TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Automatická kompozícia webových služieb s využitím znalostných prístupov

2010 **Ing. Zoltán ĎURČÍK**

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

AUTOMATICKÁ KOMPOZÍCIA WEBOVÝCH SLUŽIEB S VYUŽITÍM ZNALOSTNÝCH PRÍSTUPOV

Dizertačná práca

Študijný odbor:	9.2.8 umelá inteligencia
Školiace pracovisko:	Katedra kybernetiky a umelej inteligencie (KKUI)
Školiteľ:	doc. Ing. Ján Paralič, PhD.

Košice 2010 **Ing. Zoltán ĎURČÍK**

Analytický list

Autor:	Ing. Zoltán Ďurčík
Názov práce:	Automatická kompozícia webových služieb s využitím znalostných prístupov
Jazyk práce:	slovenský
Typ práce:	Dizertačná práca
Počet strán:	125
Akademický titul:	PhD.
Univerzita:	Technická univerzita v Košiciach
Fakulta:	Fakulta elektrotechniky a informatiky (FEI)
Katedra:	Katedra kybernetiky a umelej inteligencie (KKUI)
Študijný odbor:	9.2.8 umelá inteligencia
Mesto:	Košice
Školiteľ:	doc. Ing. Ján Paralič, PhD.
Dátum odovzdania:	30. júl 2010
Dátum obhajoby:	30. september 2010
Kľúčové slová:	webová služba, kompozícia, plánovanie, plánovač, ontológia, OWL, OWL-S, PDDL
Kategória konspekt:	Výpočtová technika; Umelá inteligencia
Citovanie práce:	Ďurčík, Ing. Zoltán: Automatická kompozícia webových služieb s využitím znalostných prístupov. Dizertačná práca. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2010. 125 s.
Názov práce v AJ:	Automatic Web Service Composition with knowledge approaches
Kľúčové slová v AJ:	web service, composition, planning, planner, ontology, OWL, OWL-S, PDDL

Abstrakt v SJ

Táto dizertačná práca sa venuje problematike kompozície webových služieb s využitím metód plánovania a znalostných prístupov. Webové služby sú programové prostriedky umiestnené na počítačovej sieti (hlavne na internete). Sú popísané pomocou popisných jazykov (najčastejšie WSDL – Web Service Description Language) a komunikujú pomocou štrukturalizovaných správ (najčastejšie s využitím SOAP – Simple Object Access Protocol). Každá webová služba poskytuje určitú funkčnosť (napr. preklad slova z jedného jazyka do druhého). V prípade, že nie je možné požiadavku splniť pomocou jednej webovej služby (napr. je potrebné preložiť slovo z angličtiny do španielčiny, ale nemáme dostupnú službu, ktorá to vie preložiť priamo), prichádza na rad kompozícia webových služieb (napr. k dispozícii je jedna služba, ktorá prekladá anglické slovo do nemčiny a druhá služba, ktorá prekladá nemecké slovo do španielčiny). Kompozícia však môže byť omnoho zložitejšia v prípade komplexných služieb. Pre riešenie takýchto problémov boli v práci navrhnuté vlastné prístupy založené na znalostnej reprezentácii. Ide jednak o ontologické popisy webových služieb (OWL-S), resp. ďalšie ontologické popisy (OWL), napr. popis riešenej domény. Samotná kompozícia sa vykonáva pomocou plánovacích metód umelej inteligencie. Pre plánovanie bol zvolený jazyk PDDL. PDDL je jazyk, ktorý plánovaciu úlohu rozdeľuje do dvoch súborov: plánovacia doména a plánovací problém. Predpokladáme, že tieto sú na začiatku reprezentované sémanticky, t.j. pomocou ontológií, resp. pomocou sémantických štandardov. V práci je preto navrhnutý spôsob transformácie ontológií do PDDL plánovacej úlohy a kompletný systém kompozície webových služieb, ktorý danú transformáciu používa. Systém bol aj čiastočne implementovaný (transformačné pravidlá) a na tomto systéme je demonštrovaná funkčnosť navrhnutého prístupu pomocou vzorových príkladov kompozície webových služieb.

Abstrakt v AJ

The topic of this dissertation thesis is web service composition with using planning methods and semantic knowledge approaches. Web services are programs located on computer networks (mainly on internet). They are described by some description language (most frequently by WSDL - Web Service Description Language) and they communicate by structuralized messages (most frequently by using SOAP - Simple Object Access Protocol). Each web service provides certain functionality (e.g. translation between two languages). If it is not possible fulfill request by one web service (e.g. it's needed translate word from English to Spanish, but there is not available service, which directly translates words between these languages), web services composition can help here (e.g. there is one web service to translate English word to German, and second service, which translates German word to Spanish). The composition but may be much complicated with more complex services. In presented thesis were introduced own approaches for handling this problems. These approaches are based on using knowledge representation, either on ontological web service description (OWL-S) and other ontological description, e.g. description of domain (OWL - Ontology Web Language). The composition is performed by artificial intelligence planning methods. For planning was choices PDDL language. PDDL (Planning Domain Definition Language) is language, which divides planning task into two subsets: planning domain and planning problem. We assume that these subsets are represented semantically, i.e. by ontologies, or by using semantic standards. Therefore is in this work introduced method for transformation ontologies into PDDL planning task. Beside this is there proposed complete web service composition system, which was also partially implemented (transformation rules). With using some examples is on this system demonstrated functionality of proposed system.

