Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný odbor: Softvérové inžinierstvo

Bc. Michal Barla

Zachytenie záujmov používateľa na webe

Diplomová práca

Vedúca diplomovej práce: prof. Ing. Mária Bieliková, PhD.
december 2006

Čestne prehlasujem, že túto prácu som vypracoval samostatne a použil som len citované zdroje.

Michal Barla

ANOTÁCIA

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný odbor: Softvérové inžinierstvo

Autor: Bc. Michal Barla
Diplomový projekt: Zachytenie záujmov používateľa na webe
Vedenie diplomového projektu: prof. Ing. Mária Bieliková, PhD.
December, 2006

Diplomová práca sa venuje problematike automatizovaného vytvárania modelu používateľa pre adaptívne webové systémy. V práci analyzujeme modelovanie používateľa z rôznych aspektov. Skúmame charakteristiky, ktoré sa do modelu používateľa ukladajú a možnosti reprezentácie modelu. Dôraz kladieme na analýzu spôsobov zberu dát a ich následného spracovania, čo sú dve základné úlohy procesu modelovania používateľa.

Hlavným výsledkom práce je návrh metódy automatizovaného získavania charakteristík používateľa, ktorá je založená na vytváraní záznamov o správaní používateľa počas interakcie s adaptívnym webovým systémom a analýze týchto záznamov. Princípom metódy je vytváranie záznamov s významom a aplikovanie heuristík pre odhad charakteristík používateľa na základe detekcie preddefinovaných vzorov správania.

Pre overenie metódy sme navrhli sadu softvérových nástrojov, ktoré realizujú jej jednotlivé kroky. Nástroje využívajú technológie webu so sémantikou, pracujú s ontologickou reprezentáciou modelu používateľa a domény.

ANNOTATION

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Software engineering

Author: Bc. Michal Barla
Diploma project: Interception of user's interests on the web
Supervisor: prof. Ing. Mária Bieliková, PhD.
2006, December

Main topic of this thesis is automatized user modeling for adaptive-web based systems. We analyze user modeling from various aspects. We explore categorization of user characteristics and possibilities of user model representations. We stress on analysis of approaches for realization of data collection and its successive processing as these two steps form base tasks of user model construction process.

Main result of the thesis is a method of automatized acquisition of user characteristics based on creation of logs about user behavior during her interaction with an adaptive web-based system and analysis of these logs. Main idea of proposed method is to create detailed user logs with semantics and to estimate user characteristics by applying heuristics determined by pre-defined behavior patterns.

To verify the feasibility of the proposed method, we devised a set of software tools which realize its individual steps. These tools take advantage of semantic web technologies, they work with an ontological representation of a user model and a domain model.

Obsah

1 Úvod

Množstvo informácií dostupných na Internete neustále narastá. Webové informačné systémy a rôzne webové sídla ponúkajú čoraz bohatší obsah, v ktorom je pre používateľa obtiažne vyhľadávať požadované informácie, resp. pohybovať sa v informačnom priestore smerom k želanému cieľu:

- používateľovi sa často prezentujú informácie, o ktoré nemá záujem alebo ktoré sú v daný moment pre neho nevhodné – to môže narúšať jeho koncentráciu, prípadne ho uviesť do omylu alebo mu úplne znemožniť prístup k tým správnym informáciám;
- používateľ sa sledovaním odkazov na webe stratí, je dezorientovaný, zabudne na svoje pôvodné ciele;
- používateľ sa často nevie rozhodnúť, ktorým odkazom má pokračovať v prehľadávaní webu.

Problém spočíva v rozsiahlom informačnom priestore a aktuálne prevládajúcej paradigme webových systémov, ktoré ponúkajú všetkým používateľom *rovnaký* obsah a možnosti navigácie, či už sa jedná o statické alebo dynamicky generované stránky. Rovnaká prezentácia však používateľom nemusí vyhovovať, keďže sa líšia vo svojich cieľoch, vzdelaní, záujmoch, preferenciách a pod.

Riešenie predstavujú tzv. adaptívne webové systémy [10], ktoré sa zameriavajú na špecifiká jednotlivých používateľov. Pri znalosti charakteristík používateľa sa dajú informácie prezentovať tak, aby sa odstránili vyššie spomínané problémy. Hovoríme, že systém sa prispôsobuje svojim používateľom. Obzvlášť veľký význam má tento prístup v aplikáciách, ktoré pokrývajú rozsiahly informačný priestor (či už uzavretý alebo otvorený) a predpokladá sa ich používanie používateľmi s rozdielnymi cieľmi a vedomosťami.

Získavanie charakteristík a cieľov používateľa je možné realizovať viacerými spôsobmi. Problémom je, že používateľ často nevie explicitne formulovať svoje charakteristiky alebo pri práci s adaptívnym webovým systémom nie je ochotný neustále explicitne vyjadrovať svoje potreby, ciele, vedomosti a pod. Preto je potrebné vyvíjať také spôsoby získavania charakteristík používateľov, ktoré minimalizujú mieru ich zapojenia do tohto procesu.

V tejto práci sa zaoberáme problematikou automatizovaného získavania charakteristík používateľa pre adaptívne webové systémy pričom sa zameriavame na analýzu správania používateľa v rámci webového systému. Vychádzame z dobre definovaného procesu modelovania používateľa pre adaptívne systémy a navrhujeme spôsoby realizácie jednotlivých úloh procesu – zberu a analýzy dát. Navrhovanú metódu sme overili vytvorením sady softvérových nástrojov a ich použitím na dvoch rôznych doménach v projektoch riešených na FIIT STU: pracovné ponuky z projektu NAZOU[1] a vedecké publikácie z projektu MAPEKUS[2].

Zber dát realizujeme ako kombináciu klientského a špeciálneho serverového zaznamenávania akcií používateľa s významom. Následná analýza je založená na pravidlách, heuristikách, ktoré spájajú vzor zložený zo zaznamenaných akcií so zmenami charakteristík v modeli používateľa.

V kapitole 2 analyzujeme možné prístupy k modelovaniu používateľa z viacerých hľadísk: reprezentácie modelu, zberu dát o používateľovi a ich následného spracovania. Kapitola 3 obsahuje zhrnutie aktuálneho stavu v danej oblasti a predstavuje hlavné ciele práce.

[1] projekt NAZOU, `http://nazou.fiit.stuba.sk`

[2] projekt MAPEKUS, `http://mapekus.fiit.stuba.sk`

V kapitole 4 opisujeme metódu pre zachytenie záujmov používateľa na webe spolu s dátovými modelmi, ktoré metóda používa. Kapitola 5 obsahuje opis návrhu a implementácie nástrojov, ktorými overujeme navrhnutú metódu. V kapitole 6 opisujeme niektoré experimenty, ktoré sme s vytvorenou implementáciou vykonali. Na záver podávame zhodnotenie a námety pre d'alšiu prácu.

Počas riešenia vznikli viaceré vedecké články predstavujúce rôzne aspekty riešenia na medzinárodných konferenciách. V prílohe A uvádzame články prijaté a prezentované na konferenciách AH 2006 v Dubline, ISD 2006 v Budapešti a Datakon 2006 v Brne.

V prílohe B sa nachádza opis ontologických modelov projektu NAZOU a MAPEKUS, na ktorých sme overovali navrhované riešenie. Technická dokumentácia k vyvinutým častiam riešenia sa nachádza v prílohe C. Príloha D predstavuje obsah priloženého dátového nosiča.

2 Modelovanie používateľa pre prispôsobovanie

Adaptívny webový systém je klasický webový systém obohatený o schopnosť prispôsobovať sa špecifikám jednotlivca. Existuje niekoľko referenčných modelov architektúry adaptívnych webových systémov z ktorých najznámejší je AHAM – *Adaptive Hypermedia Architecture Model* [19], ktorý vychádza z Dexterovského modelu klasických webových systémov [27].

Medzi hlavné modely adaptívneho webového systému patrí:

- model aplikačnej domény,
- navigačný model,
- prezentačný model,
- model kontextu (používateľa/prostredia),
- model prispôsobovania.

V tejto práci sa zaoberáme modelom používateľa adaptívneho systému a jeho vytváraním. V tomto modeli sa zachytávajú používateľove ciele, preferencie a ďalšie charakteristiky, ktoré slúžia pri prispôsobovaní sa systému špecifikám používateľa. Z povahy dát ukladaných v modeli je zrejmá úzka spätosť modelu používateľa so zvolenou aplikačnou doménou. Väčšinu príkladov uvádzame pre doménu pracovných ponúk alebo publikácií, keďže na týchto doménach overujeme výsledky tejto práce.

Personalizovaný systém nemusí mať (a často nemá) explicitne definovaný a oddelený model používateľa. Model používateľa môže byť implicitný, vytváraný nanovo pri každom navštívení systému používateľom (vtedy sa systém prispôsobuje len na základe aktuálnych akcií používateľa). Možnosti prispôsobovania takýchto systémov sú však značne obmedzené. Nevýhodou implicitného modelu používateľa je aj horšia rozšíriteľnosť a udržiavateľnosť systému.

2.1 Typy modelov používateľa

V súčasných systémoch sa používajú najčastejšie dva spôsoby realizácie modelu používateľa [10]:

- stereotypný model – jednoduchšia forma prispôsobovania, ktorá sa neprispôsobuje jednotlivým používateľom, ale určitým, väčšinou vopred, definovaným skupinám.
- prekryvný model – každý používateľ má svoju inštanciu modelu používateľa, ktorá vyjadruje jeho charakteristiky. Systém sa teda dokáže prispôsobovať jednotlivým používateľom.

Výhodou stereotypného modelu je jeho relatívne ľahká inicializácia. Stereotyp môžeme používateľovi priradiť na základe jeho odpovedí na malé množstvo otázok, prípadne podľa jeho prvotného správania sa v systéme na základe pravidiel typu: „používatelia so stereotypom X spravia ako prvú akciu Y“. Stereotypy môžu byť hierarchizované a prvotný priradený stereotyp sa môže počas používania systému používateľovi upresňovať.

Nevýhodou stereotypného modelu je prispôsobovanie sa systému skupine (čiže danému stereotypu) a nie konkrétnemu jednotlivcovi. Pritom sa nám môže ľahko stať, že sa aj v rámci jedného stereotypu budú nachádzať používatelia so značne rozličnými charakteristikami. To súvisí aj s ohraničeným počtom možných stereotypov v prípade, že ich definujeme manuálne.

Ideou prekryvného modelu je reprezentovať doménové charakteristiky pomocou doménového modelu pre každého používateľa zvlášť. Každý používateľ má vytvorenú kópiu doménového modelu, kde pre sa každý koncept dopĺňajú charakteristiky používateľa. Typickým príkladom je úroveň pochopenia obsahu konceptu.

Nevýhodou prekryvného modelu používateľa je jeho obtiažna inicializácia. Na začiatku používateľovej práce so systémom o ňom ešte nemáme dostatočný počet informácií, aby sme dokázali efektívne vykonávať prispôsobovanie, čo však neznamená, že prispôsobovanie nie je potrebné. Preto sa v praxi často používa tento prístup v kombinácii so stereotypným modelom. Pri prvom sedení používateľa sa mu priradí stereotyp (čiže vieme vykonať základné prispôsobovanie podľa skupiny) a následne sa začne napĺňať jeho vlastná inštancia modelu používateľa, ktorá bude slúžiť pre individuálne prispôsobovanie.

Problémom pri použití prekryvného modelu môže byť veľký počet používateľov systému, čo zvyšuje technické nároky na adaptívny webový systém.

Špeciálny prístup si vyžaduje rozsiahly informačný priestor, najmä ak ide o otvorený priestor, pri ktorom sa nedá vytvoriť kópia domény pre každého používateľa. Riešením je vytvárať kópiu domény, len v tých častiach, ktoré sa týkajú konkrétneho používateľa (dané koncepty už navštívil). Iným riešením je voľnejšie prepojenie modelu používateľa s modelom domény, keď sa charakteristiky nevzťahujú priamo na jednotlivé koncepty ale na ich typy, resp. vlastnosti. V otvorenej doméne pracovných ponúk teda napríklad neukladáme v modeli používateľa vzťah používateľa k jednej pracovnej ponuke, ale vzťah k vlastnostiam danej ponuky, ktoré zrejme spĺňa viacero ponúk.

2.2 Charakteristiky používateľa

Pri návrhu adaptívneho systému je potrebné zvážiť, ktoré aspekty používateľov sa budú používať pri prispôsobovaní. V modeloch adaptívnych systémoch sa zvyknú zachytávať niektoré z týchto charakteristík [13]: *znalosti*, *ciele*, *preferencie*, *záujmy*, *povahové vlastnosti*, *skúsenosti* a *zázemie používateľa*.

Znalosti sú dôležité najmä pre systémy pre podporu vzdelávania, ktoré evidujú znalosti používateľa pre každý doménový koncept. Z pohľadu domény pracovných ponúk sú dôležité najmä ciele a záujmy používateľov. Pod pojmom preferencie sa vo väčšine systémov chápu preferencie nastavenia vizuálneho vzhľadu aplikácie (farba pozadia, farba textu, rozmiestnenie textu a pod.).

Cieľ nám dáva odpoveď na otázku „Prečo sa používateľ rozhodol používať systém a čo chce dosiahnuť?“ Môžeme pritom uvažovať o celej hierarchii cieľov. V doméne pracovných ponúk je najvyšším cieľom používateľa nájsť si prácu. Nižším cieľom môže byť sformulovanie takého dopytu, ktorý používateľovi vráti pre neho zaujímavé ponuky.

2.2.1 Kategorizácia charakteristík používateľa

Charakteristiky používateľa, ktoré sú modelované v modeli používateľa môžeme rozdeliť podľa viacerých kritérií. Podľa spôsobu získania ich môžeme rozdeliť na tie, ktoré dokážeme získať automaticky a tie, ktoré môžeme získať iba s prispením používateľa. V doméne pracovných ponúk môže byť takou informáciou, ktorú v súčasnosti nedokážeme získať automaticky zoznam predchádzajúcich zamestnaní používateľa.

Charakteristiky môžeme deliť podľa ich jednoznačnosti. Niektoré charakteristiky sú už zo svojej podstaty neostré. Príkladom je preferovaný plat. To, či je nejaký plat dobrý, závisí aj od ostatných atribútov pracovnej ponuky. To znamená, že ani samotný používateľ často nevie, aký plat by si želal. Vie však zadať hodnoty, ktoré pre neho predstavujú hranice úplného zamietnutia hodnoty (takto nízky plat je už prekážkou, nech je ponuka akákoľvek) a úplného akceptovania hodnoty. Zadané hodnoty predstavujú

hranice fuzzy množiny. Iné charakteristiky môžu byť naopak úplne jednoznačné a o fuzzy prístupe pri nich nemá význam uvažovať – napríklad pohlavie.

Významným pohľadom je delenie charakteristík podľa doménovej závislosti. Doménovo nezávislé charakteristiky opisujú používateľa na všeobecnej úrovni (vek, pohlavie, vzdelanie a pod.). Tieto charakteristiky môžu využívať aplikácie z rôznych domén. Doménovo závislé charakteristiky vyjadrujú vzťah používateľa ku konceptom konkrétnej domény (oblasť výskumu, preferovaná pracovná pozícia a pod.). Použiteľnosť týchto charakteristík je viac limitovaná na jednu konkrétnu doménu.

2.2.2 Reprezentácia charakteristík

Pri charakteristikách je dôležité uvažovať o spôsobe ich reprezentácie v modeli. Niektoré modely sú navrhnuté tak, že k charakteristike sa ukladá iba jej hodnota [18] (boolovská hodnota, reťazec znakov a pod.), iné systémy si ukladajú aj zdroj, z ktorého bola charakteristika získaná [32].

Vo všeobecnosti je vhodné, ak model umožňuje uloženie ďalších informácií o charakteristike ako je len jej hodnota. Takýmito prídavnými informáciami môže byť aj relevantnosť charakteristiky pre používateľa a dôveryhodnosť charakteristiky.

Relevantnosť charakteristiky umožňuje zachytiť stav, keď používateľovi na charakteristike veľmi záleží alebo naopak ak pre neho nie je podstatná. Príkladom v doméne pracovných ponúk by mohla byt preferovaná lokalita práce. Ak by táto charakteristika mala vysokú relevantnosť, systém by mal ponuky, ktoré tejto charakteristike vyhovujú ponúkať používateľovi ako prvé. Naopak, ak by relevantnosť bola nižšia, systém túto charakteristiku nemusí brať do úvahy pri hľadaní vhodných ponúk.

Dôveryhodnosť charakteristiky súvisí so spôsobom získania charakteristiky. V [30] sa najvyššia dôveryhodnosť prisudzuje charakteristikám získaným priamo od používateľa, nižšia dôveryhodnosť je prisúdená automaticky odvodeným charakteristikám. Ešte nižšiu dôveryhodnosť môže systém prisúdiť takej charakteristike, ktorá bola odhalená iným systémom a uložená v zdieľanom modeli používateľa.

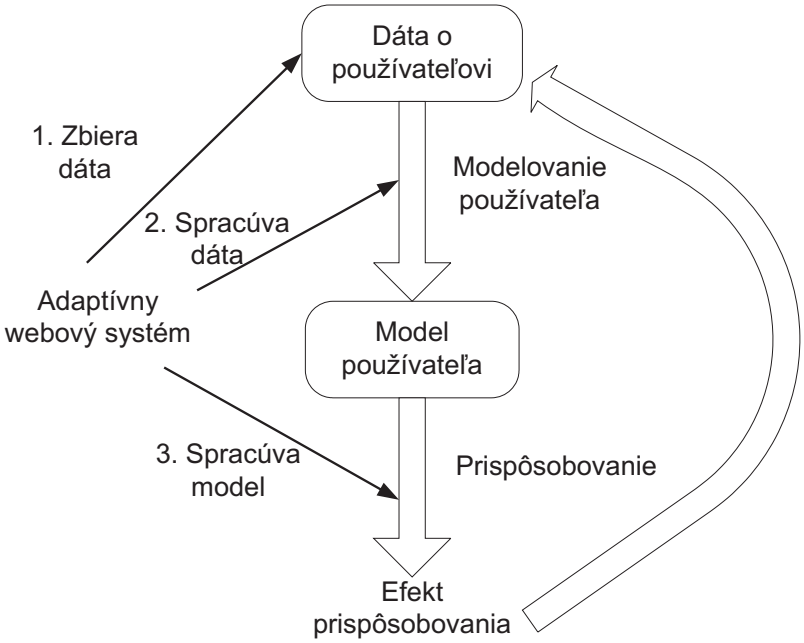
2.3 Životný cyklus modelu používateľa

Životný cyklus modelu používateľa vyjadruje postupnosť akcií vykonávaných adaptívnym webovým systémom počas používania systému používateľom. Priebeh procesu (obrázok 1, upravený podľa [13]) sa dá rozdeliť na tri fázy:

1. zber dát o používateľovi,
2. modelovanie používateľa,
3. prispôsobovanie (používanie modelu používateľa).

V prvej fáze systém zbiera dáta o používateľovi, ktoré slúžia pri vytváraní a udržiavaní modelu používateľa v druhej fáze. Vytvorený model používateľa sa v ďalšej fáze využije pre samotné prispôsobenie tak, aby sa dosiahol požadovaný efekt.

Proces ma cyklický charakter, keďže adaptívny systém po prispôsobení získa nové dáta o používateľovi, ktoré môžu spresniť budovaný model a tým znova ovplyvniť aj prispôsobovanie.



Obrázok 1: Životný cyklus modelu používateľa podľa [13].

2.4 Zber dát o používateľovi

Pri vytváraní modelu používateľa potrebujeme o používateľovi informácie, ktoré môžeme získať z viacerých zdrojov, napr. cez formulár, dodaný životopis. Často sa používa aj kombinácia viacerých prístupov.

2.4.1 Získanie informácií cez vyplnený formulár

Získanie informácií systémom formulárov je základný prístup využívaný väčšinou adaptívnych systémov. To, čo systém potrebuje o používateľovi vedieť sa jednoducho používateľa opýta cez sadu formulárov. Často sa pomocou formulárov získajú aspoň základné informácie o používateľovi (obrázok 2).

Výhodou tohto prístupu je možnosť získať prakticky ľubovoľné informácie o používateľovi – teda aj také, ktoré sa automaticky získať nedajú.

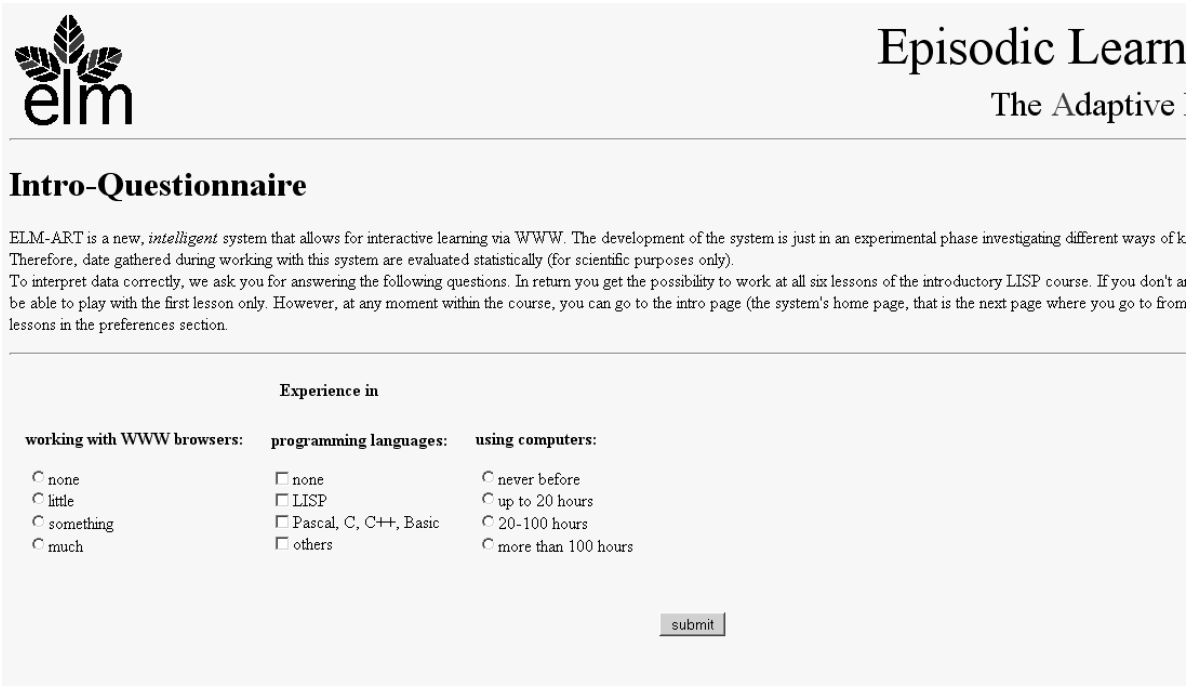
Ďalšou výhodou formulárového prístupu sa zdá byť vysoká vierohodnosť získaných informácií, keďže zdroj informácií je zároveň naším objektom záujmu. Môže však nastať situácia, keď používateľ nevie presnú odpoveď na položené otázky, nevie vyjadriť hodnoty niektorých svojich charakteristík.

Ďalšou komplikáciou môže byť používateľovo precenenie alebo podcenenie svojich charakteristík, ktoré neodrážajú realitu. Ak by sme sa napríklad používateľa opýtali nakoľko ovláda určitý programovací jazyk, používateľ si môže priradiť vynikajúci stupeň znalosti daného jazyka aj keď to nie je pravda.

V neposlednom rade je dôležité si uvedomiť, že táto metóda zberu dát núti používateľa trávit veľa času činnosťami, na ktoré systém nie je primárne určený a ktoré neboli dôvodom jeho návštevy systému.

2.4.2 Analýza predložených dokumentov

Tento prístup je založený na tom, že používateľ systému poskytne dokumenty, ktoré s ním nejakým spôsobom súvisia. Takýmto dokumentom môže byť napríklad životopis, z ktoré-



Obrázok 2: Príklad časti registračného formulára adaptívneho systému ELM-ART (`apsymac33.uni-trier.de:8080/Lisp-Course`).

ho systém dokáže zistiť používateľov vek, vzdelanie a predchádzajúce zamestnania alebo článok zaslaný na nejakú vedeckú konferenciu, z ktorého sa dá získať informácia o používateľovej doméne (napríklad informatika alebo biológia) a aj bližšie o používateľovom zameraní v danej doméne (napríklad modelovanie používateľa alebo výskum DNA).

Analýza využíva metódy extrakcie dát zvyčajne založené na štruktúre predložených dokumentov, regulárnych výrazoch a ich vzájomných vzťahoch [37]. Po jej ukončení je vhodné vzhľadom na nedeterministickú povahu analýzy zobraziť zistené charakteristiky používateľovi na kontrolu.

2.4.3 Sledovanie prístupu k zdrojom

Sledovanie prístupu k zdrojom spočíva v evidovaní prístupu používateľa ku konceptom. Typickým príkladom použitia sú adaptívne systémy pre vzdelávanie. Napríklad systém AHA! [18] vytvára dva záznamy pre každý prístup používateľa na stránku: jeden záznam pre začiatok a druhý pre koniec intervalu, počas ktorého systém predpokladá zobrazenie stránky používateľovi. Z tejto informácie potom adaptívny systém odvodí vedomosť používateľa o obsahu zobrazeného konceptu. Teda, ak sa používateľovi koncept zobrazil, systém predpokladá, že používateľ si ho prečítal a pochopil. Toto zjednodušenie nemusí byť vždy pravdivé, ale často jeho použitie dáva pri intenzívnom používaní systému dobré výsledky.

Sledovanie prístupu k zdrojom sa dá použiť aj pre iné domény. Napríklad pre doménu vedeckých publikácií by sme získali záznamy o tom, ktoré publikácie si používateľ zobrazil. Otázna je však interpretácia týchto záznamov, ktorá je určite náročnejšia ako v prípade systémov pre vzdelávanie. Môžeme totiž predpokladať, že používateľ si počas hľadania vhodných príspevkov prezrie aj také, ktoré mu nevyhovujú. V tom prípade nám obyčajný zoznam zobrazených publikácií nedáva plnohodnotnú informáciu, ktorú môžeme ďalej použiť v procese odhaľovania charakteristík. Potrebujeme poznať spôsob používateľovej navigácie k výsledkom, prípadne spätnú väzbu k jednotlivým položkám.

2.4.4 Sledovanie správania používateľa

Tento prístup je rozšírením predchádzajúceho, keď sa zaznamenáva nielen prístup k zdrojom, ale komplexné správanie sa používateľa v rámci systému. Eviduje sa čas, ktorý používateľ strávil prezeraním stránok, spôsob navigácie po webovom sídle, sekvencie kliknutí, prechody náhľad – detail, čítanie obsahu stránok (posúvanie po stránke - *scroll, page-down*), prípadné využívanie aktívnych prvkov prezeraných stránok (napríklad *hover*) či využívanie prídavnej funkcionality stránky (napríklad zaradenie pracovnej ponuky medzi obľúbené).

Tieto dáta sa následne môžu použiť pre odhad charakteristík používateľa. Z ich povahy vyplýva, že sa tento prístup obzvlášť hodí pre systémy s otvoreným a meniacim sa informačným obsahom a nelineárnym navigačným modelom. V prípade rôznych elektronických kurzov, ktoré sa snažia používateľovi „nanútit'" najvhodnejšiu cestu kurzom nemusí tento spôsob viest' k dostatočne zaujímavým dátam. V správaní používateľa sa totiž nemusia v dostatočnej miere prejavit' jeho charakteristiky, keďže väčšinou sleduje odporúčanú cestu [36].

Tento prístup získavania dát je vhodný z hľadiska úrovne zapojenia používateľa do procesu. Používateľ robí presne to, kvôli čomu sa rozhodol systém používat' (napr. si hľadá prácu) a celý proces zberu dát prebieha v pozadí, bez potreby jeho zásahu.

Väčšina systémov zbiera takýto druh dát na strane servera vo forme log záznamu. Z toho plynie nemožnost' evidencie tých akcií používateľa, ktoré sa na server nedostanú. Typickým príkladom je použitie *back* tlačidla webového prehliadača, ktoré zobrazí používateľovi predposlednú navštívenú stránku z vyrovnávacej pamäte prehliadača bez toho, aby sa klient pripájal znovu na server. Záznam na strane servera teda neobsahuje presný čas, kedy používateľ danú stránku opustil. Iným príkladom je interakcia používateľa s aktívnymi prvkami stránky, pri ktorých sa taktiež nekomunikuje so serverom.

Ak teda chceme zaznamenávat' kompletné správanie používateľa, nevystačíme si so záznamami na strane servera a potrebujeme určité akcie používateľa zaznamenávat' aj na strane klienta.

Výhodou zbierania dát na strane servera je však záruka ich získania. Server je pod kontrolou prevádzkovateľa systému a teda môžeme mat' istotu, že dáta získame. Naopak, na strane klienta nevieme kontrolovat' spustenie zaznamenávania akcií.

Zaznamenávanie akcií používateľa pre účely budovania modelu používateľa je vo svojej podstate rovnaké ako zaznamenávanie akcií pre účely vyhodnocovania rozhraní webových stránok. V práci [43] sa autori venujú analýze interakcie používateľov s webovými aplikáciami. Konštatujú, že pre účely vyhodnocovania stránok je získavanie dát na strane servera neefektívne, pretože neposkytuje potrebnú detailnost'. Potvrdzujú, že pre získanie podrobných dát o používaní webovej aplikácie je potrebné vykonávat' monitorovanie aj na strane klienta. Toto poznanie sa potvrdzuje aj v prácach z komunity adaptívnych systémov [38, 15, 24, 33].

Pre účely vyhodnocovania stránok boli vyvinuté viaceré nástroje, ktoré podporujú monitorovanie akcií na strane klienta. Takými nástrojmi sú napríklad WebVIP[3] , WET [23] alebo UAR [53]. WebVip (Web Variable Instrumenter Program) je nástroj, primárne určený na podporu testovania používateľov počas návštevy stránky. Umožňuje nastavenie akcií, ktoré chceme zaznamenávat' a vygeneruje skripty v klientskej technológii JavaScript, ktoré vloží do cieľových stránok. Tieto, na klientovi spúšt'ané, skripty zabezpečia vytvorenie záznamu vo formáte FLUD (Framework for Logging Usability Data). WebVip musí v každej HTML stránke webového sídla zmodifikovat' všetky značky predstavujúce komponent používateľského rozhrania a teda potrebuje mat' k dispozícii pri svojom

[3]WebVip, `http://zing.ncsl.nist.gov/WebTools/WebVIP/overview.html`

spustení kópiu celého sídla, čo je veľká nevýhoda tohto nástroja.

Nástroj WET (Web Event-logging Tool) taktiež využíva technológiu JavaScript na vytváranie záznamov na strane klienta. Pri jeho použití však stačí pridať odkaz na súbor so skriptom do hlavičky každej stránky, čiže nie je potrebné dodať nástroju celú kópiu webového sídla ako pri nástroji WebVip. WET pracuje na princípe spracovávania udalostí, ktoré sú generované prehliadačom. Nevýhodou takéhoto prístupu je fakt, že to, čo chápeme ako jednu akciu používateľa (interakciu) vyvolá v prehliadači celú sériu udalostí. Napríklad jednoduchá interakcia s odkazom na stránke vyvolá nasledovné udalosti: *mouseover*, *mousedown*, *mouseup* a *click*. To môže viesť k enormne veľkému množstvu dát, ktoré sa musí následne odoslať na server a spracovať. Tvorcovia preto do nástroja zakomponovali možnosť nastaviť explicitne druhy akcií, ktoré má nástroj zaznamenávať. Môžeme teda sledovať napríklad iba kliknutia, zmeny objektov, načítania stránok.

Problémom nástroja WET je jeho prílišná zviazanosť s úlohami vyhodnocovania stránok. Autori doň zakomponovali aj funkcionalitu, ktorá vytvorí dodatočné ovládacie prvky nad zobrazenou stránkou, ktorými sa riadi proces monitorovania.

Nástroj UAR (User Action Recorder) je klasickou samostatnou aplikáciou pre prostredie Windows. Dá sa ním monitorovať nielen interakcia používateľa s webovou aplikáciou, ale prakticky kompletná práca používateľa s počítačom. Nástroj monitoruje používanie klávesnice a počítačovej myši v jednotlivých otvorených oknách. Práve prakticky neobmedzené možnosti monitorovania predstavujú veľkú nevýhodu tohto prístupu z hľadiska ochrany súkromia používateľa.

2.5 Analýza a spracovanie dát o používateľovi

Po fáze zberu dát je ďalšou fázou spracovanie týchto dát a budovanie samotného modelu používateľa. Budeme sa zaoberať možnosťami spracovania automaticky získaných dát o správaní používateľa, keďže napojenie ostatných zdrojov dát spomenutých v predchádzajúcej kapitole na model používateľa je priamočiare.

2.5.1 Analýza dát dolovaním

V súčasnosti sa väčšina metód automatizovaného spracovania dát o používaní webu orientuje na techniky získavania znalostí dolovaním v dostupných dátach [28] v prostredí webu označované ako *Web Usage Mining* [22]. Ako zdroj dát slúži záznam webového servera na úrovni HTTP protokolu, z ktorého sa získavajú znalosti pomocou metód zhlukovania, klasifikácie a objavovania asociácií alebo sekvenčných vzorov. Takto získané znalosti slúžia pri zlepšovaní návrhu webových stránok alebo pri analýze ich návštevnosti. Vzhľadom na sociálny aspekt použitých techník, keď sa mapuje aktuálne sedenie používateľa na vzory skupiny používateľov nemôžeme uvedené techniky použiť priamo na získanie charakteristík jednotlivca.

Spomínané techniky (konkrétne dolovanie spojitých sekvenčných vzorov) však môžeme použiť ako zdroj prídavnej informácie o charakteristikách získaných iným spôsobom. Pri vhodne zvolených dátach (iba od jedného používateľa) nám totiž dávajú informáciu o tom, či sa používateľ správa počas aktuálnej návštevy webového sídla rovnakým spôsobom ako pri predchádzajúcich sedeniach. Jedná sa teda o informáciu o konzistentnosti správania používateľa vzhľadom na jeho predchádzajúce prístupy. Dokážeme tak sledovať krátkodobé zmeny v správaní používateľa, keď napríklad hľadá pracovné ponuky pre tretiu osobu (vtedy môžeme znížiť váhu, ktorou aktuálne sedenie prispieva do modelu používateľa) ako aj dlhodobejšie (trvalé) zmeny v správaní používateľa, keď sa menia jeho záujmy a informačné potreby. Dôležitým faktorom pri rozhodovaní, či sa jedná o krátko-

dobú alebo trvalú zmenu v správaní je informácia o čase, ktorý ubehol medzi poslednými dvoma sedeniami používateľa.

Znalosti získane pomocou techník dolovania v dátach sa ďalej môžu použiť ako riešenie *cold start* [39] problému, keď začne systém používať nový používateľ, ktorého charakteristiky ešte nie sú známe. Na základe vzorov sa môže používateľ rýchlo zaradiť do určitého zhluku používateľov – pridelí sa mu stereotyp na základe ktorého sa môže vykonávať prvé prispôsobovanie. Získané znalosti sa môžu využiť aj pri kolaboratívnom filtrovaní [46] a odporúčaní ďalšej navigácie po webovom sídle [36].

2.5.2 Spätná väzba

Všeobecne uznávaným prístupom k analýze dát o používaní webových systémov z hľadiska usudzovania o používateľovi je využívanie spätnej väzby. Tú delíme na explicitnú a implicitnú.

Explicitná spätná väzba. Explicitnú spätnú väzbu získava systém vtedy, keď používateľ explicitne vyjadrí svoj vzťah k zobrazenému obsahu. To je zvyčajne vo forme ohodnotenia obsahu v škále danej stupnice.

Explicitná spätná väzba sa vo všeobecnosti považuje za dobrý zdroj používateľových preferencií a dá sa ľahko identifikovať v zozbieraných dátach. V niektorých prípadoch má však používateľ problém zadať spätnú väzbu. Napríklad môže mať problém ohodnotiť ponuku na danej stupnici, prípadne nevyužíva celú škálu ponúkanej stupnice.

Pri využívaní explicitnej spätnej väzby vo forme ohodnotenia obsahu používateľom je potrebné používateľa motivovať, aby systému takúto spätnú väzbu poskytoval. Používateľ musí vidieť jasný zmysel ohodnocovania.

Implicitná spätná väzba. Keďže získavanie explicitnej spätnej väzby je nespoľahlivé (používateľ nevyplní formulár a pod.), výskum sa orientuje aj na využívanie implicitnej spätnej väzby. Z dát o používaní systému sa odvodí taká istá informácia ako keby používateľ poskytol explicitnú spätnú väzbu.

V [41] sú uvedené pozorovateľné správania, z ktorých sa dajú vyťažiť poznatky o používateľovi. Tieto správania sú zaradené do troch kategórií: preskúmavanie, uchovanie a odkazovanie. Z pohľadu webových systémov je najpodstatnejšia kategória preskúmavanie, do ktorej patria nasledovné pozorovateľné aspekty správania:

- Výber – informačné systémy často ponúkajú stručné zhrnutia viacerých konceptov na jednej stránke. Výber jedného konceptu na bližšie preštudovanie poskytuje určitú informáciu o záujmoch používateľa;
- Trvanie – keďže bola zistená pozitívna korelácia medzi časom čítania a explicitným ohodnotením v USENET aplikáciách, môžeme trvanie preskúmavania konceptu pokladať za ďalší aspekt určujúci záujmy používateľa;
- Opakovanie – hľadáme opakujúce sa správanie v preskúmavaní informačného obsahu portálu;
- Kúpa/zaplatenie/prihlásenie – rozhodnutie používateľa objednať si, predplatiť, prihlásiť sa a pod. vzhľadom na skúmaný koncept je silným indikátorom pozitívnej spätnej väzby na daný koncept.

