sa má filter učiť a obsahuje aj normalizáciu príkladov. Táto stratégia je jedna z rodiny stratégií využívajúcich aj vlastnosti XPath výrazov (viď Obr. 9-5).



Obr. 9-5: Stratégie učenia

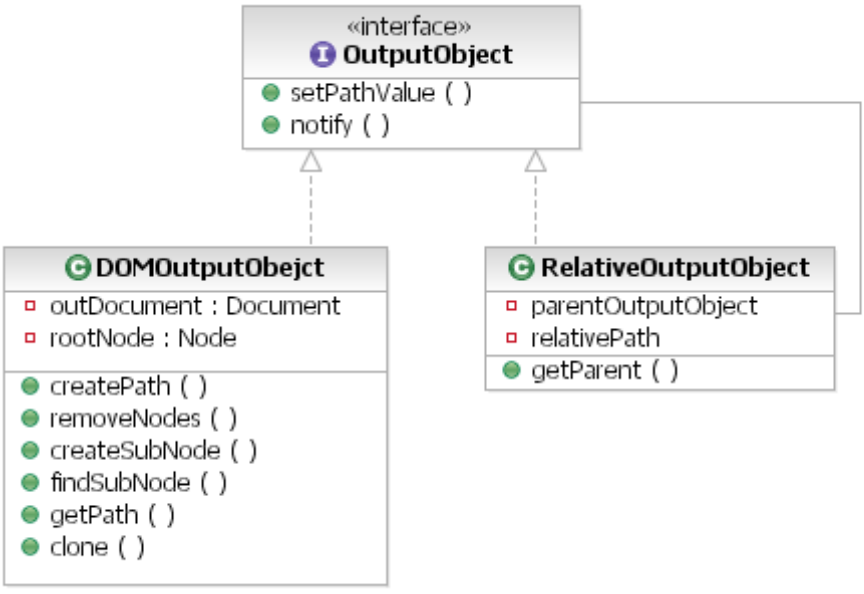
- *AttributeSelectionFilter* – trieda reprezentujúca filter filtrujúci na základe klasifikácie atribútov predkladaných elementov. Implementuje učenie klasifikátora a filtrovanie predloženého dokumentu. Elementy tohto dokumentu sú v rámci svojej štruktúry vyberané pri filtrácii v závislosti od pohybu po DOM-strome. Každý element je predložený na klasifikáciu a na základe jej výsledkov potom je alebo nie je priradený k výstupným subdokumentom, ktoré sú výsledkom filtrácie. Filter v zásade kopíruje v systéme tried pozíciu stratégie, ktorá ho vytvára (viď Obr. 9-6).



Obr. 9-6: CompoundXPathFilter a AttributeSelectionFilter v systéme filtrov

9.3 Implementácia výstupných objektov

Ďalšou časťou aplikačnej vrstvy sú výstupné objekty, ktorých kompletný model sa nachádza na Obr. 9-7.



Obr. 9-7: Kompletný model výstupných objektov

9.4 Implementácia prezentačnej vrstvy

9.4.1 Prezentačný modulu aplikácie

V tejto časti opíšeme dôležité aspekty procesu implementácie modulu aplikácie, ktorá vychádza z návrhu uvedeného v časti 8.5.1.

Dialógy jednotlivých akcií

Po výbere konkrétnej akcie z ľavej postrannej lišty sa spustí nasledovný proces:

1. Registrácia akcie - priradenie `WrappeProgramu` k akcii.
2. Vyvolanie príslušného dialógu prostredníctvom *reflexie* založenej na mene požadovanej akcie.
3. Pridanie akcie do stromu akcií a jej zobrazenie v strome obaľovača.
4. Tvorba kontextu akcie – akcia sa vykoná s kontextom od svojej rodičovskej akcie a pripraví ho tak pre ďalšiu akciu.

Na tvorbu dialógových okien využívame Swing a pre umožnenie ďalšej rozšíriteľnosti systému o nové funkcie a pre sprehľadnenie a zjednodušenie implementácie akcií, je vytvorená šablóna `ActionDialog` združujúca spoločné znaky všetkých dialógov. Jednotlivé akcie z nej preťažujú a rozširujú nasledovné štyri metódy:

- `addDialogWidgets()` – metóda umožňujúca naplnenie dialógu komponentami reprezentujúcimi parametre danej akcie.
- `fillDialogFromAction()` – metóda umožňujúca naplnenie prislúchajúcich komponentov hodnotami z akcie.
- `fillActionFromDialog()` – metóda umožňuje naplnenie parametrov akcie z prislúchajúcich komponentov dialógu.
- `validateDialog()` – metóda umožňuje kontrolovať korektnosť užívateľom zadaných hodnôt parametrov danej akcie.

Celková hierarchia dialógov reprezentovaná UML diagramom tried je na nasledujúcom obrázku.

Obr. 9-8: Hierarchia dialógov pre jednotlivé akcie

Ostatné metódy, spoločné pre všetky dialógy, sú implementované v šablóne `ActionDialog`. Z týchto metód je najzaujímavejšou metóda `createDialogControls()`, ktorá zabezpečuje obsluhu udalostí pri stlačení tlačidiel `OK` a `Cancel`. Obsluha tlačidla `OK` je riešená pomocou reflexie, kedže je implementovaná v šablóne, nevieme, o ktorý dialóg, a tým pádom ani o ktorú akciu, sa jedná.

Akcie vyžadujúce interakciu s používateľom prostredníctvom integrovaného prehliadača sú ďalej oddedené od triedy `PatternActionDialog`, ktorá do ich dialógov umožňuje pridať tlačidlo `JRex` na jeho spustenie.

Postranná lišta akcií

Všetky implementované akcie sú používateľovi prístupné z ľavej postrannej lišty, v ktorej sú kategorizované podľa ich významu. V súčasnosti sú to tieto:

- Navigačné akcie – `LoadPage, FollowLink, Form`
- Extrakčné akcie – `ExtractData, PushData`

Celá lišta je implementovaná v triede `WrapperDesignerActionPalette` obsahujúcej nasledovné metódy:

- `addCategory()` – metóda umožňujúca pridávanie ďalších kategórii a položiek (akcií) na lištu.
- `createCategoryPanel()` – vytvorí panel obsahujúci dané položky.
- `createCategoryButton(), createCategoryItemButton()` – metódy slúžiace na vytvorenie tlačidla reprezentujúceho danú kategóriu, resp. samotnú položku danej kategórie.
- `initialize()` – inicializuje rozloženie jednotlivých položiek, kategórii a ich ikoniek.

Samotná reakcia na stlačenie tlačidla s menom akcie, ktorú chce používateľ pridať, je obsluhovaná v triede `ActionButtonListener`. Z metód tejto triedy je zaujímavá najmä metóda `actionPerformed()`, ktorá obsahuje samotnú obsluhu stlačenia tlačidla s názvom požadovanej akcie a spúšťa celý proces otvárania dialógu tak ako je opísaný v časti 9.4.1.

Strom akcií

Každý obaľovač je zložený z akcií, ktoré spolu tvoria stromovú štruktúru. Samotná štruktúra je implementovaná v triede `ActionTreeImpl`, ktorá implementuje rozhranie `ActionTree` a pri vytváraní sa zobrazuje v hlavnej časti okna obaľovača. Zaujímavé sú najmä nasledujúce metódy:

- `registerAction()` – registruje novú akciu vo WrapperProgram-e a pripraví jej kontext.
- `addChildAction()` – metóda realizuje pridanie potomka aktuálnej akcii.
- `removeActiveAction()` – odstráni aktuálne zvolenú akciu zo stromu aj s jej potomkami.

9.4.2 Integrovaný webový prehliadač

V tejto časti opíšeme dôležité aspekty procesu implementácie integrovaného prehliadača, ktorá vychádza z návrhu uvedeného v časti 8.5.2.

Použité komponenty

Pri implementácii integrovaného prehliadača prezentačnej vrstvy sme použili existujúci projekt JRex. Tento projekt umožňuje vkladanie webového prehliadača do Java aplikácií

a zabezpečuje potrebnú funkcionalitu na modifikáciu prehliadaného dokumentu. Projekt JRex využíva jadro systému Mozilla, ktoré je potrebné pre správny beh aplikácie.

Implementácia integrovaného prehliadača

Implementáciu integrovaného prehliadača môžeme rozdeliť do dvoch častí. Prvou časťou je kontrolér, ktorý vytvára rozhranie medzi dialógmi akcií a prehliadačom, pričom zabezpečuje vzájomné riadenie a komunikáciu. Druhou časťou je už samotný JRex prehliadač.

Implementácia kontroléra

Z pohľadu jadra obaľovača pristupujeme k prehliadaču pomocou rozhrania `LearningController`, ktoré obsahuje metódy na riadenie prídavných modulov pre učenie filtrov. Z tohto rozhrania je odvodená trieda `JRexLearningController`, ktorá priamo komunikuje s JRex prehliadačom. Táto trieda bola rozšírená o ďalšie metódy, ktoré umožňujú vkladanie príkladov do procesu učenia. Všetky metódy triedy `JrexLearningController` vyzerajú nasledovne:

- `teachFilter()` – preťažená metóda z rozhrania `LearningController`, ktorá otvára okno prehliadača a dodáva mu údaje získané z konkrétnej akcie (aktuálny subdokument, nad ktorým sa pracuje).
- `viewPattern()` – preťažená metóda z rozhrania `LearningController`, ktorá zobrazuje naučený vzor nad aktuálnym subdokumentom.
- `addPositiveExample()` – metóda umožňuje vkladanie pozitívneho príkladu, ktorý je vybraný používateľom v prehliadači, do procesu učenia.
- `addNegativeExample()` – metóda je podobná s predchádzajúcou metódou, ale s tým rozdielom, že pracujeme s negatívnym príkladom.
- `getFilteredElementsToMark()` – metóda vráti zoznam elementov, ktoré sa majú zobraziť po aplikovaní filtra na daný subdokument.

Implementácia JRex prehliadača

JRex prehliadač je implementovaný pomocou tried `WrapperJRexFactory` a `JRexBrowser`. Trieda `WrapperJRexFactory` je v podaní klasického návrhového vzoru *Singleton*. Znamená to, že môžeme vytvoriť iba jednu inštanciu tejto triedy. Úlohou tejto triedy je prvotná inicializácia JRex prostredia, ktorá sa vykoná iba na začiatku spustenia obaľovača. Neskôr sa už pracuje so samotným JRex prehliadačom. Metódy triedy `WrapperJRexFactory` sú:

- `WrapperJRexFactory()` – konštruktor triedy, ktorý vykonáva inicializáciu JRex prostredia (vytvorenie okna prehliadača – bez jeho zobrazenia). Metóda konštruktora je privátna, čo znamená, že nie je možné vytvoriť inštanciu triedy mimo nej.
- `getInstance()` – metóda vracia inštanciu vlastnej triedy. Ak nie je vytvorená, tak sa vytvorí, v opačnom prípade sa už nevytvára.

- `getWinManager`() – metóda vráti referenciu na inštanciu manažéra okien, ktorý spravuje okno prehliadača.
- `createJRexBrowser()` – metóda vytvorí a zobrazí JRex prehliadač a načíta daný dokument.
- `createJRexParser()` – metóda vráti referenciu na JRex parser, ktorý bolo nutné implementovať kvôli problémom so štruktúrou elementov na zobrazovanej stránke (viac v časti 9.5)

Trieda `JRexBrowser` reprezentuje okno prehliadača a obsahuje metódy pracujúce nad týmto oknom. Táto trieda je zdedená z triedy `JDialog`, aby sme dosiahli modálnosť prehliadača, a tým zabezpečili konzistenciu údajov. Trieda ďalej implementuje „poslucháčov“ (listeners) tried `WindowListener`, `ProgressListener` a `EventListener`, ktoré reagujú na udalosti vyvolané oknom (vytvorenie, zatvorenie okna), parserom (dokončenie načítania dokumentu) a interakciou používateľa (stlačenie tlačidla myši):

- `windowCreated()` – metóda reaguje na udalosť vytvorenia okna. V tomto prípade sa nastavia štýly okna prehliadača a zobrazí sa webová stránka.
- `windowDisposing()` – metóda signalizuje zatvorenie okna prehliadača, kedy je potrebné uvoľniť „poslucháčov“, reagujúcich na udalosti.
- `onStateChange()` – metóda reaguje na zmenu stavu dokumentu. Existuje veľa možností, ktoré môžu nastať, no my sa zameriavame na udalosť dokončenia načítania webovej stránky.
- `handleEvent()` – metóda reaguje na interakciu používateľa nad elementami webovej stránky a podrobnejšie je popísaná v časti 9.6.2.

Medzi ďalšie funkcie JRex prehliadača môžeme zahrnúť metódy na zobrazenie webovej stránky, nad ktorou sa pracuje, a metódy výberu, označenia a distribúcie príkladov z tejto stránky. Sú to metódy:

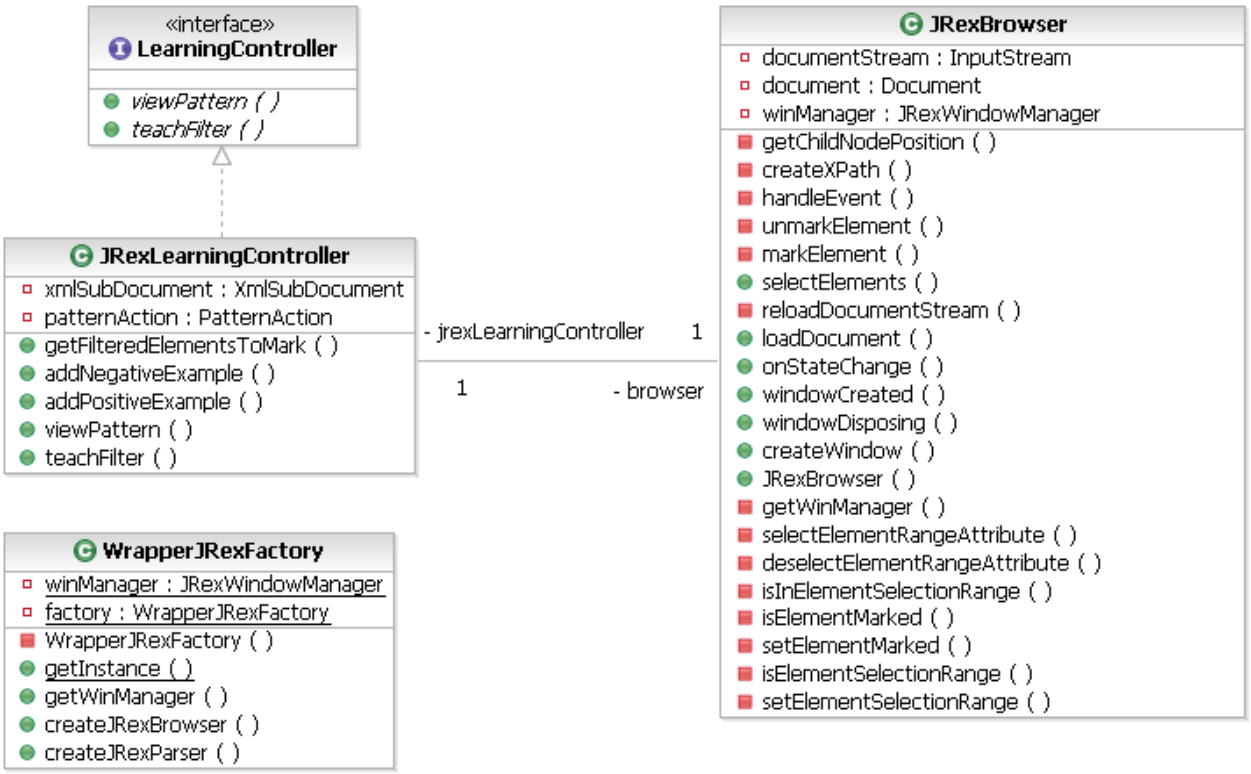
- `JRexBrowser()` – metóda konštruktora, ktorá vytvára okno JRex prehliadača a načíta zadanú webovú stránku vo forme reťazca (zdrojový kód stránky). Metóda taktiež nastaví „poslucháča“ na odchytávanie oknových udalostí.
- `createWindow()` – metóda vytvorí a zobrazí JRex prehliadač, pričom nastaví jeho modálnosť.
- `reloadDocumentStream()` – metóda uloží webovú stránku do vstupného prúdu (input stream), ktorý sa uloží na neskoršie spracovanie.
- `loadDocument()` – metóda načíta webovú stránku zo vstupného prúdu a následne ju zobrazí.
- `createXPath()` – metóda vytvorí XPath výraz pre daný uzol v DOM strome. Od tohto uzla sa v DOM strome vždy postupuje smerom hore (od dieťaťa k rodičovi), a tak sa vyskladá cesta (XPath výraz) až ku koreňovému uzlu, pričom sa vyberajú názvy uzlov (HTML tagy).
- `getChildNodePosition()` – metóda je volaná z predchádzajúcej metódy a slúži na zistenie poradia identického typu uzla na rovnakej úrovni v DOM strome (nájdu sa všetci

rovnakí súrodenci uzla a vyhodnotí sa jeho poradie - index).