Príkladom môže byť práca [14], v ktorej sa autori venovali modelovaniu používateľa pre vytváranie personalizovaných správ o priebehu návštevy v múzeu. Počas návštevy má

každý návštevník k dispozícii zariadenie – elektronického sprievodcu exponátmi múzea. Implicitnú spätnú väzbu predstavuje stláčanie tlačidiel „Viac“ (pozitívna spätná väzba) alebo „Dost’“ (negatívna spätná väzba) počas výkladu. Stlačením tlačidla „Viac“ sa používateľ dostane k ďalším, podrobnejším informáciám o koncepte a zároveň týmto využívaním funkcionality oznamuje časti systému zodpovednej za tvorbu a aktualizáciu modelu používateľa svoje preferencie. Stlačením tlačidla „Dost’“ používateľ zastaví prebiehajúci výklad. Vedľajším efektom je informácia pre systém o tom, čo ho veľmi nezaujalo. Tlačidlo „Viac“ predstavuje *Výber* z uvedenej kategorizácie podľa [41]. Ďalšou implicitnou spätnou väzbou je neprerušenie prezentácie a jej vypočutie až do konca alebo použitie tlačidla „Dost’“ (*Trvanie*) prípadne návrat návštevníka k už videnému exponátu múzea (*Opakovanie*).

Teóriu implicitnej spätnej väzby využila vo svojej práci aj Rachael Rafter v projekte Casper [48] na doméne pracovných ponúk. Na odvodenie implicitnej spätnej väzby využíva tri metriky: znovu-navštívenie ponuky, čas čítania ponuky a aktivita s ponukou (prihlásenie sa o prácu, zaslanie ponuky e-mailom). V práci sa kladie dôraz na vyčistenie dát od nechcených javov, ako je viacnásobné kliknutie na jednu ponuku, ktoré sa nemá počítať ako znovu-navštívenie ponuky a úprava časov čítania ponúk vzhľadom na priemerný čas čítania ponúk používateľom.

Problematike analýzy správania sa používateľov na webe sa venuje aj [33]. Autori skúmajú časové súvislosti (implicitnú spätnú väzbu *Trvanie*) pri používaní elektronického kurzu, v ktorom majú študenti možnosť počúvať prednášky a prezerať si poznámky.

2.5.3 Spracovanie spätnej väzby

Existuje viacero možností, ako využiť ohodnotenie obsahu používateľom (či už explicitné alebo implicitné). Najčastejšie sa táto informácia používa pri odporúčaní ďalšieho informačného obsahu, resp. filtrovaní obsahu. Pre tieto účely existuje viacero predikčných (filtrovacích) techník, ktoré sa delia na dve hlavné skupiny: obsahovo závislé a obsahovo nezávislé [46].

Obsahovo závislé techniky pracujú s obsahom a štruktúrou spracovávaných informácií a potrebujú navzájom porovnať dve rôzne entity (napr. dokumenty, ontologické inštancie), pričom nemusí ísť vždy o entity rovnakého typu. Porovnávanie môže byť realizované na rôznych úrovniach abstrakcie, napríklad filtrovanie pomocou kľúčových slov porovnáva vektorovú reprezentáciu dokumentov s vektorovou reprezentáciou modelu používateľa (pre ohodnotenie dokumentu sa jednoducho porovnajú tieto dva vektory s využitím kosínovej miery). Pri odporúčaniach sa entity navzájom porovnávajú, aby sa odhadlo ohodnotenie, ktoré používateľ ešte nevykonal na základe predchádzajúcich hodnotení.

Obsahovo nezávislé techniky (kolaboratívne, sociálne techniky) sú založené na základe hodnotení používateľov s podobným vkusom. Nepotrebujú poznať a rozumieť samotnému obsahu, pretože pracujú iba s hodnoteniami jednotlivých používateľov. Pre každého používateľa sa určí množina podobných používateľov pomocou korelačného koeficientu a následne sa pre odporúčanie využívajú hodnotenia používateľov z tejto množiny.

2.6 Príklady nástrojov na tvorbu modelu používateľa

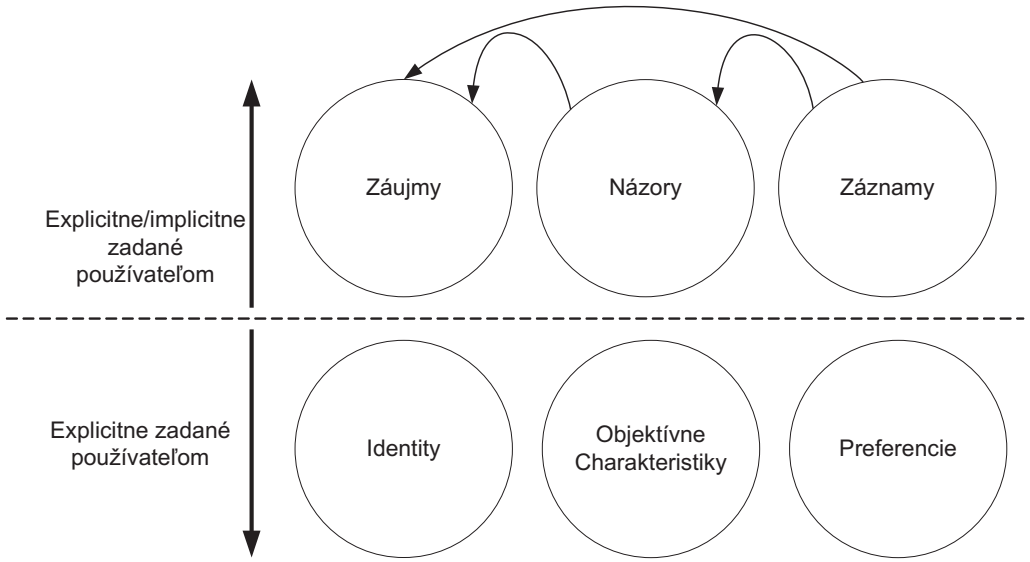
V tejto podkapitole uvádzame príklady existujúcich nástrojov (komponentov) na modelovanie používateľa. Pri výbere sme sa sústredili na také riešenia, ktoré nie sú napevno zviazané so žiadnym existujúcim systémom a predstavujú všeobecné riešenie problému budovania modelu používateľa.

2.6.1 Duine

Duine[4] [51] je nástroj, ktorý zabezpečuje modelovanie používateľa na základe spätnej väzby. Tento nástroj umožňuje vývojárom využívať predikčné mechanizmy pre odporúčacie systémy v ich vlastných aplikáciách, ktoré tak získajú schopnosť odporúčať používateľovi obsah na základe predikcie jeho záujmu o tento obsah.

Duine implementuje niekoľko predikčných metód z obidvoch skupín – obsahovo závislých (*GenreLMS* [50], *Case-based Reasoning*, *Information Filtering*) aj obsahovo nezávislých (*User Average*, *TopN Deviation*, *Social Filtering*, *Already Known*). Pre obsahovo závislé techniky je definovaný obaľovač obsahu, pomocou ktorého dokážu predikčné metódy nástroja Duine pracovať s obsahom ľubovoľného systému. Predikčné techniky sa môžu ľubovolne reťazit' do predikčných stratégií.

Z pohľadu modelovania používateľa je podstatné to, že nástroj si sám udržuje model používateľa (obrázok 3). Tento model sa skladá z dvoch častí. Jednu časť tvoria tie aspekty o používateľovi, ktoré sa nedajú zistiť automatizovane. Informácie z druhej časti modelu sa dajú získať buď priamo od používateľa alebo implicitne.



Obrázok 3: Časti modelu používateľa vytváraného nástrojom Duine.

Identity v modeli slúžia na prepojenie viacerých aliasov (prihlasovacích mien) jedného používateľa. *Objektívne charakteristiky* predstavujú také informácie ako napríklad meno, pohlavie alebo emailová adresa používateľa. *Preferencie* zachytávajú používateľovo nastavenie výzoru aplikácie. *Záznamy* obsahujú všetky akcie používateľa počas používania systému. *Názory* sú hodnotenia jednotlivých jednotiek informačného obsahu. *Záujmy* vyjadrujú mieru záujmu používateľa o určité koncepty.

Najmä techniky založené na poznaní obsahu môžu do modelu zaznamenávať zistené doménové charakteristiky používateľa. V modeli sa môžu objaviť záujmy používateľa o jednotlivé typy informácií. Problematická je interpretácia (a prípadná validácia) modelu inak, ako cez prostriedky nástroja *Duine*. To je spôsobené aj zvolenou reprezentáciou modelu pomocou tabuliek relačnej databázy.

[4]Duine, `http://sourceforge.net/projects/duine`

2.6.2 BGP-MS

BGP-MS [35] patrí do skupiny tzv. *User Modeling Shells*, teda znovupoužiteľných komponentov pre modelovanie používateľa [34], ktoré umožňujú softvérovému systému prispôsobovať sa. Obsahuje niekoľko metód pre odovzdávanie informácií o pozorovaniach týkajúcich sa používateľa a prijímaní aktuálne platných domnienok o používateľovi. Skladá sa z viacerých modulov, pričom vývojár systému môže určiť, ktoré sa majú používať a prípadne ich nakonfiguruje pre prácu v konkrétnej doméne.

BGP-MS sa zvyšku systému javí ako čierna skrinka s ktorou systém komunikuje pomocou výmeny správ. Systém môže posielať nástroju *BGP-MS* správy informujúce o používateľových cieľoch, presvedčeniach či vykonaných akciách. Nástroj dokáže aktivovať používateľovi príslušný stereotyp a v prípade, že podmienky pridelenia stereotypu prestanú platiť, ho automatizovane odobrať. *BGP-MS* obsahuje taktiež modul, ktorý dokáže vygenerovať a zaslať systému otázky, ktoré sa majú položiť používateľovi a následne vie spracovať odpovede do internej reprezentácie modelu používateľa.

Pri analýze akcií používateľa *BGP-MS* používa predpripravené typy dialógov (interakcie používateľa so systémom), ktoré majú definované predpoklady [45]. V prípade, že nastane dialóg známeho typu, vytvorí sa inštancia zodpovedajúca vzoru predpokladu.

Nevýhodou *BGP-MS* je nízka zdieľateľnosť vytváraného modelu používateľa. Vzhľadom na zvolený spôsob komunikácie musia byť *BGP-MS* a systém, ktorý využíva jeho služby spustené pod tým istým operačným systémom. Model teda môže využívať iba jedna aplikácia, čo je podmienené aj proprietárnou reprezentáciou modelu používateľa.

2.7 Uplatnenie modelu používateľa pri prispôsobovaní

Keď máme vytvorený model používateľa môžeme prejsť k samotnému prispôsobovaniu. Podľa [31] existujú tri hlavné spôsoby uplatnenia modelu používateľa pri interakcii používateľa so systémom:

- interpretácia vstupu od používateľa – v prípade, že vstup od používateľa je viacznačný, neurčitý, môže model umožniť systému jednoznačnú interpretáciu vstupu;
- úprava výstupu systému tak, aby bol vhodný pre používateľa – môže ísť o nastavenie jazyka prezentácie, veľkosti a typu písma, počtu položiek na jednej obrazovke ale aj o zmeny v samotnom obsahu či prispôsobení odkazov ktorými sa definuje navigácia po webovom sídle;
- vedenie interných akcií systému – systém vykonáva svoje interné procedúry v súlade so znalosťami uloženými v modeli používateľa (napríklad personalizované filtrovanie a vyhľadávanie informácií [47]).

Každá akcia systému, pri ktorej sa využíva model používateľa, vyvoláva nové reakcie používateľa, spätnú väzbu. Jednotlivé fázy procesu prispôsobovania (zber, analýza, prispôsobovanie) prebiehaju nepretržite. To vedie k neustálemu spresňovaniu a zmenám odhadov o charakterakteristikách používateľa.

2.8 Reprezentácia modelu používateľa

V adaptívnych webových systémoch existuje viacero možných reprezentácii modelu používateľa [1]. Tieto reprezentácie sa líšia v úrovni expresivity a flexibility, ktoré poskytujú, ako aj v možnostiach ich zdieľania medzi viacerými aplikáciami a ďalšej práce s reprezentovanými charakteristikami.

Relačná databáza. Často využívaným spôsobom reprezentácie modelu používateľa sú relačné databázy [10]. Väčšina informačných systémov tento typ dátového zdroja používa pre uloženie doménových informácií a tak pridanie modelu používateľa neprináša zvýšené nároky. Profil používateľa je vyjadrený entitno-relačným modelom, ktorému zodpovedá sada poprepájaných tabuliek v databáze.

Aj keď je použitie databáz priamočiare a prináša hneď niekoľko výhod (výkonnosť, bezpečnosť, celková vyspelosť technológie) nemusí byť najvhodnejšou reprezentáciou modelu používateľa pre webový informačný systém. Databázy sa nehodia pre reprezentáciu semi-štruktúrovaných dát, čo býva častý prípad modelu používateľa, ktorý musí zachytiť množstvo rôznorodých charakteristík vo vzťahu k modelu domény.

XML. Ďalším často používaným prístup je reprezentácia modelu pomocou XML štandardov (napr. v systéme AHA! [20]). XML poskytuje dostatočnú expresivitu pričom charakteristiky sú uložené ako hodnoty značiek, prípadne ako ich atribúty.

Keďže vybudovanie modelu používateľa nie je triviálne, existujú snahy vyvinúť mechanizmy pre ich zdieľanie medzi viacerými aplikáciami [11, 32]. Tu obidva spomínané prístupy reprezentácie zlyhávajú. Databázové riešenia sú platformovo závislé a pre ich efektívne zdieľanie potrebujú aplikácie presne poznať použitý entitno-relačný model (databázovú schému). Riešenia založené na XML jazykoch sú síce platformovo nezávislé a pripravené pre použitie na webe, ale bez definovania spoločného slovníka a pravidiel zápisu je zdieľanie medzi aplikáciami prakticky nerealizovateľné.

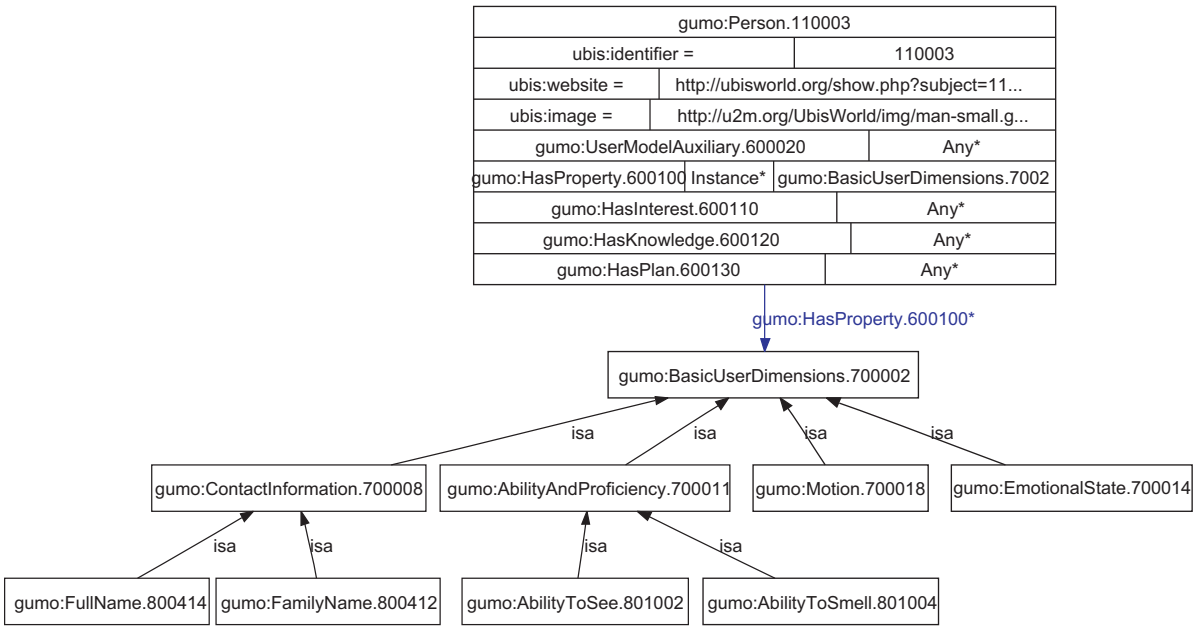
Ontológia. Ontológia je v informatike najčastejšie definovaná ako „explicitná formálna špecifikácia zdieľanej konceptualizácie“ [52]. Konceptualizácia sa pritom chápe ako formálne reprezentovaný abstraktný, zjednodušený pohľad na svet. Pretože pojem ontológia zahŕňa celý rad rôznych modelov s rôznym stupňom sémantickej bohatosti a zložitosti [42], špecifikujeme bližšie, že pod ontológiou budeme myslieť model zapísaný pomocou jazyka OWL[5], ktorý vychádza z jazyka RDF[6]. Ontológie zapísané pomocou jazyka OWL tvoria ontologickú vrstvu webu so sémantikou [44] a tento jazyk zohráva kľúčovú úlohu pri postupnom realizovaní tejto vízie [9].

Jazyk OWL patrí do rodiny XML jazykov, spĺňa teda základné predpoklady pre zdieľateľnosť modelu. Spolu s RDF (presnejšie RDF Schema) definujú spôsob a slovník, akým sa v tomto jazyku opisujú zdroje – základným konceptom je trieda, ktorá má vlastnosti, inštancie, vzťahy s inými triedami a vlastnosťami.

Príklad modelov používateľa reprezentovaných ontológiou

V súčasnosti sa jazyky OWL a RDF začínajú uplatňovať v čoraz väčšom počte projektov týkajúcich sa adaptívnych webových systémov. Okrem modelovania domény sa používajú aj na modelovanie používateľov týchto systémov, pričom sa prirodzene hľadá také vyjadrenie modelu, ktoré umožňuje čo najväčšiu znovupoužiteľnosť a zdieľanie modelu. Priamo zdieľané modely zjednodušujú znovupoužitie znalostí o používateľovi viacerými aplikáciami, nedá sa však predpokladať, že v dynamickom prostredí webu vznikne jeden univerzálny model, ktorý budú akceptovať všetky adaptívne webové systémy. Aj preto v sebe jazyk OWL obsahuje prostriedky pre mapovanie ontológií (`owl:equivalentClass`, `owl:equivalentProperty`).

[5] OWL, Web Ontology Language, `http://www.w3.org/2004/OWL/`
[6] RDF, Resource Description Framework, `http://www.w3.org/RDF/`



Obrázok 4: Vizualizácia vybraných častí ontológie GUMO. Základný koncept *Person* je napojený na rozsiahlu klasifikáciu základných dimenzií používateľa.

GUMO. General User Model Ontology (GUMO) [29] je snahou o vytvorenie všeobecne akceptovanej ontológie pre modelovanie používateľa. Ontológia GUMO je vyvíjaná v rámci projektu *Ubiquitous User Modeling*[7], čo významne ovplyvnilo aj stavbu ontológie (obrázok 4). Autori sa sústredili na vybudovanie pomerne rozsiahlej taxonómie tried modelujúcich koncepty aktuálneho kontextu používateľa (napr. aktuálny pohyb alebo emočný stav). V modeli sú takisto aj triedy pre zachytenie statických vlastností používateľa (meno a pod.) a vlastností ako sú záujmy či ciele používateľa.

Ontológia však takmer vôbec nepredpisuje vlastnosti jednotlivých tried ani relácie medzi jednotlivými triedami. Takisto autori aktuálne (model sa stále vyvíja) nevyužívajú podmienky a reštrikcie, ktoré ponúka jazyk OWL. GUMO má ambíciu stať sa tzv. *top-level* (*upper*) ontológiou v oblasti modelovania používateľa, ktorá slúži na zjednotenie pojmov. V jednotlivých špecializovaných modeloch sa môže nadefinovať mapovanie na triedy ontológie GUMO.

Model používateľa v systéme OntoAIMS. Systém OntoAIMS (Ontology-based Adaptive Information Management System) poskytuje prostredie pre vyhľadávanie a navigáciu v informáciách, ktoré odporúča používateľom (študentom) najvhodnejšie úlohy na riešenie a pomáha im pri objavovaní domény [21]. Systém predstavuje typický príklad využitia ontológií, ktorými reprezentujú doménu a používateľa. Model používateľa, vytváraný samostatným komponentom *OWL-OLM*, prekrýva doménový model a pridáva k nemu používateľovu predstavu o konceptualizácii. Prístup je výhodný, keďže informačný priestor definovaný doménou (výučba konceptov OS Linux) je uzavretý a nemení sa príliš často. Konceptuálny model používateľa, ktorý systém buduje na základe interaktívneho dialógu s používateľom, obsahuje o koncepte údaje ako počet použití, počet správnych použití, počet prípadov, kedy používateľ vyhlásil, že danému konceptu rozumie a pod. Tento konceptuálny model obohatený o údaje súvisiace s používateľom sa následne používajú pri odporúčaní obsahu.

[7] Ubiquitous User Modeling, `http://www.u2m.org/`

Zhodnotenie reprezentácií modelu používateľa

Z uvedených troch spôsobov reprezentácie modelu predstavuje ontológia najvhodnejšiu alternatívu. Základná koncepcia ontologického modelovania je blízka ľudskému mysleniu, čo umožňuje prizvať do procesu modelovania domény a súvisiaceho používateľa aj expertov danej domény. Jazyk OWL (OWL-Lite, -DL, -FULL) je dostatočne expresívny jazyk, ktorým dokážeme zachytiť a namodelovať aj zložité vzťahy vyskytujúce sa v realite. Ontológie predstavujú základ pre usudzovacie mechanizmy, ktoré dokážu nielen overiť konzistentnosť ontológie, ale odvodiť aj nové vzťahy na základe podmienok a ohraničení, ktoré tvoria inherentnú časť ontológie. Ďalšou výhodou ontologickej reprezentácie oproti zvyšným dvom spôsobom je reálna zdieľatelnosť modelu, podporená aj mapovacími konštrukciami jazyka OWL.

Nevýhodou ontologickej reprezentácie je v súčasnosti najmä nedostatok nástrojov a technológií (editorov, mapovačov, úložísk a pod.), ktoré by poskytovali aspoň rovnakú efektivitu práce ako je to pri relačných databázach alebo XML súboroch.

3 Zhodnotenie súčasného stavu a ciele práce

Prístupov k riešeniu problematiky budovania modelu používateľa je viacero, od generických riešení v podobe samostatných konfigurovateľných nástrojov a komponentov až po úzko špecializované riešenia aplikovateľné pre konkrétnu doménu a implementáciu. Prístupy sa líšia aj v úrovni podrobnosti, ktorú produkovaný model poskytuje.

Populárny spôsob získavania znalostí o používateľovi je využitie dotazníkov, testov a dialógov s používateľom pre počiatočné nastavenie modelu v kombinácii so záznamami webového servera pre získanie základných dát o pohybe používateľa na webovom sídle.

Záznamy webového servera postačujú iba pre analýzu na nižšej úrovni podrobnosti, keď je pre analýzu podstatná najmä informácia, či o daný koncept používateľ prejavil záujem, resp. koncept bol zobrazený používateľovi (čo častokrát postačuje pre odvodenie požadovaných charakteristík). V prípade, že sa požadujú podrobnejšie dáta, rieši sa zber dát pomocou špecializovaných prístupov. Častým je použitie špecializovanej aplikácie na strane klienta, prípadne použitie Java appletov. V jednotlivých riešeniach týkajúcich sa webových systémov však mnohokrát nie je objasnený spôsob, akým systém získava dáta o používateľoch a ako ich reprezentuje. V [49] autori naznačujú použitie ontológie záznamov (*Log Ontology*), ktorá definuje sémantiku akcií používateľa, neuvádzajú však detaily.

Reprezentácia dostupných dát o používateľovi a s nimi súvisiaca sémantika je teda daná implicitne a je pevnou súčasťou analytickej časti väčšiny prístupov. Tá býva často zúžená do polohy udržiavania štatistík o používateľovi v modeli (napr. koľkokrát používateľ navštívil koncept, koľkokrát správne použil koncept, ako ohodnotil koncept). V modeli používateľa sa teda ukladajú fakty, ktoré síce nepredstavujú priame charakteristiky používateľa, ale môžu byť využité pri prispôsobovaní v závislosti od prispôsobovacích pravidiel.

V prípade, že analýza dáta spracúva vo väčšom rozsahu, riadi sa väčšinou pevne danými (často implicitnými) pravidlami, aplikovateľnými na danú doménu. Často používaným princípom je odhad záujmu používateľa na základe času, pričom sa kladie menší dôraz na samotnú navigáciu. Zistené charakteristiky sú väčšinou ukladané bez prídavných informácií, ktoré by hovorili o dôveryhodnosti odhadu. To nie je závažný nedostatok pri uzavretých a dobre definovaných doménach, kde často vopred poznáme cieľ používateľov (napr. v prípade adaptívnych kurzov s konkrétnou témou je cieľ získať poznatky z danej témy), pretože charakteristiky sú všeobecne odhadnuté dobre. V rozsiahlych a otvorených informačných doménach, kde je predpoklad vyššej rôznorodosti používateľov a ich charakteristík a cieľov je však vhodné pri odhadovaní charakteristík brať do úvahy aj ich dôveryhodnosť a prípadne aj zdroj, ktorý charakteristiku získal ak existuje viacero ciest ako sa charakteristika môže do modelu dostať. Takéto delenie charakteristík je použité napríklad v [32].

Cieľom našej práce je riešenie, ktorým chceme prispieť k aktuálnemu stavu v oblasti budovania modelu používateľa. Konkrétne ciele môžeme nadefinovať zvlášť pre fázu zberu dát, pre fázu analýzy dát a zvlášť pre zvolenú reprezentáciu modelu používateľa.

1. Ciele v oblasti zberu dát

   (a) mimimalizovať mieru zapojenia používateľa do procesu získavania dát,

   (b) dosiahnuť detailnú úroveň výsledného záznamu akcií s presnými časovými údajmi,

   (c) explicitne zadefinovať *flexibilnú* reprezentáciu zozbieraných dát vyjadrujúcu sémantiku jednotlivých zachytených akcií,

(d) umožniť jednotný spôsob zberu dát z viacerých prezentačných nástrojov.

2. Ciele v oblasti analýzy dát

   (a) vytvoriť riešenie, ktoré odhaduje charakteristiky používateľa na základe sémantických záznamov akcií používateľa,

   (b) dosiahnuť takú konfigurovateľnosť riešenia, ktorá umožní znovupoužitie analýzy vo viacerých aplikačných doménach,

   (c) navrhnúť spôsoby detekovania zmien charakteristík používateľa.

3. Ciele v oblasti reprezentácie dát

   (a) vytvoriť flexibilný model, ktorý dokáže pružne reagovať na zmeny a ktorý je potencionálne zdieľateľný viacerými aplikáciami,

   (b) vytvoriť základný model použiteľný vo viacerých doménach a navrhnúť spôsob prepojenia so špecifickým modelom používateľa z konkrétnej domény.

4 Metóda zachytenia záujmov používateľa na webe

V tejto kapitole opisujeme navrhnutú metódu zachytenia záujmov používateľa na webe založenú na analýze správania používateľa. Keďže neoddeliteľnou súčasťou metódy je reprezentácia dát zachycujúcich správanie používateľa a samotného modelu používateľa, uvádzame ich základné vlastnosti. Následne opisujeme špecifické postupy realizované v obidvoch hlavných fázach procesu budovania modelu používateľa (zber dát, analýza dát).

Metóda pozostáva z nasledovných krokov:

1. odchytenie udalosti na strane klienta a jej odoslanie logovacej službe na server,
2. odoslanie udalosti z prezentačného nástroja na spracovanie logovacej službe,
3. vytvorenie záznamu o aktivite používateľa,
4. prespracovanie záznamov,
5. detekcia vzorov,
6. ak detekovaný vzor predstavuje vzor implicitnej spätnej väzby:
   (a) vyhodnotenie spätnej väzby – získanie číselného ohodnotenia konceptov,
   (b) porovnanie ohodnotených konceptov,
7. zistenie konzistentnosti správania používateľa,
8. úprava modelu používateľa.

Kroky 1 až 3 postupnosti predstavujú fázu zberu dát o používateľovi. Kroky 4 až 8 predstavujú fázu analýzy a spracovania dát o používateľovi.

4.1 Reprezentácia dát v modeli používateľa

Model používateľa je na fyzickej úrovni tvorený dvoma modelmi:

- záznamy o správaní používateľa a
- ontologický model používateľa

4.1.1 Záznamy o správaní používateľa

Keďže na analýzu správania potrebujeme oveľa podrobnejšie záznamy o aktivitách používateľa ako dokáže poskytovať webový server, súčasťou nášho riešenia je aj logovací podsystém, ktorý takéto záznamy vytvára. Základná požiadavka, kladená na záznamy je, aby boli *samonosné*, teda aby na ich interpretovanie neboli potrebné žiadne ďalšie implicitné informácie týkajúce sa pôvodu (zdroja) záznamov ani jednotlivých častí záznamu. Sémantika záznamov teda musí byť vyjadrená uplne a explicitne priamo v záznamoch.

Ďalšiou požiadavkou je aby reprezentácia dát bola flexibilná a umožňovala jednotným spôsobom uchovávať záznamy o interakcii používateľa pri použití viacerých prezentačných rozhraní a zároveň bola odolná voči zmenám systému a jeho prezentačnej časti.

Výsledkom je všeobecný dátový model, ktorého flexibilnosť bola dosiahnutá dôslednym typovaním použitých entít a ich atribútov. Model má teda dve úrovne:

- metaúroveň, ktorá predpisuje vzťahy medzi typmi entít a

- operatívnu úroveň, obsahujúcu konkrétne hodnoty.

Základnou entitou je udalosť (*event*), ktorá má časovú pečiatku a patrí do niektorého sedenia (*session*) používateľa (*user*) v systéme. Každá udalosť môže mať neobmedzený počet atribútov (*eventAttribute*), pričom každý atribút môže mať ďalšie atribúty, ktoré rodičovský atribút bližšie špecifikujú. Pod udalosťou nerozumieme ľubovoľnú udalosť systému, ale len takú udalosť, ktorá mení zobrazený obsah v dôsledku vykonanej akcie používateľa (čiže aj keď akcia používateľa vyvolá v skutočnosti sériu akcií systému, ktorými sa dosiahne zmena zobrazenia, budeme v záznamoch evidovať iba jednu udalosť). Každá udalosť teda môže mať definovaný logický stav zobrazenia (*displayState*), v ktorom bola vyvolaná a stav zobrazenia, do ktorého sa v dôsledku udalosti systém dopracoval. Každý stav zobrazenia je definovaný sériou zobrazených prvkov (*displayedItem*). Každý zobrazený prvok má podobne ako udalosť ľubovoľný počet atribútov (*displayedItemAttribute*), ktoré môžu byť špecifikované ďalšími atribútmi.

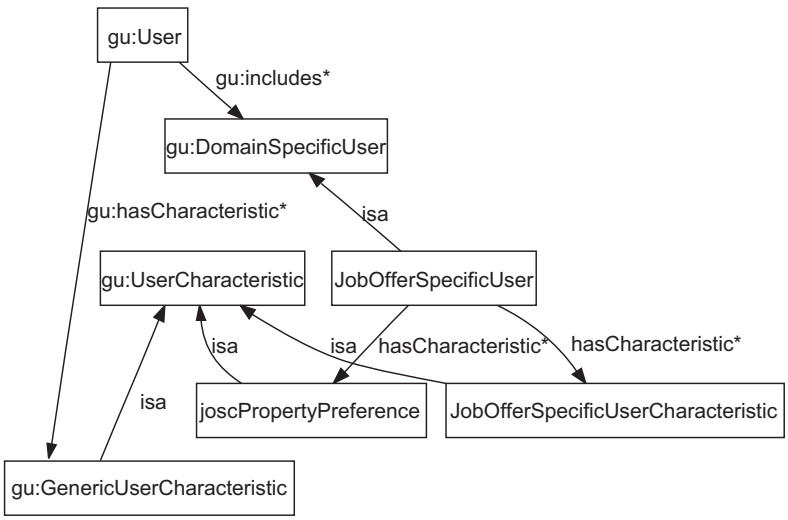
Kompletný opis dátového modelu sa nachádza v technickej dokumentácii v prílohe.

4.1.2 Ontologický model používateľa

Pre reprezentáciu charakteristík používateľa používame modely (nenaplnené štruktúry) vytvorené v rámci projektov NAZOU [1] a MAPEKUS [12]. Štruktúra modelu používateľa je vytvorená kombináciou viacerých modelov a je vždy napojená na príslušný doménový model.

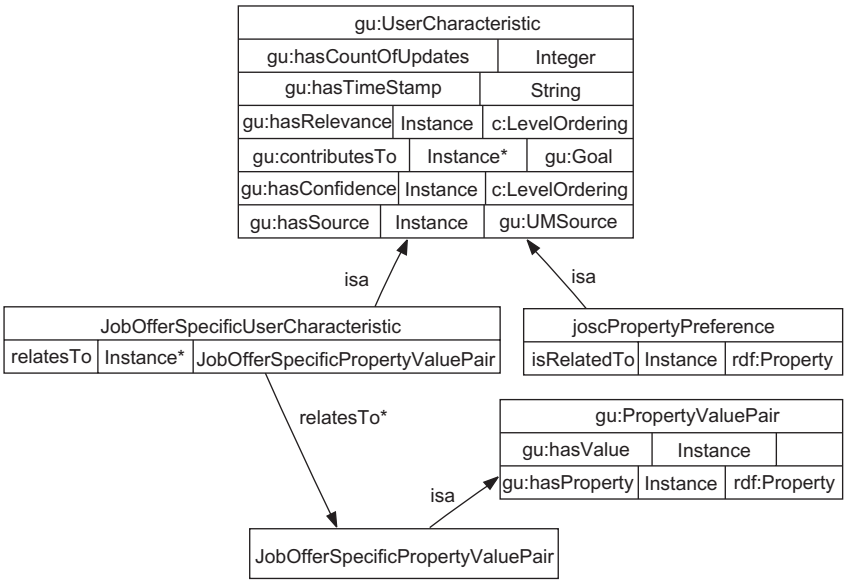
Keďže modely používateľa oboch domén sú vytvorené na základe podobnej myšlienky, opíšeme podrobnejšie iba jeden z nich. Kompletná dokumentácia modelov používateľa spolu s prislúchajúcimi doménovými modelmi sa nachádza v prílohe.

Základom modelu je entita reprezentujúca všeobecného používateľa (*gu:User*). Tá okrem toho, že má doménovo nezávislé charakteristiky (*gu:GenericUserCharacteristic*) zahŕňa aj viacero doménovo špecifických častí modelu (*gu:DomainSpecificUser*). Na obrázku 5 je zobrazená špecifická časť pre doménu pracovných ponúk (*JobOfferSpecificUser*). V tejto doménovej časti sú zadefinované dve charakteristiky. Jedna vyjadruje vzťah používateľa k jednotlivým vlastnostiam použitým v doméne (*joscPropertyPreference*, napr. miesto práce) zatiaľ čo druhá vyjadruje vzťah používateľa ku konkrétnym hodnotám vlastností (*JobOfferSpecificUserCharacteristic*, napr. Bratislava).



Obrázok 5: Základná štruktúra modelu používateľa.

Charakteristiky sú podtriedami základnej charakteristiky (*gu:UserCharacteristic*), ktorá im predpisuje spoločné vlastnosti (obrázok 6). Každá charakteristika má časovú pečiatku a zdroj. Charakteristika má určený cieľ, voči ktorému je udávaná jej relevancia. Keďže sa jedná o odhad, má charakteristika určenú úroveň dôveryhodnosti a udržuje sa počet, koľkokrát bola táto hodnota menená. Kompletná dokumentácia modelov je uvedená v prílohe.



Obrázok 6: Reprezentácia charakteristík používateľa.

4.2 Zber dát

Na základe analýzy možností zaznamenávania dát o aktivitách používateľa sme sa rozhodli vyvinúť nový spôsob zberu dát na strane servera, ktorý podporíme monitorovaním na strane klienta.

4.2.1 Zaznamenávanie na strane klienta

Základnou požiadavkou kladenou na záznamenávanie na strane klienta je automatizácia celého procesu vykonávania do takej miery, aby nebol potrebný žiadny zásah zo strany používateľa. Túto požiadavku nespĺňajú riešenia realizované ako samostatné aplikácie spúšťané na lokálnom stroji používateľa, pretože ich treba inštalovať a spúšťať. Vhodné sú klientské webové aplikácie, ktoré sa po načítaní stránky automaticky stiahnu a vykonajú v prostredí webového prehliadača.

Ďalšia požiadavka sa týka ochrany súkromia a bezpečnosti používateľa. Z pohľadu používateľa je vhodné, aby aplikácia bola schopná sledovať iba prácu používateľa s konkrétnym systémom a neumožňovala sledovanie kompletnej práce používateľa na počítači, či interakciu používateľa s inými aplikáciami. Túto požiadavku klientské webové technológie spĺňajú, keďže sú spúšťané s obmedzenými právami a možnosťami.

Metóda predpokladá zachytávanie a zaznamenávanie niektorých udalosti vygenerovaných webovým prehliadačom počas interakcie používateľa s jednotlivými prvkami na zobrazenej stránke. Monitorujeme minimálne tieto udalosti:

- *Load* – načítanie stránky;

- *Unload* – opustenie stránky;
- *Click* – nasledovanie odkazu na stránke;
- *Mouseover* – nasmerovanie kurzoru na aktívny element stránky;
- *Mouseout* – odstránenie kurzoru z aktívneho elementu stránky

Odchytávať sa môžu aj ďalšie udalosti, ktoré reprezentujú akcie, ktoré sa neprenášajú na server a nezaznamenajú v logu. Takou udalosťou je napríklad udalosť *change*, vyvolaná pri zmene prvku formulára. Postupnosť týchto udalostí nám dáva informáciu o poradí, v akom používateľ vypĺňal formulár, čo môže mať vplyv na jeho model v systéme. Iným príkladom je odchytávanie udalosti *scroll*. Táto udalosť je vyvolaná pri posúvaní obsahu stránky, ktorý presahuje obrazovku.

O každej zachytenej udalosti sa zaznamenávajú nasledovné údaje:

- typ udalosti;
- čas spustenia udalosti;
- kontext udalosti, ak je k dispozícii (typicky identifikácia elementu, ktorý vyvolal udalosť).

Komunikácia so serverom môže byť realizovaná buď po každej udalosti alebo dávkovo po zachytení sekvencie N rôznych udalostí. Komunikácia samotná prebieha na pozadí, bez potreby akéhokoľvek zásahu zo strany používateľa.

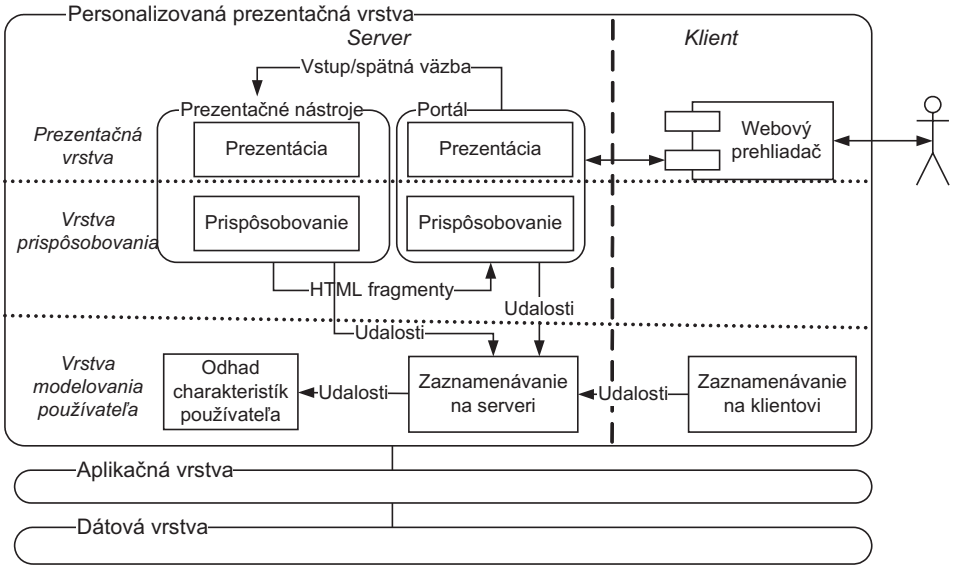
Na serveri sa prijaté dáta spájajú so záznamom produkovaným na strane servera, ktorý je obohatený o význam jednotlivých akcií. V prípade, že sa akcia vyskytuje aj v klientskom aj v serverom zázname, dostane sa do výsledného záznamu akcií iba jedna z nich.

4.2.2 Zaznamenávanie na strane servera

Zaznamenávanie na strane servera vychádza z charakteru prezentačnej vrstvy použitej v projektoch NAZOU [40] a MAPEKUS [12]. Vrstva je tvorená viacerými prezentačnými nástrojmi [56] (napr. adaptívny semantický fazetový prehliadač [55]), ktoré sú samostatne zodpovedné za prispôsobovanie a celkovú interakciu s používateľom. Jednotlivé prezentačné nástroje sú zasadené do spoločného prostredia – portálu, ktorý poskytuje nástrojom služby autentizácie a autorizácie používateľov.

Ak nechceme použiť nepostačujúce záznamy webového servera (dôvody sú opísané v predchádzajúcich kapitolách) je v takto realizovanej prezentačnej vrstve potrebné poveriť vytváraním záznamu každý nasadený prezentačný nástroj. Len nástroj samotný totiž pozná sémantiku skrytú za interakciou a dokáže ju zaznamenať (napr. URI inštancie, o ktorej detaily sa používateľ zaujíma, tak ako je uložené v doménovom modeli).

Pre oddelenie prezentačných nástrojov od použitej reprezentácie záznamov o akciách používateľa, navrhujeme špeciálnu logovaciu službu, ktorá zabezpečí príjem záznamov udalostí od jednotlivých prezentačných nástrojov ako aj od nástroja realizujúceho monitorovanie na strane klienta a zapíše tieto záznamy do úložiska v požadovanej forme. Spolupráca jednotlivých častí je naznačená na obrázku 7 spolu s prepojením na následnú analýzu dát (na obr. 7 „odhad charakteristík používateľa“).



Obrázok 7: Architektúra prezentačnej vrstvy obohatenej o prispôsobovanie a s ním spojený zber dát. Jednotlivé prezentačné nástroje zasadené v portáli ako aj portál samotný zasielajú notifikácie o udalostiach službe zaznamenávania na serveri. Tá ich spája so záznamami z klientského monitorovania a posúva na následné spracovanie, odhad charakteristík používateľa.

4.3 Analýza dát

Po získaní záznamov o akciách používateľa sa v kroku analýzy zo zachyteného správania extrahujú významy, ktoré sú s týmto správaním nepriamo spojené alebo z neho vyplývajú [5].

4.3.1 Základná koncepcia

O používateľových preferenciách mnoho vypovedá to, akým spôsobom sa pohybuje po navštívenom webovom sídle. Samozrejme to predpokladá takú navigačnú štruktúru, ktorá dáva používateľovi istú možnosť výberu akým smerom chce pokračovať v prehľadávaní webového sídla. Príkladom nie veľmi vyhovujúcej navigačnej štruktúry je sekvenčná navigačná štruktúra elektronických kurzov, v ktorej sa všetci používatelia pohybujú približne rovnako. Vhodnú navigačnú štruktúru poskytuje napríklad použitie fazetového prehliadača [55]. Za navigáciu v tomto prípade môžeme považovať postupný výber ohraničení vo fazetách, ich rušenie, prechod z náhľadu na detail a navigáciu po jednotlivých stránkach výsledkov.

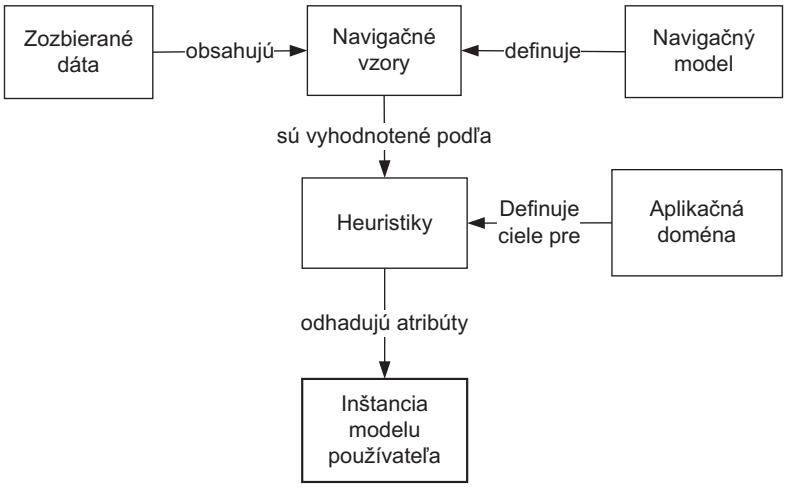
Okrem spôsobu, akým sa používateľ dostane k pre neho vhodným informáciám je dôležitá aj jeho reakcia na konkrétnu zobrazovanú informáciu (napr. pracovnú ponuku). Informácie, ktoré používateľa z nejakého dôvodu nezaujímajú si bude prezerať iba krátko a naopak informácie, ktoré používateľ študuje detailne a dlhšiu dobu sú zrejme pre neho zaujímavé. Tak isto zaujímavé sú zrejme aj informácie na stránkach, ku ktorým sa používateľ opakovane vracia.

Analýza dát využíva pravidlový prístup s využitím heuristík. V získaných dátach sa vyhľadávajú navigačné vzory (obrázok 8), ktoré vychádzajú z možností, ktoré poskytuje navigačný model. Nájdený vzor predstavuje ľavú stranu pravidla (podmienku). Pri jej splnení sa vykoná úprava modelu používateľa (pravá strana pravidla) na základe heuristi-

ky spájajúcej vzor s dôsledkom (z tohto pohľadu je analýza špecializovaným produkčným systémom). Heuristiky sú definované s ohľadom na aplikačnú doménu, ktorá definuje cieľ používateľa na najvyššej úrovni, napr. nájsť vhodnú vedeckú publikáciu alebo pracovnú ponuku. V prípade, že sa jeden spôsob navigácie používa vo viacerých doménach, dá sa znovupoužiť väčšina heuristík (s prípadnou zmenou cieľa asociovaného s charakteristikou).

Príklady heuristík

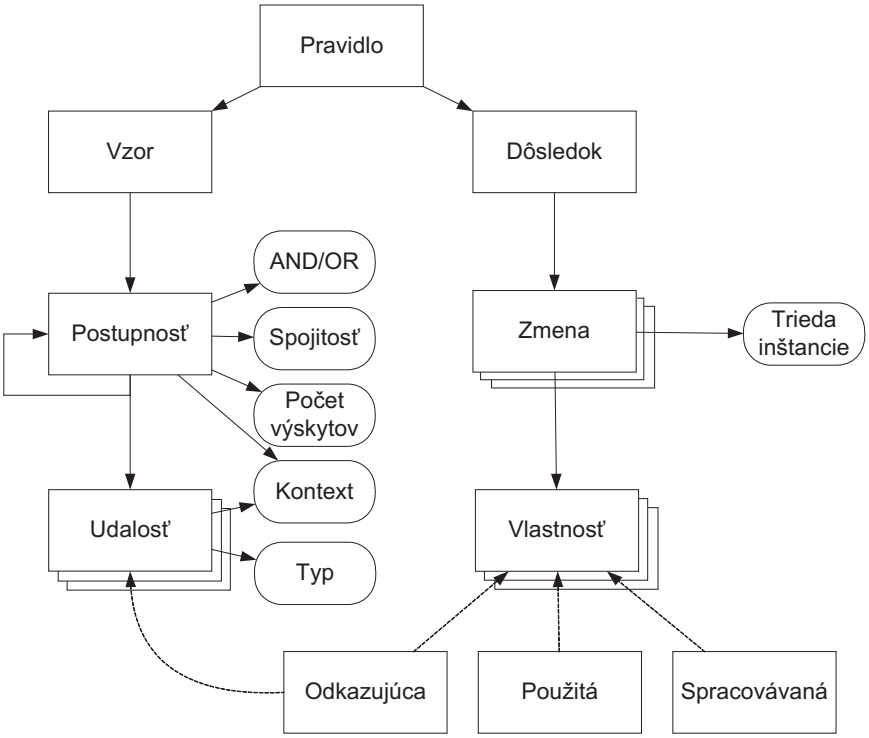
- Nech si používateľ nechal zobraziť detaily o najmenej N ponukách, ktoré spĺňajú množinu ohraničení X. V jeho modeli sa toto správanie pretaví do nastavenia charakteristík vyjadrujúcich preferencie ponúk s týmito ohraničeniami. V prípade, že sa charakteristika pre niektoré ohraničenie už v modeli nachádza, zvýši sa dôveryhodnosť tejto charakteristiky.
- Nech si používateľ ako prvú akciu po začatí sedenia vybral ohraničenie informačného priestoru pomocou atribútu ponuky X. V jeho modeli sa nastaví charakteristika vyjadrujúca záujem o ponuky s touto hodnotou a zároveň sa zvýši relevantnosť tejto charakteristiky.
- Nech si používateľ vybral množinu N ohraničení, prezrel si časť výsledných ponúk a následne jedno ohraničenie odobral. V modeli znížime relevantnosť charakteristiky spojenej s odobratým ohraničením.



Obrázok 8: Základný princíp analýzy záznamov akcií používateľa a aktualizáciu jeho modelu.

V analýze sa zameriavame na dve relatívne nezávislé cesty (pričom väčší dôraz kladieme na prvú z nich): analýzu navigácie používateľa po webovom sídle (s dôrazom na akcie nasledujúce bezprostredne po prihlásení používateľa [16]) a analýzu reakcií používateľa na ponúkaný obsah. Obidva prístupy využívajú vopred definované pravidlá, avšak zatiaľ čo pri navigačných vzoroch sa využíva priamo príslušná heuristika, pri vzoroch predstavujúcich spätnú väzbu sa vypočíta ohodnotenie obsahu používateľom a následne sa skúmajú príčiny tohto ohodnotenia, ktoré predstavujú charakteristiky používateľa.

Výsledkom analýzy dát je odhad (najmä) doménových charakteristík používateľa. Predpokladáme, že doménovo nezávislé charakteristiky sa získajú pomocou vstupného formulára a ďalších metód ako je analýza životopisov [37].



Obrázok 9: Štruktúra pravidiel pre odhadovanie charakteristík.

4.3.2 Pravidlá

Každé pravidlo sa skladá z dvoch častí: vzoru a jedného alebo viac dôsledkov (obrázok 9). Pravidlá sú dôležitou súčasťou metódy, preto kladieme dôraz na ich reprezentáciu, ktorá musí byť schopná zachytiť rôzne možnosti, ktoré môžu nastať pri interakcii používateľa so systémom. V tejto časti opisujeme zloženie jednotlivých častí pravidla.

Vzor

Kľúčovým prvkom analýzy záznamov sú preddefinované vzory, ktoré sa detekujú v dátach o aktivite používateľa. Vzor je na najvyššej úrovni postupnosť typov udalostí a ďalších postupností (obrázok 9). Vzor je detekovaný, ak sa nájde postupnosť na najvyššej úrovni. Postupnosť je nájdená ak sa v záznamoch podarí nájsť všetky jej podpostupnosti, resp. namapovať všetky predpísané typy udalostí na konkrétne inštancie udalostí zo záznamu o aktivite používateľa.

Postupnosť. Postupnost môže byť dvoch základných typov:

- *AllRequired*: základná postupnosť, ktorá sa v zázname nájde ak sa postupne nájdu všetky jej udalosti a podpostupnosti (ekvivalent logickej operácie *AND*);
- *OneRequired*: postupnosť, na ktorej nájdenie postačuje výskyt jednej z jej udalostí alebo podpostupností (ekvivalent logickej operácie *OR*).

Ďalej postupnosti delíme na *spojité* a *nespojité*. Spojitá postupnosť vyžaduje, aby všetky udalosti patriace do danej postupnosti a všetkých podpostupností nasledovali bezprostredne za sebou. Pri nespojitých postupnostiach môže byť medzi udalosťami ľubovoľný počet iných udalostí a postupnosti tak môžu zasahovať aj do viacerých sedení používateľa.

Postupnosť má ďalej tieto atribúty:

- *počet výskytov*: predpisuje požadovaný počet opakovaní postupnosti. Vykonávanie sa nepresunie na ďalšiu postupnosť v poradí (resp. nedetekuje vzor) kým sa tento počet nedosiahne. Tento atribút môže nadobudnúť špeciálnu hodnotu, ktorou sa postupnosť označí ako nepovinná;

- *kontext*: nepovinný atribút. Kontextom sa určujú podmienky, ktoré musí spĺňať každá udalosť, ktorá sa mapuje na predpísané typy udalostí danej postupnosti a jej podpostupností. Postupnosť môže mať neobmedzený počet kontextových podmienok. Ako príklad môžeme uviesť vzor, ktorý detekuje prehliadanie výsledkov. V tomto vzore má postupnosť kontextovú podmienku, aby všetky udalosti boli spojené s takými logickými stavmi zobrazenia, v ktorých sa *nemenia* aktuálne zvolené reštrikcie.

Udalosť. Udalosť predstavuje elementárnu časť vzoru. Pri detekovaní vzorov sa mapujú udalosti zo záznamu na udalosti predpísané vzorom. Každá udalosť má definovaný *typ*, ktorý zodpovedá niektorému známemu typu udalosti z metaúrovne reprezentácie záznamov o akciách používateľa. Každá udalosť môže mať definovanú váhu. Váha môže byť vypočítaná z času, ktorý uplynul do ďalšej udalosti alebo sa môžu pre výpočet použiť aj iné faktory (napr. predvolená váha niektorých typov udalostí alebo váha počítaná na základe existujúceho modelu používateľa).

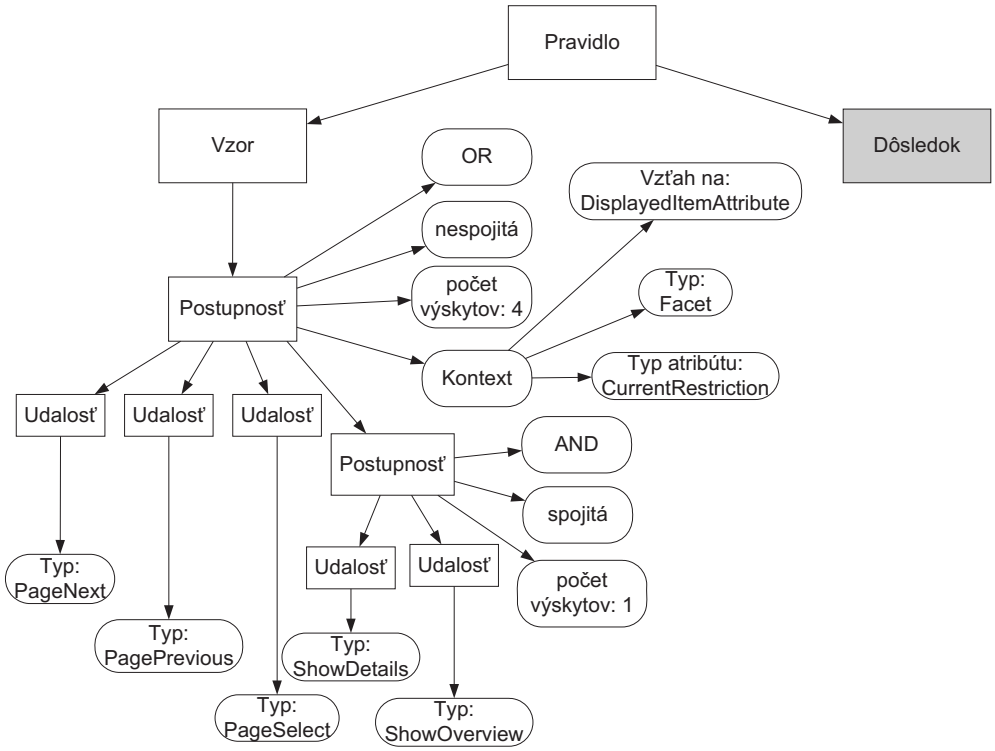
Každá udalosť môže mať ľubovoľné množstvo kontextových podmienok podobne ako pri postupnostiach. Kontext udalosti sa vzťahuje výlučne na atribúty udalosti zatiaľ čo kontext postupnosti kladie podmienky na atribúty stavu zobrazenia. Kontextová podmienka udalosti môže byť nasledovného typu:

- *SameAsPrevious*: hodnota zadaného typu atribútu udalosti musí byť rovnaká ako v predchádzajúcej udalosti (napr. pri vzore, ktorý vyjadruje upresňovanie reštrikcie v rámci jednej fazety musí byť stále rovnaká hodnota atribútu, ktorý definuje fazetu);

- *DifferentThanPrevious*: táto kontextová podmienka vyžaduje, aby sa hodnota zadaného typu atribútu líšila od hodnoty v predchádzajúcej udalosti;

- *MinValueOfWeight*: táto kontextová podmienka vyžaduje, aby váha danej udalosti bola vyššia ako zadaná hodnota. Príkladom využitia môže byť udalosť zobrazenia detailu, ktorá trvá príliš krátko na to, aby používateľ stihol prečítať aspoň časť informácií. V takom prípade udalosť nebudeme započítavať do vzoru vyjadrujúceho prehliadanie výsledkov.

Na obrázku 10 je ukážka vzoru, ktorý predstavuje prezeranie výsledkov. Na najvyššej úrovni je nespojitá postupnosť typu *OneRequired* (OR na obrázku). Postupnosť sa musí v záznamoch nájsť štyrikrát, aby bol detekovaný vzor. Postupnosť má určenú kontextovú podmienku, ktorá sa vzťahuje na atribút zobrazeného prvku (*DisplayedItemAttribute*). Všetky udalosti musia byť spojené s takými stavmi zobrazenia, v ktorých majú zobrazené prvky typu fazeta (*Facet*) nemennú hodnotu aktuálne zvolenej reštrikcie (atribút typu *CurrentRestriction*) (inými slovami, používateľ nemení aktuálne reštrikcie). Samotné udalosti môžu byť typu *PageNext*,*PagePrevious*, *PageSelect*, *ShowDetails* a *ShowOverview*. Prvé tri typy udalostí predstavujú navigáciu po jednotlivých stránkach výsledkov a posledné dve zobrazenie detailu a návrat späť na zoznam výsledkov. Posledné dve udalosti sú združené v spojitej postupnosti, čiže musia nasledovať za sebou.



Obrázok 10: Ukážka vzoru pravidla *prezeranie výsledkov*.

Dôsledok

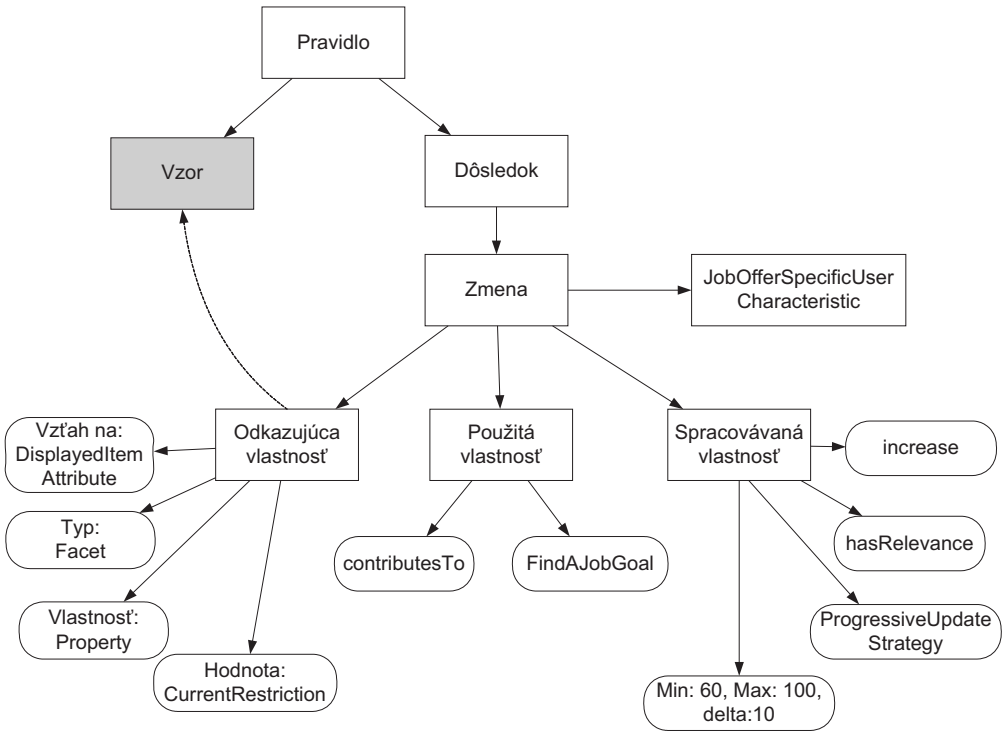
Dôsledok určuje čo a ako sa má zmenit' v modeli používateľa v prípade, že sa detekoval vzor. Dôsledok sa skladá z neobmedzeného počtu *zmien* (obrázok 9).

Každá zmena má definovanú jednu triedu a l'ubovoľný počet vlastností pomocou ich URI z ontológie. Trieda nám určuje typ inštancie, ktorá sa má pridat', resp. zmenit' v modeli používateľa (v našom modeli máme dva typy charakteristík: *joscPropertyPreference* a *JobOfferSpecificUserCharacteristic*). Vlastnosti hovoria o jednotlivých objektových a dátových vlastnostiach vytváranej/menenej inštancie.

Vlastnost' je jedného z nasledovných typov:

- *použitá*: jej hodnota sa priamo použije bez akéhokoľvek d'alšieho spracovávania;
- *spracovávaná*: obsahuje pokyny pre výpočet hodnoty danej vlastnoti, používa sa najmä pre hodnoty vlastností *confidence* a *relevance* charakteristiky. Základnou informáciou je, či sa má hodnota zvýšit' alebo znížit'. Ďalej je zadaný prírastok, minimálna a maximálna hodnota, ktorú vlastnost' získaná daným pravidlom môže získat'. Poslednou informáciou je určenie stratégie, ktorou sa má charakteristika menit' (napr. progresívne alebo rovnomerne);
- *odkazujúca*: odkazuje sa na niektorú udalost' zo vzoru. Hodnota vlastnosti je hodnotou niektorého atribútu referencovanej udalosti.

Na obrázku 11 je ukážka dôsledku pravidla *prezeranie výsledkov*. Dôsledok mení inštanciu triedy *JobOfferSpecificUserCharacteristic*. Inštancia má mat' vlastnost' *contributesTo* nastavenú na hodnotu *FindAJobGoal*. Dôsledok d'alej definuje odkazujúcu vlastnost', ktorá sa vzt'ahuje na atribúty zobrazených faziet. Vlastnost' sa neodkazuje na žiadnu špecifickú udalost' zo vzoru, takže sa môže použit' l'ubovoľná. Z každej fazety sa vytvorí



Obrázok 11: Ukážka dôsledku pravidla *prezeranie výsledkov*.

dvojica *PropertyValuePair* s hodnotami: atribút definujúci fazetu – aktuálna reštrikcia fazety. Dôsledok definuje aj spracovávanú vlastnosť: inštancia má vlastnosť *hasRelevance*, ktorá sa zvýši progresívnou stratégiou o hodnotu 10, maximálne do hodnoty 100.

4.3.3 Proces analýzy

Proces analýzy je zobrazený na obrázku 12. V tejto časti opisujeme detailne jednotlivé kroky analýzy a spracovania dát.

Predspracovanie dát

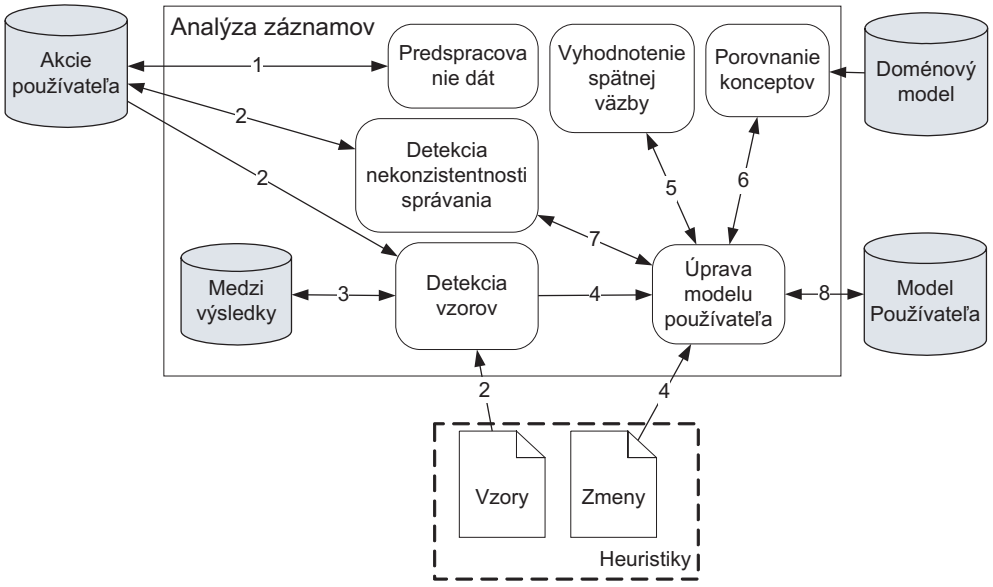
Vzhľadom na zvolený spôsob zberu dát a použitú reprezentáciu týchto dát (kapitola 4.1.1) je predspracovanie dát podstatne jednoduchšie ako je to pri bežných riešeniach, ktoré pracujú so záznamami webového servera [**?**].

V tomto kroku sa zo záznamov vyfiltrujú bezprostredne po sebe idúce *rovnaké* akcie, ktoré sú zvyčajne spôsobené netrpezlivosťou používateľa pri pomalších odozvách systému. Takisto sa v predspracovaní priradia jednotlivým udalostiam váhy na základe zvolenej stratégie.

Detekcia vzorov

Detekcia vzorov predstavuje základ procesu analýzy. Detekcia vzorov funguje podobne ako odvodzovanie v štandardných produkčných systémoch, snaží sa naviazať udalosť predpísanú pravidlom na konkrétnu udalosť záznamu. Udalosti sa viažu na inštancie pravidiel platné pre konkrétneho používateľa. Tie si udržujú väzbu na inštancie postupností, ktoré slúžia na evidenciu počtu výskytov postupností.

Pre každú spracovávanú udalosť sa vykonajú tieto kroky:



Obrázok 12: Prehľad procesu analýzy záznamov.

1. Zo všetkých pravidiel sa vyberú kandidáti – tie pravidlá, na ktoré sa potencionálne udalosť môže naviazať:

   - typ prvej udalosti vzoru pravidla sa zhoduje s typom aktuálnej udalosti alebo
   - existuje taká inštancia pravidla (pre daného používateľa), ktorej očakávaný typ udalosti sa zhoduje s typom aktuálnej udalosti.

   Rýchlosť spracovania tohto kroku je lineárne závislá od počtu pravidiel a počtu inštancií pravidla pre daného používateľa.

2. Vyberú sa inštancie pravidiel – kandidátov, ktoré skutočne vyhovujú podmienkam naviazania udalosti. V prípade, že ide o prvú udalosť vzoru pravidla, vytvorí sa nová inštancia.

   - overí sa, či udalosť spĺňa kontextové podmienky postupnosti (ak vzor nejaké definuje)
   - ak je postupnosť spojitá, overí sa, či udalosť nasleduje bezprostredne po poslednej naviazanej udalosti postupnosti (ak táto podmienka nie je splnená, môže sa inštancia pravidla vymazať, keďže vzor sa už nikdy nemôže detekovať),
   - overí sa, či udalosť spĺňa kontextové podmienky predpísanej udalosti (ak vzor nejaké definuje)

   Rýchlosť spracovania tohto kroku je lineárne závislá od počtu inštancií pravidla pre daného používateľa a náročnosti pravidiel (počtu kontextových podmienok, ktoré treba overiť).

3. Pre každú inštanciu pravidla z predchádzajúceho kroku sa vykoná naviazanie udalosti:

   - inštancia pravidla zaeviduje väzbu medzi udalosťou vzoru a udalosťou záznamu,

- zistí sa ďalšia očakávaná udalosť, upravia sa počty opakovaní v asociovaných inštanciách postupností (ak je to potrebné),
- v prípade, že je detekovaný vzor (neexistuje žiadna ďalšia očakávaná udalosť) spustí sa vykonanie dôsledku pravidla (úprava modelu používateľa). Po vykonaní zmeny sa inštancia pravidla vymaže.

Rýchlosť spracovania tohto kroku nie je závislá od počtu pravidiel či inštancií.

Úprava modelu používateľa

Úprava modelu používateľa sa riadi zmenami špecifikovanými v pravej časti pravidla (dôsledok). Pre každú zmenu sa vykonáva takáto postupnosť:

1. Načíta sa inštancia, ktorá reprezentuje menenú charakteristiku pre konkrétneho používateľa (s daným URI),
   - musia sa zhodovať hodnoty všetkých *odkazujúcich* a *použitých* vlastností,
   - zdrojom charakteristiky musí byť nástroj realizujúci túto metódu (pravidlo môže túto vlastnosť zmeniť),
   - ak takáto charakteristika neexistuje, tak sa vytvorí.
2. Upravia sa hodnoty všetkých *spracovávaných* vlastností (dôveryhodnosť, relevantnosť, časová pečiatka, počet aktualizácií) podľa príslušnej stratégie aktualizácie.

Opis modelu používateľa. Aby úprava modelu nebola viazaná na konkrétny model používateľa, má k dispozícii opis použitého modelu používateľa, ktorý mapuje všeobecné pojmy na konkrétne entity použitého modelu.

Metóda predpokladá tieto mapovania:

- URI vlastnosti, ktorou sa prepája všeobecná inštancia používateľa na doménovú inštanciu používateľa združujúcu všetky charakteristiky platné pre danú doménu (`gu:includes`),
- URI vlastnosti, ktorou sa spája charakteristika s doménovou inštanciou používateľa (musí byť zadaná pre každý typ charakteristiky),
- ak vlastnosť charakteristiky smeruje na inštanciu typu *PropertyValuePair* (pozri podkapitolu 4.1.2), je potrebné uviesť vzťah tejto inštancie ku atribútom vlastnosti zadefinovanej v pravidle (teda, ako sa vypĺňajú jednotlivé vlastnosti inštancie *PropertyValuePair*),
- pre tie *spracovávané* vlastnosti, ktorých hodnota sa nezadáva priamo ako vlastnosť charakteristiky je potrebné uviesť typ asociovanej inštancie a vlastnosť, ktorá sa má vyplniť (napr. `c:LevelOrdering` a `c:hasValue` pre uloženie hodnôt *confidence* a *relevance*),
- URI vlastnosti, ktorou sa spája charakteristika s inštanciou predstavujúcou zdroj charakteristiky (`gu:hasSource`),
- URI slúžiace ako identifikátor zdroja pre charakteristiky odhalené nástrojom realizujúcim metódu,
- URI vlastnosti, ktorými sa spája charakteristika s hodnotou predstavujúcou počet aktualizácií charakteristiky a hodnotou predstavujúcou časovú pečiatku.

Vyhodnotenie spätnej väzby

V prípade, že detekovaný vzor predstavuje vzor implicitnej spätnej väzby, vypočíta sa číselné ohodnotenie konceptu, na ktorý sa spätná väzba vzťahuje. Ohodnotenie predstavuje odhad systému, ako by používateľ ohodnotil obsah explicitne v škále danej stupnice (5, resp. 7 miestnej).

Ohodnotenie sa z implicitnej spätnej väzby môže vypočítať rôznymi stratégiami, v závislosti od vzoru implicitnej spätnej väzby [41], ktorý môže brať do úvahy rôzne indikátory spätnej väzby [17].

To, že implicitnú spätnú väzbu spracujeme do podoby odhadu hodnotenia oddeluje ďalšiu prácu so spätnou väzbou od spôsobu jej získania. Oddelenie umožňuje nahradiť implicitnú spätnú väzbu aj väzbou explicitnou, keď používateľ sám hodnotí zobrazený obsah.

Porovnávanie konceptov

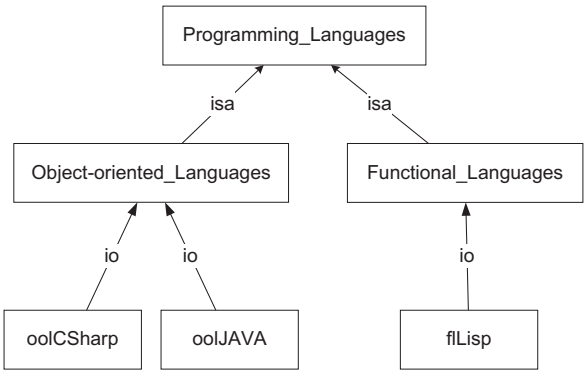
Vyhodnotenie spätnej väzby poskytne ohodnotenie konceptov. Jednoduché zvýšenie (resp. zníženie) úrovní všetkých doménových charakteristík zodpovedajúcich atribútom ohodnoteného konceptu však môže spôsobovať značné nepresnosti. Je totiž možné, že nízke ohodnotenie konceptu zapríčinil iba jeden atribút a charakteristiky pre ostatné atribúty nemusia byť znížené. Je preto dobré pátrať po dôvodoch vysokého/nízkeho ohodnotenia konceptu.

Takéto dôvody môžeme získať z porovnania rôzne a rovnako ohodnotených konceptov a zistenia ich spoločných a rovnakých atribútov. Ak sú napríklad dve ponuky ohodnotené rôzne a líšia sa iba v atribúte X, vieme odhadnúť, že atribút X bol dôvodom rôzneho hodnotenia. Hodnotu atribútu X z kladne hodnotenej ponuky tak môžeme posilniť, hodnotu z negatívne hodnotenej ponuky môžeme potlačiť.

V prípade dátových vlastností (literálov) má význam zisťovať výskyt spoločných slov v porovnávanom reťazci a počty znakov v porovnávaných slovách, ktoré sa zhodujú (tým sa môže odhaliť rozdielnosť, ktorá vznikla na základe preklepu). Je potrebné navrhnúť metriku, ktorá určí celkovú mieru podobnosti na základe výskytov rovnakých reťazcov a znakov [54].

V prípade objektových vlastností je pre mieru podobnosti významné umiestnenie tried v taxonómii ontológie. Dve inštancie jednej triedy sú si samozrejme blízke. Napr. C#, JAVA a spoločná trieda Objektovo-orientované jazyky (obrázok 13). Podobnosť dvoch inštancií klesá ak sa spoločná nadtrieda dá nájsť až na vyššej úrovni – napr. Lisp inštancia triedy Funkcionálne jazyky je od inštancie triedy C# vzdialenejšia ako JAVA, pretože majú spoločnú až nadtriedu Programovacie jazyky. V prípade pracovných ponúk by teda ponuka na pozíciu programátora v C# bola bližšia ponuke na pozíciu programátora v jazyku JAVA ako ponuke pre programátora v jazyku Lisp.