Trieda `JRexBrowser` obsahuje aj ďalšie metódy, ktoré sa týkajú označovania elementov v DOM strome. Označovanie elementov je založené na princípe modifikácie ich kaskádových štýlov zobrazovania (CSS). Ak potrebujeme označiť element, nastavíme mu potrebný atribút, ktorý určuje štýl orámovania.

Elementy sa označujú/odznačujú vtedy, ak na nich ukazuje/neukazuje kurzor myši. Táto črta umožňuje jednoduchú orientáciu používateľa pri výbere príkladov. Po výbere sa už tieto elementy označia nastálo a pri ďalšom výbere sa označujú len tie elementy, ktoré sa nachádzajú v ohraničenej oblasti určenej predchádzajúcim príkladom. Oblasť je určená fiktívnym atribútom, ktorý sa nastavuje vybranému elementu z predchádzajúceho výberu.

Výsledný diagram tried pre integrovaný prehliadač je znázornený na Obr. 9-9. Diagram nezobrazuje niektoré nevyužité metódy, ktoré bolo nutné preťažiť, ale nebolo potrebné implementovať.



Obr. 9-9: Diagram tried integrovaného prehliadača

9.5 Súhrn zmien implementácie voči návrhu

9.5.1 Zmeny v jadre

Počas implementácie modulu *jadro* sme prišli na určité problémy kompatibility technológií používaných jadrom systému. Problém spočíva v tom, že integrovaný prehliadač, ktorý je

v projekte použitý, vie zobraziť dokument iba tak, že si ho prečíta z reťazca, čiže nie je možné integrovať ho s už spracovaným DOM stromom. Táto skutočnosť má za následok to, že uvedený reťazec si integrovaný prehliadač načíta do svojej reprezentácie DOM stromu pomocou vlastného nástroja (parser). Tento parser si získaný dokument prispôsobí a upraví. Jednou z týchto úprav je napr. pridanie značky `TBODY` do každej tabuľky v HTML kóde.

Úprava spracovaného dokumentu spôsobuje, že XPath výrazy, ktoré tento integrovaný prehliadač vytvára pri selekcii elementov dokumentu, nepracujú správne nad dokumentami v kontexte akcie. Riešenie tohto problému spočíva v použití rovnakého parsera v integrovanom prehliadači aj v triede `WrapperProgram`. Najjednoduchším riešením bolo použiť časť integrovaného prehliadača na vytváranie DOM stromu. Integrovaný prehliadač však používa DOM strom, ktorý je uložený v tzv. „natívnom“ (C++) kóde, čo znamená, že každá operácia nad týmto stromom je natívna, a keďže v systéme sa s týmto stromom pracuje veľmi často výkonnosť aplikácie sa stala neprípustnou. Operatívne sme tento problém vyriešili tak, že natívny DOM strom transformujeme na java DOM strom, čím síce znížime rýchlosť získavania dokumentu, ale ďalšia práca s dokumentom už dosahuje požadovanú výkonnosť.

9.5.2 Zmeny v prezentačnej vrstve

Použitie metód automatického učenia sa vzorov nám umožnilo vypustiť iteračné akcie implementované v obaľovači verzie 1.0, z ktorého sme vychádzali. Tak isto zmeny v jadre vyústili k potrebným zmenám v inicializácii dialógov príslušných akcií. Po kliknutí na tlačidlo danej akcie sa otváral dialóg akcie a až po jeho vyplnení a potvrdení používateľom sa akcia registrovala a pridávala do stromu akcií. Takýto priebeh bolo treba zmeniť a registráciu akcie vykonať ešte pred inicializáciou jej dialógového okna, pretože viacero akciami požadovaných používateľských parametrov vyžadovalo, aby daná akcia bola v obaľovači registrovaná skôr, ako sa zobrazí jej dialóg a používateľ ich bude môcť zadať. Ide napríklad o zadávanie URI pri akcii `LoadPage`, ktoré vyžaduje `HTTPDocumentResolver`, teda triedu, ktorá sa stará o získanie dokumentu prostredníctvom protokolu HTTP.

V prototype sme na demonštrovanie výberu pozitívnych príkladov pre stratégie učenia využívali `ExampleDialog`. Tento bol implementovaný len pre potreby prototypu a do budúcnosti sa s ním nepočítalo. Bol nahradený integrovaným prehliadačom.

Ten umožňuje zadávanie príkladov pre stratégie učenia interaktívnym spôsobom. Používateľ môže vyberať pozitívne a aj negatívne príklady jednoduchým výberom pomocou myši (pozitívny - kliknutie ľavého tlačidla myši, negatívny – klávesa Ctrl + ľavého tlačidla myši), a to priamo na skutočnej webovej stránke, ktorú si zadá v akcii `LoadPage`, prípadne v inej navigačnej akcii. Táto interakcia vracia rovnakú reprezentáciu príkladov, tak ako tomu bolo v prototype, a to v podobe XPath výrazov. Ďalšou zmenou, ktorú priniesol integrovaný prehliadač, je vyznačenie už vybratých príkladov alebo vyznačenie všetkých elementov, ktoré vyhovujú naučenému vzoru.

9.6 Opis realizácie vybraných častí modulov

9.6.1 Príklad využitia reflexie v implementácii dialógov akcií

Zaujímavou ukážkou je metóda `actionPerformed()` implementovaná v triede `WrapperDesignActionPalette`, ktorá obsluhuje stlačenie tlačidla akcie v postrannej lište akcií a implementuje príslušnú reakciu naň.

Metóda najskôr overuje existenciu aktívneho okna obaľovača a následne sa pomocou reflexie a jej metódy `createObject()`, ktorá vráti novovytvorený objekt triedy so zadaným menom, získa príslušná akcia. Akcia sa zaregistruje a opäť pomocou reflexie sa spúšťa dialógové okno prislúchajúce danej akcii. Až po korektnom vyplnení údajov v dialógu a potvrdení `OK` používateľom sa akcia priraďuje do stromu akcii obaľovača. Na záver sa ešte pripraví kontext pre ďalšiu akciu, a to tak, že sa novopridaná akcia spúšťa s kontextom rodičovskej akcie. Opísaný kód potom vyzerá nasledovne:

```
public void actionPerformed(ActionEvent e) {
      WrapperProgramFrame activeFrame = owner.getActiveFrame();
      if(activeFrame == null) //ak nie je aktivne okno obalovaca, nepokracujeme
            return;

      Reflections reflections = new Reflections();  //reflexia
      Action action = (Action)reflections.createObject(qActionName);
      Class[] actionTypes = {action.getClass()};
      Object[] actions = {action};

      ActionTree tree = activeFrame.getActionTree();
      if (tree.registerAction(action)){  //registracia akcie

            ActionDialog actionDialog =(ActionDialog)reflections.createObject
                              (qActionDialogName, actionTypes, actions);
            action = actionDialog.showDialog();  //dialog pomocou reflexie
            if(actionDialog.isCanceled())
                        return;

            tree = activeFrame.getActionTree(); //pridanie akcie do stromu
            tree.addChildAction(action);

            Context oldContext = action.getContext(); //priprava kontextu
            Context newContext = new Context();
            newContext = action.executeWithContext(oldContext);
            action.getProgram().setLearnedContext(action, newContext);
      }
}
```

9.6.2 Zachytenie udalosti nad DOM elementom v JRex prehliadači

Úlohou metódy `handleEvent` triedy `JRexBrowser` je reagovanie na podnety, ktoré vznikli pri interakcii používateľa s integrovaným prehliadačom. Táto metóda je volaná asynchrónne a to iba v prípadoch, ak používateľ klikne na ktorýkoľvek element, alebo ak nastane situácia, že sa

kurzor myši posunul mimo (*mouseout*) alebo nad (*mouseover*) daný element.

V prípade, že nastala nejaká udalosť, skontroluje sa, či sa jedná o požadovanú udalosť a určí sa typ danej udalosti. Podľa typu vykonáme príslušné metódy:

- `mouseover()` – označenie elementu, na ktorý ukazuje kurzor myši
- `mouseout()` – odznačenie elementu, na ktorý ukazoval kurzor myši
- `click()` – zistenie, či je element označený, a ak je, vytvoríme k nemu prislúchajúci XPath výraz a predáme ho kontroléru

Opísané správanie zobrazuje nesledovný zdrojový kód:

```
public void handleEvent(Event event) {
     JRexMouseEventImpl mouseEvent = (JRexMouseEventImpl) event;
     EventTarget target = mouseEvent.getTarget();
     target.getClass();

     if (target instanceof JRexEventTargetImpl) {

          JRexEventTargetImpl localTarget = (JRexEventTargetImpl) target;
          Element element = (Element)localTarget.getNode();

          if (mouseEvent.getType().compareToIgnoreCase("mouseover") == 0) {
               markElement(element);
          }
          else if (mouseEvent.getType().compareToIgnoreCase("mouseout") == 0) {
               unmarkElement(element);
          }
          else if (mouseEvent.getType().compareToIgnoreCase("click") == 0) {
               if (isElementMarked(element)) {
                    mouseEvent.preventDefault();
                    mouseEvent.stopPropagation();
                    try {
                         if (mouseEvent.getCtrlKey())
                              controller.addNegativeExample(createXPath(element));
                         else
                              controller.addPositiveExample(createXPath(element));
                    }
                    catch (FilterException e) {
                         e.printStackTrace();
                         JOptionPane.showMessageDialog(this, "Could not add
                         example " + e.getMessage(), "Example add",
                         JOptionPane.ERROR_MESSAGE);
                    }
               }
          }
     }
}
```

10 Testovanie produktu

V tejto časti dokumentu sa nachádza opis vykonaných testov. Produkt bol testovaný na úrovni komponentov už počas vývoja prostredníctvom unit testov. Po dokončení komponentov bolo vykonané integračné a systémové testovanie produktu. Testy sme rozdelili do nasledovných častí:

- testovanie časti JRex – testovanie vstavaného prehliadača (zobrazovanie príkladov, pridávanie príkladov, načítanie príkladov a iné),
- testovanie GUI – testovanie všetkých častí používateľského rozhrania,
- testovanie jadra – testovanie častí jadra produktu (testovanie akcií, výstupných objektov a iné).

Produkt sme rozdelili na časti, ktoré budeme testovať a časti, na ktoré sa nebudeme sústrediť pri testovaní. Medzi testované črty produktu patria vyššie spomenuté časti produktu (JRex, GUI a jadro), v ktorých sme sa sústredili, či pre zadané vstupy (vybraných viacero rôznych vstupov) sa výstupy testovanej časti zhodujú so špecifikovanými výstupmi. Pri testovaní sme sa nesústredili na metriky škálovateľnosti alebo efektívnosti získania výsledkov, prvoradým cieľom bola správnosť výsledkov. Testované charakteristiky jednotlivých častí (JRex, GUI a jadro) sú opísané v príslušných kapitolách o testovaní častí. Medzi charakteristiky produktu, ktoré sme v nižšie opísaných testov nezahrnuli, patrí aj testovanie s novou stratégiou – *AttributeSelection*, keďže pre túto stratégiu ešte neboli vykonané všetky testy na úrovni komponentov.

Pri testovaní sme vychádzali z existujúceho štandardu IEEE 829/1998, ktorý definuje špecifikáciu testov na základe návrhu testov, špecifikácií testovacích prípadov a špecifikácií testovacích procedúr. Jednotlivé testy boli vykonávané manuálne a výsledky testov sú zhrnuté v časti súhrnnej správe o teste.

10.1 Testovanie JRex časti

Časť JRex predstavuje vstavaný prehliadač, ktorého úlohou je umožňovať používateľovi zadávať príklady a následne zobrazovať filtrovanú časť dokumentu. Z tohto vyplývajú aj základné charakteristiky vykonaných testov, kde sme sa sústredili najmä na správnosť interakcie prehliadača s používateľom pri zadávaní a zobrazovaní častí dokumentov.

10.1.1 Návrh testov

Tab. 10-1: Návrh testov pre testovanie JRex časti

ID	DJRex1
Testované funkcie	Vyznačovanie príkladov v dokumente, umožnenie výberu príkladov iba z dovolených častí (aktuálny subdokument), práca s rôznymi dokumentmi, testovanie linky v dokumente
Opis postupu	Testovanie prebiehalo ručne so snahou pokryť viaceré typy vstupov. Kontrola s požadovanými výstupmi prebiehala vizuálne
Testovacie prípady	TCJrex1, TCJrex2, TCJrex3, TCJrex4

10.1.2 Testovacie prípady

V tabuľkách 10-2 až 10-5 je uvedená špecifikácia testovacích prípadov. Keďže prehliadač pracuje s webovými stránkami, požiadavky všetkých prípadov predstavuje dostupné internetové spojenie.

Tab. 10-2: Testovací prípad TCJRex1

ID	TCJrex1
Testovaná položka	Vyznačovanie príkladov - vyber príkladov v JRex-e (pozitívne, negatívne). Označený príklad by sa mal v prehliadači zvýrazniť a následne pridať medzi príklady vo vzore
Vstupy	1. Načítaný dokument v prehliadači JRex
	2. Vyznačenie časti načítaného dokumentu
Výstupy	1. Vyznačené požadované časti dokumentu v prehliadači
	2. Príklady pridané vo vzore (overenie v dialógu ViewPattern)
Požiadavky na prostredie	-

Tab. 10-3: Testovací prípad TCJRex2

ID	TCJrex2
Testovaná položka	Označovanie povolenej časti dokumentu – príklady možno vyberať iba z časti dokumentu (subdokument) získanej od rodičovskej akcie (postupná dekompozícia dokumentu)
Vstupy	1. Načítaný dokument v prehliadači JRex
	2. Vyznačenie časti načítaného dokumentu
	3. Pridanie potomkov akcie a vyznačenie príkladov v pridaných akciách
Výstupy	1. Vyznačenie časti dokumentu v prehliadači pre prvú akciu
	2. Ohraničenie dokumentu pre potomkov akcie (krok 1)
Požiadavky na prostredie	-

Tab. 10-4: Testovací prípad TCJRex3

ID	TCJrex3
Testovaná položka	Načítanie rôznych typov dokumentov – testovanie správnosti práce s rôznymi stránkami (HTML)
Vstupy	1. Vytvorená akcie LoadPage so zadanou adresou dokumentu
Výstupy	1. Načítaný dokument a zobrazený v JRex okne
	2. Práca s dokumentom v súlade s požiadavkami (napr. vyznačovanie príkladov)
Požiadavky na prostredie	-

Tab. 10-5: Testovací prípad TCJRex4

ID	TCJrex4
Testovaná položka	Práca s linkami v JRex – linky v dokumente (odkazy na iné dokumenty) by malo byť možné pridávať medzi príklady a akcia FollowLink by mala načítať odkazovaný dokument
Vstupy	1a. Načítaný dokument s linkou v akcii
	1b. Načítaný dokument s linkou v akcii FollowLink, ku ktorej sú pridaní potomkovia
Výstupy	1a. Pridanie príkladu s linkou (TCJrex1)
	1b. Vyznačený príklad v akcii FollowLink, v ďalších akciách načítaný

	odkazovaný dokument
Požiadavky na prostredie	-

10.1.3 Testovacie procedúry

Testovacia procedúra 1

Identifikátor: PJrex1

Účel: Testovanie produktu pre testovacie scenáre TCJrex1, TCJrex2, TCJrex3, TCJrex4.