Pri porovnávaní dvoch konceptov môžeme definovať váhy jednotlivých vlastností, ktoré vplývajú na výpočet celkovej miery podobnosti konceptov Zmenou týchto váh môžeme zadefinovať, ktorá vlastnosť konceptu je pre nás pri porovnávaní najdôležitejšia. Napríklad podobnosť/rozdielnosť dátumu nástupu do zamestnania nemusí vplývať na celkovú podobnosť ponúk takou mierou ako je to pri ponúkanej pozícii. Porovnávanie môže brať do úvahy viacero metrík na rôznych úrovniach s rôznymi váhami podobne ako pri váhach vlastností [2].



Obrázok 13: Časť taxonómie programovacích jazykov.

Zisťovanie konzistentnosti správania používateľa

Aj keď sa metódy dolovania v dátach spomenuté v kapitole 2.5.1 nehodia na priame zisťovanie charakteristík používateľa, dajú sa pri vhodne zvolených vstupných dátach použiť pre zisťovanie konzistentnosti správania používateľa v systéme.

Ak spustíme metódu dolovania spojitých sekvenčných vzorov iba nad záznamami získanými z predchádzajúcich sedení jedného používateľa, dostaneme jeho typické vzory správania v systéme. Postupnosť z aktuálneho sedenia môžeme porovnávať so získanými vzormi. V prípade, že sa aktuálna postupnosť nedá namapovať ani na jeden vydolovaný vzor, jedná sa o nekonzistentné správanie používateľa.

Nekonzistentné správanie môže mať dve rôzne vysvetlenia. Jedným je to, že používateľ je v stave, v ktorom navigáciou po stránke nesleduje cieľ nájsť potrebnú informáciu (ideálnu ponuku, publikáciu a pod.). Príkladom takého správania je hľadanie práce pre kamaráta alebo prezeranie záznamov o publikáciách zo zvedavosti. Systém by v takom prípade mal znížiť dôveryhodnosť charakteristík objavených počas aktuálneho sedenia.

Ak je však od posledného prístupu používateľa do systému uplynula značná doba, mohlo sa stať, že sa zmenil samotný používateľa a jeho správanie odzrkadľuje tieto zmeny. Príkladom môže byť používateľ, ktorý si v systéme našiel vhodnú pracovnú ponuku a zamestnal sa. Po pár rokoch sa rozhodne úplne zmeniť pracovnú pozíciu, odbor a znova navštívi webový systém, aby si našiel inú prácu. Systém by mal v takomto prípade upraviť dôveryhodnosť evidovaných charakteristík spojených s cieľom nájsť si prácu a začať objavovať charakteristiky nanovo.

5 Nástroje realizujúce metódu

Pre overenie metódy sme navrhli niekoľko softvérových nástrojov, ktoré realizujú jednotlivé kroky metódy získavania charakteristík používateľa:

- *Click* – realizuje zber dát na strane klienta
- *SemanticLog* – realizuje zber dát na strane klienta
- *LogAnalyzer* – realizuje analýzu zobieraných dát
- *ConceptComparer* – realizuje porovnávanie dvoch ontologických inštancií

Pri overení navrhovanej metódy sme sa sústredili na analýzu navigácie. Vytvorili sme softvérové prototypy nástrojov realizujúcich zber dát *Click*, *SemanticLog* a tie časti nástroja *LogAnalyzer*, ktoré realizujú metódu analýzy navigácie používateľa po webovom sídle. Nástroj *ConceptComparer* nebol vzhľadom na rozsah práce implementovaný. Je potrebný pri analýze spätnej väzby, ktorá nebola predmetom overenia.

5.1 Client Side Action Recorder – Click

Nástroj *Click* [8] realizuje metódu zberu dát na strane klienta opísanú v kapitole 4.2.1. Nástroj je implementovaný v jazyku JavaScript, čo umožňuje jeho jednoduchú integráciu s webovým prehliadačom. Vďaka použitému prostrediu má nástroj natívny prístup k DOM (*Document Object Model*) reprezentácii stránky a udalostiam generovaným prehliadačom.

Na odchytávanie udalostí sa používa *W3C DOM Level 2 Event Model*[8] podobne ako v [3]. Ten umožňuje dynamické pridávanie funkcií na obsluhu udalostí, vďaka čomu sa nástroj veľmi jednoducho integruje s existujúcimi a novými stránkami. V každom HTML dokumente stačí v jeho hlavičke uviesť referenciu na súbor so skriptom.

Odosielanie záznamov je realizované pomocou technológie AJAX, ktorá umožňuje asynchrónnu komunikáciu so serverom, z ktorého bola stránka so skriptom načítaná. Komunikácia prebieha cez SOAP protokol. Pre účely komunikácie sa využíva knižnica Web Services Javascript Library od IBM[9].

Podrobnosti o implementácii a použití nástroja *Click* sa nachádzajú v technickej dokumentácii v prílohe.

5.2 Semantic Log

SemanticLog je nástroj (služba), ktorá realizuje metódu zberu dát na strane servera navrhnutú v kapitole 4.2.2. Na jeho návrhu a implementácii sa významnou mierou podieľal aj Bc. Michal Tvarožek.

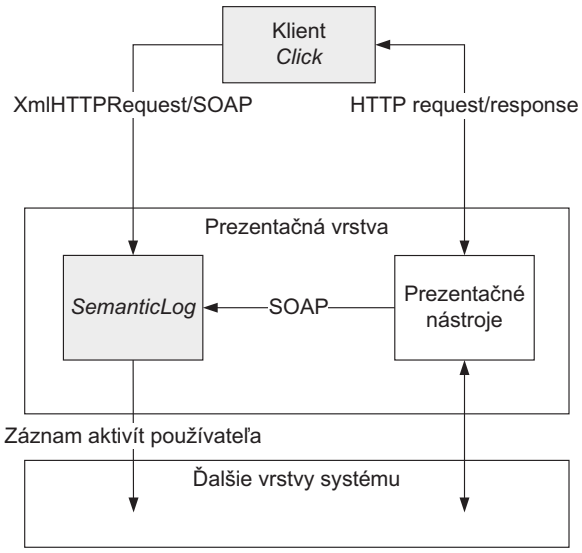
SemanticLog je implementovaný v jazyku Java SE 5.0 a môže byť nasadený buď ako webová služba alebo knižnica, ktorú využíva iný (prezentačný) nástroj. Ako úložisko záznamov o aktivitách používateľov definovaných v kapitole 4.1.1 *SemanticLog* používa relačnú databázu a systém riadenia bázy dát *MySQL*. K databáze nástroj pristupuje pomocou objektovo-relačného mapovača *Hibernate*[10], ktorý zabezpečuje perzistenciu použitého objektového modelu. Vďaka použitiu mapovania sa stal nástroj robustnejším voči zmenám systému riadenia bázy dát alebo zmenám v štruktúre databázy. Aktuálne používaná databázová schéma sa nachádza v technickej dokumentácii v prílohe.

[8] DOM Level 2 Events Specification, `http://www.w3.org/TR/DOM-Level-2-Events/`

[9] Call SOAP Web services with Ajax,
`http://www.ibm.com/developerworks/webservices/library/ws-wsajax/`

[10] Hibernate, `http://www.hibernate.org/`

Na obrázku 14 je zobrazené zapojenie nástroja *SemanticLog* do celkovej architektúry systému [**?**]. Nástroj *SemanticLog* prijíma notifikácie o udalostiach od jednotlivých prezentačných nástrojov a od nástroja *Click*, ktorý realizuje klientský monitoring.



Obrázok 14: Všeobecná architektúra časti systému vykonávajúcej zber dát.

5.3 Log Analyzer

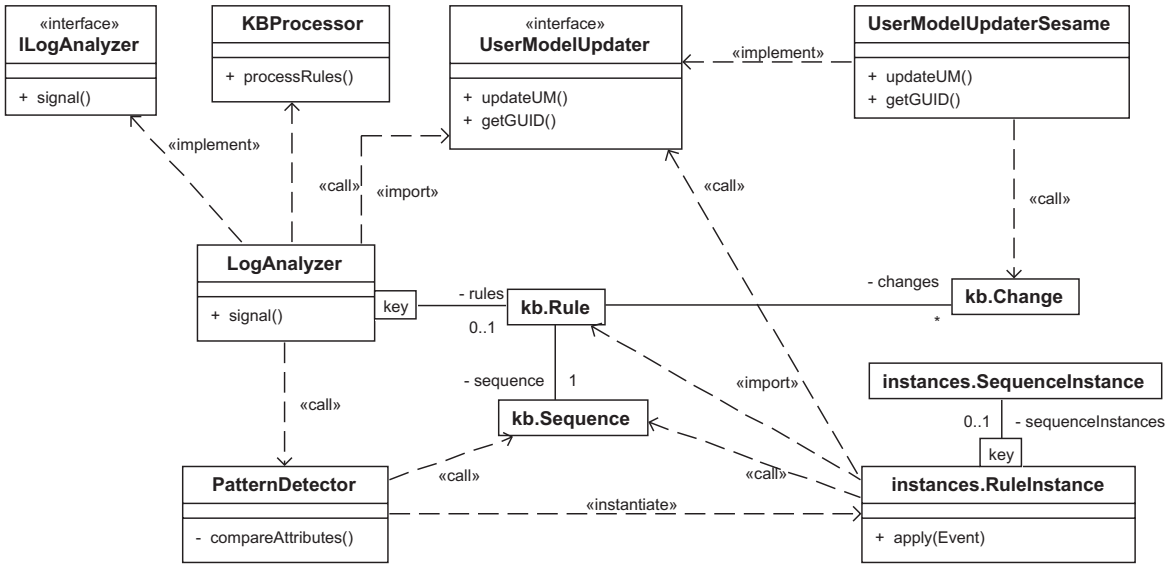
Nástroj Log Analyzer [4] realizuje metódu analýzy navigácie opísanú v kapitole 4.3. Je implementovaný v jazyku Java SE 5.0. Keďže metóda nepredpokladá priamu komunikáciu s používateľom, nástroj nemá prezentačnú vrstvu a nerealizuje žiadne používateľské rozhranie. Je navrhnutý v dvoch štandardných vrstvách architektúry: aplikačnej a dátovej vrstve.

Aplikačná vrstva sa riadi architektonickým vzorom *Doménový model* [25]. Objektový model aplikácie teda obsahuje dáta a definuje aj správanie, logiku aplikácie. Dátová vrstva je realizovaná kombináciou viacerých prístupov v závislosti od použitého typu dátového zdroja. *LogAnalyzer* používa tri typy dátových zdrojov: súbor, relačnú databázu a ontologickú databázu.

Na obrázku 15 sú zobrazené základné triedy nástroja. Hlavná trieda `LogAnalyzer` prepája všetky ostatné potrebné komponenty – trieda `KBProcessor` spracúva konfiguračný súbor s pravidlami a vytvára ich objektový model (na vrchnej úrovni reprezentovaný množinou pravidiel `Rule`, z ktorých každé má práve jednu postupnosť `Sequence` na najvyššej úrovni (vzor) a niekoľko zmien `Change`. Metódy triedy `PatternDetector` vyhľadávajú vzory, ktoré vyhovujú aktuálne spracovaváným udalostiam. Pre vyhovujúce vzory vytvára, resp. načítava ich inštancie `RuleInstance`. Jednotlivé inštancie vzorov zabezpečujú mapovanie udalostí záznamov na udalosti pravidiel. Sledujú stav splnenia vzoru a podľa potreby spúšťajú úpravu modelu používateľa cez triedu `UserModelUpdater`. Tá zabezpečí vykonanie jednotlivých zmien `Change`.

5.3.1 Detekovanie vzorov

Detekovanie vzorov vykonáva trieda *PatternDetector*. Pri detekovaní vzorov sa hľadajú všetky spôsoby naviazania udalostí na pravidlá (niektoré produkčné systémy hľadajú po prvé úspešné naviazanie). Výsledkom pre každú udalosť je séria inštancií pravidiel, na



Obrázok 15: Základné triedy nástroja LogAnalyzer. Prerušovaná čiara značí závislosť medzi triedami, plnými čiarami sú vyznačené základné asociácie medzi triedami. Podrobný opis objektového modelu sa nachádza v technickej dokumentácii v prílohe.

ktoré sa dá udalosť aplikovať. Aplikovaním udalosti na inštanciu pravidla sa vykoná samotné naviazanie udalosti a overenie podmienok splnenia vzoru pravidla. Ak vzor pravidla nie je splnený, určí sa množina očakávaných udalostí. Ak príchodzia udalosť splnila vzor pravidla, spustí sa aktualizácia modelu používateľa podľa dôsledkovej časti pravidla.

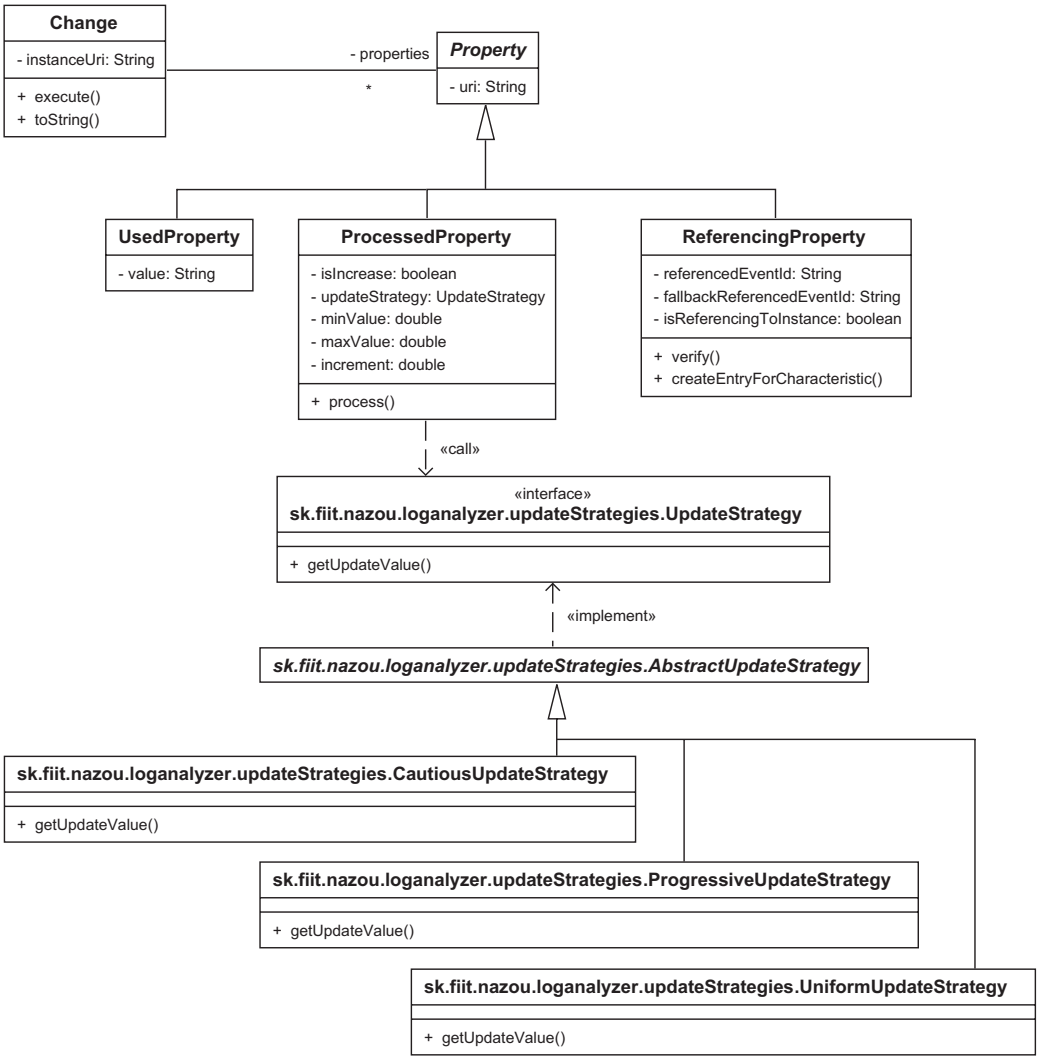
5.3.2 Úprava modelu používateľa

Opis modelu používateľa, ktorý sme uviedli v kapitole 4.3.3 je dostupný vo forme súboru *Java properties*. Komponent zodpovedný za úpravu modelu používateľa (trieda implementujúca rozhranie `UserModelUpdater`) vyvoláva postupne všetky zmeny zadefinované v príslušnom pravidle a poskytuje im prístup k tomuto opisu. V našom prípade je model používateľa uložený v RDF úložisku *Sesame*[11], ku ktorému sa pristupuje cez rozhranie ontologickej korporatívnej pamäti projektu NAZOU *OntoCM* [**?**].

Na obrázku 16 je zobrazená dekompozícia zmeny. Každá zmena `Change` má okrem identifikátora triedy menenej inštancie zoznam vlastností `Property`, ktoré sa majú danej inštancii menit'. Každý typ vlastnosti definovaný v podkapitole 4.3.2 zodpovedá jednej triede. *Použitá* vlastnosť `UsedProperty` potrebuje okrem URI vlastnosti iba hodnotu, ktorá sa má použiť. *Odkazujúca* vlastnosť `ReferencingProperty` potrebuje identifikátor udalosti, na ktorú sa odkazuje. *Spracúvaná* vlastnosť `ProcessedProperty` má informácie o spôsobe spracovania numerických hodnôt. Samotná zmena hodnoty je realizovaná vzorom *Strategy* [26]. Aktuálne sú v nástroji implementované tri stratégie (pričom implementácia umožňuje jednoduché pridávanie d'alších stratégií):

- rovnomerná (`UniformUpdateStrategy`): hodnota sa vždy zvýši/zníži o zadefinovaný rozdiel,
- progresívna (`ProgressiveUpdateStrategy`): skutočný rozdiel je zadefinovaný ako rozdiel násobený počtom zmien charakteristiky,

[11] Sesame RDF framework, `http://www.openrdf.org`



Obrázok 16: Objektový model zmeny v modeli používateľa.

- opatrná (`CautiousUpdateStrategy`): skutočný rozdiel je zadefinovaný ako rozdiel delený počtom zmien charakteristiky.

Nástroj *LogAnalyzer* sme počas vývoja testovali na rôznych úrovniach. Časť funkcionality bola testovaná na úrovni modulov s využitím rámca *JUnit*. Následne bol nástroj testovaný prístupmi čiernej skrinky (funkcionálne testovanie) a bielej skrinky (štrukturálne testovanie). Nástroj bol testovaný v ladiacom režime vo vývojovom prostredí ako aj v zaintegrovanom stave v reálnej spolupráci s ostatnými nástrojmi. Pre zaznamenávanie akcií nástroja a ich následné vyhodnotenie je použitý logovací rámec *Log4J*[12].

5.3.3 Model spúšťania

Hlavná trieda `LogAnalyzer` implementuje rozhranie `ILogAnalyzer`, ktoré predpisuje metódu `signal`. Touto bezparametrovou metódou sa spúšťa spracovanie.

Nástroj si pri každom spustení načíta z konfiguračného súboru čas posledného spustenia. Tento čas spolu s aktuálnym časom predstavujú hranice intervalu, ktorý *LogAnalyzer* v danom behu spracuje (teda, do úvahy sa berú iba tie udalosti, ktoré majú časovú

[12]Log4J logging service, `http://logging.apache.org/log4j/`

pečiatku v hraniciach spomínaného intervalu). Aktuálny čas spustenia sa zapíše do konfiguračného súboru, aby bol k dispozícii pre ďalší beh nástroja.

Konfiguračný súbor okrem času posledného spustenia definuje umiestnenie súboru s heuristikami/pravidlami a obsahuje *hash* hodnotu získanú aplikáciou hašovacej funkcie MD5 na súbor s heuristikami. Pri *vytvorení inštancie* nástroj overí zhodu zapísanej hash hodnoty s aktuálne vypočítanou hodnotou. Rozdielnosť hodnôt indikuje zmenu heuristík. V takom prípade nástroj odstráni všetky uložené medzivýsledky, keďže boli získané už neplatnými pravidlami.

5.3.4 Pravidlá – zápis a spracovanie do vnútornej reprezentácie

Pravidlá, ktoré boli opísané v kapitole 4.3.2 sú zapísané v XML a uložené v súbore. XML umožňuje efektívne zapísať všetky údaje, ktoré vyžaduje metóda. XML súbor je spracovaný do vnútornej reprezentácie metódami triedy `KBProcessor`, ktorá so súborom pracuje pomocou knižnice *Commons Configuration*[13].

Pravidlá sa spracúvajú pri vytváraní inštancie triedy `LogAnalyzer`. Trieda je implementovaná ako *Singleton* [26], takže k spracovaniu pravidiel dochádza iba raz pri prvom spustení a zavedení nástroja do pamäte.

Príklad 1 obahuje zjednodušenú ukážku zápisu pravidla. Pravidlo vychádza z predpokladu, že po začatí sedenia si používateľ pravdepodobne vyberie ohraničenie priestoru podľa hodnoty (pre používateľa) najdôležitejšej vlastnosti. Vzor pravidla teda obsahuje jednoduchú postupnosť troch udalostí: prihlásenie, prvotné zobrazenie výsledkov a výber reštrikcie. Sekvencia, ktorú vzor predpisuje musí byť spojitá, udalosti teda musia nasledovať bezprostredne za sebou.

Dôsledok pozostáva z jednej zmeny: úpravy inštancie `joscPropertyPreference`, ktorá zachytáva preferencie používateľa voči použitým ontologickým vlastnostiam. *Odkazujúca* vlastnosť `isRelatedTo` sa inštancii nastaví na hodnotu atribútu typu `Property` udalosti zo vzoru s identifikátorom `ev009`. Charakteristika sa viaže na cieľ nájsť si prácu (predpokladáme doménu pracovných príležitostí), takže *použitá* vlastnosť `contributesTo` sa nastaví na hodnotu `FindAJobGoal`. *Spracúvané* vlastnosti *hasRelevance* a *hasConfidence* sa upravia podľa príslušnej stratégie a parametrov. Dané pravidlo nemôže zvýšiť hodnotu dôveryhodnosti charakteristiky na viac ako 50 a hodnotu relevancie na viac ako 80. Dôveryhodnosť sa pritom zvyšuje s krokom 7 s použitím opatrnej stratégie (skutočný prírastok sa teda s počtom úprav znižuje). Relevancia sa zvyšuje rovnomerne o hodnotu 5 bez ohľadu na počet úprav charakteristiky.

[13] Commons configuration, `http://jakarta.apache.org/commons/configuration/`

```
<rule id="http://fiit.sk/loganalyzer#AfterLoginRestriction">
  <sequence id="s001" count-of-occurence="1" isContinuous="true">
    <event id="ev007" type="http://fiit.sk/semanticlog#UserLogin"/>
    <event id="ev008" type="http://fiit.sk/semanticlog#ShowOverview"/>
    <event id="ev009" type="http://fiit.sk/semanticlog#SelectRestriction"/>
  </sequence>
  <consequences>
    <change>
      <instance name="http://fiit.sk/job-offer-user#joscPropertyPreference"/>
      <property name="http://fiit.sk/job-offer-user#isRelatedTo" type="replace"
ref="ev009" typeOfRelatedEntity="http://fiit.sk/loganalyzer#EventAttribute">
          http://fiit.stuba.sk/semanticlog#Property
      </property>
      <property name="http://fiit.sk/gu#contributesTo" type="use">
          http://fiit.sk/job-offer-user#FindAJobGoal
      </property>
      <property name="http://fiit.sk/job-offer-user#hasRelevance"
type="process" isIncrease="true" strategy="UniformUpdateStrategy">
        <minValue>30</minValue>
        <maxValue>80</maxValue>
        <increment>5</increment>
      </property>
      <property name="http://fiit.sk/job-offer-user#hasConfidence"
type="process" isIncrease="true" strategy="CautiousUpdateStrategy">
        <minValue>10.0</minValue>
        <maxValue>50.0</maxValue>
        <increment>7.0</increment>
      </property>
    </change>
  </consequences>
</rule>
```

Príklad 1: Ukážka XML zápisu pravidla.

6 Experimenty a výsledky

S prototypom nástroja *LogAnalyzer* na odhadovanie charakteristík používateľa sme vykonali niekoľko experimentov. Nástroj *LogAnalyzer* sme prepojili s nástrojom *SemanticLog*, pomocou ktorého vytváral záznamy o aktivite používateľa nástroj *Factic*. Nástroje boli spustené v integračnom prostredí projektu NAZOU (Portal engine rámca Apache Cocoon).

Pri experimentovani sme sledovali nasledovné ciele:

1. zistiť reálne výkonnostné vlastnosti implementácie – čas spracovania udalosti v zaintegrovanom tvare. Z týchto vlastností vyplýva najvhodnejší model spúšťania nástroja;

2. overiť navrhnuté sady pravidiel – heuristík a podľa potreby upraviť ich parametre.

6.1 Výkonnostné vlastnosti implementácie

Pre zistenie výkonnostných parametrov sme simulovali používateľa, ktorý si hľadá prácu v regióne Amerika na pozíciu programátora. Po aplikovaní príslušných reštrikcií si používateľ prezrie aspoň štyri zobrazené ponuky. Experimenty sme vykonávali v informačnom priestore projektu NAZOU, ktorý obsahoval 101 ručne vyplnených pracovných ponúk.

V tabuľke 1 je zobrazený výpis zaznamenaných udalostí. Experiment sme overovali s dvoma sadami pravidiel (dostupné na priloženom elektronickom médiu):

1. *4 pravidlá*: preferencia vlastnosti fazety a vybranej hodnoty (s možnosťou spresňovania) po zvolení reštrikcie *hneď po prihlásení*, prehliadanie výsledkov, odobratie reštrikcie;

2. *6 pravidiel*: predchádzajúce štyri pravidlá obohatené o preferenciu vlastnosti fazety a jej hodnoty (s možnosťou spresňovania) po zvolení reštrikcie.

Tabuľka 1: Zaznamenané udalosti, zoskupené podľa volaní nástroja *LogAnalyzer*.

id	opis
1	prihlásenie, zobrazenie prehľadu
2	reštrikcia na pozíciu programátor
3	reštrikcia na región svet (rozbalenie fazety)
4	reštrikcia na región Amerika
5	detail ponuky
6	zobrazenie prehľadu
7	detail ponuky
8	zobrazenie prehľadu
9	detail ponuky
10	zobrazenie prehľadu
11	detail ponuky
12	zobrazenie prehľadu

Overenie na dvoch sadách pravidiel nám umožnuje vyhodnotiť citlivosť rýchlosti spracovania na malú zmenu počtu pravidiel. V tabuľke 2 sú zobrazené jednotlivé hodnoty časov spracovania z piatich totožných simulácií (S1,..,S5) so štyrmi pravidlami spolu s výslednými priemermi sa sumami. Tabuľka 3 obsahuje tie isté údaje pre simuláciu so

šiestimi pravidlami. Každá simulácia bola spustená nad čistou databázou udalostí a neexistujúcim modelom používateľa (pri prvej aktualizácii charakteristiky sa vytvorili všetky potrebné inštancie). Pre porovnanie sú na obrázku 17 v grafe vynesené priemerné hodnoty spracovania pre každé spustenie nástroja.

Z obrázku vidieť udalosti, ktoré spúšťali úpravu modelu používateľa (2, 3 a 12 pre obe sady a 5 pre sadu šiestich pravidiel). Udalosť 2 vyvolala zápis preferencie vlastnosti *offersPosition* pracovnej ponuky. Udalosť 3 vyvolala zápis hodnoty charakteristiky *programmer* a v prípade sady šiestich pravidiel aj nepatrné zvýšenie relevancie a dôveryhodnosti oboch charakteristík. Udalosť 5 vyvolala zápis preferencie vlastnosti *hasDutyLocation* pracovnej ponuky spolu s hodnotou *America*. Udalosť 12 vyvolala zápis charakteristiky, ktorá nastavila aktuálne zvolenej pozícii *a zároveň* zvolenému regiónu vysokú relevanciu a dôveryhodnosť.

Čas spracovania spomínaných udalostí výrazne presahuje spracovanie ostatných udalostí. Priemerný čas spracovania udalosti, ktorá nespúšťa úpravu modelu je: 226 ms oproti 8 948,486 ms, čo je priemerný čas spracovania udalosti, ktorá upravuje model. Tieto hodnoty sú z veľkej miery dané limitmi použitého ontologického úložiska a výpočtových prostriedkov.

Tabuľka 2: Časy spracovania (v milisekundách) jednotlivých udalosti pri inviduálnom volaní metódy `signal()` a použití štyroch pravidiel.

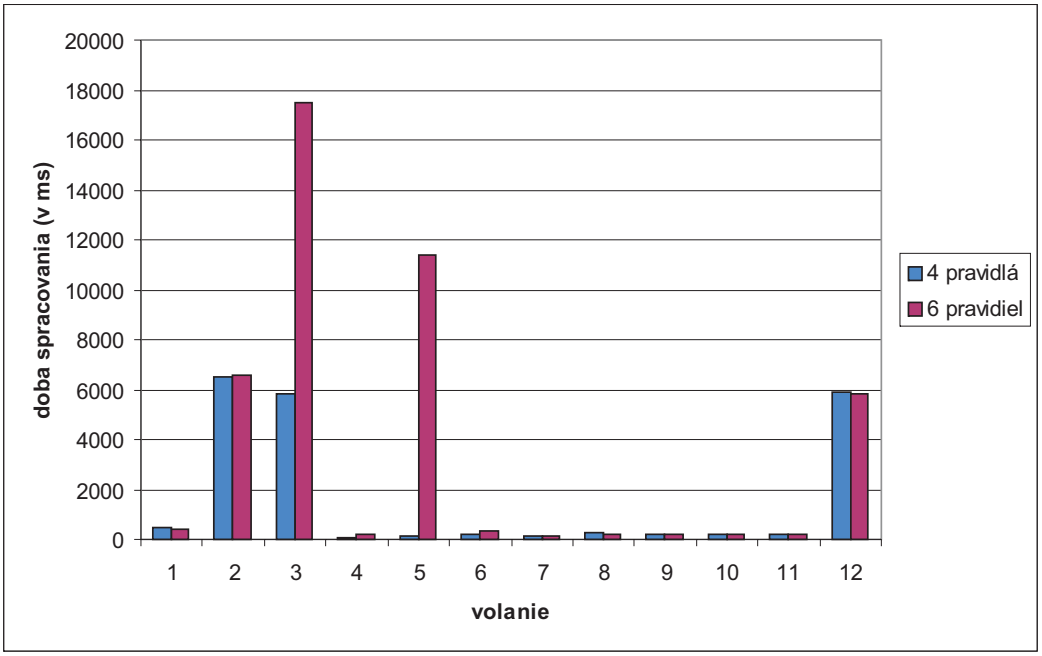
id\simulácia	$\mathbf{S}_1[ms]$	$\mathbf{S}_2[ms]$	$\mathbf{S}_3[ms]$	$\mathbf{S}_4[ms]$	$\mathbf{S}_5[ms]$	**priemer**
1	375	672	390	391	562	478
2	6 219	6 609	6 672	6 594	6 515	6 521,8
3	6 672	5 578	5 657	5 703	5 648	5 851,6
4	78	78	78	78	78	78
5	157	140	140	140	141	143,6
6	219	204	203	203	203	206,4
7	172	172	172	156	156	165,6
8	203	218	218	640	218	299,4
9	219	172	187	157	218	190,6
10	218	203	219	203	218	212,2
11	172	171	172	172	172	171,8
12	5 797	6 422	5 625	5 672	5 813	5 865,8
suma	20 501	20 639	19 733	20 109	19 942	20 184,8

Z uvedených tabuliek a obrázku ďalej vyplýva, že zmena počtu pravidiel zo štyroch na šesť nemala štatisticky významný dopad na rýchlosť spracovania jednotlivých udalostí avšak vďaka vyššiemu počtu pravidiel dochádzalo k častejším úpravám modelu, čo viedlo k celkovo dlhšiemu spracovaniu udalostí (priemerne 20 184,8 ms oproti 43 146,6 ms). Keďže spracovanie pravidiel má lineárnu zložitosť v závislosti od počtu pravidiel, dá sa pri väčšom počte pravidiel predpokladať vyšší čas spracovania. Výsledky však ukazujú, že malý počet pravidiel (rádovo jednotky) dokáže zabezpečiť odchytávanie dôležitých charakteristík používateľa.

Uvedený experiment bol vykonávaný v modeli spúšťania, keď prezentačný nástroj *Factic* volal synchrónne (blokujúco) logovací nástroj *SemanticLog*, ktorý takisto synchrónne volal nástroj *LogAnalyzer*. Experiment ukázal, že spracovanie udalosti môže trvať niekoľkonásobne dlhšie ako je doba, ktorú je štandardný používateľ ochotný akceptovať ako maximálnu dobu odozvy systému. Ako najvhodnejší model spúšťania procesu ana-

Tabuľka 3: Časy spracovania (v milisekundách) jednotlivých udalosti pri inviduálnom volaní metódy `signal()` a použití šiestich pravidiel.

id\simulácia	$\mathbf{S}_1[ms]$	$\mathbf{S}_2[ms]$	$\mathbf{S}_3[ms]$	$\mathbf{S}_4[ms]$	$\mathbf{S}_5[ms]$	**priemer**
1	438	391	391	391	468	415,8
2	6 547	6 984	6 282	6 703	6 329	6 569
3	16 188	17 063	18 343	17 344	18 359	17 459,4
4	171	234	172	281	172	206
5	10 860	11 485	11 360	11 437	11 765	11 381,4
6	203	750	187	188	203	306,2
7	172	156	172	187	156	168,6
8	187	187	344	187	203	221,6
9	218	172	156	172	156	174,8
10	203	203	219	203	219	209,4
11	313	156	172	157	172	194
12	5 656	5 922	5 641	5 890	6 093	5 840,4
suma	41 156	43 703	43 439	43 140	44 295	43 146,6



Obrázok 17: Doba spracovania jednotlivých volaní nástroja LogAnalyzer.

lýzy záznamov sa teda javí asynchrónne (neblokujúce) volanie v d'alšom vlákne alebo prepojenie na základe mechanizmu posielania správ, ktoré by sa hromadili vo vstupnom zásobníku nástroja *LogAnalyzer* a postupne by sa spracúvali. Do úvahy pripadá aj úplné offline spracovanie, keď sa analýza dát spúšťa dávkovo v stanovených intervaloch.

6.2 Overenie pravidiel

Aktuálne sú v systéme nastavené také pravidlá (dostupné na priloženom elektronickom nosiči), ktorých vzory spájajú viacero rôznych udalostí (napr. login a reštrikcia, reštrikcia a prezeranie výsledkov). Takéto pravidlá odhadujú charakteristiky s vyššou dôveryhodnosťou ako pri použití jednoduchších vzorov pozostávajúcich z atomických udalostí (aj keď riešenie takéto vzory umožňuje nadefinovať a používať).

Pre vyhodnotenie vhodnosti aktuálnych pravidiel môžeme brať opäť do úvahy príklad programátora hľadajúceho prácu v Amerike, ktorý v systéme vykoná akcie zodpovedajúce udalostiam v tabuľke 1.

Prvou akciou používateľa je zvolenie reštrikcie na pozíciu *programmer*. To vyvolá prvú zmenu v modeli používateľa na základe pravidla o preferencii vlastností (tabuľka 4).

Tabuľka 4: Model používateľa po prvej akcii.

typ:	joscPropertyPreference
vzťah k:	offersPosition
relevancia:	60
dôveryhodnosť:	25

Pravidlo nastavuje relatívne vysokú relevanciu charakteristiky, čím sa vyzdvihuje dôležitosť prvej akcie používateľa po prihlásení. Na to, aby mala charakteristika vyššiu dôveryhodnosť je potrebných viac sedení používateľa, počas ktorých si vždy vyberie ako prvú reštrikciu práve hodnotu vlastnosti *offersPosition*.

Následne sa používateľ rozhodne volit' reštrikcie vo fazete pre miesto výkonu práce (Svet – Severná Amerika – Amerika). V okamihu, keď zvolí hodnotu z inej fazety ako v predchádzajúcom prípade sa splní vzor d'alšieho pravidla (tabuľka 5).

Tabuľka 5: Model používateľa po druhej akcii.

typ:	JobOfferSpecificUserCharacteristic
vzťah k vlastnosti:	offersPosition
hodnota vlastnosti:	pcProgrammer
relevancia:	60
dôveryhodnosť:	10

Zatiaľ čo relevancia je rovnaká ako v prípade predchádzajúceho pravidla (jedná sa o hodnotu v prvej reštrikcií po prihlásení), dôveryhodnosť sme nastavili na nižšiu hodnotu.

Ak používateľ začne interagovať s aktuálne zobrazenými ponukami (detail – náhľad, prehliadanie stránok s výsledkami) a vykoná nad zoznamom ponúk aspoň 4 akcie, splní sa d'alší vzor (tabuľka 6).

Táto charakteristika spája vlastnosť ponúk s pozíciou *programátor* s miestom výkonu práce *Amerika*. Keďže v samotnom vzore pravidla je zadefinovaný opakujúci sa proces (aspoň štyrikrát vykonať určitý typ akcie) má odhad charakteristiky vyššiu dôveryhodnosť ako pri iných pravidlách.

Tabuľka 6: Model používateľa po prezretí časti výsledkov.

typ:	JobOfferSpecificUserCharacteristic
vzt'ah k vlastnosti:	offersPosition
hodnota vlastnosti:	pcProgrammer
vzt'ah k vlastnosti:	hasDutyLocation
hodnota vlastnosti:	America
relevancia:	60
dôveryhodnost':	50

Používateľ teda v systéme zatiaľ neklikol viac ako desat'krát a v modeli máme uložené tri charakteristiky, s ktorými sa dá ďalej pracovat' (pre používateľa je podstatná pozícia, zaujíma sa o prácu programátora v Amerike). Charakteristiky majú zatiaľ nižšiu dôveryhodnost', avšak pravidlá sme nastavili tak, aby sa pri opakovanom odhalení rovnakej charakteristiky táto hodnota zvyšovala rýchlejšie: pri druhom prihlásení používateľa a vybraní reštrikcie z fazety pre pozíciu je dôveryhodnost' charakteristiky z tabuľky 4 zvýšená na hodnotu 40 (pôvodne 25). Relevancia rastie pomalšie, zvýši sa na 80 (pôvodne 70).

7 Zhodnotenie a d'alšia práca

Cieľom práce bolo navrhnúť metódu automatizovaného získavania charakteristík používateľa webového informačného systému. Orientovali sme sa na analýzu správania používateľa v rámci webového systému založenú na heuristikách vyjadrených pravidlami, pričom sa analyzujú záznamy o aktivite používateľa s významom.

Z hľadiska cieľov definovaných v kapitole 3 sme navrhli a overili metódu zberu dát o používateľovi s minimálnym zapojením používateľa do procesu. Vďaka kombinácii zberu dát na strane servera a klienta sme dosiahli požadovanú granularitu výsledného záznamu, ktorá nekladie žiadne obmedzenia na následnú fázu analýzy. Pri zbere dát na strane servera sme určili zodpovednosť logovania pre jednotlivé prezentačné nástroje, čo nám umožňuje dodať jednotlivým akciám význam a dosiahnúť tak oveľa lepšie východiská pre analýzu ako je to pri použití štandardného záznamu webového servera.

Navrhli sme metódu analýzy zozbieraných dát, ktorá kombinuje viaceré prístupy (analýza navigácie, spätnej väzby, konzistencie správania) pre lepší odhad charakteristík. Samotná metóda analýzy je generická a znovupoužiteľná vo viacerých doménach, pričom doménovo špecifické sú vždy heuristiky a mapovanie na použitý model používateľa.

Z hľadiska reprezentácie dát sme navrhli flexibilný model používateľa, ktorý dôsledne oddeľuje jednotlivé doménovo špecifické sub-modely. Navrhli sme niekoľko typov charakteristík spolu so spôsobom ako konzistentne pridávať d'alšie typy charakteristík.

Na základe návrhu sme implementovali prototyp, v ktorom sme sa sústredili na analýzu navigácie a na tie časti zberu dát, ktoré tento typ analýzu nevyhnutne potrebuje. Zároveň sme kládli dôraz na integráciu vytvoreného riešenia v kontexte vybraných projektov NAZOU a MAPEKUS. Riešenie sme však vytvárali ako nezávislé s dôsledným oddelením od ostatných častí systému (najmä dátových úložísk) pomocou konfigurovateľných rozhraní.

S prototypom sme vykonali niekoľko experimentov, v ktorých sme overovali výkonnostné vlastnosti riešenia a hľadali vhodné parametre pravidiel. Základné experimenty ukazujú, že systém analýzy môže byť nasadený ako online riešenie za predpokladu, že jeho volanie nebude blokovať prezentačnú vrstvu alebo ako offline riešenie spúšťané v naplánovaných časoch (resp. dynamicky pri splnení podmienky aktivačného mechanizmu). Zároveň sa ukazuje, že systém je schopný odhadnúť niektoré charakteristiky veľmi rýchlo a tak rýchlejšie prekonať tzv. *cold-start* problém. S prototypom by bolo vhodné vykonať podrobnejšie experimenty s viacerými používateľmi, ktoré by reálne ukázali silu aktuálnych pravidiel a ich parametrov.

Metóda, jednotlivé softvérové nástroje a použité modely boli odprezentované na viacerých medzinárodných konferenciách. Príspevky prezentujúce myšlienky tejto práce boli ocenené cenou dekana na študentskej vedeckej konferencii IIT.SRC 2006 [6] a piatym miestom vo finále česko-slovenskej vedeckej súťaže študentských projektov ACM Student Research Competition 2006 [7].

Už počas riešenia sme identifikovali niekoľko možných vylepšení systému a smerov, ktorými by sa mohla uberať d'alšia práca. Okrem overenia navrhovanej analýzy spätnej väzby, ktorá by mala poskytnúť ešte detailnejšie charakteristky sa ako zaujímavý javí problém udržiavania modelu používateľa. Základom je navrhovaná detekcia konzistentnosti správania, ktorá ovplyvňuje dôveryhodnosť charakteristík a je schopná odhaliť zásadné zmeny v správaní používateľa. Rozšírením by mohli byť d'alšie metódy, ktoré analyzujú existujúci model a odstraňujú prípadne nekonzistencie alebo zlučujú veľmi podobné charakteristiky.

Tvorba pravidiel by mohla byť podporená technikami dolovania v dátach, ktoré by odhalili typické vzory správania používateľov systému. Tieto vzory môžu slúžiť na kontrolu množiny vzorov používaných v analýze. V prípade, že by sa v správaní používateľov vy-

skytoval zaujímavý vzor, ktorý nie je v analýze pokrytý, mohol by administrátor systému k vzoru doplniť význam a vytvoriť tak pravidlo pre analýzu.

Z hľadiska formalizmu pravidiel by bolo vhodné umožniť dynamické parametre (napr. dynamický počet opakovaní sekvencie na základe kontextu, maximálneho možného počtu opakovaní pre daný stav) a najmä meta-pravidlá, ktoré by mohli predpisovať zmeny pravidiel pre konkrétnych používateľov. Tieto meta-pravidlá by mohli určovať maximálny počet použití niektorého pravidla pre jedného používateľa, ovplyvnovať parametre pravidiel (počet opakovaní postupností, numerické hodnoty používané pri výpočtoch a pod.)

Použitá literatúra

[1] Andrejko, A., Barla, M., Bieliková, M.: Ontology-based User Modeling for Web-based Information Systems. In: *Information Systems Development (ISD)*, Budapest, Hungary, Springer, 2006.

[2] Andrejko, A., Barla, M., Tvarožek, M.: Comparing Ontological Concepts to Evaulate Similarity. In Návrat, P. et al., ed.: *Tools For Acquisition, Organisation and Presenting of Information and Knowledge*, Bystrá dolina, Nízke Tatry, SR, Vydavateľstvo STU, Bratislava, 2006, pp. 71–78.

[3] Atterer, R., Wnuk, M., Schmidt, A.: Knowing the User's Every Move: User Activity Tracking for Website Usability Evaluation and Implicit Interaction, booktitle = WWW '06, year = 2006, isbn = 1-59593-323-9, pages = 203-212, address = Edinburgh, Scotland, publisher = ACM Press.

[4] Barla, M.: Estimation of User Characteristics from Logs of User Activity. In Návrat, P. et al., ed.: *Tools For Acquisition, Organisation and Presenting of Information and Knowledge*, Bystrá dolina, Nízke Tatry, SR, Vydavateľstvo STU, Bratislava, 2006, pp. 175–181.

[5] Barla, M.: Interception of User's Interests on the Web. In Wade, V., Ashman, H., Smyth, B., eds.: *Adaptive Hypermedia and Adaptive Web-Based Systems, AH'06*. LNCS 4018, Dublin, Ireland, Springer, 2006, pp. 435–439.

[6] Barla, M.: Interception of User's Interests on the Web. In Bieliková, M., ed.: *IIT.SRC 2006: Student Research Conference*, FIIT SUT, Bratislava, SR, 2006, pp. 105–112.

[7] Barla, M.: Interception of User's Interests on the Web. In Mannová, B., Šaloun, P., Bieliková, M., eds.: *ACM CZ Student Research Competition 2006, Proceedings of Finalists Papers*, Prague, ČR, 2006, pp. 1–8.

[8] Barla, M., Tvarožek, M.: Automatic Acquisition of Comprehensive Semantic User Activity Log. In Návrat, P. et al., ed.: *Tools For Acquisition, Organisation and Presenting of Information and Knowledge*, Bystrá dolina, Nízke Tatry, SR, Vydavateľstvo STU, Bratislava, 2006, pp. 169–174.

[9] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. *Scientific American*, 2001, vol. 284, no. 5, pp. 34–43.

[10] Bieliková, M.: Presentation of Adaptive Hypermedia on the Web. In Popelínský, L., ed.: *DATAKON 2003*, Brno, ČR, 2003, pp. 72–91.

[11] Bieliková, M., Kuruc, J.: Sharing User Models for Adaptive Hypermedia Applications. In: *Intelligent Systems Design and Applications, ISDA'05*, Wroclaw, Poland, ACM Press, 2005, pp. 506–511.

[12] Bieliková, M. et al.: Modelovanie a získavanie, spracovanie a využívanie znalostí o konaní používateľa v hyperpriestore Internetu. Priebežná správa o riešení projektu APVT-20-007104, 2006.

[13] Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. *User Model. User-Adapt. Interact.*, 1996, vol. 6, no. 2-3, pp. 87–129.

[14] Callaway, C., Kufflik, T.: Using a Domain Ontology to Mediate between a User Model and Domain Applications. In Brusilovsky, P., Callaway, C., Nürnberger, A., eds.: *Workshop on New Technologies for Personalized Information Access (PIA 2005)*, Edinburgh, Scotland, UK, 2005.

[15] Cassel, L., Wolz, U.: Client Side Personalization. In: *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, Dublin, Ireland, 2001.

[16] Church, K., Keane, M., Smyth, B.: The First Click is the Deepest: Assessing Information Scent Predictions for a Personalized Search Engine. In: *Third Workshop on Empirical Evaluation of Adaptive Systems in conjunction with AH 2004*, Eindhoven, Netherlands, 2004, pp. 173–182.

[17] Claypool, M. et al.: Implicit Interest Indicators. In: *Intelligent User Interfaces*, Santa Fe, New Mexico, USA, 2001, pp. 33–40.

[18] De Bra, P., Calvi, L.: AHA: a Generic Adaptive Hypermedia System. In: *2nd Workshop on Adaptive Hypertext and Hypermedia*, Pittsburgh, USA, 1998, pp. 5–12.

[19] De Bra, P., Houben, G., Wu, H.: AHAM: a Dexter-based reference model for adaptive hypermedia. In: *Hypertext'99*, Darmstadt, Germany, ACM Press, 1999, pp. 147–156.

[20] De Bra, P. et al.: AHA! The adaptive hypermedia architecture. In: *Hypertext 2003*, Nottingham, UK, ACM, 2003, pp. 81–84.

[21] Denaux, R., Dimitrova, V., Aroyo, L.: Integrating Open User Modeling and Learning Content Management for the Semantic Web. In Ardissono, L., Brna, P., Mitrovic, A., eds.: *User Modeling 2005*. LNCS 3538, Edinburgh, Scotland, UK, Springer, 2005, pp. 9–18.

[22] Eirinaki, M., Vazirgiannis, M.: Web mining for web personalization. *ACM Trans. Internet Techn.*, 2003, vol. 3, no. 1, pp. 1–27.

[23] Etgen, M., Cantor, J.: What does getting WET (Web Event-logging Tool) Mean for Web Usability? In: *Human Factors & The Web: The Future of Web Applications*, NIST, Gaithersburg, Maryland, USA, 1999.

[24] Fenstermacher, K., Ginsburg, M.: Mining Client-Side Activity for Personalization. In: *Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, WECWIS '02*, Newport Beach, California, USA, 2002, pp. 205–212.

[25] Fowler, M.: *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, Boston, MA, USA, 2002.

[26] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[27] Halasz, F., Schwartz, M.: The Dexter Hypertext Reference Model. *Commun. ACM*, 1994, vol. 37, no. 2, pp. 30–39.

[28] Han, J., Kamber, M.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[29] Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M., von Wilamowitz-Moellendorff, M.: GUMO – The General User Model Ontology. In Ardissono, L., Brna, P., Mitrovic, A., eds.: *User Modeling 2005*. LNCS 3538, Edinburgh, Scotland, UK, Springer, 2005, pp. 428–432.

[30] Kay, J.: The um Toolkit for Cooperative User Modeling. *User Model. User-Adapt. Interact.*, 1995, vol. 4, no. 3.

[31] Kay, J. Human Factors and Ergonomics Series. In: *User Modeling for Adaptation*. Lawrence Erlbaum Associates, 2000, pp. 271–294.

[32] Kay, J., Kummerfeld, B., Lauder, P.: Personis: A Server for User Models. In De Bra, P., Brusilovsky, P., Conejo, R., eds.: *Adaptive Hypermedia and Adaptive Web-Based Systems, AH 2002*. LNCS 2347, Malaga, Spain, Springer, 2002, pp. 203–212.

[33] Kay, J., Lum, A.: Creating User Models from Web Logs. In: *Intelligent User Interfaces Workshop: Behavior-Based User Interface Customization*, Funchal, Madeira, Portugal, 2004, pp. 17–20.

[34] Kobsa, A.: Generic User Modeling Systems. *User Model. User-Adapt. Interact*, 2001, vol. 11, no. 1-2, pp. 49–63.

[35] Kobsa, A., Pohl, W.: The User Modeling Shell System BGP-MS. *User Model. User-Adapt. Interact.*, 1995, vol. 4, no. 2, pp. 59–106.

[36] Krištofič, A., Bieliková, M.: Improving adaptation in web-based educational hypermedia by means of knowledge discovery. In Reich, S., Tzagarakis, M., eds.: *Hypertext 2005*, Salzburg, Austria, 2005, pp. 184–192.

[37] Lenčucha, L.: Mining User's Characteristics in the Text, Bachelor thesis, FIIT STU, Bratislava, 2006.

[38] Lu, H., Luo, Q., Shun, Y.: Extending a Web Browser with Client-Side Mining. In Zhou, X., Zhang, Y., Orlowska, M., eds.: *Web Technologies and Applications, 5th Asian-Pacific Web Conference, APWeb 2003*. LNCS 2642, Xian, China, Springer, 2003, pp. 166–177.

[39] Middleton, S., Shadbolt, N., De Roure, D.: Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst.*, 2004, vol. 22, no. 1, pp. 54–88.

[40] Návrat, P. et al.: Acquiring, Organising and Presenting Information and Knowledge in an Environment of Heterogenous Information Sources. In Návrat, P. et al., ed.: *Tools For Acquisition, Organisation and Presenting of Information and Knowledge*, Bystrá dolina, Nízke Tatry, SR, Vydavateľstvo STU, Bratislava, 2006, pp. 1–12.

[41] Oard, D., Kim, J.: Implicit Feedback for Recommender Systems. In: *AAAI Workshop on Recommender Systems, July 1998.*, Madison, Wisconsin, USA, 1998.

[42] Obrst, L.: Ontologies for semantically interoperable systems. In: *Information and Knowledge Management*, New Orleans, Louisiana, USA, ACM Press, 2003, pp. 366–369.

[43] Paganelli, L., Paternò, F.: Intelligent Analysis of User Interactions with Web Applications. In: *Intelligent User Interfaces, IUI '02*, San Francisco, California, USA, ACM Press, 2002, pp. 111–118.

[44] Passin, T.: *Explorer's Guide to the Semantic Web*. Manning Publications, 2004.

[45] Pohl, W., Kobsa, A., Kutter, O.: User Model Acquisition Heuristics Based on Dialogue Acts. In: *International Workshop on the Design of Cooperative Systems*, Antibes-Juan-les-Pins, France, 1995, pp. 471–486.

[46] Polčicová, G.: *Topographic Organization of User Preference Patterns in Collaborative Filtering*. PhD thesis, FIIT STU, Bratislava, Slovakia, 2004.

[47] Qiu, F., Cho, J.: Automatic Identification of User Interest for Personalized Search. In: *WWW '06*, Edinburgh, Scotland, ACM Press, 2006, pp. 727–736.

[48] Rafter, R., Smyth, B.: Passive Profiling from Server Logs in an Online Recruitment Environment. In: *IJCAI Workshop on Intelligent Techniques for Web Personalisation (ITWP 2001)*, Seattle, Washington, USA, 2001, pp. 35–41.

[49] Razmerita, L., Angehrn, A., Maedche, A.: Ontology-Based User Modeling for Knowledge Management Systems. In Brusilovsky, P., Corbett, A., Rosis, F., eds.: *User Modeling 2003*. LNCS 2702, Johnstown, PA, USA, Springer, 2003, pp. 213–217.

[50] Setten, M.: Experiments with a Recommendation Technique that Learns Category Interests. In: *International Conference WWW/Internet, ICWI 2002*, Lisbon, Portugal, IADIS, 2002, pp. 722–725.

[51] Setten, M.: *Supporting People In Finding Information: Hybrid Recommender Systems and Goal-Based Structuring*. PhD thesis, Telematica Instituut, 2005.

[52] Studer, R., Benjamins, R., Fensel, D.: Knowledge Engineering: Principles and Methods. *Data Knowledge Engineering*, 1998, vol. 25, no. 1-2, pp. 61–197.

[53] Thomas, R. et al.: Generic Usage Monitoring of Programming Students. In Crisp, G., Thiele, D., Scholten, I., Barker, S., Baron, J., eds.: *20th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education, ASCILITE'03*, Adelaide, Australia, 2003, pp. 715–719.

[54] Tury, M., Bieliková, M.: An Approach to Detection Ontology Changes. In: *Workshop on Adaptation and Evolution in Web Systems Engineering (AEWSE'06)*. Volume 155 of *Conf. Proc. Series*., Palo Alto, California, USA, ACM Press, 2006.

[55] Tvarožek, M.: Personalized Navigation in the Semantic Web. In Wade, V., Ashman, H., Smyth, B., eds.: *Adaptive Hypermedia and Adaptive Web-Based Systems, AH'06*, Dublin, Ireland, Springer, LNCS 4018, 2006, pp. 467–471.

[56] Tvarožek, M., Barla, M., Bieliková, M.: Personalized Presentation in Web-Based Information Systems. In J, van Leeuwen, et al., ed.: *SOFSEM 2007*. LNCS 4362, Harrachov, ČR, Springer, 2007, pp. 796–807.

A Príspevky z medzinárodných konferencií

V tejto prílohe sa nachádzajú príspevky, v ktorých sme na medzinaródnych fórach prezentovali jednotlivé časti riešenia.

A.1 Príspevok prijatý na konferenciu AH 2006

V tejto prílohe sa nachádza príspevok prijatý na konferenciu *Adaptive Hypermedia and Adaptive Web-Based Systems – AH 2006*, ktorá sa konala v júni 2006 v Dubline. V príspevku sme prezentovali návrh metódy na zachytenie záujmov používateľa na webe. Príspevok bol prijatý do časti *Doctoral Consortium* konferencie.

Interception of User's Interests on the Web

Michal Barla*

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
barla@fiit.stuba.sk

Abstract. Current adaptive systems acquire information about users mainly by simple tracking of resources, a user has requested and by asking users to supply the needed information. In this paper, we discuss user modeling based on observing a user's interaction with the system. We propose to collect usage data on the server side as well as on the client side. Collected data are then processed into knowledge about user's intentions and preferences. This processing relies on a set of heuristics, which help to interpret the usage patterns found in the collected data.

1 Introduction

Adaptivity is becoming ever more important feature of web-based systems. An adaptive system reflects the particular needs of an individual user in a particular context and improves the efficiency of the user – system interaction. It is a response to the permanent information growth on the Internet, where finding the right information becomes difficult and time consuming.

Each adaptive system can only perform personalization if it already has some knowledge about the user. This knowledge is stored in various attributes in the user model. As the user continues to use the system, additional knowledge is acquired and added to the user model resulting in better adaptation. This leads to the cyclic loop "user modeling – adaptation" in an adaptive system, as mentioned in [1].

Our work focuses on the user modeling part of adaptive systems. Many user modeling systems gain information about users by simply asking them, however we chose to focus on an approach based on user observation. This includes the collection of data about user activity and the transformation of this data into knowledge about the user – creating the user model. We identify the main problems in this area and discuss possible solutions. Results of our work, as part of the project [2] are verified in the domain of job offers in a system used for job finding.

The paper is structured as follows. Section 2 discusses the approaches of gathering information about user actions in the information space in a non-intrusive manner. Next, in Section 3, we describe the methods used to transform acquired usage data to the user model. Finally, we draw some conclusions.

* Supervisor: prof. Mária Bieliková.

V. Wade, H. Ashman, and B. Smyth (Eds.): AH 2006, LNCS 4018, pp. 435–439, 2006.

2 Data Collection

There are several ways to acquire information about a user which serve for user model constructing. One is to monitor the user's interaction with the system – logging each user action for further analysis. One main drawback is the unreliability of user characteristics deduced from the acquired information, because there is no explicit relation between user actions and characteristics in the user model. However, this approach has the advantage that the system does not force the user to provide information explicitly, but instead it implicitly gathers the sequence of user actions during a session and interprets the acquired data to make statements about the user.

There are several approaches to user monitoring, which can either be performed on the server side or on the client side of the system. As a third option, one can combine both of these approaches.

Server side monitoring tracks user requests for resources. Its main drawback is that it does not provide precise time-related data, because it relies on the behavior of web browsers, which usually do not re-demand an already visited page from the server, but instead use the copy stored in the local cache. Thus the system does not know the exact time that the user spent viewing a certain page. The current most widespread web browsers do not respect the cache-control directives of the HTTP protocol forbidding the use of the local cache, so they cannot be used to bypass the cache problem. It is also mentioned in [3] that client side monitoring is necessary to get the precise records about a user's interaction with a system. Despite this, server side monitoring is still suitable for many adaptive systems. For example, AHA![1] uses server side logging to track what reading material is presented to the user [4].

Client side monitoring allows for the creation of a detailed log of user actions with exact timestamps. It can be performed by a specific client side application (e.g., User Action Recorder in [5]) or by employing a client web technology like JavaScript or Java applets. Since we consider the first approach as very invasive and not flexible enough, we focus on the second approach. The mentioned web technologies are common in the majority of web browsers on all major platforms. The possible drawbacks are that not every user accepts this kind of detailed monitoring and some of them block the execution of embedded scripts. Furthermore users may not have the necessary software installed on their computers (e.g., Java virtual machine).

Several tools with support for client side logging exist that exploit JavaScript such as WebVip[2] or WET[3]. Both tools are primarily designed for the purpose of web site usability evaluation. These tools are either too focused on the evaluation process or demand the entire copy of the web site for their operation. Therefore, we developed our own client side logging tool based on JavaScript combined with

[1] Adative Hypermedia for All, `http://aha.win.tue.nl/`
[2] Web Variable Instrumenter Program, `http://zing.ncsl.nist.gov/WebTools/WebVIP/overview.html`
[3] Web Event-logging Tool, [6].

the DOM2 event handling and asynchronous server communication using AJAX technology.

To summarize, it is not possible to gather any data if the user is not willing to enable client side user monitoring, which is a strong argument against the sole use of the above mentioned tools. On the other hand, server side monitoring is more reliable since it always acquires some data, but carries the risk of loosing precious time-related information. Our approach is based on the idea of combining the two aforementioned approaches – on the extraction of a maximum amount of data from the server log and on the use of the client log as a source of optional additional, precise information about the user's activity.

3 Data Analysis

After the data collection stage, we are supposed to transform the sequence of user's activities into statements about her cognitive processes. In another words, we have to determine non-behavioral meanings, which are either implied by or associated with the users' behavior [7] (e.g., to find out user goals, estimate user knowledge about certain concepts). The binding between actions and cognitive processes is not deterministic and is never definite. This is why the problem is widely discussed in the user modeling community (e.g., [8, 7]).

Patterns and Heuristics. When interpreting the data we look for interesting usage patterns, which describe the implicit feedback of the user. We analyze the sequences of "clicks" on the web-site and usage of the *back* button in the browser. We use sequential pattern mining algorithms to find such sequences of actions that differ only slightly from the predefined ones. For the initialization of the system, we plan to define patterns related to the user's goal and evaluate these patterns as the system will be used by real users.

We assign higher weights to the "first click" of the found sequence, as it usually has stronger relation to the user's intentions than the rest of the sequence. Successively we identify the appropriate usage pattern and use heuristics to infer user characteristics (see fig. 1).

An example of a simple heuristic in the domain of job offers is: "If a user chose to view at least "sufficient number" of offers from sector A (e.g., health-care or IT), raise the relevance level of this sector in the model of the user's ideal job offer".

During the analysis stage we also consider the navigation model of the system, which actually determines the possible sequences of user actions and thus makes all heuristics system-specific. Educational systems with sequential structure of pages forming an e-course, would have different usage patterns compared to a job offer portal, whose content is not sequential. The system must support easy navigation and searching in the information content, what results in a hierarchically organized navigation structure of the portal.

Relations between concepts. User actions in the context of adaptive web-based systems can be regarded as navigation between concepts. Our idea is that by

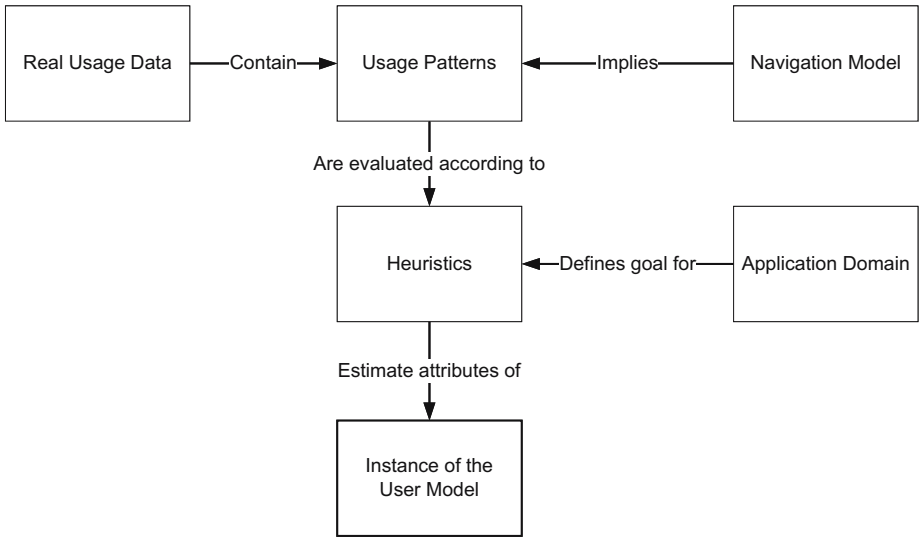


Fig. 1. Sources for creating an instance of the user model. We search for interesting *Usage Patterns* in the *Usage Data*. These patterns are determined by the *Navigation Model*. Knowing the related *Heuristics*, we can evaluate the located *Usage Patterns* to estimate attributes of an *Instance of the User Model*. *Heuristics* are bound to the goals of the user, determined by the *Application Domain*.

comparing the visited concepts and finding out their common and different aspects, we may gain knowledge about reasons (user preferences), why a user reacted differently to each of them. This comparison can either point out the values of different attributes of two concepts or compute their distance.

The distance represents the measure of dissimilarity of two concepts. For instance, C# is different from JAVA but it is definitely closer – less different to JAVA than to Lisp. A heuristic, which use the differences of two concepts must consider the distance of these concepts and its impact on the user model.

Afterward, it is possible to estimate user characteristics from the different or similar user actions related to the compared concepts. For instance, if the user "refuses" one offer but "accepts" another and these offers are quite close to each other, with their main difference being in the duty location, we can surmise that the user prefers the region from the second offer.

4 Conclusion

We described our research in the field of user modeling that is focused on user modeling based on user observation, which comprises the collection of information about the user activities within an information system and the successive analysis of the acqired information to create a user model.

We have identified several ways of user activity data acquisition, where client side monitoring appears to be the most efficient based on the richness of data, but also has a serious drawback in the unreliability of execution. Therefore, we use a combination of client and server side monitoring to achieve good results.

Analysis of the acquired data transforms the acquired user behavior into the knowledge about user characteristics or about user goals. We identified aspects, which influence the creation of heuristics that estimate some user characteristics from the recorded usage patterns.

Future work includes the design and verification of a method for the creation of an instance of the user model based on analysis of server and client side logs. This method would map the preferences of a user to a particular sequence of actions, use the comparison of the visited concepts to reveal user preferences and semi-automatically fill the user model with relevant data. To verify our design, we evaluate it with the user model used in the project [2] for adaptation in the domain of job offers.

Acknowledgments

This work was partially supported by the Science and Technology Assistance Agency under the contract No. APVT-20-007104 and the State programme of research and development "Establishing of Information Society" under the contract No. 1025/04.

References

1. Brusilovsky, P.: Methods and techniques of adaptive hypermedia. User Modeling and User-Adapted Interaction **6**(2-3) (1996) 87–129
2. Návrat, P., Bieliková, M., Rozinajová, V.: Methods and tools for acquiring and presenting information and knowledge in the web. In: International Conference on Computer Systems and Technologies – CompSysTech' 2005, Varna, Bulgaria (2005)
3. Paganelli, L., Paterno, F.: Intelligent analysis of user interactions with web applications. In: IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces, New York, NY, USA, ACM Press (2002) 111–118
4. Bra, P.D., Calvi, L.: AHA: a generic adaptive hypermedia system. In: 2nd Workshop on Adaptive Hypertext and Hypermedia. (1998) 5–12
5. Thomas, R., et al.: Generic usage monitoring of programming students. In Crisp, G., Thiele, D., Scholten, I., Barker, S., Baron, J., eds.: 20th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCILITE). (2003)
6. Etgen, M., Cantor, J.: What does getting wet (web event-logging tool) mean for web usability? (In: 5th Conference on Human Factors & The Web)
7. Judd, T., Kennedy, G.: Making sense of audit trail data. Australasian Journal of Educational Technology **20**(1) (2004) 18–32
8. Kay, J., Lum, A.: Creating user models from web logs. In: Intelligent User Interfaces Workshop: Behavior-Based User Interface Customization. (2004)

A.2 Príspevok prijatý na konferenciu ISD 2006

V tejto prílohe sa nachádza príspevok prijatý na konferenciu *Information Systems Development – ISD 2006*, ktorá sa konala v auguste 2006 v Budapešti. V príspevku porovnávame rôzne spôsoby reprezentácie modelu používateľa, uvádzame možnosti, ktoré poskytuje ontologická reprezentácia modelu. Predstavujeme model používateľa vyvíjaný v rámci projektu NAZOU[14] a nástroje, ktoré s ním pracujú.

[14] projekt NAZOU, `http://nazou.fiit.stuba.sk`

Ontology-based User Modeling for Web-based Information Systems

Anton Andrejko, Michal Barla and Mária Bieliková

Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, Slovak university of Technology in Bratislava, Slovakia
(andrejko, barla, bielik)@fiit.stuba.sk

The Web represents dynamically growing information space where the amount of the information that is provided by the web-based information systems grows exponentially. This calls for personalized interaction between a user and a web-based information system. Current web-based information systems provide certain level of personalization, which allows a user to set up her preferences (related mostly to the page layout) manually. Improving efficiency in information acquisition can be achieved by personalization where a concern is on user's particularities employed for adapting not only the layout but also the content or navigation through the information space. A user model that reflects the real user who requires information provided by an information system is the necessary part for the successful personalization. We present an ontology-based approach to the user modeling and describe the user model that we have designed for the web-based information system aimed at job acquisition. We point out several advantages of ontology-based approach, namely sharing ontology with other applications and the reusability.

1 Introduction

People are often overloaded by the information and finding the relevant one can be nearly impossible. This problem is yet more exposed in the information systems that cover large information space (e.g., the Web) where we suppose that individual users can have different knowledge and information needs. The system's general interface and behavior designed as "one size for all" is obviously not effective for all categories of users, thus adaptation is desired.

An approach to solve this problem sits in increasing efficiency of the user interaction with the information system by focusing on user's particularities. This brings to the web-based information systems an additional feature – personalization. A user model that reflects the real user who requires information provided by an information system is the necessary part for the successful personalization. The goal of this paper is to discuss possible user

model representations in order to show advantages and disadvantages resulting from ontology-based approach to the user modeling in web-based information systems. We discuss an approach to the user modeling where the model is expressed by an ontology and present the approach on an example of the user model being developed in the course of the research project Tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources [14]. In this project the Web is considered as an environment of heterogeneous information source and software tools are being developed for building web-based information system aimed at job acquisition. Since we model the user looking for a job, we assume that the application domain for examples throughout the paper is labor market.

2 User characteristics

User model represents various user characteristics, which can be used for adaptation of the content, presentation or the navigation. Kay and Lum define a user model as beliefs about the user that include preferences, knowledge and attributes for a particular domain [13]. At 9th International conference on User Modeling (2003) was the user model qualified as an explicit representation of properties of individual users or user classes [3].

Designers use for describing the user model terms like attributes, features, characteristics or properties. For the purpose of this paper we use the term *characteristic*. From differences that are not only in the terminology, it is obvious that the user modeling area needs to be standardized. First attempt to describe user modeling area provides User Modeling Meta-Ontology [20].

As an example, how the user's characteristics can help the adaptive process, let us assume that we have a characteristic in the user model expressing the minimal wage per month acceptable by the user. If the system knows that the user is not interested in job offers where offered wage per month is less (or much less to consider fuzzy nature of this characteristic) than her expected wage, it will not present her offers that do not fulfill this condition. The system filters the information on behalf of the user and the user model assists in this action.

Other actions, where a system can use the user model content, is an interpretation of the user's input [11] (which can be ambiguous, incomplete, with errors, etc.) and personalization of the system's output (sorting of the results, number of results per page, font, colors, etc.). The more relevant characteristics describing the user are included in the user model the more accurate can be the adaptation provided by the information system. Designers exploit invaluable knowledge of the specialists who work in the application domain for which the user model is designed to. Their experiences might help to construct the user model reflecting the real user as accurately as possible.

3 User model representations

There are several approaches to representing and storing a user model of the web-based information system. For user modeling it is important to analyze to what extent is particular representation flexible for different kinds of user characteristics in uniform manner together with possibility of reasoning directed to decisions on information content presented to the user. We do not discuss representation using proprietary formats as this would block almost totally the sharing and reuse of the user model.

3.1 Non-ontological representations

Markedly the most obvious is the use of the relational database to store data about the user as most of information systems already use this kind of application data storage. User model is represented in this case as a set of database tables. User characteristics are mapped to the attributes in relational data model, which store values assigned to the particular user characteristics.

Using relational database is quite straightforward, offers good performance and several other advantages like security, data recovery etc. that follow from good theoretical background of relational calculus and the maturity of its realization by database management systems. However, user models of web-based information systems contain often semi-structured data as they use an *overlay model*, which follows the representation of information space with various characteristics defined for particular concepts from the domain model. Relational databases are not primarily designated to express semi-structured data. Moreover, relational databases are not well suited when frequent changes in data structure need to be done, which is often the case of user modeling.

Other frequently used approach in current web-based adaptive systems is the user model represented by an XML based language using the file system. Representing the user model using an XML based language offers enough powerful expressiveness. As an example we give an open source general-purpose adaptive web-based system AHA! [6]. The part of the user model which stores information about the user's name is defined in the AHA! as follows:

```
<record>
  <key>personal.name</key>
  <value>John Smith</value>
  <firsttimeupdated>false</firsttimeupdated>
</record>
```

The performance of this solution is limited by the performance of used file system (it is effective for user models with a few instances and rich structure of user characteristics). Reusability and sharing is better than in the database approach, thanks to the platform independence of the used XML. Using XML has the advantage that it can be used directly in the Web environment. Problem is that XML is a meta-language and defines only the general syntax for

the model representation without formally defined semantics, which leads to difficulties when specifying a reasoning. Moreover, everyone can come with own name for tags; somebody stores attributes as tags; somebody uses the attributes of tags defined by XML syntax.

Both above mentioned approaches offer only the way how to describe user characteristics and do not offer any added value from the user modeling view. The way how to move the user modeling from the low level describing of the user characteristics to the higher level with additional possibilities offers ontology-based approach to the user modeling.

3.2 Representing user model by ontology

According the most cited definition of ontology in the Semantic Web community, an ontology is an explicit specification of the conceptualization of a domain [7]. The term ontology includes whole range of various models with various semantics richness. In this paper we consider representing the ontology by RDF[1]/OWL[2] formalisms. Approach based on RDF and its extension OWL takes the previous mentioned XML representation (syntax) and eliminates its disadvantage by defining a vocabulary for describing properties and classes. OWL serves as a common language for automated reasoning about the content for the vision of the Semantic Web.

For illustration, bellow is a fragment representing user's name and working experience that is a part of the ontology-based user model for job acquisition web-based information system:

```
<rdf:Description rdf:about="#name">
  <rdfs:label xml:lang="en">name</rdfs:label>
  <rdf:type rdf:resource= "http://www.w3.org/2002/07/
                                        owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#User"/>
  <rdfs:range rdf:resource=
                    "http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Description>
<rdf:Description rdf:about="#hasExperience">
  <rdfs:label xml:lang="en">has working experience</rdfs:label>
  <rdfs:domain rdf:resource="#User"/>
  <rdfs:range rdf:resource="http://www.fiit.sk/
                     classification#ExperienceClassification"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/
                                         owl#ObjectProperty"/>
</rdf:Description>
```

The advantages that lead to using ontologies for user modeling come from the fundamentals of this formalism. Ontologies provide a common understand-

[1] Resource Description Framework, `http://www.w3.org/RDF/`
[2] Web Ontology Language, `http://www.w3.org/2004/OWL/`

ing of the domain to facilitate reuse and harmonization of different terminologies [13]. They support reasoning, which is considered as important contribution of the ontology-based models. Having some user's characteristics represented, we can use the ontology and its relations, conditions and restrictions to provide the basis for infering additional user characteristics. For example, considering a user who is a programmer and works for a company that develops web-based applications using Java technologies we can infer that she is skillful in Java technologies.