Kroky procedúry:

1. **Zahájenie**:

   Spustenie programu obaľovača a vytvorenie prázdneho obaľovača, pridanie akcie LoadPage

2. **Procedúra**:

   2.1. Nastavenie adresy v akcii LoadPage – www.profesia.sk

   2.2. Pridanie akcie FollowLink (akcia 1)

   2.3. Pridanie pozitívneho príkladu do FollowLink – linka “hľadáte prácu”

   2.4. Pridanie ExtractData (akcia 2) – potomok akcie 1

   2.5. Pridanie pozitívneho príkladu do ExtractData, ktorý dekomponuje celý dokument na subdokument obsahujúci viaceré elementy – stredná časť dokumentu s nadpisom “Hľadáte prácu”

   2.6. Pridanie akcie ExtractData (akcia 3) – potomok akcie 2

   2.7. Pridanie pozitívnych a negatívnych príkladov – pozitívne príklady – pracovné inzeráty práce, negatívne príklady – text ponuky

3. **Meranie výsledkov:**

   Výsledky sú merané vizuálne na základe očakávaného stavu v okne prehliadača Jrex. Správne pridanie príkladov je kontrolované v dialógu ViewPattern.

4. **Ukončenie:**

   Ukončenie programu obaľovača

Testovacia procedúra 2

Identifikátor: PJrex2

Účel: Testovanie produktu pre testovacie scenáre TCJrex1, TCJrex3.

Kroky procedúry:

1. **Zahájenie**:

   Spustenie programu obaľovača a vytvorenie prázdneho obaľovača, pridanie akcie LoadPage

2. **Procedúra**:

   2.1. Nastavenie adresy v akcii LoadPage – www.pravda.sk

      2.1.1. Zmena adresy v akcii LoadPage – www.sme.sk, www.post.sk, www.praca.sk

   2.2. Pridanie ExtractData

   2.3. Pridávanie pozitívnych a negatívnych príkladov

3. **Meranie výsledkov:**

   Výsledky sú merané vizuálne na základe očakávaného stavu v okne prehliadača Jrex. Správne pridanie príkladov je kontrolované v dialógu ViewPattern.

4. **Ukončenie:**

   Ukončenie programu obaľovača

10.1.4 Súhrnná správa o teste

ID	RJRex1
Požadované výsledky	TCJRex1 – Príklady sú vyznačené v prehliadači (na základe zvoleného typu - pozitívny, negatívny). Príklady sú pridané do vzoru.
	TCJRex2 – Vyznačenie časti, z ktorej možno vyberať príklady, zakázanie výberu z inej časti.
	TCJRex3 – Správna práca s rôznymi dokumentmi – na základe vytvorených testov.
	TCJRex4 – Odkaz (link) možno zvoliť ako príklad. Ak je daný príklad pridaný do akcie FollowLink, v ďalších akciách je načítaný nový dokument
Dosiahnuté výsledky	TCJRex1 – Problém zobrazovania negatívnych príkladov – nie všetky časti sa dali zaznačiť medzi negatívne, vyznačené sa nepridali medzi príklady. Problém s vyznačovaním rôznych častí na stránke (www.profesia.sk). Pri výbere viacerých príkladov sa príklady v JRexe prestanú vyznačovať.
	TCJRex2 – Výsledky sú zhodné s očakávanými.
	TCJRex3 – Jediný nedostatok bol zistený pri stránke www.praca.sk, ktorú sa nepodarilo načítať, čo môže spôsobovať, že stránka nemusí byť well-formed.
	TCJRex4 – Výsledky sú zhodné s očakávanými.
Ohodnotenie a návrhy na zlepšenie	Zistené nedostatky môžu súvisieť s chybami v projekte JRex, ktorý sme použilil, preto treba skúmať, ktoré z nedostatkov sú spôsobené chybami v našich častiach a odstrániť nájdené chyby. Zistenie dôvodov nefunkčnosti načítania stránky www.praca.sk

10.2 Testovanie GUI časti

Grafické používateľské rozhranie prezentuje systém a jeho výsledky používateľovi a odovzdáva pokyny od používateľa na spracovanie systému. Vzhľadom na túto úlohy bolo testovanie zamerané na správnu funkcionalitu jednotlivých ovládacích prvkov.

10.2.1 Návrh testov

Tab. 10-6: Návrh testov pre testovanie GUI časti

ID	DGui1
Testované funkcie	Validácia polí dialógov, kontrola správnej funkčnosti položiek menu, následnosti akcií, dialógov akcií a ich záložiek
Opis postupu	Testovanie bolo vykonané ručne. Kontrola funkcionality GUI prebiehala počas testu na základe správania sa systému v porovnaní s očakávanou funkcionalitou.
Testovacie prípady	TCGui1, TCGui2, TCGui3, TCGui4, TCGui5, TCGui6

10.2.2 Testovacie prípady

V tabuľkách 10-7 až 10-12 je uvedená špecifikácia testovacích prípadov. Používateľské rozhranie poskytuje vo svojej funkcionalite aj spúšťanie internetového prehliadača a zobrazovanie z neho získaných údajov, pre kompletné vykonanie testov je potrebné internetové pripojenie.

Tab. 10-7: Testovací prípad TCGui1

ID	TCGui1
Testovaná položka	Práca s programom obaľovača ako celkom – vytvorenie programu, uloženie do súboru, zatvorenie programu, načítanie zo súboru. Testovanie – klávesových skratiek, tlačidiel a položky menu – File
Vstupy	1. Wrapper Designer neobsahujúci žiaden otvorený program obaľovača. 2. Vytvorenie programu obaľovača a jeho uloženie, načítanie a zatvorenie
Výstupy	1. Vytvorený a uložený program obaľovača
Požiadavky na prostredie	-

Tab. 10-8: Testovací prípad TCGui2

ID	TCGui2
Testovaná položka	Zmeny zobrazenia okna programu obaľovača vo Wrapper Designeri. Otestovanie tlačidiel a menu.
Vstupy	1. Aspoň 2 programy obaľovačov otvorené vo Wrapper Designeri 2. Zmeny zobrazenia okna programu obaľovača
Výstupy	1. Rôzne formy zobrazenia okna programu obaľovača
Požiadavky na prostredie	-

Tab. 10-9: Testovací prípad TCGui3

ID	TCGui3
Testovaná položka	Testovanie dialógov jednotlivých akcií a editácia akcií
Vstupy	1. Vytvorený program obaľovača
Výstupy	1. Akcie so správne vyplnenými parametrami zaradené do stromu akcií
	2. Akcie s editáciou zmenenými parametrami
Požiadavky na prostredie	-

Tab. 10-10: Testovací prípad TCGui4

ID	TCGui4
Testovaná položka	Testovanie položky Nástroje (Tools)
Vstupy	1. Vytvorený spustiteľný program obaľovača
Výstupy	1. Predvedenie činnosti nástrojov
Požiadavky na prostredie	-

Tab. 10-11: Testovací prípad TCGui5

ID	TCGui5
Testovaná položka	Testovanie manipulácie so stromom akcií
Vstupy	1. Vytvorený program obaľovača s akciami
Výstupy	1. Modifikovaný strom akcii
Požiadavky na prostredie	-

Tab. 10-12: Testovací prípad TCGui6

ID	TCGui6
Testovaná položka	Testovanie poradia akcií
Vstupy	1. Spustený Wrapper Designer (bez ďalších zvláštnych požiadaviek)
Výstupy	1. Strom logicky správne zoradených akcií
Požiadavky na prostredie	-

10.2.3 Testovacie procedúry

Testovacia procedúra 1

Identifikátor: PGui1

Účel: Testovanie produktu pre testovacie scenáre TCGui1.

Kroky procedúry:

1. **Zahájenie**:
   Spustenie programu Wrapper Designer-u.

2. **Procedúra**:
   2.1. Vytvorenie nového programu - Ctrl+N

2.1.1. Pridanie akcie LoadPage:
2.1.2. Kliknutím na tlačidlo Navigation sa otvorí zoznam navigačných akcií
2.1.3. Výber LoadPage
2.1.4. Vyplnenie položiek v záložke General - Name na "Načítanie stránky" a Uri:"http://www.praca.sk/Applicants/searchJobs.aspx?page=1&c=1"
2.2. Potvrdenie OK
2.3. Uloženie pomocou Ctrl+S
2.4. V dialógu - nastavenie cesty k ukladaným súborom a uloženie pod menom gui_test1. Potvrdenie pomocou OK
2.5. Uzatvorenie programu: Ctrl+W
2.6. Otvorenie programu: Ctrl+O
2.7. Zvolenie pripraveného súboru gui_test1
2.8. Zatvorenie WrapperDesigneru: Ctrl+X

3. **Meranie výsledkov:**
Výsledok je vyhodnotení počas vykonávania procedúry: Program obaľovača sa musí uložiť do súboru a rovnaký sa z neho aj načítať – porovnané vizuálne.

4. **Ukončenie:** (ukončenie v rámci procedúry)

Testovacia procedúra 2

Identifikátor: PGui2

Účel: Testovanie produktu pre testovacie scenáre TCGui1.

Kroky procedúry:

1. **Zahájenie**:
Spustenie programu Wrapper Designer-u.

2. **Procedúra**:
2.1. Vytvorenie nového programu: File - New
2.2. Pridanie akcie LoadPage (Name:"Načítaj brigády" Uri:"http://www.brigady.sk/vysledky-vyhladavania.aspx?IDRegionu=2&TypPracovnejPonuky=1"). Potvrdenie OK.
2.3. Uloženie pomocou: File - Save (meno súboru: gui_brigády)
2.4. Uzatvorenie programu: File - Close
2.5. Otvorenie programu: File - Open... (meno súboru: gui_brigády)
2.6. Zatvorenie WrapperDesigneru: File - Exit

3. **Meranie výsledkov:**
Výsledok je vyhodnotení počas vykonávania procedúry: Program obaľovača sa musí uložiť do súboru a rovnaký sa z neho aj načítať – porovnané vizuálne.

4. **Ukončenie:** (ukončenie v rámci procedúry)

Testovacia procedúra 3

Identifikátor: PGui3

Účel: Testovanie produktu pre testovacie scenáre TCGui1, TCGui2, TCGui3.

Kroky procedúry:

1. **Zahájenie**:
   Spustenie programu Wrapper Designer-u.

2. **Procedúra**:
   2.1. Otvorenie súboru: tlačítko otvorenia súboru (meno súboru gui_test1)
   2.2. Kliknutie pravým tlačítkom na LoadPage v strome akcií - Edit.
   2.3. V dialógu - zmena Name na "Načítanie stránky z práce". OK
   2.4. File - Save as... (meno súboru: gui_práca)
   2.5. Minimalizácia okna programu obaľovača tlačítkom pre minimalizáciu okna programu.
   2.6. Vytvorenie nového programu tlačidlom na vytvorenie nového programu.
   2.7. Pridanie LoadPage (Name: "Načítanie z profesie" Uri: "http://www.profesia.sk/praca/user_details.php3")
   2.8. Pridanie ExtractData akcie:
      2.8.1. Kliknutie v menu na Extraction a zo zobrazeného menu na ExtractData
      2.8.2. Záložka General - Name: "Názvy ponúk"
      2.8.3. Záložka Source - Learning Strategy: "SimpleXPath Strategy", stlačenie Jrex... tlačidla - výber 2 názvov brigád, zatvorenie otvoreného okna
      2.8.4. Záložka Destination - outputObjectPath: "ponuky", outputObjectID:"názvy"
      2.8.5. OK
   2.9. Pridanie PushData akcie:
      2.9.1. Kliknutie v Extraction menu na PushData
      2.9.2. Záložka General - Name: "Uloženie ponúk"
      2.9.3. Záložka Destination - outputObjectPath: "pracovné_ponuky", outputObjectID: "názvy"
      2.9.4. OK
   2.10. Kliknutie na zatvorenie programu. V dialógu - zrušiť.
   2.11. Pokus o zatvorenie cez Ctrl+W. V dialógu – zrušiť.
   2.12. Stlačenie tlačidla pre uloženie programu (meno súboru: gui_profesia).
   2.13. Stlačenie tlačidla pre zatvorenie programu.
   2.14. Otvorenie programu: Ctrl+O (meno súboru: gui_profesia).

2.15. Window – Minimize

2.16. Window - Maximize

2.17. Window - Minimize

2.18. Kliknutie na tlačidlo gui_práca

2.19. Window - Cascade

2.20. Kliknutie na tlačidlo gui_profesia

2.21. Window - Cascade

2.22. Kliknutie do časti okna patriacej gui_práca.

2.23. Kliknutie na LoadPage v programe.

2.24. Pridanie ExtractData (Name:"Výber ponúk práce", LearningStrategy:"Simple XPath Strategy", JRex... - výber 2 názvov a zatvorenie okna, outputObjectPath:"oopPráce", outputObjectID:"ooidPráce"). OK

2.25. File - Close All. V dialógu "Yes"

3. **Meranie výsledkov:**
Správnosť činnosti je ohodnocovaná priebežne na základe pozorovania činnosti: správania sa okien, uloženie, načítanie programu, pripájanie akcií do stromu akcii (programu).

4. **Ukončenie:** (ukončenie v rámci procedúry)

Testovacia procedúra 4

Identifikátor: PGui4

Účel: Testovanie produktu pre testovacie scenáre TCGui3

Kroky procedúry:

1. **Zahájenie**:
Vytvorenie programu: Ctrl+N

2. **Procedúra**:
2.1. Pridavanie LoadPage - stornovanie pomocou Cancel

2.2. Pridanie LoadPage s prázdnou položkou Name

2.3. Pridanie LoadPage s prázdnou položkou Uri

2.4. Pridanie LoadPage s oboma prázdymi položkami

2.5. Pridanie LoadPage (Name:"Stránka1", Uri:"www.yahoo.com")

2.6. Pridavanie ExtractData - stornovanie pomocou Cancel

2.7. Pridanie ExtractData s prázdnou položkou Name (na testovanie používame hodnoty -

Name:"Extrakcia", outputObjectPath:"oop", outputObjectID:"ooid", Attribute Selection Strategy

2.8. Pridanie ExtractData s prázdnou položkou outputObjectPath

2.9. Pridanie ExtractData s prázdnou položkou outputObjectID

2.10. Pridanie ExtractData s prázdnymi položkami záložky Destination

2.11. Pridanie ExtractData so všetkými položkami

2.12. Zmena Uri editáciou akcie LoadPage (cez menu Action - Edit)

2.13. Opakovanie bodov 8,9,10,11,12 pre stratégiu Repeating XPath Strategy. Zmena stratégie editáciou akcie ExtractData vo programe obaľovača cez Action - Edit. A zadanie príkladov.Kontrola naučeného vzoru cez Action - ViewPattern

2.14. Pridávanie príkladov cez Action - Add example a cez kontektové menu.

2.15. Zmena Uri editáciou akcie LoadPage (cez kontextové menu – Edit)

2.16. Opakovanie bodov 8,9,10 pre stratégiu Simple XPath Strategy. Zmena stratégie editáciou akcie ExtractData v programe obaľovača, pravým kliknutím na ňu a Edit z kontextového menu. A zadanie príkladov. Kontrola naučeného vzoru cez View Pattern z kontextového menu.

2.17. Pridávanie PushData (Name:"Ulož", outputObjectPath:"push_oop", outputObjectID:"X") postupne bez jednotlivých položiek a ich kombinácií.

2.18. Vymazanie PushData cez Action Remove

2.19. Pridávanie PushData (Name:"Ulož", outputObjectPath:"push_oop", outputObjectID:"ooid") postupne bez jednotlivých položiek a ich kombinácií.

2.20. Vymazanie PushData cez kontextové menu - Remove

3. **Meranie výsledkov:**
   Ohodnocovanie správnosti možnosti pridania akcie na základe poskytnutých parametrov, kontrola pridávania príkladov.

4. **Ukončenie:** Zatvorenie programu obaľovača: Ctrl+W (Neukladať).

Testovacia procedúra 5

Identifikátor: PGui5

Účel: Testovanie produktu pre testovacie scenáre TCGui3, TCGui5

Kroky procedúry:

1. **Zahájenie**:
   Spustenie programu Wrapper Designer-u.

2. **Procedúra**:
   2.1. Načítanie programu: Ctrl+O (súbor gui_profesia)
   2.2. Rozvinutie stromu akcii (skúška zvinutia jednotlivých častí) - pomocou ovládača vľavo od názvu akcie
   2.3. Pridanie FollowLink za LoadPage
      2.3.1. Označenie LoadPage - kliknutím ľavým tlačidlom na akciu v strome akcií
      2.3.2. Pridávanie FollowLink (Name:"Prechod", Uri:"http://", Interation limit:"0","1","-1","", Infinity) - pre rôzne kombinácie hodnôt a ich absenciu
   2.4. Pridanie Form za FollowLink (bez hodnoty parametra Name, s hodnotou Name:"Formulár")
   2.5. Označenie akcie ExtractData - kopírovanie cez Edit - Copy
   2.6. Označenie akcie LoadPage - prilepenie cez Edit - Paste
   2.7. Označenie akcie PushData - vystrihnutie cez Edit - Cut
   2.8. Označenie akcie ExtractData - prilepenie cez Edit - Paste
   2.9. Body 5 až 8 zopakované s klávesomvými skratkami (Ctrl+C pre Copy, Ctrl+V pre Paste, Ctrl+X pre Cut)

3. **Meranie výsledkov:**
   Ohodnocovanie správnosti možnosti pridania akcie na základe poskytnutých parametrov, kontrola správnosti modifikácie stromu akcií.

4. **Ukončenie:** Zatvorenie programu obaľovača: Ctrl+W (Neukladať).

Testovacia procedúra 6

Identifikátor: PGui6

Účel: Testovanie produktu pre testovacie scenáre TCGui4

Kroky procedúry:

1. **Zahájenie**:
   Dostupný program obaľovača s akciami

2. **Procedúra**:
   2.1. Načítanie programu obaľovača: Ctrl+O (súbor gui_profesia).
   2.2. Zobrazenie výstupnej konzoly cez tlačidlo a cez Tools - Console
   2.3. Spustenie interpretera cez tlačidlo a cez Tools - Interpret
   2.4. Nastavenie volieb: Tools - Options

3. **Meranie výsledkov:**
   Vizuálne zhodnotenie zobrazenia nástrojov a možností.

4. **Ukončenie:** Zatvorenie programu obaľovača: Ctrl+W

Testovacia procedúra 7

Identifikátor: PGui7

Účel: Testovanie produktu pre testovacie scenáre TCGui5, TCGui6

Kroky procedúry:

1. **Zahájenie**:
   Vytvorenie programu

2. **Procedúra**:
   2.1. Vloženie LoadPage akcie
   2.2. Vloženie 3 ExtractData akcií ako jej potomkov
   2.3. Priradenie jednej PushData ako potomka ku každej ExtractData
   2.4. Označenie 2 ExtractData, zadanie Remove (cez kontextové menu, cez Action - Remove)
   2.5. Označenie 1 ExtractData, zadanie Remove recursive (cez kontextové menu, cez Action - Remove recursive)
   2.6. Pridávanie akcii jednotlivých druhov v kombináciach, kde sa každé dve vystriedajú vo vzťahu rodič potomok.

3. **Meranie výsledkov:**
   Vizuálne zhodnotenie správnosti odstránenia akcií a logickej správnosti následnosti akcií.

4. **Ukončenie:** Zatvorenie programu obaľovača: Ctrl+W (Neukladať).

10.2.4 Súhrnná správa o teste

ID	RGui1
Požadované výsledky	TCGui1 – Program obaľovača sa vytvoril, uložil, zatvoril a správne načítal zo súboru
	TCGui2 – Okná sa správajú podľa názvu vykonávanej funkcie (napr. maximalizácia na celé okno vyhradené pre programy obaľovačov v designeri)
	TCGui3 – Akcie, ktoré sa pridali iba ak mali požadované parametre. Parametre sa podarilo editáciou zmeniť.
	TCGui4 – Zobrazenie výstupnej štandardnej a chybovej konzoly, spustenie interpretera programu so zobrazením činnosti.

	TCGui5 – Strom akcií zmenený na základe kopírovania, vkladanie, vystrihovania a mazania jeho podstromov.
	TCGui6 – Akcie sa dajú radiť do stromovej štruktúry na základe svojich logických nadväzností.
Dosiahnuté výsledky	TCGui1 – Dosiahli sa požadované výsledky
	TCGui2 – Minimalizované okno sa nemaximalizovalo pomocou položky menu Maximize, ale iba stlačením tlačidla s jeho menom.
	TCGui3 – Formulárová akcia nemá hotové GUI, pri zadávaný cesty v PushData sa nekontroluje prítomnosť medzier. Pri stornovaní akcie LoadPage sa nedá už nič pridať. Kontextové Add Example nefunguje, Attribute Selection Strategy nemá náhľad naučených vzorov.
	TCGui4 – Interpreter sa nezobrazil
	TCGui5 – Kopírovanie, vkladanie a vystrihovania akcií nič nerobí
	TCGui6 – Akcie sa dajú zoraďovať ľubovoľne
Ohodnotenie a návrhy na zlepšenie	Zistené nedostatky predstavujú obmedzenia aj v oblasti správnosti používania, čiže sú možné aj nesprávne činnosti, ďalej je obmedzené zobrazovanie informácií o stave a v oblasti funkcionality chýba prístup k niektorým funkciám – akcia Form. Je potrebné chýbajúce časti doplniť a odstrániť aspoň zásadné nedostatky.

10.3 Testovanie jadra

Jadro systému predstavuje vnútornú časť, v ktorej sa nachádza celá funkcionalita vytvoreného systému. Ide o funkcie, ktoré zabezpečujú získavanie a zapisovanie údajov z webových stránok. Pri testovaní tejto časti systému sa zameriame hlavne na zapisovanie získaných údajov, otestujeme všetky typy akcií v programe, správne uloženie a načítanie programu obaľovača, ako aj stratégiu učenia na základe atribútov.

10.3.1 Návrh testov

Tab. 10-13: Návrh testov pre testovanie jadra

ID	DCore1
Testované funkcie	Napĺňanie a zapisovanie výstupných objektov, kontrola správnej funkčnosti všetkých typov akcií, ukladanie a načítanie programu obaľovača
Opis postupu	Testovanie sa vykonávalo vizuálne počas behu obaľovača. Získané výsledky sa porovnali s očakávanými.
Testovacie prípady	TCCore1, TCCore2, TCCore3, TCCore4, TCCore5, TCCore6

10.3.2 Testovacie prípady

V tabuľkách 10-14 až 10-19 je uvedená špecifikácia testovacích prípadov. Keďže testujeme správnosť funkcionality jadra systému, musíme zabezpečiť stále pripojenie do Internetu pre otestovanie funkcií na získavanie údajov. Ďalšou požiadavkou je potreba testovacích nástrojov (Sesame, MySQL) pre zapisovanie týchto údajov.

Tab. 10-14: Testovací prípad TCCore1

ID	TCCore1
Testovaná položka	Zápis získaných údajov do XML súboru. Po vytvorení a spustení programu obaľovača sa naplní výstupný XML súbor.
Vstupy	1. Vytvorený program v obaľovači 2. Spustenie programu
Výstupy	1. Vytvorený a naplnený výstupný XML súbor
Požiadavky na prostredie	

Tab. 10-15: Testovací prípad TCCore2

ID	TCCore2
Testovaná položka	Zápis získaných údajov do ontológie. Po vytvorení a spustení programu obaľovača sa naplní ontológia.
Vstupy	1. Vytvorený program v obaľovači 2. Spustenie programu
Výstupy	1. Naplnená ontológia
Požiadavky na prostredie	Použitie ontológie Sesame

Tab. 10-16: Testovací prípad TCCore3

ID	TCCore3
Testovaná položka	Zápis získaných údajov do databázy. Po vytvorení a spustení programu obaľovača sa naplní databáza.
Vstupy	1. Vytvorený program v obaľovači 2. Spustenie programu
Výstupy	1. Naplnená databáza
Požiadavky na prostredie	Použitie databázy MySQL

Tab. 10-17: Testovací prípad TCCore4

ID	TCCore4
Testovaná položka	Testovanie úplného programu obaľovača – testovanie všetkých typov akcií a ich vzájomnej spolupráce.
Vstupy	1. Vytvorený prázdny program obaľovača
Výstupy	1. Spustiteľný úplný program obaľovača
	2. Výstupné údaje zodpovedajúce programu
Požiadavky na prostredie	

Tab. 10-18: Testovací prípad TCCore5

ID	TCCore5
Testovaná položka	Testovanie zápisu a načítania programu obaľovača
Vstupy	1. Vytvorený program obaľovača s akciami
	2. Uloženie a načítanie programu obaľovača
Výstupy	1. Zhodný program obaľovača – rovnaký strom akcií, rovnaké údaje príslušných akcií
Požiadavky na prostredie	

Tab. 10-19: Testovací prípad TCCore6

ID	TCCore6
Testovaná položka	Testovanie stratégie učenia na základe atribútov
Vstupy	1. Vytvorený prázdny program obaľovača
Výstupy	1. Spustiteľný program obaľovača
Požiadavky na prostredie	

10.3.3 Testovacie procedúry

Testovacia procedúra 1

Identifikátor: PCore1

Účel: Testovanie produktu pre testovacie scenáre TCCore1, TCCore2 a TCCore3.

Kroky procedúry:

5. **Zahájenie**:
   Spustenie programu Wrapper Designer-u.

6. **Procedúra**:
   6.1. Vytvorenie nového programu
   6.2. Pridanie akcie LoadPage (URI: „http://www.praca.sk/Applicants/searchJobs.aspx?page=1“)
   6.3. Pridanie akcie ExtractData – potomok akcie LoadPage
      6.3.1.Nastavenie output objektov (OutputObjectPath = objects, OutputObjectID = id)
      6.3.2.Pridanie dvoch pozitívnych príkladov – prvé dve pracovné ponuky
   6.4. Pridanie akcie PushData – potomok akcie ExtractData
      6.4.1.Nastavenie output objektov (OutputObjectPath = object, OutputObjectID = id)
   6.5. Spustenie programu obaľovača

7. **Meranie výsledkov:**
   Výsledok je vyhodnotení po vykonaní procedúry: získané údaje musia byť uchované v príslušnej forme (XML súbor, Sesame ontológia, databáza MySQL) – porovnané vizuálne s webovou stránkou a získanými údajmi.

8. **Ukončenie:**
   Ukončenie programu obaľovača

Testovacia procedúra 2

Identifikátor: PCore2

Účel: Testovanie produktu pre testovacie scenáre TCCore1, TCCore2, TCCore3 a TCCore4.

Kroky procedúry:

1. **Zahájenie**:
   Spustenie programu Wrapper Designer-u.

2. **Procedúra**:
   2.1. Vytvorenie nového programu
   2.2. Pridanie akcie LoadPage (URI: „http://www.profesia.sk“)
   2.3. Pridanie akcie FollowLink – potomok akcie LoadPage
      2.3.1.Pridanie pozitívneho príkladu – linka „hľadáte prácu“
   2.4. Pridanie akcie Form – potomok akcie FollowLink
      2.4.1.Pridanie pozitívneho príkladu – výber komboboxu „Aktuálne ponuky za“
      2.4.2.Pridanie pozitívneho príkladu – výber položky z komboboxu „1 deň“
      2.4.3.Pridanie pozitívneho príkladu – kliknutie na tlačidlo „Hľadať ponuky“
   2.5. Pridanie akcie ExtractData – potomok akcie Form
      2.5.1.Nastavenie output objektov (OutputObjectPath = objects, OutputObjectID = id)
      2.5.2.Pridanie dvoch pozitívnych príkladov – prvé dve pracovné ponuky
   2.6. Pridanie akcie PushData – potomok akcie ExtractData
      2.6.1.Nastavenie output objektov (OutputObjectPath = object, OutputObjectID = id)
   2.7. Spustenie programu obaľovača

3. **Meranie výsledkov:**
   Výsledok je vyhodnotení počas vykonávania procedúry: akcie musia spolupracovať a dcérskym akciám musia poskytovať správne údaje, získané výsledné údaje musia byť uložené v príslušnej forme.

4. **Ukončenie:**
   Ukončenie programu obaľovača

Testovacia procedúra 3

Identifikátor: PCore3

Účel: Testovanie produktu pre testovacie scenáre TCCore5.

Kroky procedúry:

1. **Zahájenie**:
   Spustenie programu Wrapper Designer-u.

2. **Procedúra**:
   2.1. Vytvorenie nového programu
   2.2. Pridanie akcie LoadPage (URI: „http://www.profesia.sk/praca/user_details.php3“)
   2.3. Pridanie akcie ExtractData – potomok akcie LoadPage
      2.3.1.Nastavenie output objektov (OutputObjectPath = objects, OutputObjectID = title)
      2.3.2.Pridanie dvoch pozitívnych príkladov – prvé dve pracovné ponuky
   2.4. Pridanie akcie PushData – potomok akcie ExtractData

2.4.1.Nastavenie output objektov (OutputObjectPath = object, OutputObjectID = title)
2.5. Spustenie programu obaľovača
2.6. Uloženie programu do súboru (profesia.xml)
2.7. Zatvorenie programu
2.8. Načítanie programu zo súboru (profesia.xml)
2.9. Spustenie programu obaľovača

3. **Meranie výsledkov:**
Správnosť uloženia a načítania programu sa určuje podľa údajov získaných po spustení vytvoreného programu obaľovača a spustení načítaného (rovnakého) programu obaľovača

4. **Ukončenie:**
Ukončenie programu obaľovača

Testovacia procedúra 4

Identifikátor: PCore4

Účel: Testovanie produktu pre testovacie scenáre TCCore6

Kroky procedúry:

1. **Zahájenie**:
Spustenie programu Wrapper Designer-u.

2. **Procedúra**:
2.1. Vytvorenie nového programu
2.2. Pridanie akcie LoadPage (URI: „http://www.profesia.sk/praca/user_details.php3“)
2.3. Pridanie akcie ExtractData – potomok akcie LoadPage
2.3.1.Výber stratégie Attribute Selection Strategy
2.3.2.Nastavenie output objektov (OutputObjectPath = objects, OutputObjectID = title)
2.3.3.Pridanie dvoch pozitívnych príkladov – prvé dve pracovné ponuky
2.4. Pridanie akcie PushData – potomok akcie ExtractData
2.4.1.Nastavenie output objektov (OutputObjectPath = object, OutputObjectID = title)
2.5. Spustenie programu obaľovača

3. **Meranie výsledkov:**
Výsledok je vyhodnotení počas a po vykonaní procedúry: pozitívne príklady sa musia zvýrazniť a vo výstupnom XML súbore sa musia uchovávať získané údaje.

4. **Ukončenie:**
Ukončenie programu obaľovača

10.3.4 Súhrnná správa o teste

ID	RCore1
Požadované výsledky	TCCore1 – Obaľovač správne uloží získané údaje do XML súboru.
	TCCore2 – Obaľovač správne uloží získané údaje do ontológie Sesame.
	TCCore3 – Obaľovač správne uloží získané údaje do databázy MySQL.
	TCCore4 – Všetky akcie pracujú správne, poskytujú správne údaje dcérskym akciám.
	TCCore5 – Program obaľovača sa správa po vytvorení a pri opätovnom načítaní zo súboru rovnako. Vytvorený a načítaný program bol rovnaký, obsahoval rovnaké akcie a údaje.
	TCCore6 – Stratégia Attribute Selection správne určí potrebné údaje a tie sa zapíšu do výstupného XML súboru
Dosiahnuté výsledky	TCCore1 – Dosiahli sa požadované výsledky.
	TCCore2 – Ontológia Sesame sa nenaplnila. Problémy nastali už pred pripojením sa na ontológiu.
	TCCore3 – Rovnako ako ontológia Sesame, tak aj databáza MySQL sa nenaplnila získanými údajmi.
	TCCore4 – Všetky akcie pred akciou Form pracovali správne. Akcia Form vrátila zlé výsledky jej dcérskej akcii, a kvôli tomu nebolo možné otvoriť v tejto dcérskej akcii okno JRex na zadanie pozitívnych príkladov.
	TCCore5 – Pri ukladaní programu obaľovača nastala chyba.
	TCCore6 – Stratégia nedokázala určiť potrebné údaje, JRex nezobrazil vybrané pozitívne príklady, spustenie programu obaľovača zlyhalo.
Ohodnotenie a návrhy na zlepšenie	Nájdené nedostatky rapídne ovplyvňujú použiteľnosť vytvoreného systému. Chyby vznikajú pri zložitejšej štruktúre programu obaľovača, ako aj pri zápise získaných údajov do niektorých foriem. Ďalšou chybou je zápis vytvoreného programu do súboru. Načítanie zo súboru nebolo možné otestovať. Všetky tieto chyby boli silne ovplyvnené použitím projektu JRex, a preto navrhujeme použiť inú techniku, ktorá pracuje nad DOM dokumentom.