By creating ontology-based user model and deriving it from the domain ontology, we raise the probability that the user's characteristics would be shared among a range of systems of the same domain (especially on the Web, where most ontologies is currently represented by the OWL). We consider user models sharing as one of main advantages of using ontologies for user modeling. One of the most obvious advantages of shared model is that one system can use the initialized data for personalization from the others what may prevent the user to type the same information again and again in every system (e.g., name, locale settings). However, the key advantage of shared user model is an availability of user's characteristics discovered by other systems as user characteristics acquisition is considered as a bottleneck of personalization.

As an example consider the web-based information system for the job acquisition discovering that the user's education is in the domain of information technologies with deep knowledge on object-oriented paradigm of programming. As the user searches for a job, she visits another adaptive job offer portal. Because it uses the same user modeling server it has access to information about user's education and automatically displays the offers demanding for specialists on object-oriented design on the first place.

Some authors believe that the solution to syntactical and structural differences between user models which interfere with sharing is in commonly accepted user model ontology [9]. As we agree that building common vocabularies is important and useful (we remark a role of standards), considering a large distributed information space (e.g., the Web) we need to come into some compromise between enabling diversity and looking for mappings between various models. Commonly accepted one user ontology is simply impossible to reach in such diverse and distributed environment.

For certain unified representation by ontologies can move the personalization on the Web further and give new possibilities of using user characteristics derived by other applications. Considering structural unification a problem arises when the applications using the shared user model evaluate some user's characteristic differently. This characteristic would constantly change as the user uses various applications, which can lead to unfit personalization among all applications using the mentioned characteristic. One solution to this problem is to keep track of model changes [18]. This would allow each application to use this track as an additional information for personalization.

4 User model for job acquisition domain

We developed the user model and software tools for its employing for personalization in the context of research project aimed at support of acquisition, organization and presentation of information on the Web [14, 15]. The outcome of the project is the web-based information system in domain of labor market (both for people who are looking for a job and companies which are looking for employees). The system itself is being developed by means of several cooperating software tools, which support various stages of data-information-knowledge transformation from raw unknown data from the Web to presented information and knowledge related to specific interests of particular user.

Data and knowledge repository is designed as a heterogeneous space [5] where several formalisms for representation are used. Raw data from the Web are stored as files, data extracted from the Web source files are stored in relational database, domain model together with the user model both used for personalized presentation of acquired job offers are represented by ontologies.

4.1 User model ontology

We have designed the user model in several iterations according a user dependency criterion, which divides user characteristics into domain-dependent and independent groups. This criterion has led us to representing the user model by two sub-ontologies corresponding to mentioned groups of characteristics:

- `DomainDependentUser` – domain-dependent part of the user ontology describes domain-dependent user characteristics,
- `DomainIndependentUser` – contains domain-independent part of the user ontology that reflects domain-independent user characteristics.

Domain-dependent part

Vocabulary of the domain-dependent part of the user model goes out from the domain model ontology developed for the project. Domain ontology here represents explicit conceptualization of job offers. It profits from the advantage of ontology reuse and uses also other ontologies (which domain is independent from labor market domain) to achieve the desired conceptualization.

The domain model consists of the following ontologies:

- ontology *classification* (prefix "c") – hierarchies for industrial segments, professions, educational levels, qualifications and various organizations;
- ontology *region* (prefix "r") – domain of the regions, countries, languages and currencies that are used in these regions;
- ontology *offer* (prefix "ofr") – hierarchies for industrial segments, professions, educational levels, qualifications and various organization;
- ontology *classification* (prefix "c") – general offer domain; general offer is represented by the `ofr:Offer` class; any offer has a source and a validity interval.

The `JobOffer` class is the key class of the ontology. It represents the standalone job offer. `JobOffer` has several object and datatype properties. Selected object properties of the `JobOffer` class are shown in the Figure 1.

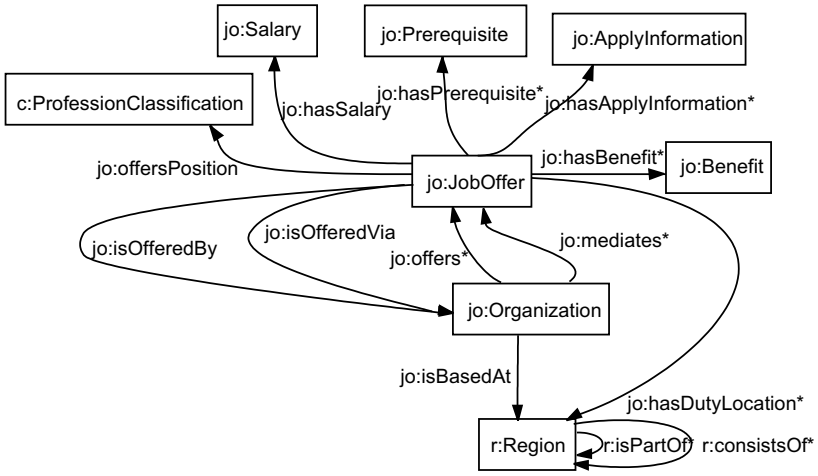


Fig. 1. Selected object properties of the `JobOffer` class.

Mentioned ontologies provide the base for the user model and a mapping between the domain and user models. Domain-dependent part of the user model exploits domain model sub-ontologies – classification, region and offer. In the Figure 2 is depicted domain-dependent part of the user model that describes user's preferences towards the job offer. The term *preference* expresses user's expectations towards the application domain. In the case of job offer domain we store in the user model information about user's desirable job offer.

The user can specify following attributes:

- *ContractType* – user has a chance to choose from these options: contract, non executive, permanent, self employed, temporary.
- *Region* – defines the destination where the looking job should be engaged,
- *Salary* – expected wage,
- *DomainClassification* – defines the domain of user's interests, in which the job position should be engaged (usually the standard study programs at universities).

Information about job offers that the user has already visited plays important role in the personalization. It is modeled by the `VisitedJobOffer` class. The user model stores information about the source of the visited offer (`relatesTo`) and evaluation assigned by the user (`hasRating`).

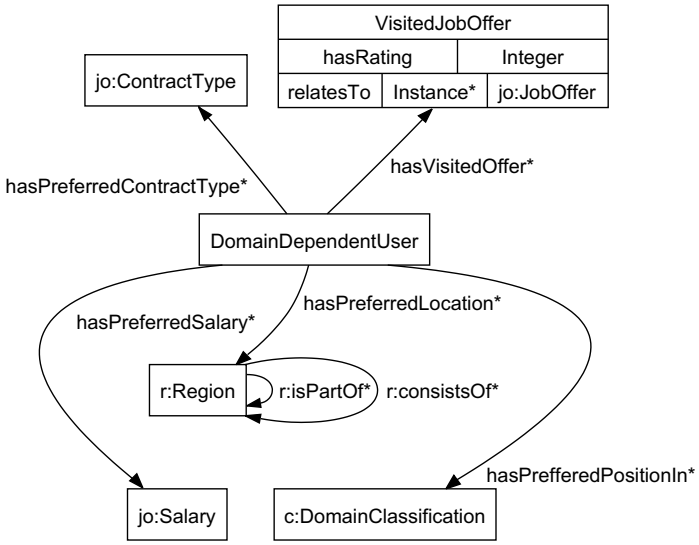


Fig. 2. Domain-dependent part of the user model.

Domain-independent part

Domain-independent part involves user characteristics that describe a user as a person. This part consists of datatype and object properties (see Figure 3). Datatype properties are `dateOfBirth` and `name`. Object properties are mapped only to the domain ontology parts that are independent from our labor market domain. That way we can express user's language skills including mother language (using reference `hasLanguageKnowledge`), personal characteristics, e.g. gender (`hasPersonalAttribute`), computer skills (`hasUserLevel`) and previous education (`hasEducation`).

4.2 Tools for user modeling support

We employ described user model to provide a personalization [1]. To fill the user model with data we observe user's actions within the web based system combined with an explicit input from the user. However, we stress on automatic part of the user characteristics acquisition. We collect as much data as possible about the user's actions by employing standard server side logging mechanisms as well as special client side javascript logging tool called *Click*. Click records such actions, which are not visible on the server (e.g., reload of a page stored in browser's cache, hover on page elements, or using the back button, which is important for discovering user interests on the portal).

Afterwards, collected data is analyzed to estimate selected user's characteristics. Estimation uses heuristics and predefined patterns of navigation on the site. Some heuristics need to compare two domain concepts to find out

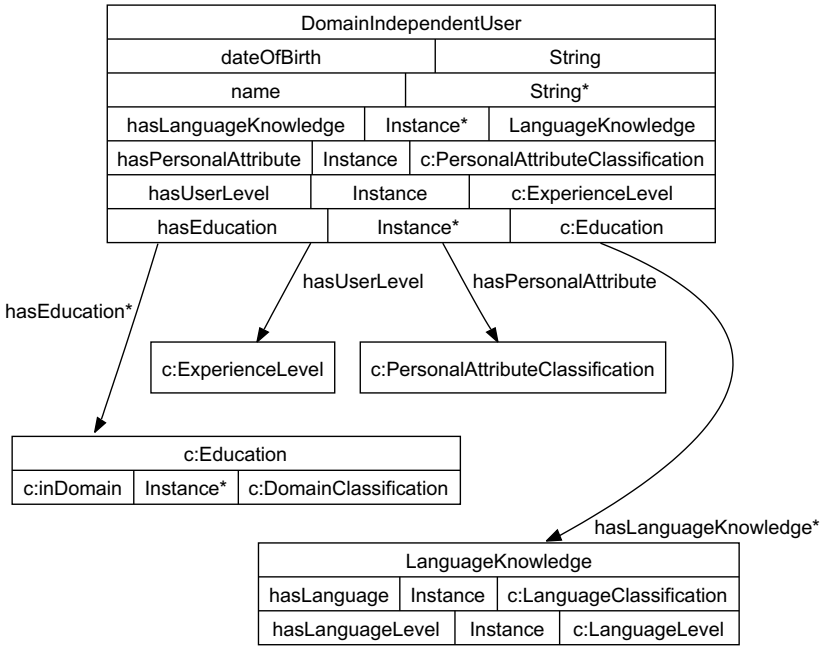


Fig. 3. Domain-independent part of the user model.

their common and different aspects. Once an instance of a model exists in a system, it can be used by other components of the system to perform adaptation itself, which can be of various types – annotation of displayed content, its sorting etc. Therefore, the comparison between the user ontology and the domain ontology instances is necessary. This process is not straightforward.

In contrast to the common attribute-value models, ontology provides structured data and a one-on-one comparison does not provide satisfactory results because two individuals usually may provide semantically similar information even though they are not on the same level in the hierarchical structure of ontology. Since a part of the structure of the ontology is known (property `subClassOf`) we use a recursive algorithm to traverse the hierarchy. Because we consider attributes that have identical parent nodes to be closer, we take into account both a straight path between attributes in the hierarchy and also what branch of the hierarchy tree they belong to.

The user characteristics are used by several software tools aimed at further refinement of user interests. *Top-k aggregator* tool retrieves some top job offers with respect to user preferences (e.g., salary, education requirement, place) based on ordered lists of user preferences [8]. *Aspect* tool searches for similar documents (job offers) based on probabilistic model for soft clustering [16]. Using described approach to the comparison of domain and user ontology instances devised clusters are presented the user according her characteristics.

5 Related works

Although several possible representations of a user model are currently used, the user modeling community has changed focus recently to ontology based approach, which brings several advantages as discussed above. Several projects, which are either concentrating on building reusable user model ontologies or employ the user model ontology as a part of adaptive web-based system, exist.

An extension to the XML structure to be able represent graph structure adds UserML – the RDF-based user model exchange language [10]. It uses two cooperative levels. First one defines simple XML structure for the user model entries and second one are categories defined in the ontology. The advantage of this approach is that different ontologies can be used with the same UserML tools. UserML served as a base for reusable user model ontology GUMO – General User Model Ontology [9] represented by the OWL language. GUMO provides a collection of the user's dimensions (e.g., user's heart beat, age, position, etc.) that might be helpful for several information systems intending to provide personalization based on the user model. These characteristics can be shared also with our user ontology when the web-based information system realizes adaptation according such personal characteristics.

OntobUM (Ontology based User Model) is a generic ontology-based user modeling architecture developed for knowledge management system [17]. The user model consists of implicit and explicit part. While explicit part contains characteristics such as identity, preferences, the implicit part is related to experiences related to the system usage. Our criterion based on domain dependence extends this classification.

Among the projects, which uses the ontology based user model representation we mention ADAPT^2 – Advanced Distributed Architecture for Personalized Teaching & Training [4]. It stands for a general framework for distributed education. This framework employs Ontology Server to user model exchange.

The idea of shared user model is elaborated also in [12]. Here, Personis server that uses a proprietary representation of a model based on triplets *component–evidence–source* is described. There is not an explicit definition of triplets semantics, each application can define its own triplets not regarding the others, which limits the reusability. Another project, UMoWS [2] uses OWL representation of a model. Because the same knowledge can be represented by different ontologies on different levels of abstraction UMoWS supports the representation in multiple ontologies and can provide the mapping between them to the applications, so they can really share a model.

6 Conclusions

In this paper we discussed ontology-based representation of user models aimed at providing personalization in web-based information systems. We concentrated on comparison with other approaches to user model representation and pointed advantages of ontology-based representation.

We consider the simplification of exchanging user model data between different applications as the major advantage of using ontology. Presented ontology developed in the course of research project aimed at job acquisition application domain contributes to the state of the art by a separation of domain-independent and domain-dependent parts of the user model. Separating the domain-independent part of user characteristics allows us to build general user model. This kind of the user model can be used in wide-range applications only with adding parts, which differs from one application to another. Actually we use presented ontology for the user model for research project aimed at developing a recommendation layer for digital library that serves for personalization its services.

Ontology is not the only representation that is advantageous for user modeling in web-information systems. Systems that build user model based on the monitoring a user represent logs that are also considered as a part of the user model using simpler data structures as ontologies (often XML file is sufficient). Another example of non-ontological part of the user model is statistics related to the user behavior. Ontology representation is advantageous for such part of the user model that is related to user characteristics where some reasoning is useful, i.e. the checking consistency of values for various user characteristics.

Software tools working on the user ontology mentioned in the paper can be used also for other domains as that of labor market. The tools are designed to be domain independent realizing their methods with optional domain dependent layer. Navigation in the domain is done through a faceted semantic browser [19]. This tool is also responsible for creation server side logs enhanced with semantics, which is designed for the use in various application domains.

References

1. A. Andrejko, M. Barla, M. Bieliková, and M. Tvarožek. Tools for user characteristics acquisition. In *Web Personalisation, Recommender Systems and Intelligent User Interfaces Workshop, AH'06*, 2006. Submitted.
2. M. Bieliková and J. Kuruc. Sharing user models for adaptive hypermedia applications. In *5th Int. Conf. on Intelligent Systems Design and Applications, ISDA'05*, pages 506–511, Wroclaw, Poland, 2005. ACM Press.
3. P. Brusilovsky, A. Corbett, and F. de Rosis. User modeling 2003: Preface. In *9th Int. Conf. on User Modelling*, Johnstown, USA, 2003. Springer, LNCS 2702.
4. P. Brusilovsky, S. Sosnovsky, and M. Yudelson. Ontology-based framework for user model interoperability in distributed learning environments. In G. Richards, editor, *E-Learn 2005*, pages 2851–2855, Vancouver, Canada, 2005. AACE.
5. M. Ciglan, M. Babik, M. Laclavik, I. Budinska, and L. Hluchy. Corporate memory: A framework for supporting tools for acquisition, organization and maintenance of information and knowledge. In *9th Int. Conf. on Inf. Systems Implementation and Modelling, ISIM'06*, Prerov, Czech Republic, 2006.
6. P. de Bra, A. Aerts, B. de Lange, B. Rousseau, T. Santic, D. Smith, and N. Stash. AHA! the adaptive hypermedia architecture. In *ACM Conf. on Hypertext and Hypermedia*, pages 81–84, Nottingham, UK, 2003.

7. T.R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.
8. P. Gurský, R. Lencses, and P. Vojtáš. Algorithms for user dependent integration of ranked distributed information. In M. Böhlen, J. Gamper, and M. Wimmer, editors, *TCGOV 2005 Poster Proceedings*, Bozen-Bolzano, Italy, 2005.
9. D. Heckmann, T. Schwartz, B. Brandherm, M. Schmitz, and M. von Wilamowitz-Moellendorff. Gumo – the general user model ontology. In L. Ardissono, P. Brna, and A. Mitrovic, editors, *10th Int. Conf. on User Modeling, UM'05*, pages 428–432, Edinburgh, Scotland, UK, 2005. Springer, LNCS 3538.
10. Dominik Heckmann and Antonio Krueger. A user modeling markup language (UserML) for ubiquitous computing. In P. Brusilovsky, A. Corbett, and F. de Rosis, editors, *9th Int. Conf. on User Modelling, UM'03*, pages 393–397, Johnstown, USA, 2003. Springer, LNCS 2702.
11. J. Kay. User modeling for adaptation. In C. Stephanidis, editor, *User Interfaces for All*, Human Factors Series, pages 271–294, Florence, Italy, 2000.
12. J. Kay, B. Kummerfeld, and P. Lauder. Personis: A server for user models. In P. de Bra, P. Brusilovsky, and R. Conejo, editors, *2nd Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems, AH'02*, pages 29–31, Malaga, Spain, 2002. Springer, LNCS 2347.
13. J. Kay and A. Lum. Ontology-based user modeling for the semantic web. In *10th Int. Conf. on User Modeling, UM'05, Workshop 8*, pages 11–19, Edinburgh, Scotland, UK, 2005.
14. P. Návrat, M. Bieliková, and V. Rozinajová. Methods and tools for acquiring and presenting information and knowledge in the web. In *Int. Conf. on Computer Systems and Technologies, CompSysTech 2005*, Varna, Bulgaria, 2005.
15. G. Nguyen, M. Laclavik, Z. Balogh, E. Gatial, M. Ciglan, M. Babik, I. Budinska, and L. Hluchy. Data and knowledge acquisition with ontology background. In W. Abramowicz, editor, *Business Information Systems, BIS'06*, Poznan, Poland, 2006.
16. G. Polčicová and P. Tiňo. Making sense of sparse rating data in collaborative filtering via topographic organisation of user preference patterns. *Neural Networks*, 17:1183–1199, 2004.
17. L. Razmerita, A. Angehrn, and A. Maedche. Ontology-based user modeling for knowledge management systems. In *9th Int. Conf. on User Modeling, UM'03*, pages 213–217, Johnstown, PA, USA, 2003. Springer, LNCS 2702.
18. M. Tury and M. Bieliková. Change indentification in ontologies. In P. Vojtáš, editor, *ITAT 2005 – Workshop on Theory and Practice of Information Technologies*, pages 381–390, 2005.
19. M. Tvarožek. Personalized navigation in the semantic web. In V. Wade, H. Ashman, and B. Smyth, editors, *4th Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems, AH'06*, pages 467–471, Dublin, Ireland, 2006. Springer, LNCS 4018.
20. M. Yudelson, T. Gavrilova, and P. Brusilovsky. Towards user modeling meta-ontology. In L. Ardissono, P. Brna, and A. Mitrovic, editors, *10th Int. Conf. on User Modeling, UM'05*, pages 448–452, Edinburgh, Scotland, UK, 2005. Springer, LNCS 3538.

A.3 Príspevok prijatý na konferenciu Datakon 2006

V tejto prílohe sa nachádza príspevok prijatý na konferenciu *Datakon 2006*, ktorá sa konala v nobembri 2006 v Brne. V príspevku predstavujeme ret'az nástrojov podporujúcich proces automatizovaného získavania charakteristík používateľa.

Softvérové nástroje pre získavanie charakteristík používateľa

Anton ANDREJKO, Michal BARLA, Mária BIELIKOVÁ, Michal TVAROŽEK

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
{andrejko,barla,bielik,tvarozek}@fiit.stuba.sk

Abstrakt. Prispôsobovanie sa charakteristikám používateľov sa stáva významnou vlastnosťou súčasných webových informačných systémov. Bez prispôsobovania prezentácie môže byť veľmi ťažké a dokonca často nemožné používať tieto systémy na efektívnu prácu s informáciami. Na prispôsobovanie je potrebné poznať charakteristiky jednotlivých používateľov. V príspevku opisujeme získavanie charakteristík používateľa pomocou spolupracujúcich softvérových nástrojov, kde každý nástroj vykonáva špecifickú časť tohto procesu. Charakteristiky získavame na základe monitorovania správania sa používateľa, čo minimalizuje mieru potrebného zapojenia používateľa do procesu tvorby modelu používateľa. Opisujeme štyri nástroje s dôrazom na ich vzájomné prepojenia, ktoré vedú k získavaniu charakteristík používateľa.

Kľúčové slová: model používateľa, objavovanie charakteristík používateľa, vzory správania, zaznamenávanie akcií používateľa.

1 Úvod

Dôležitou súčasťou adaptívnych webových systémov je modelovanie používateľa, ktoré zahŕňa najmä získanie a vyhodnotenie charakteristík používateľa tak, aby sa mohli použiť pri prispôsobovaní. S nárastom množstva dostupných informácií sa prispôsobovanie zamerané na konkrétneho používateľa – personalizácia stáva nevyhnutnou súčasťou súčasných webových informačných systémov. Dopyt po generických riešeniach adaptívnych systémov vyúsťuje do potreby metód a nástrojov, ktoré umožňujú automatické získavanie a vyhodnocovanie charakteristík používateľa.

Vytvorenie vhodného modelu používateľa je zložitý proces, ktorý zahŕňa viacero úloh. K ich realizácii môžeme pristúpiť vytvorením sady softvérových nástrojov, ktoré sa špecializujú na jednotlivé úlohy. Príkladom môže byť nástroj, ktorý spája informácie získané od používateľa počas registrácie z príslušných formulárov s informáciami získanými z iných nástrojov do modelu používateľa, ktorého cieľom je reprezentovať dobre odhadnuté charakteristiky používateľa.

Cieľom nášho výskumu je tvorba modelu používateľa na odporúčanie vhodných ponúk z konkrétnej domény (napr. pracovné alebo cestovné ponuky), ktorý tvorí základ pre personalizovanú navigáciu a filtrovanie obsahu pre jednotlivých používateľov. Na vyhodnotenie výsledkov nášho výskumu používame doménový model a model

používateľa vytvorený v rámci výskumného projektu „Nástroje pre získavanie, organizovanie a udržovanie znalostí v prostredí heterogénnych informačných zdrojov“ [6], ktorý sa orientuje na doménu pracovných ponúk.

V tomto príspevku prezentujeme niekoľko softvérových nástrojov, ktoré podporujú proces získavania charakteristík používateľa spolu s ich prepojeniami. V časti 2 prezentujeme proces vytvárania modelu používateľa a možné zdroje informácií, ktoré do procesu vstupujú. Definujeme sadu nástrojov, ktoré realizujú potrebné úlohy. V časti 3 sa venujeme špecifikám fázy zberu dát a opisujeme príslušné nástroje tejto fázy (*Click* – Client Side Action Recorder a *SemanticLog*). V časti 4 opisujeme spracovanie zozbieraných informácií do podoby odhadov charakteristík používateľa a ich realizáciu nástrojom *LogAnalyzer*. Nakoniec sumarizujeme príspevok a závery z našej práce prezentovanej v tomto príspevku.

2 Proces vytvárania modelu používateľa

Tvorba modelu používateľa pozostáva z dvoch hlavných fáz. Najprv sa získajú dáta o správaní používateľa, ktoré sa ďalej analyzujú a interpretujú s cieľom objavenia relevantných charakteristík používateľa. Takto objavené charakteristiky sa použijú pri personalizácii. Existuje viacero spôsobov získavania dát o používateľoch:

- formuláre, ktoré umožňujú používateľom priamo zadať požadované charakteristiky alebo odpovedať na špecifické otázky, ktoré charakteristiky odhaľujú nepriamo;
- explicitná spätná väzba používateľa na zobrazený obsah; spätná väzba môže byť buď pozitívna alebo negatívna;
- dokumenty dodané používateľom (napr. CV), z ktorých sa dajú extrahovať relevantné charakteristiky;
- záznam prístupov používateľa k zdrojom; používa sa na odvodenie stupňa znalosti používateľa o prezentovaných informáciách;
- záznam správania používateľa v systéme, ktorý zahŕňa implicitnú spätnú väzbu používateľa na zobrazený obsah.

Keďže je vhodné minimalizovať mieru spoluúčasti používateľa na procese vytvárania modelu používateľa, sústredíme sa na modelovanie používateľa založené na pozorovaní správania používateľa a automatickom odvodení jeho charakteristík. Identifikovali sme sadu softvérových nástrojov, ktoré pokrývajú obidve fázy procesu. V tomto príspevku opisujeme nástroje *Click* a *SemanticLog*, ktoré sú určené pre fázu získavania dát a nástroj *LogAnalyzer*, ktorý je zameraný na objavovanie charakteristík používateľa v získaných dátach.

3 Zber dát

Existujú dva hlavné prístupy monitorovania akcií používateľa: monitorovanie vykonávané na strane servera a monitorovanie na klientskej strane systému, pričom sa môže použiť aj kombinácia oboch prístupov.

Monitorovanie na strane servera sleduje prístupy používateľa ku zdrojom. Hlavná nevýhoda tohto prístupu je v tom, že neumožňuje zaznamenávanie niektorých typov interakcie používateľa so systémom (napr. interakcia s prvkami formulára) a neposkytuje presné informácie o čase zobrazenia informácií. Monitorovanie na strane servera závisí od správania webových prehliadačov, ktoré zvyčajne nežiadajú znovu od servera už navštívené stránky, ale použijú namiesto toho kópiu uloženú vo vyrovnávacej pamäti. Systém teda nemá žiadne informácie o dobe, ktorú používateľ strávil prezeraním určitej stránky. Väčšina v súčasnosti najrozšírenejších webových prehliadačov nerešpektuje direktívy HTTP protokolu, ktoré zakazujú používanie lokálnej vyrovnávacej pamäte, takže ich použitie nerieši uvedený problém. Takisto treba vziať do úvahy znížený komfort používania systému bez vyrovnávacej pamäte. Aj v [8] autori uvádzajú závery, že pre získanie presných záznamov o interakcii používateľa so systémom treba využiť monitorovanie na strane klienta. Aj napriek vyššie uvedeným problémom je prístup s využitím záznamov servera postačujúci pre veľa adaptívnych systémov. Napr. systém AHA! (Adaptive Hypermedia for All, `aha.win.tue.nl/`) tieto záznamy využíva pre sledovanie toho, aké dokumenty sa už zobrazili používateľovi [1].

Monitorovanie na strane klienta vytvára podrobný záznam akcií používateľa s presnými časovými pečiatkami. Monitorovanie sa môže vykonávať špeciálnou klientskou aplikáciou (napr. *User Action Recorder* v [10]) alebo použitím klientských webových technológií akými sú JavaScript alebo Java applety. Pretože považujeme prvý zmienený prístup za pomerne invazívny a málo flexibilný, sústreďujeme sa druhý prístup. JavaScript a Java applety v súčasnosti podporujú všetky významné webové prehliadače na hlavných platformách. Možnou nevýhodou je, že nie každý používateľ akceptuje takéto podrobné monitorovanie a niektorí môžu vo svojich prehliadačoch zablokovať vykonávanie vložených skriptov. Napokon môže nastať aj prípad, keď používateľ nemá na počítači nainštalovaný potrebný softvér.

Existuje viacero nástrojov, ktoré využívajú JavaScript na monitorovanie na strane klienta ako sú WebVip (Web Variable Instrumenter Program, `zing.ncsl.nist.gov/WebTools/WebVIP/overview.html`) alebo WET (Web Event-logging Tool, [3]). Obidva nástroje sú primárne určené pre účely vyhodnocovania použiteľnosti webových stránok. Nevýhodou nástroja WebVIP je vytvorenie kópie celého webového sídla, do ktorej nástroj do jednotlivých stránok pridáva identifikátory a obsluhy udalostí na HTML odkazoch. Nástroj sa preto nedá použiť pri dynamicky vytváraných webových stránkach, keďže mu nedokážeme dodať kompletné webové sídlo. Ďalšou nevýhodou nástroja WebVIP je jeho obmedzenie na štandardné HTML odkazy ignorujúce ďalšie HTML elementy stránok. Aj keď nástroj WET rieši uvedené problémy, je príliš naviazaný na doménu vyhodnocovania stránok, čo znemožňuje jeho použitie ako všeobecného nástroja na zaznamenávanie akcií používateľa. Nástroj vytvára nad stránkou ďalšiu vrstvu, ktorá obsahuje tlačidlá na ovládanie procesu zaznamenávania, pričom dáta sa posielajú na server až po stlačení tlačidla stop.

Z uvedenej analýzy vyplýva, že pri použití monitorovania na strane klienta sa vystavujeme riziku, že používateľ nebude súhlasiť s podrobným monitorovaním, čo by znamenalo, že nezískame žiadne dáta. To je silný argument proti použitiu čistého monitorovania na strane klienta. Monitorovanie na strane servera je z tohto pohľadu spoľahlivejšie, keďže sa mu vždy podarí získať nejaké dáta. Nevýhodou je, že môžeme stratiť časový aspekt zaznamenaných akcií. Náš prístup je založený na myšlienke

skombinovania oboch prístupov – naše nástroje extrahujú maximum informácií na strane servera a používajú záznam z klientskeho monitoringu ako zdroj prídavných, časovo presných informácií o aktivitách používateľa.

3.1 Nástroj na zaznamenávanie akcií používateľa na strane klienta

Click je nástroj, ktorý realizuje monitorovanie používateľa na strane klienta a zachytáva udalosti, ktoré vyvoláva webový prehliadač počas interakcie používateľa s elementmi zobrazovaných stránok:

- *Load:* zobrazenie stránky používateľovi,
- *Unload:* keď používateľ stránku opustí,
- *Click:* keď používateľ nasleduje niektorý odkaz na stránke,
- *Mouseover:* keď používateľ nasmeruje kurzor na aktívny element stránky,
- *Mouseout:* keď používateľ odstráni kurzor z aktívneho elementu stránky.

Uvažujeme aj zachytávanie ďalších udalostí, ktoré sú špecifické pre stranu klienta a nie sú dostupné na serveri. Medzi také udalosti patrí udalosť *change*, ktorá sa vyvolá po zmene obsahu prvku formulára. Sekvencia týchto udalostí informuje o poradí, v ktorom používateľ vypĺňal formulár. Iným príkladom je udalosť *scroll*, ktorá sa vyvolá, ak sa používateľ posunie po stránke.

Click zbiera nasledovné údaje o každej zachytenej udalosti:

- typ udalosti,
- čas spustenia udalosti a
- kontext zachytenej udalosti, napr. ktorý odkaz používateľ nasledoval.

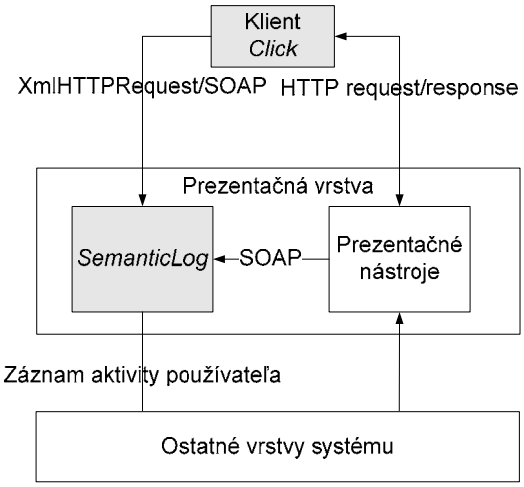
Komunikácia so serverom je realizovaná v dávkach *N* zachytených udalostí. Ak je *N* nastavené na jedna, *Click* posiela na server záznam po každej zachytenej udalosti.

Click je implementovaný s využitím JavaScript technológie, ktorá nekladie žiadne špeciálne požiadavky na klienta, keďže je podporovaná väčšinou súčasných webových prehliadačov. JavaScript poskytuje natívny prístup k objektovému modelu HTML dokumentu (DOM), takže *Click* má prístup k štruktúre aj obsahu stránky.

Zachytávanie udalostí je založené na špecifikácii *DOM Level 2 Events* (`www.w3.org/TR/DOM-Level-2-Events`), ktorá definuje obsluhovače udalostí dynamicky priraďované k elementom objektového modelu. To umožňuje jednoducho integrovať nástroj do existujúcich ako aj dynamicky generovaných HTML stránok pridaním odkazu na príslušný skript v hlavičke HTML dokumentu.

Komunikácia so serverom je asynchrónna s použitím objektu *XmlHTTPRequest* (pozri obr. 1), ktorý predstavuje jadro technológie AJAX. Nástroj *Click* posiela záznamy zachytených udalostí na SOAP rozhranie nástroja *SemanticLog*. Ten agreguje získané dáta s informáciami získanými z prezentačných nástrojov, ktoré poznajú sémantiku zobrazeného obsahu. Nakoniec sa záznam obohatený o sémantiku zašle na ďalšie spracovanie vnútornými vrstvami systému.

Serverovú časť monitorovacieho systému sme realizovali ako samostatný nástroj, ktorý prijíma záznamy udalostí vytvárané nástrojom *Click* cez SOAP rozhranie webovej služby. Pre volanie metódy webovej služby z prostredia JavaScript používame knižnicu od IBM (Call SOAP Web services with Ajax).



Obr. 1. Štruktúra časti pre získavanie dát.

3.2 Nástroj na zaznamenávanie udalostí so znalosťou sémantiky

SemanticLog je generický nástroj na vytváranie serverového záznamu, ktorý vykonáva dve hlavné úlohy. Prvou je zaznamenávanie udalostí dodaných od nástrojov prezentačnej vrstvy, druhou je prijímanie a zaznamenávanie udalostí od nástrojov vykonávajúcich monitorovanie na strane klienta ako napr. nástroj *Click.*

Nástroj je implementovaný ako webová služba s verejným rozhraním, ktoré umožňuje ostatným nástrojom posielať dáta, ktoré sa spájajú do jedného spoločného záznamu. Napr., fazetový prehliadač, ktorý slúži v projekte [6] ako prezentačný nástroj, môže zaznamenávať akcie používateľa pomocou SOAP protokolu. Keďže nástroj pozná sémantiku vykonaných akcií a "rozumie" obsahu, ktorý produkuje, dokážeme v konečnom dôsledku vytvoriť podrobnejší a zároveň sémanticky obohatený záznam v porovnaní so štandardným záznamom webového servera.

Napr. ak používateľ kliknutím vyberie ohraničenie informačného priestoru založené na mieste výkonu práce zo zoznamu možných miest, nástroj nezaznamená *URL* odkazu, na ktorý používateľ klikol, ale význam vykonanej akcie spolu s *URI* zvolenej inštancie tak, ako je uložená v ontológii.

Príklad udalosti zaznamenanej nástrojom *SemanticLog:*

```
<event>
        <datetime>2006-04-23 00:54:06</datetime>
        <type>SelectRestriction</type>
        <facet>Region</facet>
        <value>http://job-offers/region#US_AZ</value>
</event>
```

Navyše, nástroje pre monitorovanie na strane klienta, ako napr. *Click*, môžu tiež využiť SOAP rozhranie a dodať udalosti z klientskej strany interakcie ako napr. použitie tlačidla „späť" alebo umiestnenie kurzora nad vybranými položkami. Tieto informácie možno asociovať s udalosťami získanými na strane servera a vytvoriť tak podrobný záznam interakcie používateľa so systémom.

Hlavnou výhodou navrhnutého prístupu je zachovanie sémantiky akcií, ktoré vykonáva používateľ ako aj sémantiky zobrazených stránok pre ďalšie spracovanie nástrojmi pre analýzu správania používateľa.

Nástroje, ktoré spracúvajú záznamy potrebujú mať informácie o zaznamenaných URL adresách a asociovaných parametroch tak, aby „pochopili“ sémantiku obsiahnutú v záznamoch, čo vedie k silnej zviazanosti jednotlivých nástrojov. Pridanie ďalšieho prezentačného nástroja alebo prípadná zmena už existujúceho nástroja vedú vždy k (väčším) zmenám nástrojov analyzujúcich záznamy.

Nástroj *SemanticLog* rieši problém silnej zviazanosti prezentačných nástrojov s nástrojmi pre analýzu poskytnutím spoločnej reprezentácie udalostí získaných od všetkých prezentačných nástrojov. Ďalšou výhodou použitia nástroja *SemanticLog* oproti tradičnému zaznamenávaniu URL adries je teda aj to, že nástroje pre analýzu záznamov (napr. *LogAnalyzer*) nepotrebujú rozumieť špecifickým parametrom jednotlivých prezentačných nástrojov a sú teda len voľne zviazané.