10.4 Opis testovacích prostredí

Počas testovania zapisovačov sme použili ako testovacie prostredia nástroje Sesame a MySQL. Tieto nástroje slúžia na uchovávanie získaných údajov z obaľovača. Tieto údaje sa namapujú na existujúce šablóny a tie sa poskytnú spomínaným nástrojom.

Sesame

Sesame je open-source Java framework na ukladanie a dopytovanie súborov Resource Description Framework (RDF) a RDF schém (šablóny pre uchovávané údaje). Môže byť použitý ako databáza pre tieto súbory, alebo ako Java knižnica pre aplikácie pracujúce s RDF. Sesame poskytuje potrebné nástroje na parsovanie, interpretáciu, dotazovanie a uchovávanie všetkých

informácií v oddelenej databáze alebo na vzdialenom serveri.

MySQL

MySQL je open-source databáza, ktorá umožňuje uchovávanie údajov vo forme tabuliek. Tieto tabuľky je možné vytvárať, rušiť a meniť. Databáza MySQL umožňuje dotazovanie sa na uchovávané údaje, a to vďaka jazyku Structured Query Language (SQL). Pomocou tohto dotazovania môžeme vytvárať rôzne náhľady nad existujúcimi tabuľkami. MySQL databáza pracuje na vzdialenom serveri, ale môže fungovať aj na lokálnej stanici použitím ďalších nástrojov.

11 Zhodnotenie

Náš projekt nadväzoval na prácu minuloročného tímu, ktorý vytvoril hotový rámec WrapperDesigner. Medzi základnými cieľmi, ktoré sme si stanovili, bolo vytvorenie nového prístupu tvorby a interpretácie obaľovača. Základom tohto prístupu bolo vytvorenie akcií, ktoré by pracovali s relatívnymi časťami dokumentu prostredníctvom lokálnych kontextov. Ďalšou dôležitou črtou vytvoreného obaľovača bola zmena lokalizácie údajov v dokumente.

Údaje na stránke sú lokalizované prostredníctvom filtrov, ktoré sú súčasťou vzorov asociovaných k akciám. Filtre sú vytvárané pomocou stratégií učenia, ktoré určujú spôsob reflexie príkladov vyznačených používateľom (pozitívne alebo negatívne) do hodnoty filtra. Jedným prístupom v stratégií učenia bola reprezentácia filtra pomocou XPath výrazu, kde jednotlivé príklady boli generalizované so zachovaním podmienky zahŕňať všetky pozitívne a žiadne negatívne príklady. Druhým spôsobom reprezentácie filtrov zavedených v obaľovači bol klasifikátor, ktorý na základe vytvorených pravidiel určí triedu zadaného príkladu (pozitívna, negatívna trieda). V tomto prístupe sú príklady reprezentované množinou atribútov s priradenými hodnotami. Učenie vzorov je možné jednoducho rozšíriť pri zachovaní jadra aplikácie vytvorením novej stratégie a implementovaním potrebných metód.

Akcie vytvoreného obaľovača postupne dekomponujú extrahovaný dokument prostredníctvom predávania si lokálnych kontextov. Každá akcia tak pracuje nad vlastným subdokumentom, ktorý je relativizovaný vzhľadom na rodičovskú akciu. Podobný princíp sme použili aj pre implementáciu výstupných objektov, ktoré sú buď akciou priamo vytvárané a niektorým z dostupných zapisovačov transformované do výstupného prostredia, alebo akcia pokračuje v zápise na mieste kde skončila jej rodičovská akcia.

Ďalšou charakteristikou vytvoreného obaľovača bolo integrovanie webového prehliadača, ktorý ma za cieľ zjednodušiť prácu používateľa a odbremeniť ho od zložitej lokalizácie údajov v dokumente pomocou XPath výrazu. Používateľ interaktívnou formou zadáva svoje voľby a nástroj využitím niektorej zo stratégii učenia automaticky vytvára a následne v obaľovači zobrazuje vytvorené vzory.

Možných rozšírení aplikácie je viacero. Keďže medzi naše priority patrilo najmä vytvorenie jadra aplikácie na základe nového pohľadu na problematiku a použitie integrovaného prehliadača, veľký priestor môžeme vidieť v rozšírení najmä prezentačnej vrstvy aplikácie. Súčasťou prezentačnej vrstvy aplikácie môže byť umožnenie modifikácie stromu akcií (napr. presúvaním podstromov), atď. Medzi ďalšie výzvy patrí zlepšenie interakcie integrovaného prehliadača s aplikáciou a zvýšenie jeho výkonnosti. V aplikačnej vrstve je možné dopĺňať nové stratégie učenia a tvorby filtrov, ktoré by pracovali aj s inou reprezentáciou dokumentu (napr. reťazcová reprezentácia). Zmeny reprezentácií dokumentov je možné realizovať veľmi jednoducho pomocou implementovaného adaptačného rámca. Tak isto je v prípade potreby možné do ponuky prostredia doplniť nové akcie pre obaľovače (napríklad extrahovanie multimediálneho obsahu, a pod). Ďalšie rozšírenia funkcionality aplikácie sa môžu týkať implementácie zapisovačov do

nových výstupných prostredí, či zlepšenia a spríjemnenia ovládateľnosti v integrovanom prehliadači stránok (použitie viacerých farieb s odlišným významom pri vyznačovaní objektov na stránke).

Cieľom tímového projektu bolo tiež vytvorenie modulárneho rámca obaľovača implementujúceho nový prístup k tvorbe a interpretácie obaľovača. Vytvorený rámec je ľahko rozšíriteľný, kedže bol viackrát aplikovaný návrhový vzor stratégií (*strategy*) [10].

12 Použité pramene

[1] Kapow Technologies. *RoboSuite Quick Start Guide* [online]. Verzia 6.1. 2006 [cit. 2006-11-11]. Dostupné na: http://kdc.kapowtech.com/documentation_6_1/robosuite/RoboSuiteQuickStartGuide.pdf.

[2] Kapow Technologies. *ModelMaker User's Guide* [online]. Verzia 6.1. 2006 [cit. 2006-11-11]. Dostupné na: http://kdc.kapowtech.com/documentation_6_1/robosuite/ModelMakerUsersGuide.pdf.

[3] Kapow Technologies. *RoboMaker User's Guide* [online]. Verzia 6.1. 2006 [cit. 2006-11-11]. Dostupné na: http://kdc.kapowtech.com/documentation_6_1/robosuite/RoboManagerUsersGuide.pdf.

[4] Lixto Software GmbH. *Lixto Visual Wrapper: User Manual.* Veriza 2.2. 2006. Dostupné ako súčasť inštalačného balíka Lixto Visual Wrapper.

[5] BERTA, I., JANŽO, A., JEMALA, M. et al. *Analýza textov pracovných ponúk z prostredia webu*. Bratislava, 2006. Tímový projekt na Fakulte informatiky a informačných technológií Slovenskej technickej univerzity v Bratislave. Vedúci projektu Mgr. György Frivolt. 158 s.

[6] XUL Planet. *XPCOM Interfaces* [online]. 2006 [cit. 2006-11-11]. Dostupné na: http://www.xulplanet.com/tutorials/xultu/xpcom.html.

[7] KOSEK, J. *XUL* [online]. 2002 [cit. 2006-11-13]. Dostupné na: http://www.kosek.cz/clanky/xul/index.htm.

[8] CARME, J., CERESNA, M., GOEBEL, M.: *Web Wrapper Specification Using Compound Filter Learning*. [s.l.] : [s.n.], 2006. 8 s.

[9] SZINEK, P. *W3C Mozilla DOM Connector will be released soon!* [online]. 2006 [cit. 2006-11-14]. Dostupné na: http://www.rubyrailways.com/w3c-mozilla-dom-connector-will-be-released-soon/.

[10] GoF. *Design Patterns: Elements of Reusable Object-Oriented Software*. London, 1994. ISBN 0-201-63361-2.

Príloha A: Používateľská príručka prototypu

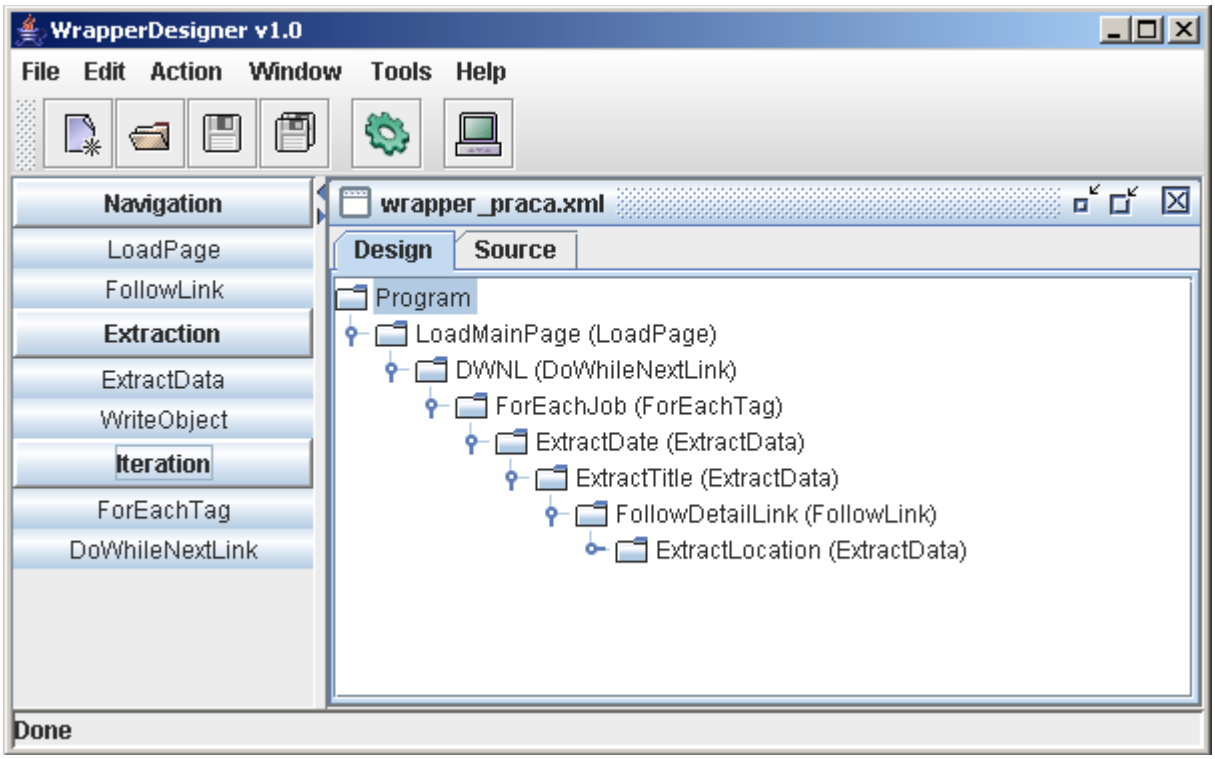
Prototyp aplikácie tvorby obaľovača nadväzuje na aplikačný rámec WrapperSuite, ktorý bol implementovaný tímom č. 10 v minulom akademickom roku. Zmenená bola iba kľúčová funkcionalita, prostredníctvom ktorej realizujeme ciele, ktoré sme si vytýčili v časti 7.1. Keďže vlaňajší aplikačný rámec obsahoval kompletnú príručku používateľa, v nasledujúcej časti kladieme dôraz najmä opis ovládania aplikácie súvisiaceho s funkcionalitou, ktorá úzko súvisí s demonštrovanými vlastnosťami prototypu.

A.1 WrapperDesigner

Na interakciu s používateľom slúži prostredie WrapperDesigner. Je rozdelené do nasledovných funkčných celkov:

1. menu,
2. panel nástrojov,
3. panel obsahujúci jednotlivé akcie, ktoré môžu byť použité pri tvorbe obaľovača,
4. návrhový a zdrojový pohľad na zdrojový súbor vytváraný obaľovač,
5. interpreter.

Snímok obrazovky je zobrazený na Obr. A-1.



Obr. A-1: Snímok obrazovky prostredia WrapperDesigner

A.1.1 Menu

Menu prostredia WrapperDesigner je rozdelené na nasledovné položky:

- File – umožňuje prístup k príkazom pre prácu s aktuálne spracovávaným zdrojovým kódom obaľovača,
- Edit – umožňuje prístup k príkazom pre prácu s textom umiestneným v schránke,
- Action – obsahuje možnosti práce s akciami, jednotlivé položky budú bližšie opísané v nasledujúcich častiach
- Window – umožňuje prístup k príkazom pre prepínanie medzi oknami jednotlivých otvorených zdrojových súborov obaľovača,
- Tools – obsahuje nástroje aplikačného rámce WrapperSuite, ako aj možnosti nastavenia vlastností aplikácie
- Help – umožňuje prístup k oknu s informáciami o tejto aplikácii.

A.1.2 Panel nástrojov

Možnosti panela nástrojov prakticky zodpovedajú najčastejšie používaným položkám menu. Išlo nám hlavne o pohodlné používateľské prostredie a intuitívne ovládanie aplikácie prototypu.

Obr. A-2: Panel nástrojov prostredia WrapperDesigner

Panel nástrojov prostredia WrapperDesigner obsahuje postupne (sprava doľava) položky umožňujúce prístup k nasledujúcim funkciám:

- Vytvorenie nového zdrojového súboru obaľovača,
- Otvorenie existujúceho zdrojového súboru obaľovača,
- Uloženie aktuálne spracovávaného zdrojového súboru obaľovača
- Uloženie všetkých otvorených zdrojových súborov obaľovačov,
- Interpretácia aktuálne spracovávaného zdrojového súboru obaľovača,
- Zobrazenie konzoly umožňujúcej prehliadanie výstupu obaľovača.

A.1.3 Panel akcií

Panel akcií prostredia WrapperDesigner, ktoré môžu byť použité pri vytváraní obaľovača je rozdelený do troch celkov, podľa typu akcie:

- navigačné akcie,
- extrakčné akcie,
- iteračné akcie.

Po kliknutí a rozbalení jednotlivých skupín akcií je možné priamo vybrať požadovanú akciu. V prípade navigačných akcií sú to nasledujúce:

- akcia LoadPage,
- akcia FollowLink.

V prípade extrakčných akcií ide o:

- akcia ExtractData,
- akcia WriteObject.

V prípade iteračných akcií sú to nasledujúce:

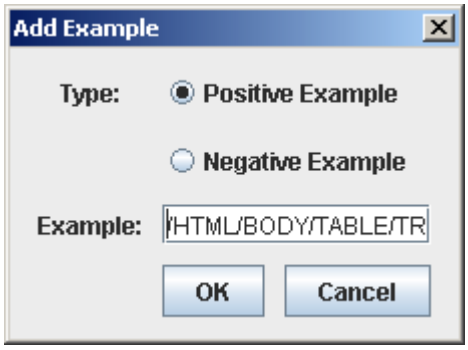
- akcia ForEachTag,
- akcia DoWhileNextLink.

Z pohľadu prototypu sú kľúčové akcie LoadPage, ForEachTag, ExtractData a WriteObject. Ich konkrétne použitie bude opísané v časti A.2.

A.1.4 Návrhový pohľad na zdrojový súbor – karta Design

Pri tomto pohľade je zdrojový súbor vytváraného obaľovača zobrazený ako strom na seba nadväzujúcich, postupne vykonávaných akcií (pozri Obr.A-1 - kartu Design). V tomto pohľade je možné editovanie samotného programu obaľovača. Editovanie realizujeme pridávaním alebo odstraňovaním akcií. Všetky úpravy je možné robiť nielen pomocou panela nástrojov alebo prostredníctvom menu, ale tiež použitím pravého tlačidla myši, čo zabezpečuje okamžitý prístup k vlastnostiam akcie, na ktorú sme klikli.

Z pohľadu prototypu je dôležité všimnúť si možnosť pridávania príkladov (položka menu Actions, resp. kliknutím pravého tlačidla a následne Add Example). Používateľ môže pre danú stratégiu učenia zadať pozitívny alebo negatívny príklad v podobe XPath výrazu (Obr. A-3).

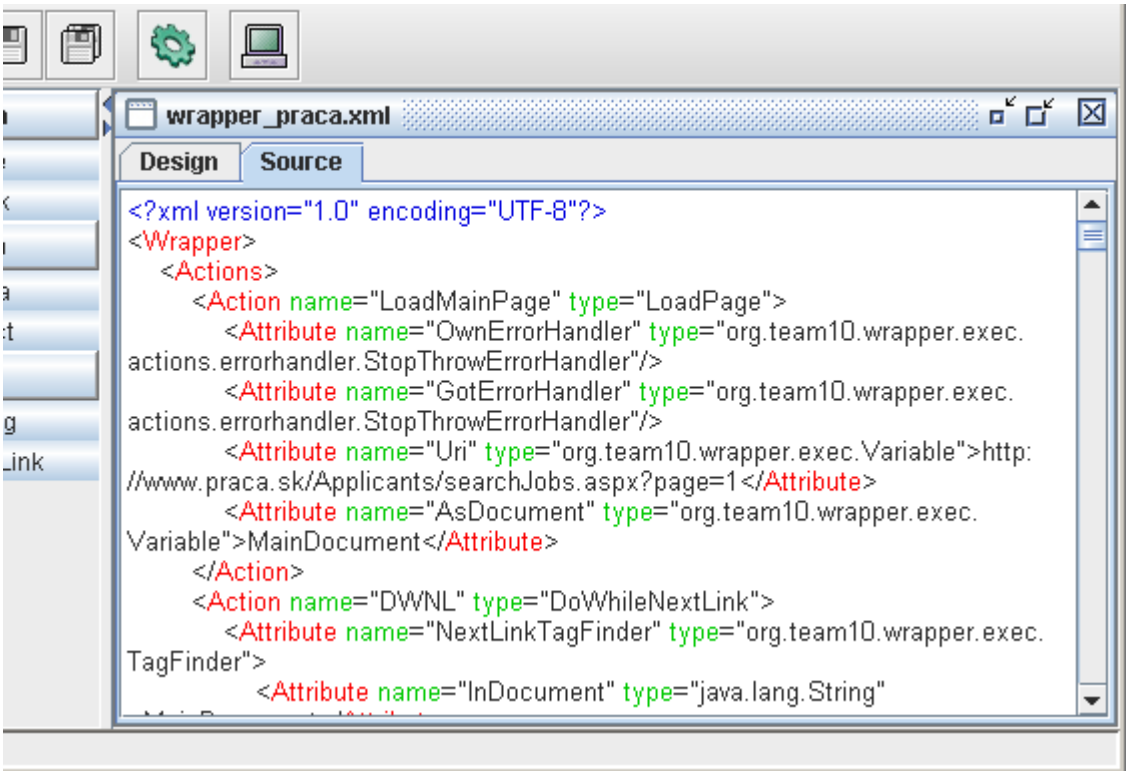


Obr. A-3: Pridanie príkladu

A.1.5 Zdrojový pohľad na zdrojový súbor – karta Source

Okrem návrhového pohľadu na vytváraný zdrojový súbor obaľovača je možné prepnutie sa aj do karty Source (Obr. A-4), ktorá poskytuje pohľad priamo vytváraný XML súbor obaľovača tak, ako bude uložený do výstupného súboru. Kvôli prehľadnosti a väčšej flexibilite kontroly

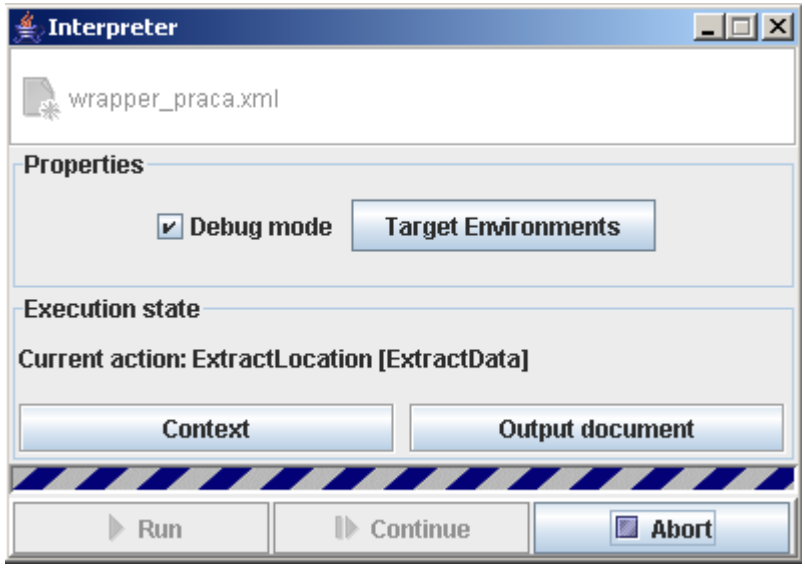
vytváraného zdrojového súboru používateľom bolo použité zvýraznenie syntaxe.



Obr. A-4: Snímok časti obrazovky, kde sa nachádza tzv. zdrojový pohľad na zdrojový súbor

A.1.6 Interpreter

Okrem možnosti vizuálneho vytvárania a editovania zdrojového súboru obaľovača poskytuje prostredie WrapperDesigner aj prostriedky pre interpretáciu takto vytvoreného zdrojového súboru (Obr. A-5).



Obr. A-5: Snímok obrazovky interpretera

Keďže práca s interpretom sa od predchádzajúcej verzie nezmenila, nemá zmysel rozoberať detaily možností použitia v prípade prototypu znovu.

A.2 Demonštrácia prototypu

Nasledujúca časť opisuje postup použitia aplikácie WrapperSuite na docielenie a ukázanie takého správania, na ktorom bude možné demonštrovať splnenie cieľov, ktoré boli kladené na prototyp. Uvidíme, ako sa na praktickom príklade uplatní jednoduchá stratégia učenia sa.

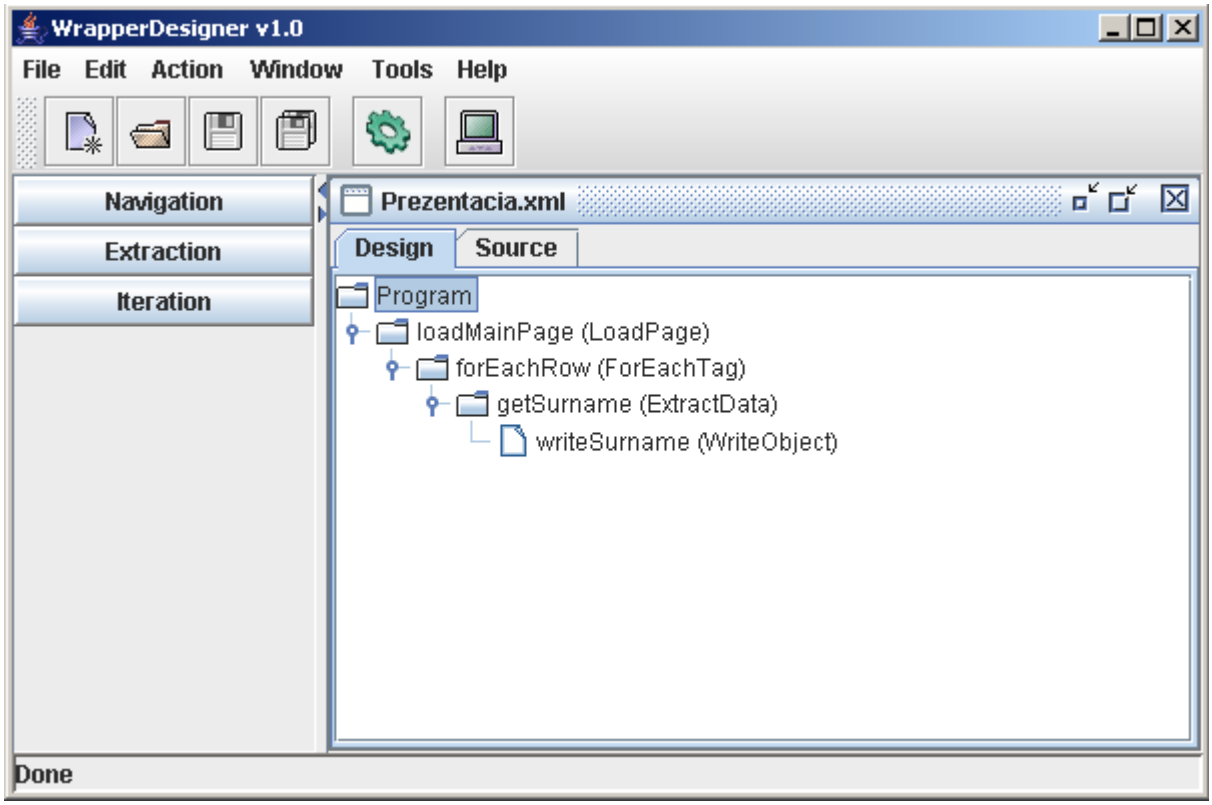
Postup vytvorenia obaľovača a nasledovného spustenia učenia sa a extrahovania potrebných dát pozostáva z nasledujúcich krokov:

1. Vytvorenie nového obaľovača kliknutím na ikonu New alebo výberom tejto položky z menu File
2. Pridanie navigačnej akcie LoadPage. Do zobrazeného dialógového okna je potrebné vložiť parametre akcie:
    1. Name – názov akcie: *loadMainPage*;
    2. URI – adresu nahrávanej stránky – *http://osiris.yweb.sk/tabulka.html* – webová stránka vytvorená pre testovacie účely
    3. asDocument – názov dokumentu, pod ktorým bude vystupovať v kontexte: *MainDocument*.

Prezentácia učenia – najprv pomocou RepeatingXPathLearningStrategy:

3. Pridanie iteračnej akcie ForEachTag. Do zobrazeného dialógového okna je potrebné vložiť parametre akcie:
    1. Name v karte General: *forEachRow*
    2. LearningStrategy v karte General: *RepeatingLearningStrategy*
    3. XPathExp v karte TagFinder: *necháme prázdne*
4. Pridanie extrakčnej akcie ExtractData. Do zobrazeného dialógového okna je potrebné vložiť parametre akcie:
    1. Name v karte General: *getSurname*
    2. LearningStrategy v karte General: *RepeatingLearningStrategy*
    3. inDocument v karte Source: *MainDocument*
    4. XPathExp v karte Soure: *necháme prázdne*
    5. OutObject v karte Destination: *objSurname*
    6. OutObjectPath v karte Destination: *Surname*
    7. ContVariable v karte Destination: *necháme prázdne*
5. Pridanie extrakčnej akcie WriteObject. Do zobrazeného dialógového okna je potrebné vložiť parametre akcie:
    1. Name: *writeSurname*
    2. ObjectName: *objSurname*

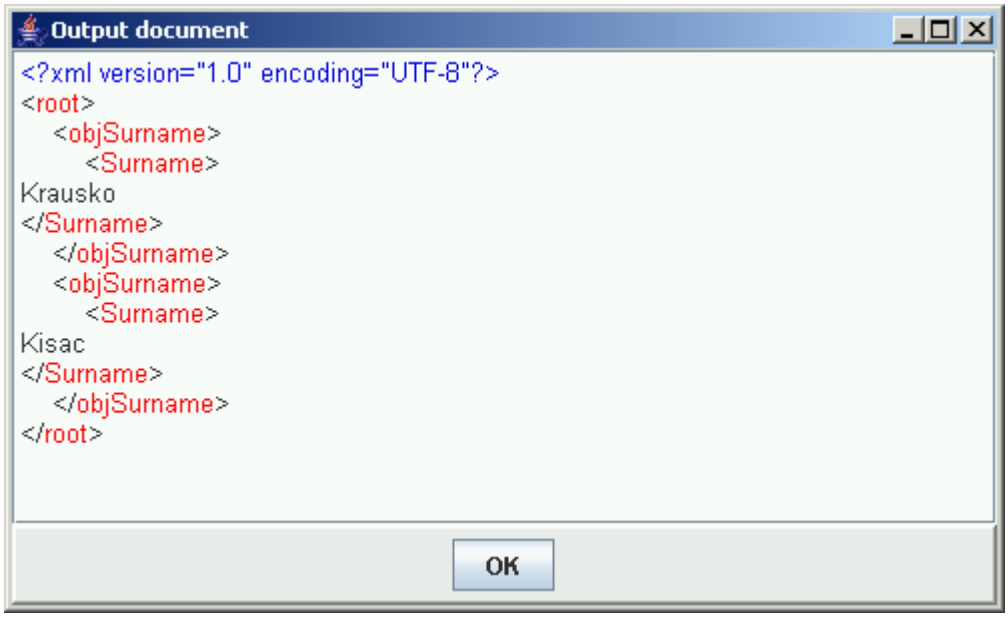
6. Zadáme prvý pozitívny príklad.
   1. Kliknutím pravým tlačidlom myši na akciu *forEachRow*
   2. V kontextovom menu kliknutie ľavým tlačidlom na *Add Example*
   3. Typ príkladu: *Positive Example*.
   4. Do políčka Example: */HTML/BODY/TABLE/TR[2]/TD[2]*
   5. Potvrdenie OK
      Aktuálny stav v návrhovom pohľade by mal zodpovedať Obr. A-6.



Obr. A-6: Obaľovač používajúci stratégiu RepeatingLearningStrategy

7. Spustíme interpret. Spustíme obaľovač – tlačidlom Run.
8. Výstup skontrolujeme tlačidlom Output document, kde vidíme jedno priezvisko z druhého riadku a druhého stĺpca tabuľky.
9. Zatvorenie interpreta (tlačidlo Close) a dokumentu (tlačidlo OK).
10. Pridanie ďalšieho pozitívneho príkladu. Postupujeme ako v kroku 6, ale za Example zadáme: */HTML/BODY/TABLE/TR[3]/TD[2]*
11. Overenie výstupu – body 7., 8., 9. Tu vidíme, že pribudlo 1 priezvisko, teraz z riadku 3.

    Zvolená stratégia učenia teda nie je schopná generalizovať, filter prispôsobí iba poslednému zadanému príkladu (Obr. A-7).