4 Objavovanie charakteristík používateľa

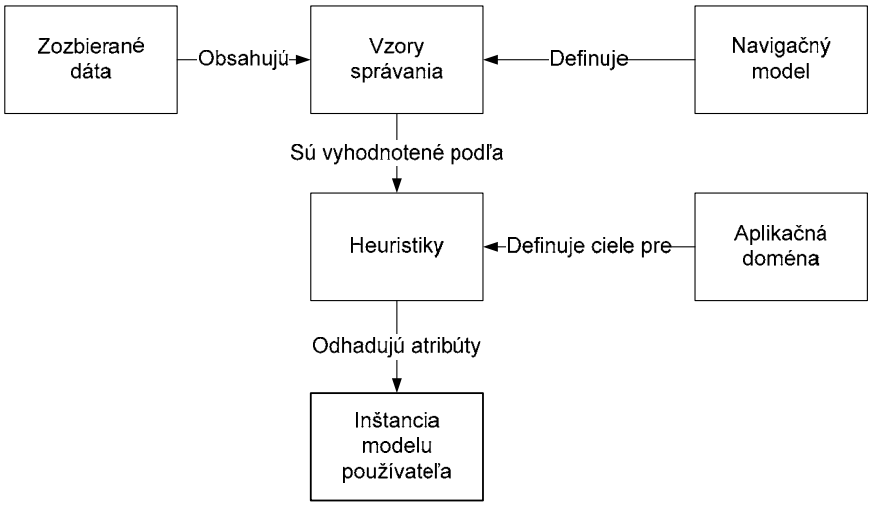
Po fáze zberu dát treba interpretovať ich význam, ktorý nepriamo súvisí so zachyteným správaním používateľa – buď zo správania vyplýva alebo je s ním asociovaný [4] (napr. identifikovať ciele alebo odhadnúť znalosti používateľa o určitých konceptoch). Niektoré charakteristiky vieme odvodiť zo spôsobu navigácie používateľa vo webovom sídle. Odvodenie predpokladá použitie takého navigačného modelu, ktorý umožňuje používateľovi relatívne slobodný pohyb medzi jednotlivými stránkami tak, aby sa v správaní prejavili charakteristiky používateľa.

Iný spôsob odhadu charakteristík používateľa predstavuje analýza reakcií používateľa na zobrazený obsah, ktorá vyhodnocuje implicitnú spätnú väzbu vyplývajúcu z akcií používateľa [7].

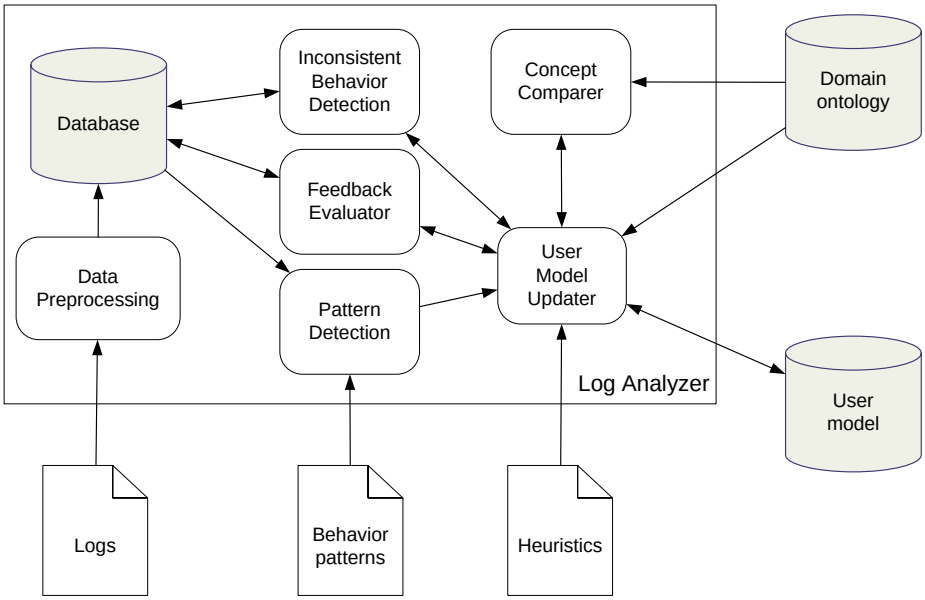
V obidvoch prípadoch hľadáme preddefinované vzory správania v postupnosti akcií používateľa vo webovom sídle. Následne využijeme heuristiku asociovanú s identifikovaným vzorom a vykonáme príslušnú zmenu v modeli používateľa. Heuristiky sú definované s ohľadom na typické ciele používateľa v príslušnej aplikačnej doméne. Obr. 2 zobrazuje zdroje informácií na vytvorenie alebo aktualizovanie inštancie modelu používateľa.

Na analýzu zozbieraných dát s využitím vyššie uvedených princípov sme navrhli softvérový nástroj *LogAnalyzer*, ktorý na základe analýzy zozbieraných dát upravuje model používateľa.

Činnosť nástroja je rozdelená do troch samostatných fáz: (i) predspracovanie získaných dát, (ii) objavovanie charakteristík používateľa a (iii) aktualizácia modelu používateľa. Architektúru nástroja *LogAnalyzer* sme navrhli s ohľadom na tieto tri fázy (pozri Obr. 3). Prvú fázu realizuje modul *DataPreprocessing*. (predspracovanie dát) Druhú fázu, objavovanie charakteristík používateľa, je zabezpečujú moduly *PatternDetection* (odhaľovanie vzorov), *FeedbackEvaluator* (vyhodnocovanie spätnej väzby) a *IncosistentBehaviorDetection* (odhaľovanie nekonzistentného správania). Poslednú fázu – aktualizáciu modelu používateľa, realizuje modul *UserModelUpdater* (aktualizácia modelu používateľa), ktorý používa modul *ConceptComparer* (porovnávanie konceptov).



Obr. 2. Zdroje informácií pre aktualizovanie inštancie modelu používateľa.



Obr. 3. Architektúra nástroja *LogAnalyzer*.

Predspracovanie dát

Modul *DataPreprocessing* identifikuje v získaných záznamoch jednotlivé sedenia používateľa a uloží dáta v podobe vhodnej pre ďalšiu analýzu. V tejto fáze sa vykonajú aj ďalšie transformácie dát, ktoré odstránia identické, po sebe idúce akcie (napr. keď používateľ kvôli pomalej odozve systému zopakuje vykonanú akciu).

Dôležitou úlohou tohto modulu je úprava hodnôt, ktoré predstavujú dobu čítania stránky. Zvyčajne je táto doba vo vzťahu so záujmom používateľa o obsah danej stránky, avšak ak je extrémne dlhá, môžeme predpokladať, že používateľ prestal pracovať so systémom a nevenuje sa zobrazenému obsahu [9]. Preto čas upravujeme do primeraných intervalov.

Odhaľovanie nekonzistentného správania

Modul *InconsistentBehaviorDetection* realizuje metódu dolovania sekvenčných vzorov nad predchádzajúcimi sedeniami používateľa. Získané vzory pokladáme za typické vzory správania používateľa. Aktuálne sedenie porovnáme so získanými vzormi a zisťujeme tak, či je konzistentné s predchádzajúcimi sedeniami. Podobný prístup je použitý aj v [5], kde sa dolovanie sekvenčných vzorov používa na objavovanie zaujímavých navigačných ciest v adaptívnych výučbových systémoch.

Odhaľovanie vzorov

Modul *PatternDetection* zisťuje výskyt preddefinovaných vzorov v spracovaných záznamoch. Odhaľovanie vzorov je založené na použitých dátových štruktúrach, v našom prípade od dátovej štruktúry *suffix tree* (komprimovaný *trie* [2]).

Vyhodnocovanie spätnej väzby

Modul *FeedbackEvaluator* vyhľadáva vzory implicitnej spätnej väzby [7] v získaných dátach a odhaduje používateľove hodnotenie zobrazeného obsahu.

Porovnávanie konceptov

Informácia o ohodnotení zobrazených informácií reprezentovaných konceptmi na odhad charakteristík používateľa nepostačuje. Dôležité sú dôvody, prečo je konkrétne ohodnotenie nízke alebo vysoké. Keďže sa toto ohodnotenie pri rôznych konceptoch líši, je vhodné poznať vzťahy medzi jednotlivými konceptmi. Túto úlohu realizuje modul *ConceptComparer*, ktorý porovnáva koncepty reprezentované ontologickými inštanciami a hľadá ich spoločné a rozdielne aspekty (mieru podobnosti).

V prípade domény pracovných ponúk môžeme uviesť ako príklad dve pracovné ponuky, ktoré majú veľmi rozdielne ohodnotenie. Ak sú tieto dve ponuky takmer identické a líšia sa len v mieste výkonu práce, vieme z rozdielneho ohodnotenia odvodiť, že je miesto výkonu práce pre používateľa dôležité. Navyše vieme povedať, ktorá hodnota je pre používateľa vhodná a ktorá nevyhovuje.

Modul *ConceptComparer* počas porovnávania skúma vzťahy porovnávaných inštancií na základe použitej taxonómie tried. Hľadá spoločnú nadtriedu týchto inštancií – čím viac krokov treba na nájdenie tejto nadtriedy, tým menej sú si porovnávané inštancie podobné. Tento modul tiež analyzuje dátové a objektové atribúty porovnávaných inštancií. Pri porovnávaní dátových atribútov (najmä reťazcov) nepostačuje jednoduché porovnanie zhody – treba vyhodnotiť zhodnosť textov na sémantickej úrovni. Metóda takéhoto vyhodnocovania je navrhnutá v [11].

Pri objektových atribútoch sa algoritmus vykonáva rekurzívne na príslušné asociované inštancie. Čiastkové výsledky sú agregované s použitím váh jednotlivých atribútov, ktoré môžu byť preddefinované alebo uložené v modeli používateľa.

Aktualizácia modelu používateľa

Modul *UserModelUpdater* agreguje výsledky ostatných modulov a vykonáva aktualizáciu inštancie modelu používateľa podľa preddefinovaných heuristík.

Nižšie uvádzame príklad jednoduchej heuristiky v doméne pracovných ponúk:

> „Ak si používateľ zobrazil detaily o aspoň ‘dostatočnom počte‘ ponúk z odvetvia *A* (napr. zdravotníctvo alebo IT), zvýš relevantnosť tohto odvetvia v modeli ideálnej pracovnej ponuky uloženej v modeli používateľa.“

Iná heuristika, ktorá predpokladá navigáciu v informačnom priestore pomocou sémantického fazetového prehliadača [12] je:

> „Ak si používateľ hneď po začiatku sedenia vybral ohraničenie *X*, zvýš relevantnosť charakteristiky spojenej s týmto ohraničením.“

5 Záver

V príspevku sme opísali výskum v oblasti automatizovaného získavania charakteristík používateľa, ktoré je založené na sledovaní používateľa. Definovali sme dvojfázový proces, ktorý postupuje od správania používateľa v rámci webového informačného systému k odhadu charakteristík používateľa v jeho modeli. Každá fáza procesu je pokrytá softvérovými nástrojmi, ktoré vykonávajú jednotlivé definované úlohy.

Prvá fáza – *zber dát* produkuje detailný záznam aktivít používateľa. Navrhli sme nástroje *Click* a *SemanticLog* spolu s ich vzájomným prepojením a napojením na zvyšok webového informačného systému. Nástroje vykonávajú monitorovanie používateľa na strane servera a klienta a vytvárajú spoľahlivý záznam aktivít používateľa v časových súvislostiach so zachovaním sémantiky jednotlivých udalostí.

V druhej fáze procesu, fáze objavovania charakteristík používateľa, sme navrhli nástroj *LogAnalyzer*, ktorý spracúva záznam z predchádzajúcej fázy a analyzuje ho z viacerých hľadísk (navigácia, implicitná spätná väzba, konzistencia správania). Výsledky čiastočných analýz kombinuje s heuristikami a upravuje model používateľa. Modul *ConceptComparer* nástroja *LogAnalyzer* by mohol byť definovaný aj ako samostatný nástroj, ktorý poskytuje službu porovnávania ontologických inštancií pre rôzne časti adaptívneho systému.

Jednotlivé nástroje realizujeme a experimentálne overujeme v rámci výskumného projektu v doméne pracovných príležitostí. V budúcnosti plánujeme tiež ich overenie v doméne vedeckých publikácií.

Poďakovanie

Táto práca vznikla za čiastočnej podpory Agentúry na podporu vedy a techniky (APVT-20-007104), vedeckej grantovej agentúry VEGA (VG1/3102/06) a štátnym programom pre výskum a vývoj „Budovanie informačnej spoločnosti“ (1025/04).

Literatúra

1. De Bra, P., Calvi, L.: AHA: A generic adaptive hypermedia system. In: Proc. of *2nd Workshop on Adaptive Hypertext*. Pittsburgh, USA (1998) 5–12.

2. Dunham, M.: *Data Mining: Introductory and Advanced Topics*. Prentice Hall PTR, Upper Saddle River, NJ, USA (2002).
3. Etgen, M., Cantor, J.: What does getting WET (Web Event-logging Tool) mean for web usability? In: *Proc. 5th Conf. on Human Factors & the Web.* (1999).
4. Judd, T., Kennedy, G.: *Making sense of audit trail data. Australasian Journal of Educational Technology* 20(1) (2004) 18–32.
5. Krištofič, A., Bieliková, M.: Improving Adaptation in Web-Based Educational Hypermedia by means of Knowledge Discovery. In *Proc. of 16th ACM Conf. on Hypertext and Hypermedia, HT'05,* ACM Press (2005) 184–192.
6. Návrat, P., Bieliková, M., Rozinajová, V.: Methods and tools for acquiring and presenting information and knowledge in the web. In: *Proc. of Int. Conf. on Computer Systems and Technologies, CompSysTech'05*, Varna, Bulgaria (2005).
7. Oard, D.W., Kim, J.: Implicit feedback for recommender systems. In *Proc. of AAAI Workshop on Recommender Systems*, Madison, WI, USA (1998) 80-82.
8. Paganelli, L., Paterno, F.: Intelligent analysis of user interactions with web applications. In *Proc. of the 7th Int. Conf. on Intelligent User Interfaces*, *IUI'02,* New York, NY, USA, ACM Press (2002) 111–118.
9. Rafter, R., Smyth, B.: Passive profiling from server logs in an online recruitment environment. In *Proc. of the IJCAI Workshop on Intelligent Techniques for Web Personalisation, ITWP'01*, Seattle, Washington, USA (2001) 35-41.
10. Thomas, R., Kennedy, G., Draper, S., Mancy, R., Crease, M.,Evans, H., Gray, P..: Generic usage monitoring of programming students. In Crisp, G., Thiele, D., Scholten, I., Barker, S., Baron, J., eds.: *Proc. of 20th Annual Conf. of the Australasian Society for Computers in Learning in Tertiary Education, ASCILITE'03,* Adelaide, Australia (2003) 715-719.
11. Tury, M., Bieliková, M.: An Approach to Detection Ontology Changes. In *Proc. of 1st Int. Workshop on Adaptation and Evolution in Web Systems Engineering, AEWSE'06 at ICWE 2006*, ACM, Palo Alto, CA, USA (2006).
12. Tvarožek, M.: Personalized navigation in the semantic web. In V. Wade, H. Ashman, and B. Smyth, editors, *Proc. of 4th Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems*, *AH'06*, Springer, LNCS 4018, Dublin, Ireland, (2006) 467–471.

Annotation

Software Tools for User Characteristics Acquisition

Nowadays, many web-based information systems need the ability to adapt to user characteristics. Otherwise it would be difficult or even impossible to use them efficiently. This paper describes an approach to user characteristics acquisition performed by a set of cooperating software tools where each tool performs an individual task of the characteristics acquisition process. Characteristics are acquired based on user behavior, what minimizes the amount of necessary user involvement. Four tools are described with stress on their interconnections in order to provide effective user characteristics acquisition.

B Ontologické modely projektov NAZOU a MAPEKUS

V tejto prílohe sa nachádzajú ontologické modely projektov NAZOU a MAPEKUS: model domény a model používateľa. Tieto modely sa používajú na overenie navrhovanej metódy zachytenia záujmov používateľa na webe.

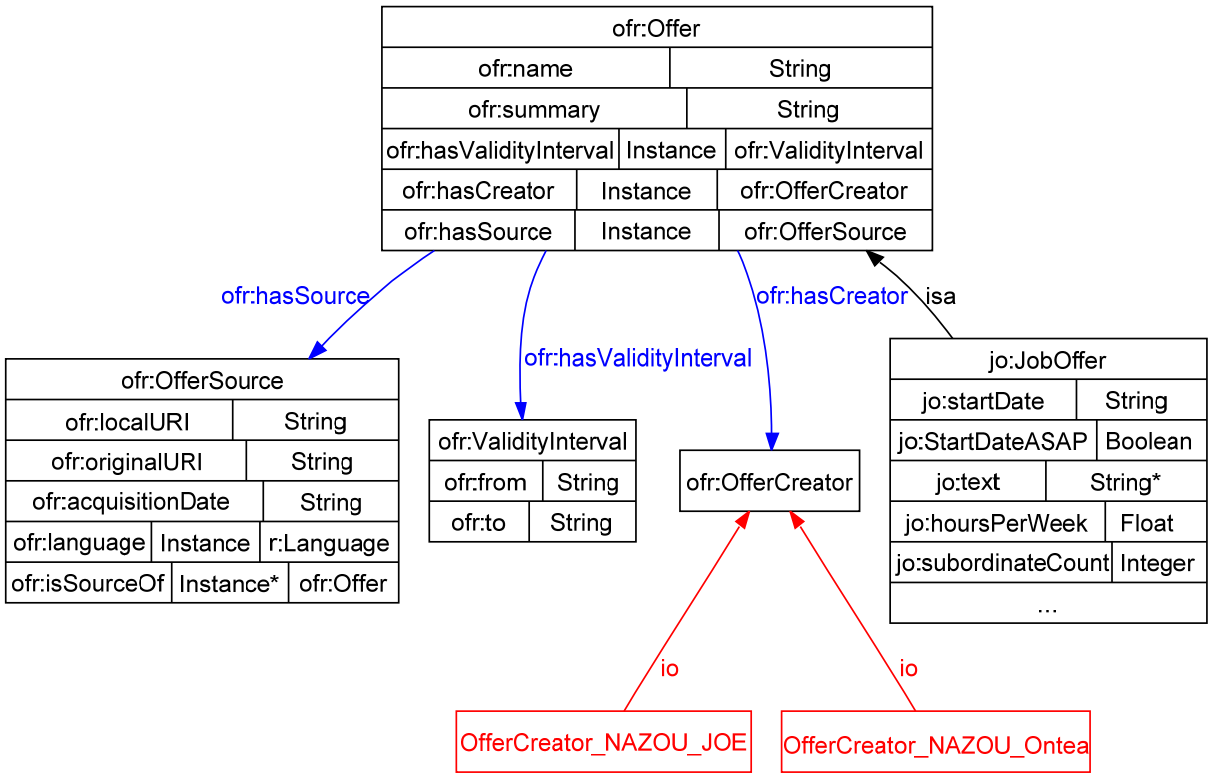
Ontologické modely projektov NAZOU a MAPEKUS

Model pracovnej ponuky – doménová ontológia projektu NAZOU

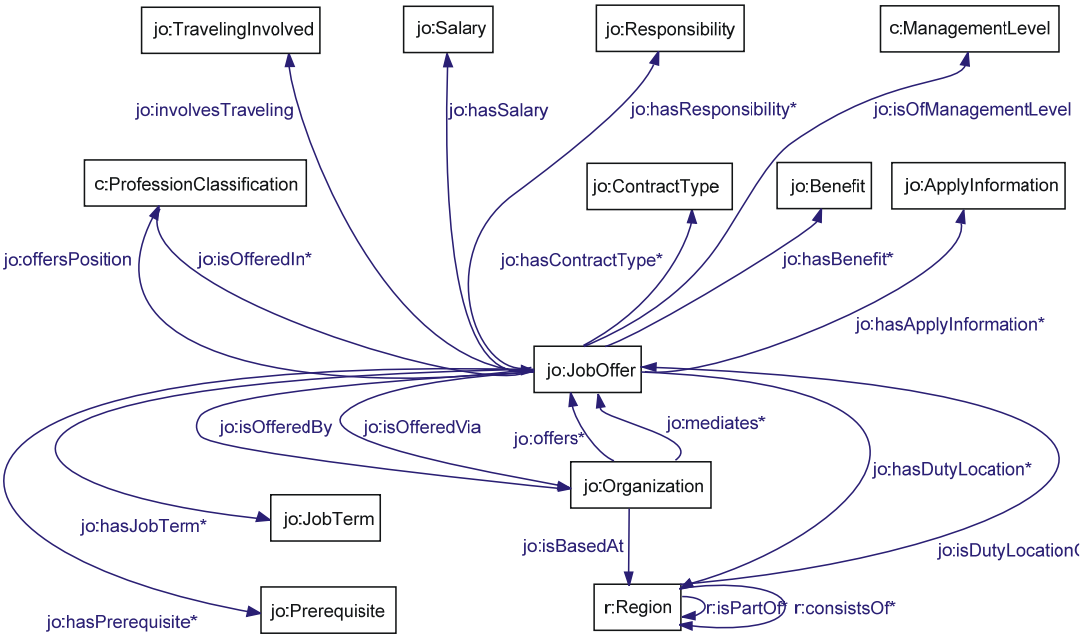
Doménová ontológia predstavuje explicitnú konceptualizáciu pracovnej ponuky. Pri konceptualizácii pracovnej ponuky využíva a prípadne aj dopĺňa ďalšie ontológie, ktorých doména je nezávislá od zvolenej aplikačnej domény pracovných ponúk. Tieto ontológie sa môžu uplatniť aj pri inej aplikačnej doméne, ktorá využíva nimi definované koncepty.

- Ontológia *region* (používaná predpona „r:“) – opisuje doménu regiónov, krajín, jazykov a mien, ktoré sa používajú v daných regiónoch;
- Ontológia *classification* (predpona „c:“) – definuje hierarchie pre priemyselné odvetvia, profesie, úrovne vzdelania, kvalifikácie a rôzne usporiadania;
- Ontológia *offer* (predpona „ofr:“) – opisuje doménu všeobecnej ponuky. Všeobecná ponuka je reprezentovaná triedou *ofr:Offer*. Každá ponuka má svoj zdroj (*ofr:OfferSource*), interval platnosti (*ofr:ValidityInterval*) a identifikáciu nástroja/spôsobu, ktorý bol použitý na vloženie ponuky do ontológie (*ofr:OfferCreator*). Ich vzájomné vzťahy k triedam sú zobrazené na obrázku 1.

Pri zdrojoch ontológie, ktorá priamo súvisí s doménou pracovných ponúk sa používa predpona „jo“.



Obrázok 1: Vzťahy medzi triedami Offer, OfferSource a JobOffer.



Obrázok 2: Objektové vlastnosti triedy JobOffer.

Opis vybraných tried

Trieda JobOffer

Kľúčovou triedou ontológie je trieda *JobOffer*, ktorá predstavuje samotnú pracovnú ponuku. Okrem objektových vlastností (pozri obrázok 2) má trieda aj dátové vlastnosti:

- `jo:hoursPerWeek` – dátového typu `xsd:float` – počet hodín práce za týždeň;
- `ofr:name` – dátového typu `xsd:string` – názov ponuky;
- `jo:startDate` – dátového typu `xsd:date` – dátum nástupu;
- `jo:startDateASAP` – dátového typu `xsd:boolean` – nástup ihneď;
- `jo:subordinateCount` – dátového typu `xsd:int` – počet podriadených.

Objektové vlastnosti spájajú triedu *JobOffer* s nasledovnými konceptmi pracovnej ponuky:

- *jo:ApplyInformation* – informácie k podávaniu žiadostí o zamestnanie;
- *jo:Benefit* – ponúkaná výhoda (auto, byt, poistenie,...);
- *jo:ContractType* – pracovný pomer (trvalý, dočasný, na dohodu);
- *jo:JobTerm* – pracovný úväzok (plný, polovičný,...);
- *c:ManagmentLevel* – vyjadruje úroveň ponúkanej pracovnej pozície z hľadiska umiestnenia v hierarchii manažmentu spoločnosti;
- *jo:Organization* – organizácia, ktorá ponúka alebo sprostredkúva prácu;
- *jo:Prerequisite* – predpoklady na uchádzača o zamestnanie požadované zamestnávateľom;
- *c:ProfessionClassification* – klasifikácia profesií podľa OSN;
- *r:Region* – región v ktorom sa vykonáva práca;

- *jo:Responsibility* – zodpovednosť;
- *jo:Salary* – plat;
- *jo:TravelingInvolved* – vyjadruje či a nakoľko je náplňou práce cestovanie.

Trieda Prerequisite

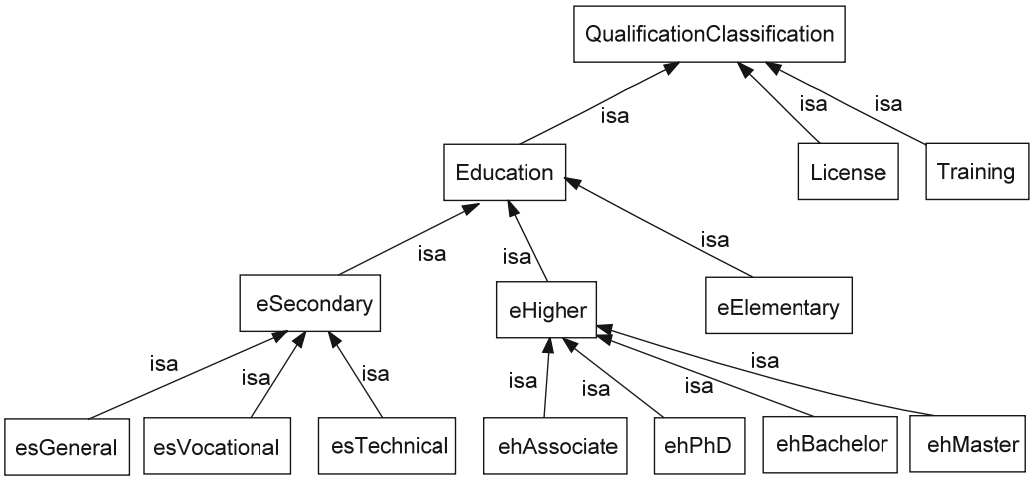
Pri pracovnej ponuke sú veľmi dôležité predpoklady, kladené na uchádzača. Tie modeluje trieda *Prerequisite*. Predpoklady delí na dve skupiny:

Požadované (angl. required) – musí uchádzač jednoznačne spĺňať;

Uprednostňované (angl. preferred) – predstavujú pre uchádzača výhodu pri ich splnení.

Požiadavky, ktoré má zamestnávateľ na zamestnanca smerujú na triedy ontológie *classification* a môžu byť troch typov:

- Požiadavky na kvalifikáciu zamestnanca – *QualificationClassification*;
- Požiadavky na skúsenosti zamestnanca – *ExperienceClassification*;
- Požiadavky súvisiace so zamestnancom ako takým – *PersonalAttributeClassification*.



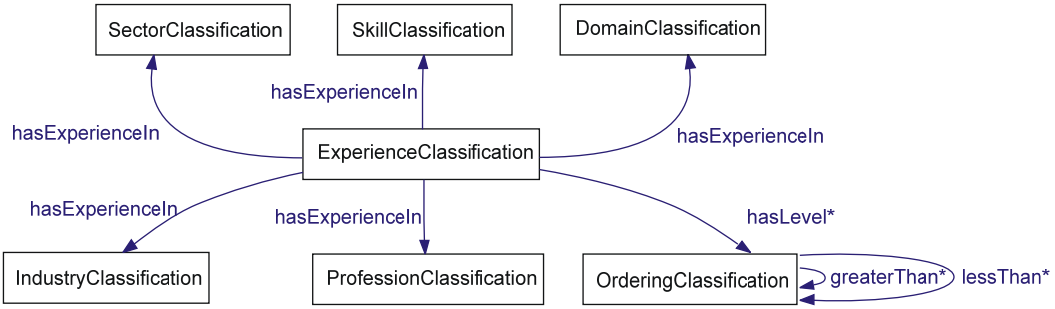
Obrázok 3: Klasifikácia kvalifikácií.

Trieda QualificationClassification

Klasifikácia kvalifikácií na obrázku 3: pozostáva z klasifikácie vzdelania (*Education*), z licencií, ktoré preukazujú absolvovanie určitej skúšky uchádzačom (*Licenses*) a zo zručností nadobudnutých na školeniach a kurzoch (*Training*).

Trieda ExperienceClassification

Klasifikácia skúseností umožňuje modelovať požiadavky na skúsenosti uchádzača o zamestnanie. Každá skúsenosť má priradený určitý stupeň (hasLevel) alebo môže byť vyjadrená hodnotou určitej veličiny (napríklad počet najazdených kilometrov). Skúsenosť môže byť vyjadrená vo vzťahu k určitej doméne znalostí (zvyčajne študijný odbor), zručnosti, sektoru (napr. akademický, súkromný), priemyselnému odvetviu, alebo profesii (obrázok 4).



Obrázok 4: Vzťah klasifikácie skúseností k iným klasifikáciám.

Trieda PersonalAttributeClassification

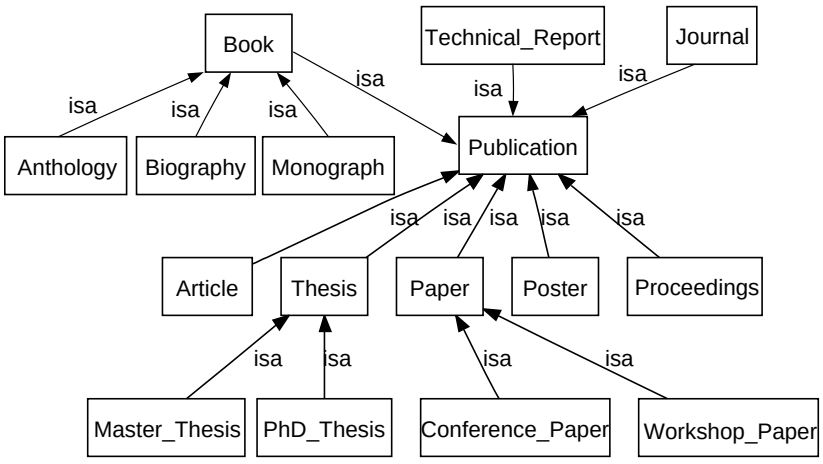
Klasifikácia osobných atribútov uchádzača o zamestnanie umožňuje modelovať požiadavky, ktoré súvisia s osobnostnými predpokladmi uchádzača o zamestnanie. Ide napríklad o schopnosti analytického myslenia, či komunikácie, alebo bezchybný zrak (nulové dioptrie).

Model publikácie – doménová ontológia projektu MAPEKUS

Doménová ontológia predstavuje explicitnú konceptualizáciu publikácie. Pri konceptualizácii publikácie využíva ďalšie ontológie, ktorých doména je nezávislá od zvolenej aplikačnej domény.

- Ontológia *region* (používaná predpona „r:“) – opisuje doménu regiónov, krajín, jazykov a mien, ktoré sa používajú v daných regiónoch;
- Ontológia *party* – konceptualizuje (zmluvnú) stranu vo vzťahu. Stranou môže byť na najvyššej úrovni fyzická osoba alebo organizácia.

No obrázku 5 sa nachádza hierarchia podtried triedy *Publication*, čiže hierarchia všetkých publikácii nachádzajúcich sa v ontológii.



Obrázok 5: Hierarchia publikácií.

Opis vybraných tried

- *Article* – trieda reprezentujúca článok v časopise.
- *Book* – predstavuje knihu a obsahuje ďalšie podtriedy :

- *Anthology* – zbierka prác od rôznych autorov,
- *Biography* – životopisné dielo,
- *Monography* – odborné dielo opisujúce jednu problematiku.

- *Journal* – reprezentuje periodikum určené pre odbornú verejnosť.
- *Paper* – predstavuje odborný príspevok. Delí sa na ďalšie podtriedy:
  - *Conference paper* – konferenčný odborný príspevok.
  - *Technical paper* – odborný príspevok opisujúci napríklad technické aspekty určitého systému.
- *Proceedings* – súbor odborných príspevkov vydaných v kontexte nejakej konferencie alebo iného stretnutia odbornej verejnosti. Obyčajne sa vydáva v knižnej podobe.
- *Poster* – reprezentuje plagáty s odbornou tematikou.
- *Technical report* – formálna správa, ktorá opisuje prínos v oblasti aplikovaného výskumu, poukazuje na detaily a výsledky riešenia nejakého vedeckého problému.
- *Thesis* – práca obsahujúca výsledky výskumu vypracovaná kandidátom na titul v rámci štúdia. Člení sa na podtriedy:
  - *MasterThesis* – práca vypracovaná na druhom stupni vysokoškolského štúdia.
  - *PhDThesis* – práca vypracovaná na treťom (doktorandskom) stupni vysokoškolského štúdia.

Vlastnosti triedy Publication

Na obrázku 6 sú zobrazené dátové a objektové vlastnosti triedy Publication.

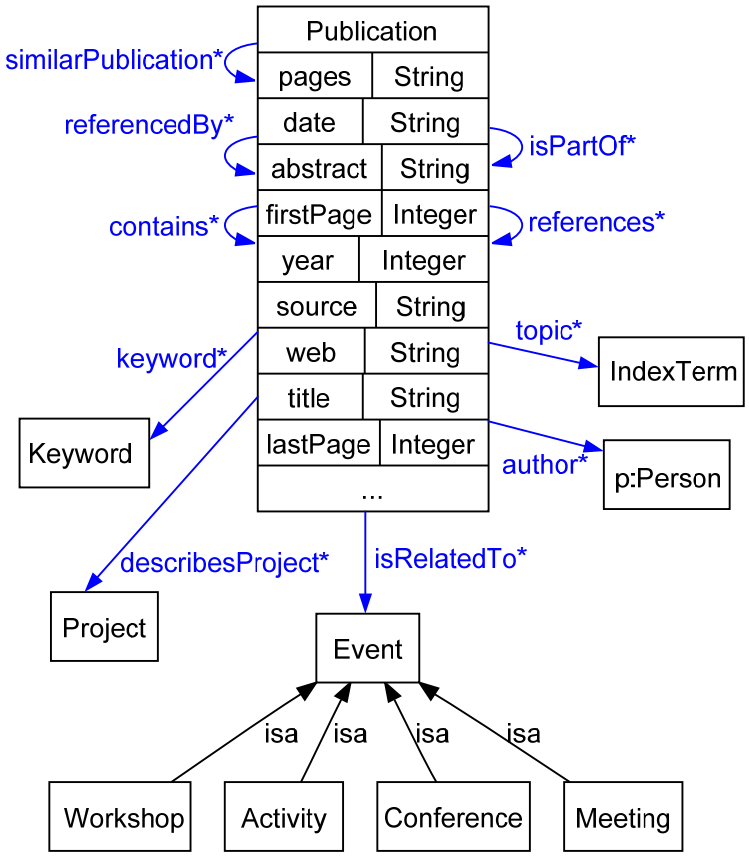
Dátové vlastnosti:

- `pages` – typu `xsd:string` – strany na ktorých sa nachádza publikácia ak je časťou inej publikácie.
- `date` – typu `xsd:string` – dátum vydania/sprístupnenia publikácie.
- `abstract` – typu `xsd:string` – abstrakt publikácie.
- `firstPage` – typu `xsd:int` – prvá strana publikácie v rámci inej publikácie.
- `year` – typu `xsd:int` – rok vydania publikácie.
- `source` – typu `xsd:string` – zdroj publikácie (napr. doi odkaz).
- `web` – typu `xsd:string` – odkaz na asociovaný zdroj na webe.
- `title` – typu `xsd:string` – názov publikácie.
- `lastPage` – typu `xsd:int` – posledná strana publikácie v rámci inej publikácie.

Objektové vlastnosti:

- *Keyword* – trieda `Keyword` – jednoduché kľúčové slovo.
- *describesProject* – trieda `Project` – výskumný/vývojový projekt.
- *isRelatedTo* – trieda `Event` – udalosť, ktorá je zvyčajne spojená s publikáciou (konferencia,...)
- *author* – trieda `Person` – autor publikácie.

- *topic* – trieda `IndexTerm` – indexovaný výraz opisujúci publikáciu.
- *contains* – trieda `Publication` – opisuje publikáciu, ktorá je časťou rodičovskej publikácie (napr. články v časopise).
- *isPartOf* – trieda `Publication` – je časťou publikácie (inverzná ku *contains*).
- *references* – trieda `Publication` – odkazuje sa na publikáciu.
- *referencedBy* – trieda `Publication` – je odkazovaná publikáciou (inverzná k *references*).
- *similarPublication* – trieda `Publication` – je podobná publikácii.



Obrázok 6: Vlastnosti triedy Publication.

Model používateľa

Model používateľa vychádza z opisu domény pomocou doménovej ontológie. Dôležité sú ďalšie možnosti použitia modelu ako aj jeho schopnosť zachytiť všetky potrebné charakteristiky. Tieto môžu byť úzko späté so zvolenou informačnou doménou alebo také, ktoré má význam modelovať aj pri iných informačných doménach, iných aplikáciách a je výhodné ich zdieľať.

Ontológia používateľa sa skladá z troch samostatných ontológií, čím oddeľuje tieto dva druhy charakteristík:

- ontológia g*eneric-user* – definuje všeobecné charakteristiky používateľa;

- ontológia *job-offer-user* – definuje charakteristiky používateľa viažuce sa na doménovú ontológiu pracovných ponúk;
- ontológia *publication-user* – definuje charakteristiky používateľa viažuce sa na doménovú ontológiu publikácií.