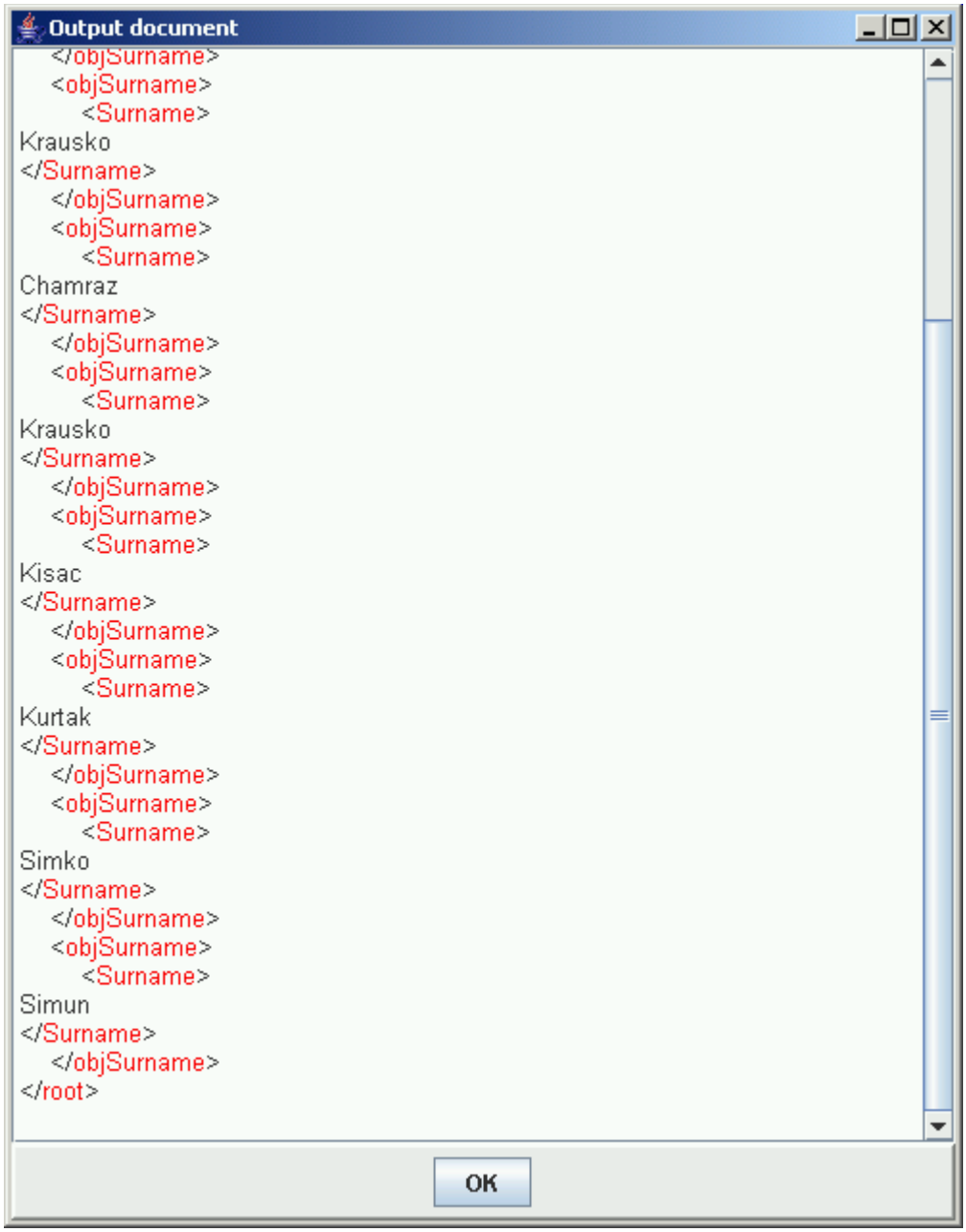


Obr. A-7: Podoba Output Document-u po učení stratégiou RepeatingLearningStrategy

Prezentácia učenia pomocou SimpleXPathLearningStrategy:

12. Zmena stratégie učenia
    1. Po pravom kliknutím na forEachTag a vybraním položky Edit sa otvorí dialógové okno, kde na karte General zmeníme Learning Strategy na SimpleLearningStrategy.
    2. Potvrdíme pomocou OK.
13. Zadanie prvého pozitívneho príkladu – rovnako ako v bode 6.
14. Overenie zopakovaním bodov 7., 8., 9. s rovnakým výsledkom.
15. Zadanie druhého pozitívneho príkladu – rovnako ako v bode 10.
16. Overenie ako v bode 11., ale do výsledku pribudol celý stĺpec s priezviskami.

    Zvolená stratégia generalizovala 2 zadané príklady, z ktorých odvodila výber celého stĺpca (Obr. A-8).



Obr. A-8: Podoba Output Document-u po učení stratégiou SimpleLearningStrategy

Príloha B: Používateľská príručka produktu

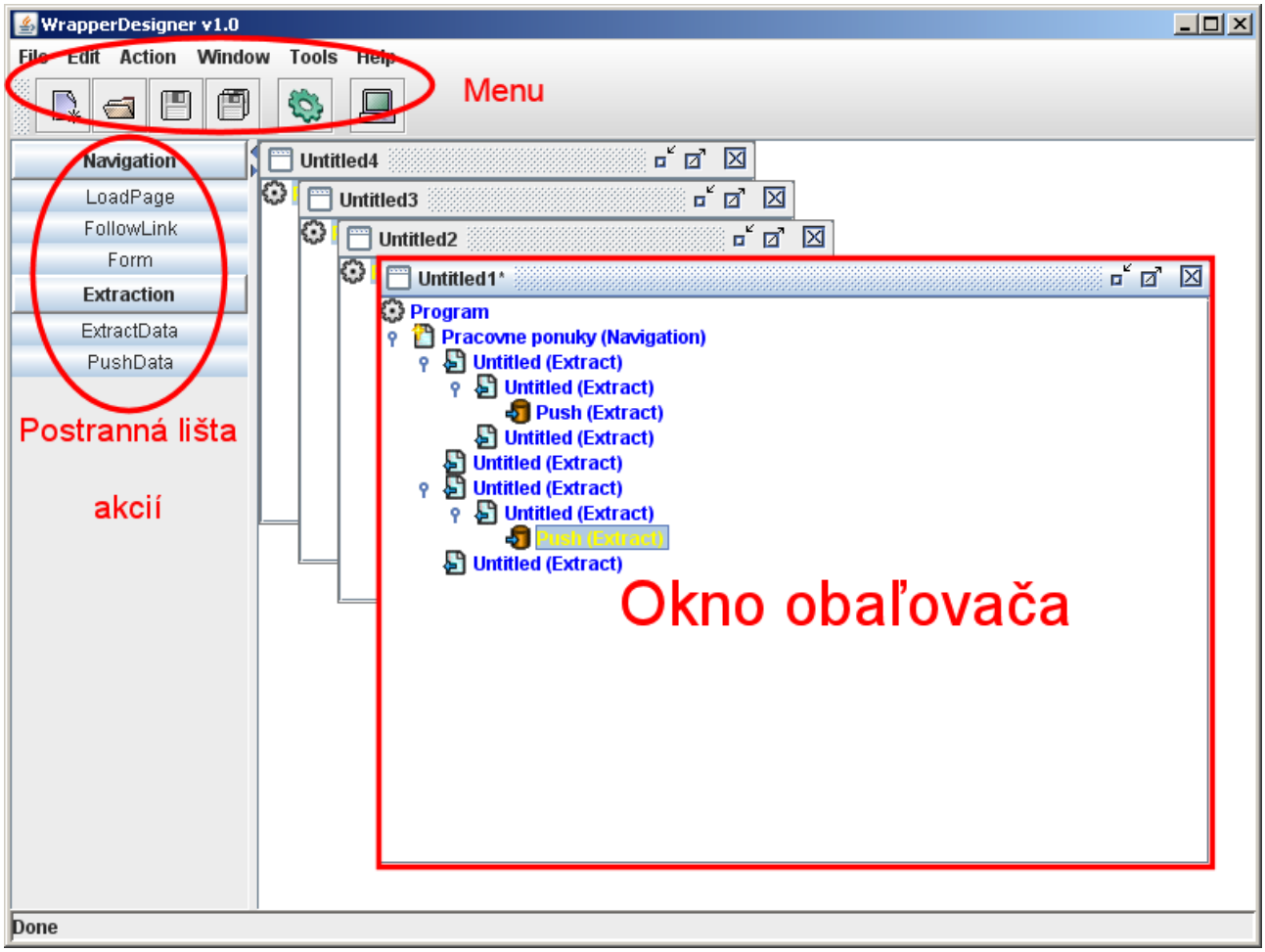
B.1 Inštalácia a spustenie

Program nevyžaduje špeciálnu inštaláciu, stačí ho na lokálny disk skopírovať a spustiť hlavný súbor wrapper.jar. Pred spustením sa uistite, že na PC je inštalované prostredie Java Runtime Enviroment (JRE) a cesta k nemu je správne nastavená v systémovej premennej PATH.

B.2 Okno WrapperDesigner a rozmiestnenie ovládacích prvkov

Okno aplikácie zobrazuje obrázok B-1. Skladá sa s 3 základných častí, a to:

- menu – textové menu a nástrojová lišta,
- postranná lišta akcii – lišta s kategóriami akcií,
- samotné okno obaľovača.



Obr. B-1: Okno aplikácie s vyznačenými aktívnymi prvkami

B.2.1 Menu

Menu je štandardné a má nasledovnú štruktúru:

- *File*
  - ◇ *New* – vytvorenie nového obaľovača a jeho okna
  - ◇ *Open...* - otvorenie obaľovača na editáciu
  - ◇ *Save* – uloženie aktívneho okna obaľovača
  - ◇ *Save As...* - nové uloženie aktívneho okna obaľovača
  - ◇ *SaveAll* - uloženie všetkých otvorených okien obaľovačov
  - ◇ *Close* – zatvorenie aktívneho obaľovača
  - ◇ *Close All* – zatvorenie všetkých obaľovačov
  - ◇ *Exit* – koniec aplikácie
- *Edit* – štandardné funkcie editácie
  - ◇ *Cut*
  - ◇ *Copy*
  - ◇ *Paste*
- *Action*
  - ◇ *Edit* – otvori na editáciu dialógové okno aktívnej akcie
  - ◇ *Remove recursive* – zmaže akciu aj s jej potomkami v strome
  - ◇ *View Pattern* – zobrazí identifikované vzory aktívnej akcie
- *Window* – standardné funkcie prace s oknami
  - ◇ *Minimize*
  - ◇ *Maximize*
  - ◇ *Cascade*
  - ◇ všetky otvorené okná
- *Tools*
  - ◇ *Interpreter* – umožňuje spúšťanie obaľovača
  - ◇ *Console* – konzola na zachytávanie štandardných a chybových výpisov
- *Help*
  - ◇ *About*

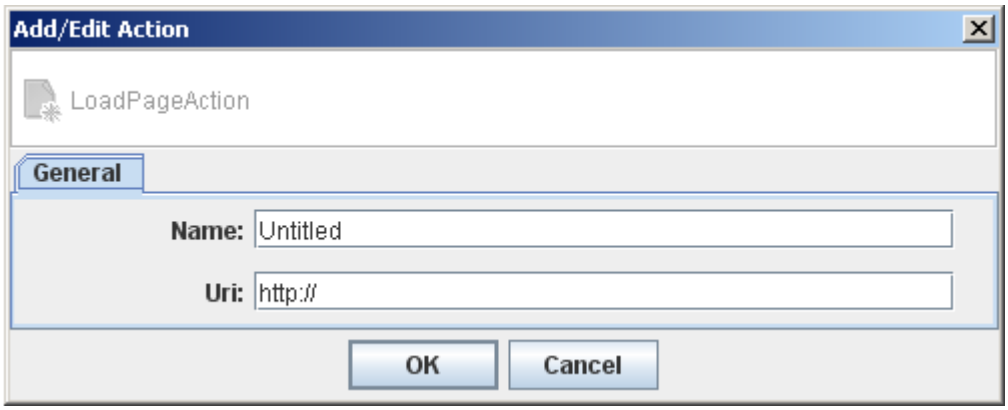
V menu nástrojovej lišty sa nachádzajú tlačidlá rýchlej voľby pre tieto funkcie z menu:

- New
- *Open...*
- Save
- Save All
- Interpreter
- Console

B.2.2 Dialógy akcií

Load Page

Dialógové okno akcie *LoadPage* zobrazuje obrázok B-2. Obsahuje len jednu záložku, ktorej najdôležitejšou a povinnou položkou je *Uri*.

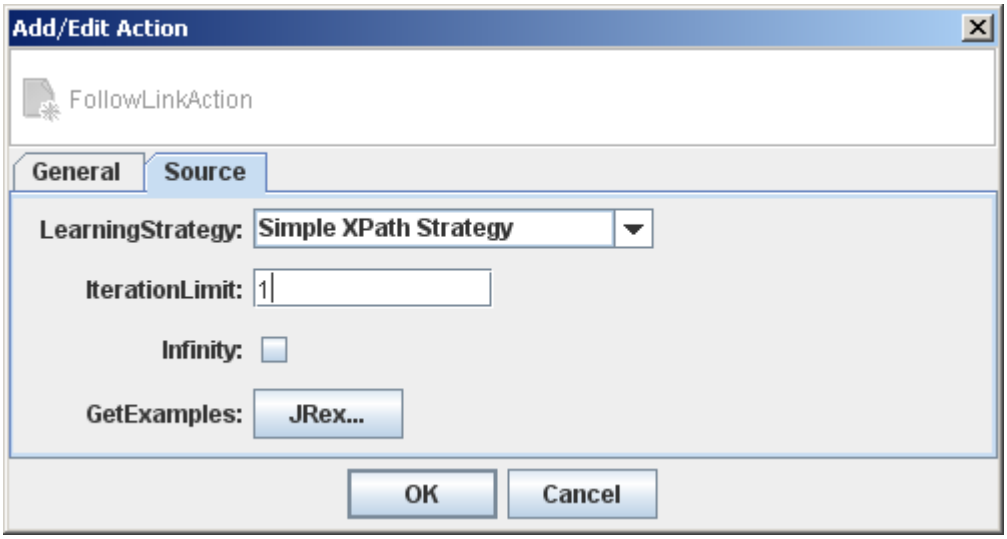


Obr B-2: Dialógové okno akcie LoadPage.

FollowLink

Dialógové okno akcie *FollowLink* zobrazuje obrázok B-3. Obsahuje dve záložky:

- *General* – obsahuje položku *Name* na nastavenie mena akcie a položku *Uri*, podobne ako pre akciu *LoadPage* je tento údaj povinný.
- *Source* – obsahuje položku pre výber stratégie učenia, zadanie počtu iterácii a spustenie integrovaného prehliadača *JRex* pre zadávanie príkladov. Položka *IterationLimit* môže byť nastavená aj na hodnotu nekonečno pomocou zaškrtávacieho tlačidla *Infinity*.

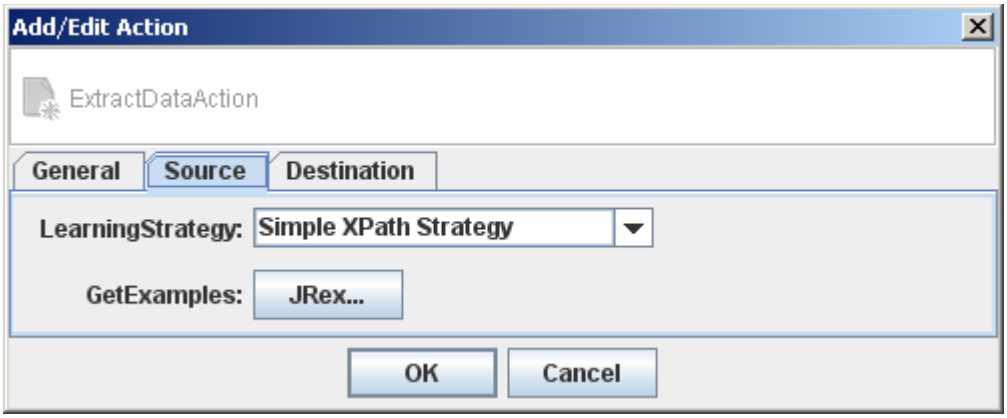


Obr. B-3: Dialógové okno akcie FollowLink s aktívnou záložkou Source.

ExtractData

Dialógové okno akcie *ExtractData* zobrazuje obrázok B-4. Obsahuje tri záložky:

- *General* – obsahuje len položku *Name* na nastavenie mena akcie.
- *Source* – pre výber stratégie učenia a spustenie integrovaného prehliadača *JRex* pre zadávanie príkladov.
- *Destination* – obsahuje položky *outputObjectPath* na zadanie cesty k výstupnému objektu a *outputObjektID* na zadanie identifikátora výstupného objektu. Parametre nie sú povinné. Ak však zadáme aspoň jeden z nich musíme zadať aj druhý!

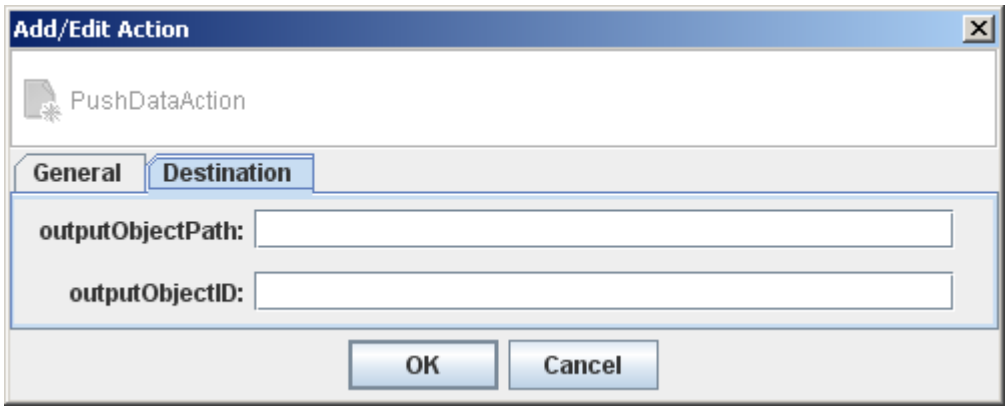


Obr. B-4: Dialógové okno akcie ExtractData s aktívnou záložkou Source.

PushData

Dialógové okno akcie *PushData* zobrazuje obrázok B-5. Obsahuje dve záložky:

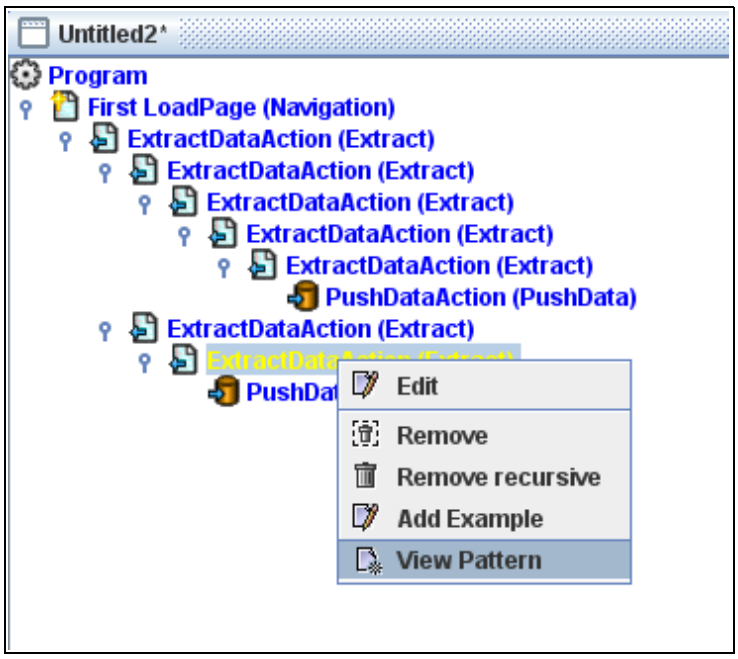
- *General* – obsahuje len položku *Name* na zadanie mena akcie,
- *Destination* – obsahuje *outputObjectPath* a *outputObjectID* rovnako ako akcia *ExtractData*. Oba parametre sú tak isto nepovinné, ale ak jednému z nich nastavíme nejakú hodnotu, musíme nastaviť aj druhý parameter.



Obr. B-5: Dialógové okno akcie PushData s aktívnou záložkou Destination.

B.2.3 Strom akcií a kontextové pop-up menu

Na obrázku B-6 je vidieť príklad stromu akcií a kontextového pop-up menu. Strom akcií vytvárame pridávaním akcií, ktoré sa automaticky ukladajú pod aktívnu (označenú) akciu. Ak teda chceme aby mala akcia v strome viacero potomkov na rovnakej úrovni, musí byť pri pridávaní potomkov vždy aktívna práve táto rodičovská akcia. Kontextové menu nám jednoduchým spôsobom sprístupňuje akcie, ktoré sú v textovom menu aplikácie pod položkou *Action*.



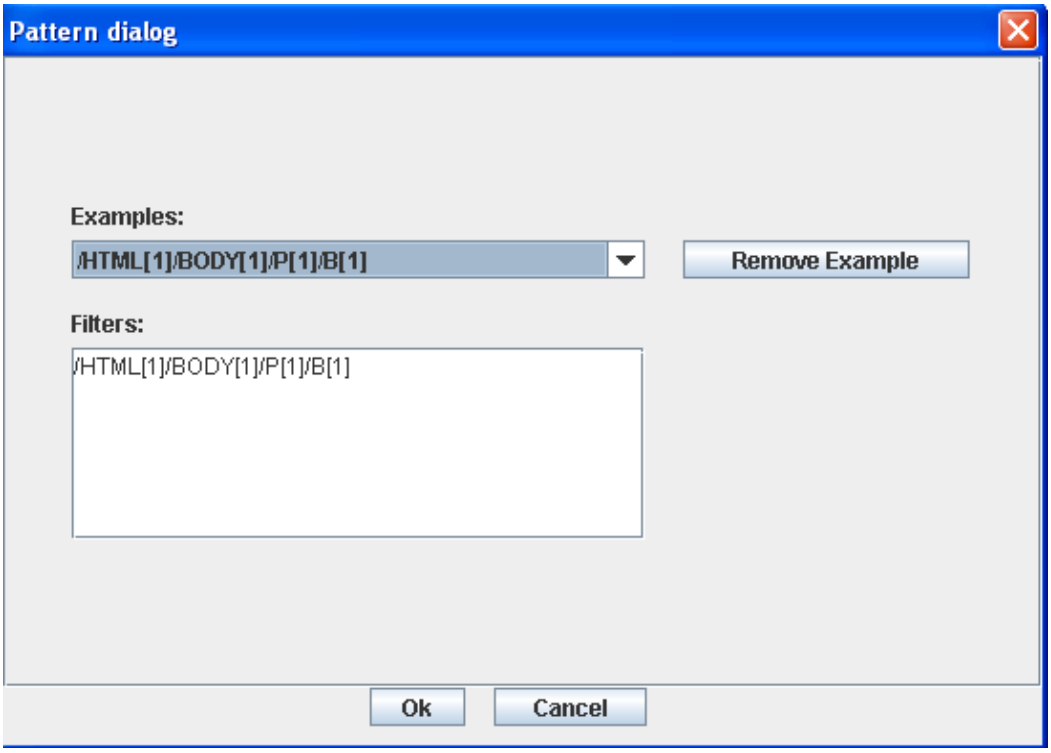
Obr. B-6: Ukážka stromu akcii a kontextového pop-up menu.

ViewPattern

Dialógové okno *zobrazenia vzorov - ViewPattern* zobrazuje obrázok B-7. Obsahuje 2 časti:

- *Examples* – naučené príklady vo vzore. Tlačidlom Remove Example sa odstráni aktuálny príklad a tým sa zmení aj naučený filter,
- *Filters* – obsahuje filter naučený z aktuálneho príkladu. Vyznačením posledného príkladu sa zobrazí aktuálny filter.

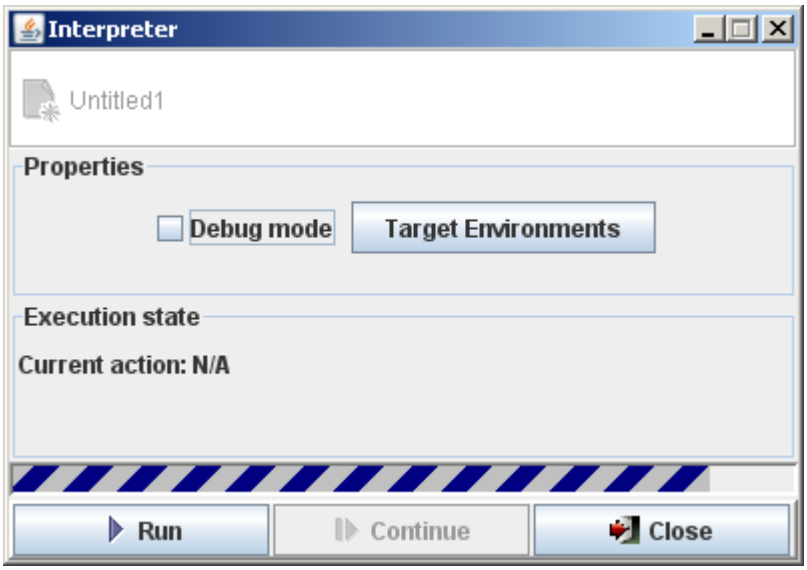
Tlačidlom *Ok* používateľ potvrdí zmeny vykonané v dialógu. Tlačidlo *Cancel* tieto zmeny zruší.



Obr. B-7: Dialógové okno zobrazenia vzorov.

B.2.4 InterpreterDialog

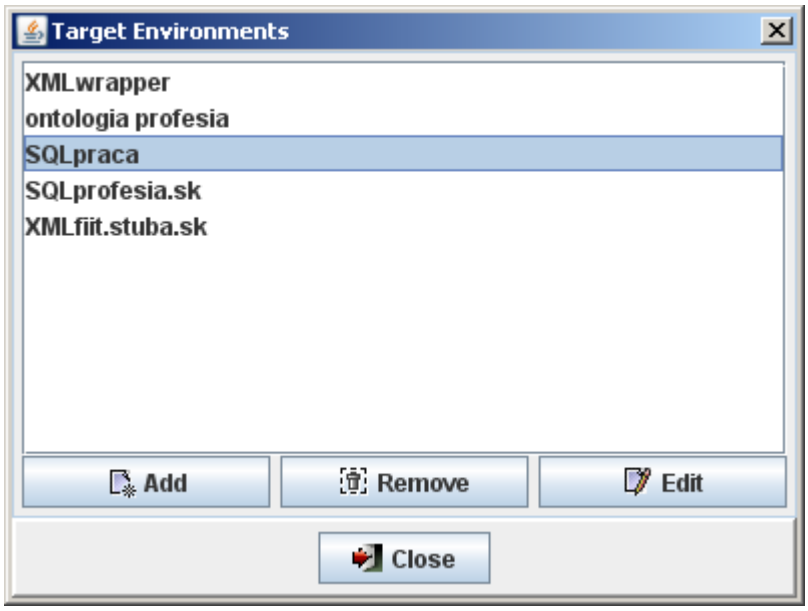
Na spúšťanie vytvorených obaľovačov slúži nástroj Interpreter. Jeho dialógové okno je na obrázku B-8. Okrem ovládacích prvkov ladenia a samotného spúšťania obaľovača, tento dialóg umožňuje prostredníctvom tlačidla „*Target Environmeents*“ aj nastavenie cieľového



Obr. B-8: Dialógové okno nástroja Interpreter.

prostredia, do ktorého bude Interpreter exportovať výsledky činnosti obaľovača. Po kliknutí na toto

tlačítko sa otvorí okno s výberom uložených profilov cieľových prostredí (obrázok B-9). Tie je možné editovať, mazať alebo pridávať úplne nové.

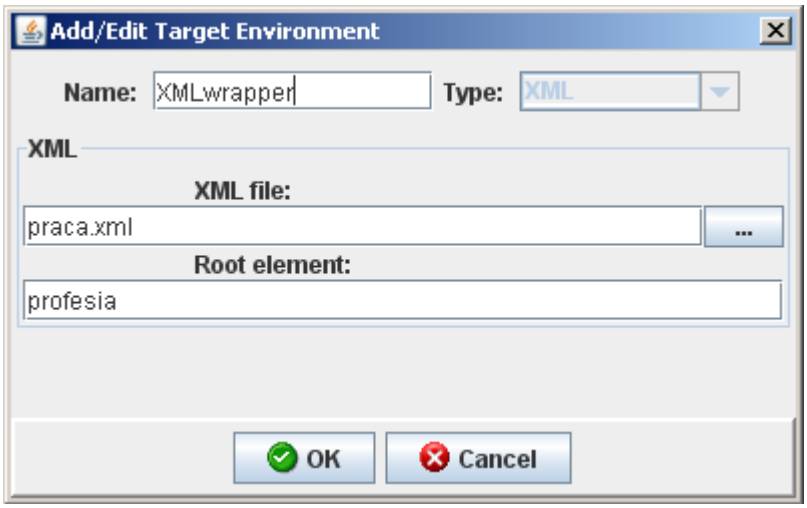


Obr. B-9: Dialógové okno pre výber profilu cieľového prostredia

Podľa vybratého typu výstupného súboru, ktorý sa volí prostredníctvom combo boxu sa zobrazia ďalšie prvky dialógového okna, ktoré je potrebné vyplniť. Aplikácia ponúka tieto cieľové prostredia:

- XML
- XSLT
- Ontology - jobOffer
- Ontology – RDF
- SQL Database

Na obrázku B-10 je okno pre výstupný súbor typu XML.



Obr. B-10: Dialógové okno pre editáciu zvoleného profilu cieľového prostredia.

B.3 Integrovaný prehliadač

Integrovaný prehliadač možno spúšťať vo vybraných akciách v záložke Source tlačidlom JRex. Následne sa otvorí okno prehliadača a používateľ môže vyznačovať pozitívne a negatívne príklady. V prehliadači je tiež vyznačená časť, z ktorej môže používateľ vyberať príklady a tiež naučený filter.

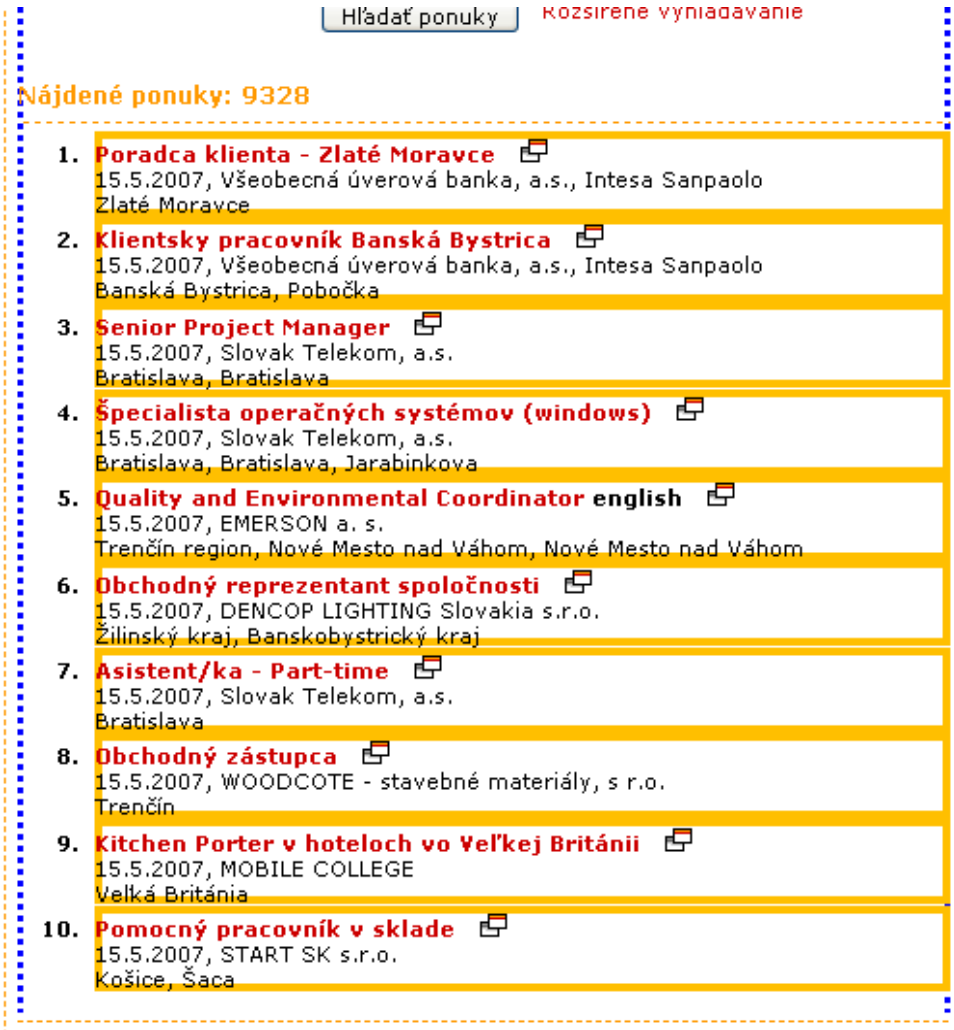
Na obrázku B-11 je naznačený výber pozitívneho príkladu. Používateľ označil časť "Hľadáte prácu?" (odkaz na pracovné ponuky). V prehliadači sa vyznačený príklad zobrazil v žltom orámovaní.

Obr. B-11: Vyznačenie pozitívneho príkladu

Na obrázku B-12 je zobrazený dokument s pridaným negatívnym príkladom (výber so stlačeným klávesom Ctrl) v pravej časti, ktorý je vyznačený modrým orámovaním.

Obr. B-12: Pridanie negatívneho príkladu v pravej časti dokumentu

Na obrázku B-13 sa nachádza dokument s vyznačenou povolenou časťou, z ktorej mohol používateľ vyberať príklady – prerušované orámovanie modrej farby (na základe extrakcie rodičovskej akcie). Z tejto časti používateľ vybral prvé 2 pracovné ponuky, následne sa na základe učiacej stratégie generalizoval filter vo vzore a vytvorený filter sa zobrazil vyznačením príkladov, ktoré sa budú extrahovať – všetky pracovné ponuky.



Obr. B-13: Vyznačenie viacerých pozitívnych príkladov v povolenej časti