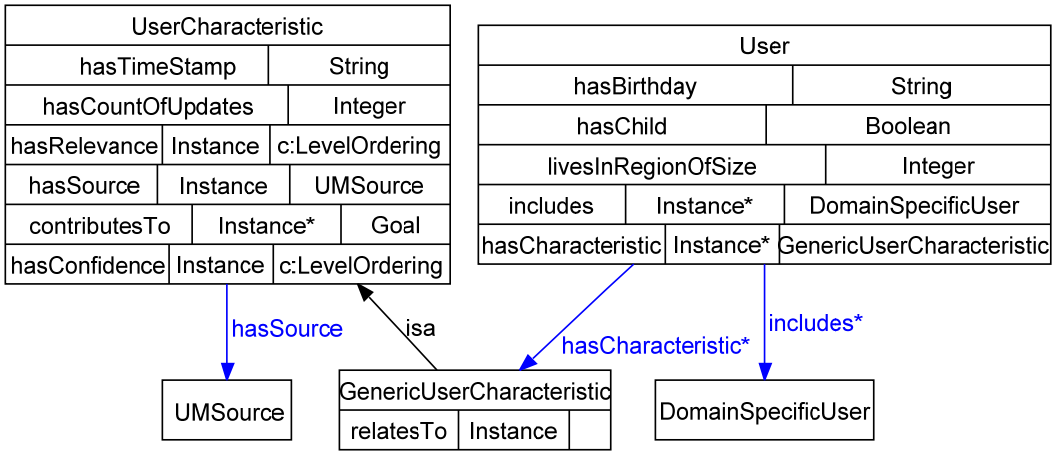
Doménovo nezávislý model

Primárnou triedou tohto modelu je trieda *User* (obrázok 7). Má nasledovné dátové vlastnosti:

- `hasName` – dátového typu `xsd:string` – meno používateľa;
- `hasBirthday` – dátového typu `xsd:date` – dátum narodenia;
- `hasChild` – dátového typu `xsd:boolean` – či má používateľ aspoň jedno dieťa.
- `livesInRegionOfSize` – dátového typu `xsd:int` – počet obyvateľov regiónu, v ktorom používateľ žije.

Objektové vlastnosti spájajú triedu *DomainIndependentUser* s nasledovnými konceptmi:

- *GenericUserCharacteristic* – doménovo nezávislé charakteristiky
- *DomainSpecificUser* – časti modelu špecifické pre jednotlivé domény



Obrázok 7: Všeobecné charakteristiky používateľa.

O každej charakteristike zaznamenávame:

- *hasConfidence* – spoľahlivosť,
- *hasRelevance* – relevantnosť,
- *relatesTo* – hodnotu,
- *contributesTo* – cieľ, ku ktorému táto charakteristika prispieva,
- *hasSource* –identifikácia nástroja, ktorý charakteristiku pridal do modelu,
- *hasCountOfUpdates* – počet úprav charakteristiky.

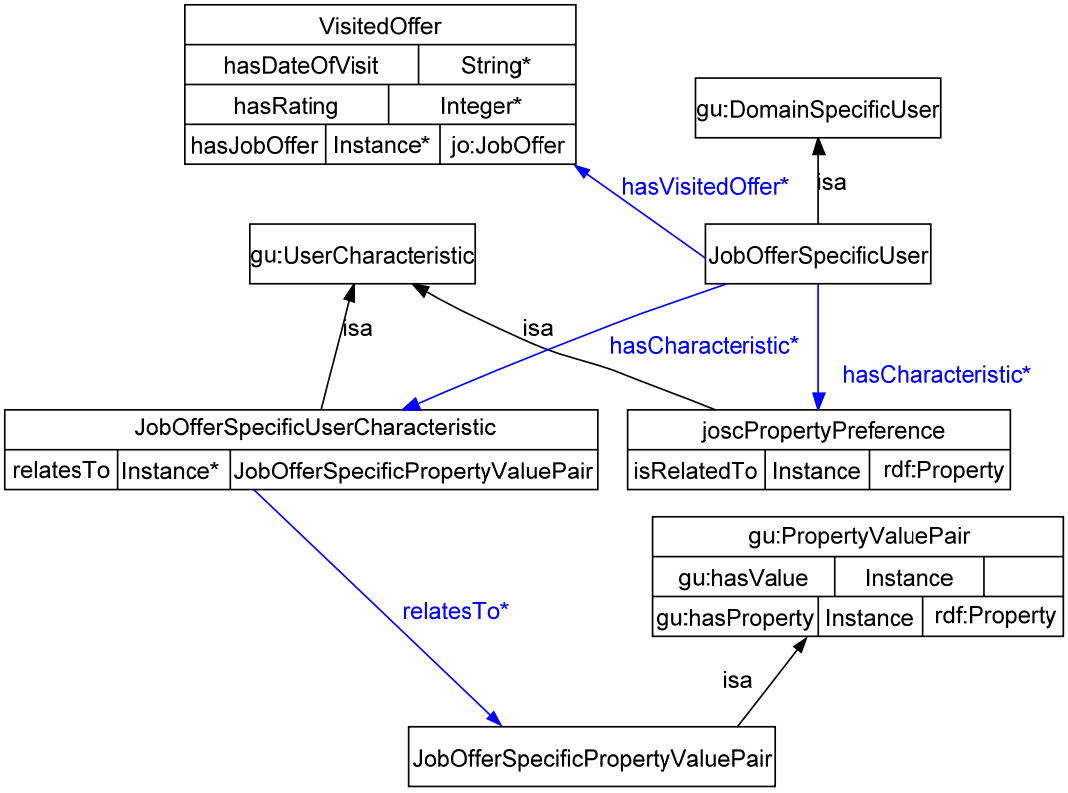
Doménovo závislý model

Vo všeobecnosti môže existovať viacero doménovo závislých modelov používateľa, ktoré sú prepojené s doménovo nezávislým modelom. Z domén projektov NAZOU a MAPEKUS vnikli dva doménovo závislé modely používané v príslušných doménach ponúk a publikácií.

V projekte NAZOU je základnou triedou *JobOfferSpecificUser* (obrázok 8), ktorá je podtriedou triedy *DomainSpecificUser* z doménovo nezávislého modelu, čím je zabezpečené prepojenie modelov.

Objektové vlastnosti spájajú triedu *JobOfferSpecificUser* s nasledovnými konceptmi:

- *VisitedOffer* – predstavuje záznam o zobrazení ponuky používateľovi a jej ohodnotení používateľom.
- *JobOfferSpecificUserCharacteristic* – charakteristika orientovaná na hodnoty doménových konceptov reprezentované párom dvojica – hodnota triedy *JobOfferSpecificPropertyValuePair*.
- *joscPropertyPreference* – charakteristika orientovaná na preferencie vlastností realizujúcich konceptualizáciu.



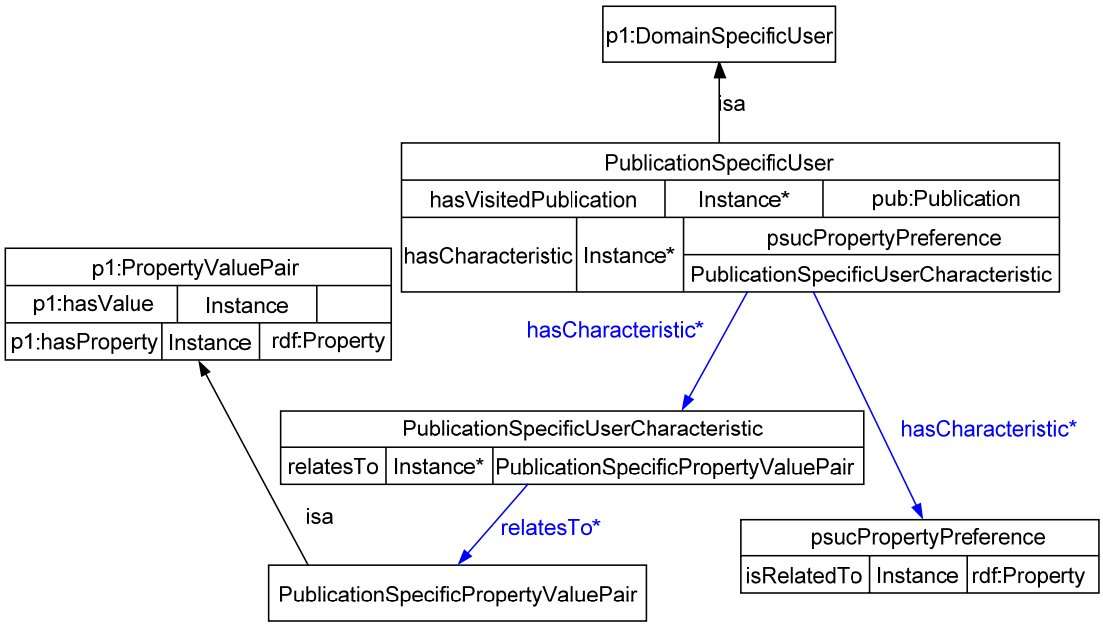
Obrázok 8: Doménové charakteristiky používateľa v projekte NAZOU.

Triedy *JobOfferSpecificUserCharacteristic* a *PropertyPreference* sú podtriedami triedy *UserCharacteristic* z doménovo nezávislého modelu. Tieto triedy majú teda tak isto ako *GenericUserCharacteristic* vlastnosti spoľahlivosť, relevantnosť, hodnotu, cieľ atď.

V projekte MAPEKUS je model používateľa riešený takmer identicky. Základnou triedou je *PublicationSpecificUser* (obrázok 9), ktorá je podtriedou triedy *DomainSpecificUser* z doménovo nezávislého modelu, čím je zabezpečené prepojenie modelov.

Objektové vlastnosti spájajú triedu *PublicationSpecificUser* s nasledovnými konceptmi:

- *PublicationrSpecificUserCharacteristic* – charakteristika orientovaná na hodnoty doménových konceptov reprezentované párom dvojica – hodnota triedy *PublicationSpecificPropertyValuePair*.
- *psucPropertyPreference* – charakteristika orientovaná na preferencie vlastností realizujúcich konceptualizáciu.



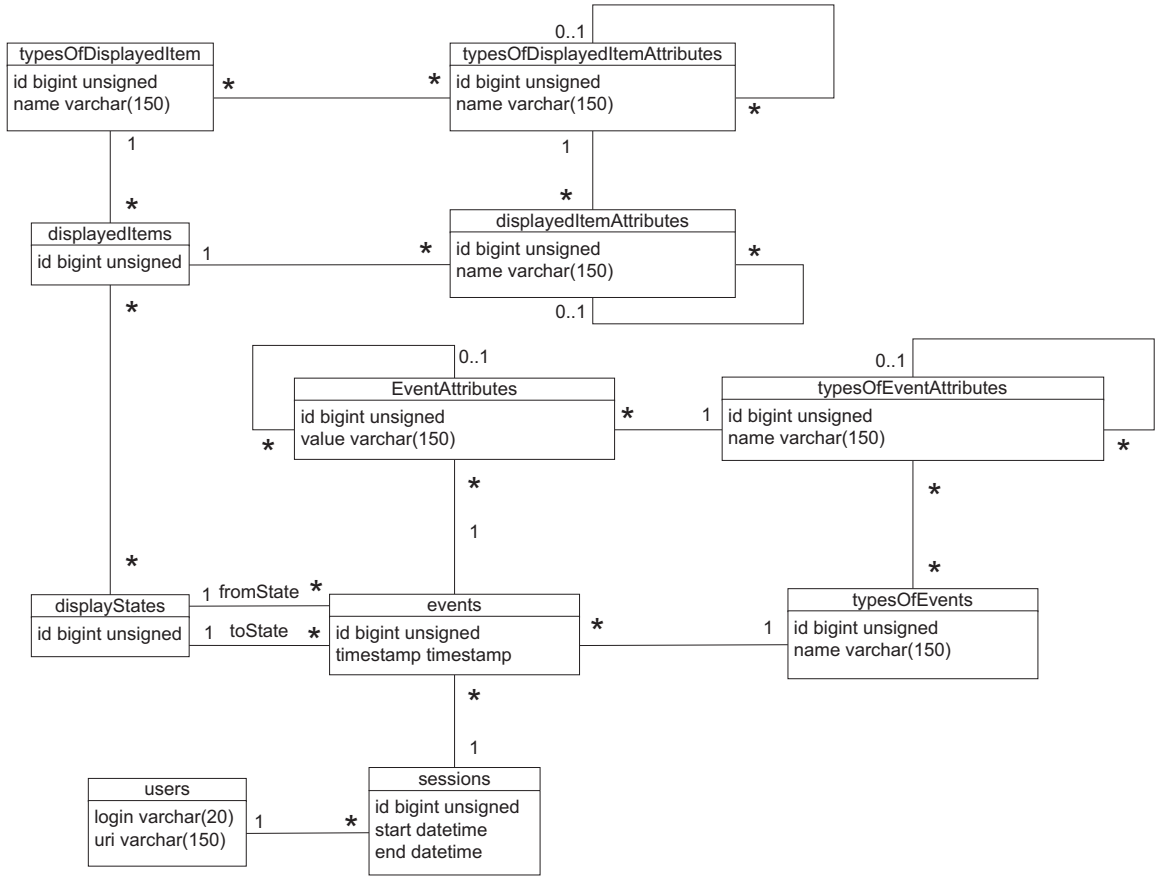
Obrázok 9: Doménové charakteristiky používateľa v projekte MAPEKUS

C Technická dokumentácia

V tejto prílohe uvádzame podrobnejší opis implementácie riešenia.

C.1 Záznamy o správaní používateľa

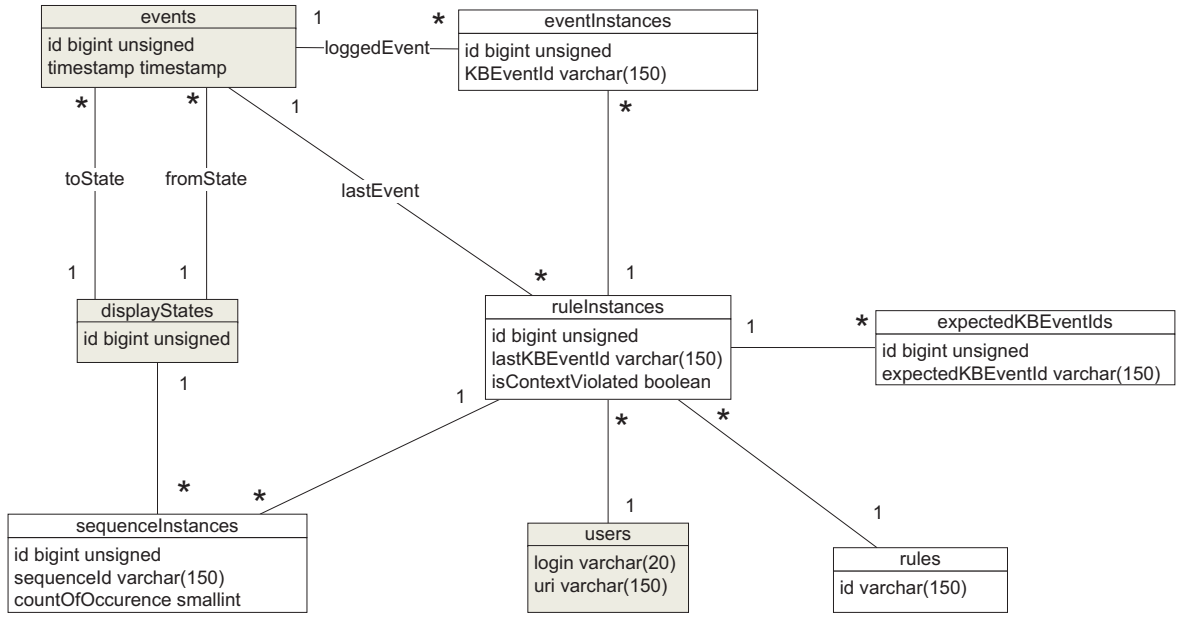
Na obrázku 1 je zobrazený logický dátový model záznamov udalostí vyvolaných akciami používateľa.



Obrázok 1: Logický dátový model záznamov udalostí.

C.2 Medzivýsledky nástroja LogAnalyzer

Na obrázku 2 je zobrazený logický dátový model medzivýsledkov nástroja LogAnalyzer. Tento model rozširuje pôvodný model záznamov o d'alšie entity. Entity z pôvodného modelu sú v obrázku farebne odlíšené.



Obrázok 2: Logický dátový model medzivýsledkov nástroja LogAnalyzer.

Nástroj pracuje s perzistentnými inštanciami pravidiel `ruleInstances`, ktoré sa vzťahujú na konkrétne pravidlá `rules` a používateľov `users`. Inštancia pravidla má definované aktuálne očakávané udalosti `expectedKBEventIds`, inštancie postupnosti `sequenceInstances` s aktuálnym počtom opakovaní a inštancie udalostí `eventInstances`, ktoré spájajú udalosti zo záznamu s udalosťami predpísanými pravidlom.

C.3 Client Side Action Recorder – Click

Nástroj je implementovaný v jazyku Javascript. Funkcionalita sa vykonáva v troch krokoch:

1. Zaregistrovanie obsluhy udalostí
2. Spracovanie udalosti
3. Odoslanie spracovanej udalosti cez SOAP protokol

Uvádzame výpisy hlavných funkcií, ktoré realizujú uvedené kroky:

Click - Hooking listeners on events

```
/*
 hooks function eventHandler to process events of all
 elements in document with name ‘‘elementName’’
*/
function hookTags(elementName)
{
  (function(e)
  {
    addEvent(e, "click", eventHandler);
    addEvent(e, "mouseover", eventHandler);
    addEvent(e, "mouseout", eventHandler);
  }).Iterate(document.getElementsByTagName(elementName));
}
```

Click - Event handling

```
/*
  creates string representation of an event
  and calls function LogEvent to process it
*/
function eventHandler(ev)
{
    var eventDump = "";
    eventDump += "time=\"" + Date() + "\"&";
    eventDump += "type = \"" + ev.type + "\"&";
    eventDump += "target =\""
    /*
      if the page was loaded or unloaded we record its URL
      otherwise, we record target of an event
      (typically HREF attribute of anchor element)
    */
    if(ev.type == "load" || ev.type == "unload")
    {
        eventDump += ev.target.URL + "\"";
    }
    else
    {
        eventDump += ev.target + "\"";
    }
    logEvent(eventDump);
}
```

Click - Event logging

```
/*
  invokes web service SemanticLog via AJAX and logs the event
*/
function logEvent(eventToBeLogged) {
  var call = new WS.Call
             ('http://localhost:8080/axis/SemanticLog.jws');
  var nsuri = 'http://sk.fiit.barla.click';
  var qn_op = new WS.QName('logEvent',nsuri);
  var qn_op_resp = new WS.QName('logEventResponse',nsuri);
  call.invoke_rpc(
    qn_op,
    new Array(
      {name:'event',value:eventToBeLogged}
    ),SOAP.NOENCODING,
    function(call,envelope) {
      // envelope is the response SOAP.Envelope
      // the XML Text of the response is in arguments[2]
    }
  );
}
```

C.4 Algoritmus výberu pravidiel – kandidátov

LogAnalyzer - findCandidateRules

```
1  private List<Rule> findCandidateRules(Event event)
2  {
3    List<Rule> result = new LinkedList<Rule>();
4    String eventType = event.getTypeOfEvent().getName();
5    UserSession uSession = event.getUserSession();
6    User user = uSession.getUser();
7    List<Rule> notExpectingRules = findNotExpectingRulesForUser(user);
8    List<Rule> rules = LogAnalyzer.getSingletonInstance().getRules();
9    for(Iterator<Rule> it = rules.iterator();it.hasNext();)
10   {
11     List<Rule> list = new LinkedList<Rule>();
12     Rule rule = it.next();
13     /* maybe we have the rule already in result
14      * because it is a "not expecting rule"
15      */
16     boolean flag = false;
17     //we iterate through all rules already present in result
18     for(Iterator<Rule> resultIterator = result.iterator();
19                                       resultIterator.hasNext();)
20     {
21       //if we have a match of IDs...
22       if(resultIterator.next().getId().compareTo(rule.getId()) == 0)
23       {
24         flag=true;
25         break;
26       }
27     }
28     if(flag == true)
29     {
30       /* if the rule is already in the result list,
31        * we can pass to the next one
32        */
33       continue;
34     }
35     /*candidate rule MUST contain event of type 'eventType'
36      * unless there is an instance which does not expect anything
37      */
38     if(rule.getSequence().contains(eventType)){
39       /* we try to determine whether it is not
40        * among the first expected events
41        */
42       Set<Event> kbEvents = rule.getSequence().getFirstExpectedEvent()
43       Iterator<Event> expEventsIterator = kbEvents.iterator();
44       while(expEventsIterator.hasNext())
45       {
46         String expEvent = expEventsIterator.next().getEventType().getUri();
47         if(expEvent.compareTo(eventType) == 0){
48           list.add(rule); break;}
49       }
50
51       /*
52        * if it was not among the first expected events,
```

```
53          * maybe there is some rule instance which is expecting such event
54          */
55         if(list.size() == 0){
56           List<RuleInstance> riList = getRuleInstances(user, rule);
57           //if there are some instance of the rule
58           if(riList != null){
59             for(Iterator<RuleInstance> riIterator = riList.iterator();
60                                                 riIterator.hasNext();)
61             {
62               RuleInstance ri = ruleInstanceIterator.next();
63               Set<String> set = ri.getExpectedKBEventIds();
64               //nothing expected ~ possibly a -1 count-of-occurence
65               set = ri.getExpectedKBEventIds();
66               if(set.size() == 0){
67                 //in fact, this should not happen
68                 list.add(rule);
69                 break;
70               } else {
71                 for(Iterator<String> expEventsIdsIterator = set.iterator();
72                                       expEventsIdsIterator.hasNext();)
73                 {
74                   /* we retrieve event with given id and
75                    * compare its type with current type
76                    */
77                    String id = expEventsIdsIterator.next();
78                   if(rule.getSequence().getEventwithId(id).getEventType().
79                                   getUri().compareTo(eventType) == 0)
80                   {
81                     list.add(rule);
82                     break;
83                   }
84                 }
85               }//else
86             }//for
87           }//if riList !=null
88         }//if list.size() == 0
89       result.addAll(list);
90       }//rule.getSequence().contains(eventType)
91     }//for
92     return result;
93   }
```

C.5 Opis vybraných tried nástroja LogAnalyzer

V tejto časti uvádzame opis niektorých tried nástroja LogAnalyzer vo formáte upravenej Javadoc dokumentácie.

Package sk.fiit.nazou.loganalyzer.instances

Provides classes representing intermediate results of the process (instances of rules and sequences).

Class RuleInstance

sk.fiit.nazou.loganalyzer.instances

```
java.lang.Object
  └sk.fiit.nazou.userlogs.PersistentObject
      └sk.fiit.nazou.loganalyzer.instances.RuleInstance
```

---

public class **RuleInstance**
extends PersistentObject

Class mapped to ruleInstances table

Method Detail

getLastEvent

`public Event getLastEvent()`

Returns last mapped event.

---

getRule

`public Rule getRule()`

Return the rule associated with this rule instance (!!from relationalDatabaseOnly!!).

---

getCompleteRule

`public Rule getCompleteRule()`

Returns rule associated with this rule instance as it was processed from configuration file.

---

setRule

`public void setRule(Rule rule)`

Set the rule associated with this rule instance (!!used mainly by Hibernate!!).

---

getUser

```
public User getUser()
```

Retrieves the user associated with this instance.

---

getExpectedKBEventIds

```
public Set getExpectedKBEventIds()
```

---

getSequenceInstances

```
public Map getSequenceInstances()
```

---

getEventInstances

```
public Map getEventInstances()
```

---

apply

```
public void apply(Event ev)
```

Performs binding of current event to this rule instance. Launch user model update if the rule instance becomes fulfilled.

---

getLastKBEventId

```
public String getLastKBEventId()
```

---

isContextViolated

```
public boolean isContextViolated()
```

---

getIsContextViolated

```
public boolean getIsContextViolated()
```

This signature is just for Hibernate, humans prefer `isContextViolated()`

Package sk.fiit.nazou.loganalyzer.kb

Provides classes representing a rule and all its parts including logic.

Class Change

sk.fiit.nazou.loganalyzer.kb

java.lang.Object
  └ **sk.fiit.nazou.loganalyzer.kb.Change**

---

public class **Change**
extends Object

Class represents one change defined in consequence part of the rule in config file. It provides a method to execute the change and actualy change the user model.

Method Detail

getInstanceUri

```
public String getInstanceUri()
```

---

getProperties

```
public List getProperties()
```

---

execute

```
public void execute(Map eventInstances,
                    String userUri)
             throws CorruptedUserModelUpdaterPropertiesException
```

This method executes the change - updates appropriately user model.

Parameters:

`eventInstances` - map from IDs of events from config file to events logged in database (for referencing properties).

`userUri` - Uri of a user (as stored in ontology) whose characteristics are going to be adjusted

Throws:

`CorruptedUserModelUpdaterPropertiesException`

Class Event

sk.fiit.nazou.loganalyzer.kb

java.lang.Object
  └ sk.fiit.nazou.loganalyzer.kb.Sequence
    └ **sk.fiit.nazou.loganalyzer.kb.Event**

---

public class **Event**
extends Sequence

Class represents event as a part of the rule (*not* event as a result of user action)

Method Detail

isSequence

`public boolean **isSequence**()`

determines whether the current instance is of Event or Sequence
Overrides:
`isSequence` in class `Sequence`

---

getEventType

`public TypeOfEvent **getEventType**()`

---

getFirstEvent

`public Event **getFirstEvent**()`

Returns this instance - provided for compatibility with super class Sequence.
Overrides:
`getFirstEvent` in class `Sequence`

---

getLastEvent

`public Event **getLastEvent**()`

Returns this instance - provided for compatibility with super class Sequence.
Overrides:
`getLastEvent` in class `Sequence`

---

getFirstExpectedEvent

`public Set **getFirstExpectedEvent**()`

Returns this instance in a set - provided for compatibility with super class Sequence.
Overrides:
`getFirstExpectedEvent` in class `Sequence`

---

getEventwithId

`public Event **getEventwithId**(String kbEventId)`

Returns this instance in a set - provided for compatibility with super class Sequence.
Overrides:
`getEventwithId` in class `Sequence`

Parameters:
`kbEventId` - id of an event as defined in configuration file.

---

getEventOfType

`public Event getEventOfType(String typeOfEvent)`

Returns this instance if its type is the same as given type of event or null. Provided for compatibility with super class Sequence.
Overrides:
`getEventOfType` in class `Sequence`

---

getNextEvent

`public Event getNextEvent()`

Retrieves next event of the parent sequence
Returns:
following event in the sequence or null

---

getPreviousEvent

`public Event getPreviousEvent()`

Retrieves previous event of the parent sequence
Returns:
preceding event in the sequence or null

---

isLastEventInSequence

`public boolean isLastEventInSequence()`

---

getNextEventOfType

`public Event getNextEventOfType(String typeOfEvent)`

Class ProcessedProperty

sk.fiit.nazou.loganalyzer.kb

```
java.lang.Object
  └sk.fiit.nazou.loganalyzer.kb.Property
    └sk.fiit.nazou.loganalyzer.kb.ProcessedProperty
```

---

public class **ProcessedProperty**
extends Property

Class represents properties whose value need to be processed (computed) according to defined action (e.g. http://fiit.stuba.sk/loganalyzer#increase)

Method Detail

getIncrement

```
public double getIncrement()
```

---

getMaxValue

```
public double getMaxValue()
```

---

getMinValue

```
public double getMinValue()
```

---

isReferencingProperty

```
public boolean isReferencingProperty()
```

Used to distinguish property types
Overrides:
`isReferencingProperty` in class `Property`

---

isUsedProperty

```
public boolean isUsedProperty()
```

Used to distinguish property types
Overrides:
`isUsedProperty` in class `Property`

---

process

```
public void process(String characteristicInstanceUri)
```

This method updates **this** property of the given instance identified by its uri
Parameters:
`characteristicInstanceUri` - uri of the instance (characteristic) as stored in ontology

Class ReferencingProperty

sk.fiit.nazou.loganalyzer.kb

```
java.lang.Object
  └sk.fiit.nazou.loganalyzer.kb.Property
      └sk.fiit.nazou.loganalyzer.kb.ReferencingProperty
```

public class **ReferencingProperty**
extends Property

Class represents property which is referencing some event and whose value needs to be retrieved from attributes connected to this event

Method Detail

getValue

`public String getValue()`

getReferencedEventId

`public String getReferencedEventId()`

getTypeOfDisplayedItem

`public String getTypeOfDisplayedItem()`

getTypeOfRelatedEntity

`public TypeOfRelatedEntity getTypeOfRelatedEntity()`

isReferencingProperty

`public boolean isReferencingProperty()`

Overrides:
`isReferencingProperty` in class `Property`

isUsedProperty

`public boolean isUsedProperty()`

Overrides:
`isUsedProperty` in class `Property`

isReferencingToInstance

`public boolean isReferencingToInstance()`

getType

```
public String getType()
```

---

getFallbackReferencedEventId

```
public String getFallbackReferencedEventId()
```

---

getQuery

```
public String getQuery(Event ev)
                throws CorruptedUserModelUpdaterPropertiesException
```

Returns part of the query to retrieve instance related to values of this property
Throws:
CorruptedUserModelUpdaterPropertiesException

---

verify

```
public boolean verify(IResultData characteristicURI)
               throws CorruptedUserModelUpdaterPropertiesException
```

Verifies whether the retrieved characteristic with given URI is the right one. The idea is to get count of attributes and count of related statements from ontology. If these numbers are equal, we assume we retrieved the right instance of characteristic.
Throws:
CorruptedUserModelUpdaterPropertiesException

---

createEntryForCharacteristic

```
public IGraph createEntryForCharacteristic(String characteristicInstanceUri)
                                    throws CorruptedUserModelUpdaterPropertiesException
```

This method creates the corresponding property in the user model.
Throws:
CorruptedUserModelUpdaterPropertiesException

Class Sequence

sk.fiit.nazou.loganalyzer.kb

java.lang.Object
  └ **sk.fiit.nazou.loganalyzer.kb.Sequence**

Direct Known Subclasses:
Event

---

public class **Sequence**
extends Object

Class represents a sequence from configuration file with all needed attributes, sub-sequences and events.

Field Detail

id

protected String **id**

id of the sequence as stored in configuration file

context

protected List **context**

List of contextual conditions valid for this sequence.

Method Detail

getCountOfOccurence

public int **getCountOfOccurence**()

Returns:
countOfOccurence

getItems

public List **getItems**()

Returns:
items

getSequenceType

public SequenceType **getSequenceType**()

Returns:
sequenceType

isSequence

public boolean **isSequence**()

differs sequence object from event object

getContext

public List **getContext**()

Returns:
context

---

getParentRule

```
public Rule getParentRule()
```

Returns:
parentRule

---

isContinuous

```
public boolean isContinuous()
```

Returns:
isContinuous

---

contains

```
public boolean contains(String eventTypeName)
```

This method finds out whether the sequence (and all subsequences) contains an event of given type
Returns:
boolean

---

getFirstEvent

```
public Event getFirstEvent()
```

Retrieves the first event of this sequence.

---

getLastEvent

```
public Event getLastEvent()
```

Retrieves the last event of this sequence.

---

getEventwithId

```
public Event getEventwithId(String kbEventId)
```

Return an Event with the given id.
Parameters:
kbEventId - id of an event as defined in configuration file.

---

getEventOfType

```
public Event getEventOfType(String typeOfEvent)
```

Retrieve first occurence of event of given type.

---

getParentSequence

```
public Sequence getParentSequence()
```

Returns:
`parentSequence`

---

getNextEvent

```
protected Event getNextEvent(Sequence childSequence)
```

Return next sibling of the given child sequence. If there is no next sibling, it pass the call to parent sequence. If there is no parent sequence, it return null.

---

getPreviousEvent

```
protected Event getPreviousEvent(Sequence childSequence)
```

Return previous sibling of the given child sequence. If there is no previous sibling, it pass the call to parent sequence. If there is no parent sequence, it return null.

TryToFinish

```
public Set TryToFinish(Sequence childSequence,
                       RuleInstance ruleInstance,
                       boolean isContextViolated)
```

This *recursive* function tries to finish current sequence and returns set of expected events if it fails to finish. Recursion is done through `getNextExpectedEventsAndTryToFinish(SequenceInstance, RuleInstance, boolean)`.

Parameters:
`childSequence` - sequence(or event) which has just finished

---

getFirstExpectedEvent

```
public Set getFirstExpectedEvent()
```

Returns set of first expected events as defined in configuration file.

---

getAllContext

```
public List getAllContext()
```

returns all contextual conditions of this sequence.

D Príručka pre správcu systému

D.1 Nástroj Click

D.1.1 Predpoklady

- webová služba, ktorá od nástroja *Click* prevezme notifikáciu o udalosti (nástroj *SemanticLog*). Odporúčame využitie SOAP implementácie Apache Axis[15] v servlet kontajneri Apache Tomcat[16].

D.1.2 Inštalácia

1. zaintegrujte nástroj

   (a) do zobrazovaných HTML stránok pridaním referencií na tieto súbory do `head` elementu stránky:

   - prototype.js
   - ws.js
   - click.js

   alebo

   (b) do portálu, ktorý je založený na *Portal Engine* riešení Apache Cocoon[17]:

   - v súbore `portal/skins/<cesta-k-skinu>/styles/portal-page.xsl` sa k transformácii elementu `head` pridajú nasledovné riadky:

     ```
     <script type="text/javascript" src="{$base}js/prototype.js"/>
     <script type="text/javascript" src="{$base}js/ws.js"/>
     <script type="text/javascript" src="{$base}js/click.js"/>
     ```

   - JavaScript súbory sa nakopírujú do adresára `portal/skins/<cesta-k-skinu>/js`

2. vo funkcii `logEvent(eventToBeLogged)` v súbore `click.js` nastavte inicializáciu premennej `call` podľa aktuálnej inštalácie serverovej časti.

D.2 Nástroj LogAnalyzer

D.2.1 Predpoklady

- Java runtime vo verzii 5.0 (Java 2 Platform Standard Edition 5.0),
- relačný databázový systém s podporou rozhrania JDBC a príslušné JDBC ovládače (napr. MySQL),
- RDF/OWL úložisko s príslušnou implementáciou rozhrania `OntoCM` (aktuálne Sesame[18]).

[15] Apache Axis, `http://ws.apache.org/axis/`
[16] Apache Tomcat, `http://tomcat.apache.org/`
[17] Apache Cocoon, `http://cocoon.apache.org`
[18] Sesame RDF framework, `http://www.openrdf.org`

D.2.2 Konfigurácia

Nástroj má šesť konfiguračných súborov:

- `LogAnalyzer.properties` – premennej `rules` je potrebné nastaviť meno súboru s pravidlami, ktoré má nástroj používať. Ostatné parametre do tohto súboru zapisuje nástroj počas behu.
- `UserModelUpdater.properties` – v tomto súbore sa definujú vlastnosti aktuálne používaných charakteristík a ich napojenie na inštanciu používateľa.
- `OntoMem.properties` – v tomto súbore sa definuje prístup k RDF/OWL úložisku pre rozhranie `OntoCM`.
- súbor s pravidlami (XML) – v tomto súbore sa definujú pravidlá, ktoré nástroj používa.
- `log4j.properties` – konfiguračný súbor pre log4j[19] knižnicu použitú na vytváranie správ o behu programu.
- `hibernate.cfg.xml` – konfiguračný súbor pre použitý objektovo-relačný mapovač Hibernate[20]. V tomto súbore je potrebné nastaviť boldom zvýraznené premenné:

```
<property name="hibernate.connection.driver_class">
    JDBC ovládač
</property>
<property name="hibernate.connection.url">
    pripojenie k databáze
</property>
<property name="hibernate.connection.username">
    login
</property>
<property name="hibernate.connection.password">
    heslo
</property>
```

Ukážkové konfiguračné súbory sa nachádzajú na priloženom elektronickom nosiči v adresári `Prototype/LogAnalyzer/src`.

D.2.3 Použitie

Funkcionalita nástroja sa vyvolá pomocou metódy `signal` rozhrania *ILogAnalyzer*. Pre úspešný beh nástroja potrebujete:

1. mať pri volaní v premennej prostredia *CLASSPATH* tieto súbory:
   - `LogAnalyzer.jar` – triedy a rozhrania samotného nástroja,
   - `UserLogs.jar` – triedy a rozhrania záznamov udalostí,
   - `OntoCM.jar` – triedy a rozhrania pre prístup do RDF/OWL úložiska,
   - a ďalšie knižnice tretích strán (dostupné na priloženom elektronickom nosiči)

[19] Log4j logging services, `http://logging.apache.org/log4j/`
[20] Hibernate – relational persistence for Java and .NET, `http://hibernate.org/`

2. mať nainicializovanú databázu. Potrebné skripty sa nachádzajú v:

   - `Prototype/LogAnalyzer/out/createTables_LogAnalyzer.txt` ,
   - `Prototype/UserLogs/out/createTables.txt` .

Ukážka volania nástroja v inom nástroji:

LogAnalyzer - Ukážka volania

```
try
{
    ILogAnalyzer la = (ILogAnalyzer) LogAnalyzer.getSingletonInstance();
    if(la != null)
    {
      la.signal();
    }
}
catch(Exception ex)
{
    //hande the exception here
}
```

E Obsah dátového nosiča

V štruktúre dátového nosiča sa dá navigovať pomocou webového prehliadača otvorením súboru `index.html`.

/**Conferences** – príspevky zaslané na konferencie

/**Prototype** – vytvorený prototyp

- /**Install** – potrebné softvérové vybavenie
- /**Click** – nástroj *Click*
- /**LogAnalyzer** – nástroj *LogAnalyzer*
- /**UserLogs** – dátová vrstva zdieľaná viacerými nástrojmi
- /**doc** – javadoc dokumentácia

/**Ontologies** – ontologické modely projektov NAZOU a MAPEKUS

/**Resources** – literatúra použitá pri vypracovaní práce v el. podobe

/**Thesis** – text diplomovej práce v elektronickej podobe