Ciele práce

Cieľom tejto práce bolo:

- analyzovať súčasný stav v oblasti webových služieb a možností ich kompozície,
- analyzovať dostupné plánovacie metódy a prístupy z pohľadu ich využiteľnosti pre kompozíciu webových služieb,
- navrhnúť vlastné riešenie pre kompozíciu webových služieb s využitím znalostných prístupov,
- implementovať a experimentálne overiť funkčnosť vlastného návrhu pre automatickú kompozíciu webových služieb na báze znalostných prístupov.

Čestné vyhlásenie

Vyhlasujem, že som celú dizertačnú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 30. júl 2010

.....

vlastnoručný podpis

Poďakovanie

Ďakujem svojmu školiteľovi doc. Ing. Jánovi Paraličovi, PhD. za veľmi cenné pripomienky, postrehy a podporu poskytnutú pri písaní mojej práce. Ďalej by som sa chcel poďakovať svojim rodičom, priateľke a najbližšej rodine, ktorá ma podporovala pri písaní.

Predhovor

Hlavným dôvodom vzniku predkladanej práce bolo navrhnúť pôvodný príspevok k riešeniu problému kompozície webových služieb s využitím znalostných prístupov. Webové služby sa dostali do pozornosti hlavne s príchodom tzv. sémantického webu. Začali vznikať rôzne snahy o štandardizovanie popisu služieb, procesu výmeny informácií medzi službami, objavovania služieb v doméne internetu, vykonávania služieb, kompozície služieb a podobne. Predkladaná práca sa zaoberá práve kompozíciou webových služieb, pričom sa snaží využiť metódy umelej inteligencie z oblastí plánovania a znalostných reprezentácií.

V prvých kapitolách je uvedený prehľad problematiky webových služieb a používaných štandardov, ktoré súvisia s webovými službami. Je tu popísaný aj spôsob popisu webových služieb, spôsob komunikácie, ontológie a ich vzťah ku webovým službám. Ďalej je tu načrtnutý problém kompozície webových služieb a je podaný prehľad v súčasnosti najčastejšie používaných prístupov na jeho riešenie. Po predstavení problému kompozície webových služieb je uvedený problém plánovania a je analyzovaná jeho vhodnosť pre kompozíciu webových služieb. Sú tu predstavené a popísané najvhodnejšie plánovacie techniky a existujúce systémy, ktoré tieto techniky pri plánovaní používajú. Dané kapitoly tvoria teoretickú časť predkladanej práce. Po nej nasleduje praktická časť, v ktorej sa nachádza pôvodný príspevok k riešeniu problému kompozície webových služieb s využitím znalostných prístupov

Hlavným cieľom práce bolo načrtnúť a čiastočne implementovať vlastný systém pre automatickú kompozíciu webových služieb s využitím znalostných prístupov. Pre potreby daného systému bol navrhnutý proces transformácie sémantického popisu služieb do plánovacej úlohy. Vlastnému návrhu systému patrí hlavná časť práce.

Práca obsahuje aj niekoľko príloh, kde okrem príkladov využívaných na vysvetlenie problematiky preberanej v práci je k dispozícii aj aplikácia a popis aplikácie, ktorá demonštruje navrhované prístupy ku automatickej kompozícii webových služieb.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam algoritmov

Zoznam príkladov

Zoznam symbolov a skratiek

ADL **A**ction **D**escription **L**anguage

AI **A**rtificial **I**ntelligence

DAML **D**arpa **A**gent **M**arkup **L**anguage

EBNF **E**xtended **B**actus-**N**euer **F**orm

FF **F**ast **F**orward

HTN **H**ierarchical **T**ask **N**etwork

HTTP **H**yper **T**ext **T**ransfer **P**rotocol

KIF **K**nowledge **I**nterchange **F**ormat

OIL **O**ntology **I**nference **L**ayer (or **L**anguage)

OWL **W**eb **O**ntology **L**anguage

OWL-S **W**eb **O**ntology **L**anguage for Web **S**ervice

PDDL **P**lanning **D**omain **D**efinition **L**anguage

RDF **R**esource **D**escription **F**ramework

SOAP **S**imple **O**bject **A**ccess **P**rotocol

STRIPS **S**tanford **R**esearch **I**nstitute **P**roblem **S**olver

SWRL **S**emantic **W**eb **R**ule **L**anguage

W3C **W**orld **W**ide **W**eb **C**onsortium

WS **W**ebová **S**lužba

WSDL **W**eb **S**ervice **D**escription **L**anguage

WWW **W**orld **W**ide **W**eb

UDDI **U**niversal **D**escription, **D**iscovery and **I**ntegration

UI **U**melá **I**nteligencia

URI **U**niform **R**esource **I**dentifier

XML e**X**tensible **M**arkup **L**anguage

XSD **X**ML **S**chema **D**efinition

Slovník termínov

Ontológia je formálne, jednoznačné vymedzenie zdieľaných pojmov (Gruber, 1993)

Atóm v matematickej logike predstavuje základné stavebné bloky pre konštrukciu výrokov v predikátovej logike prvého rádu [41].

Literál označuje atomickú formulu, teda atóm, pričom pozitívny literál predstavuje atóm a negatívny literál predstavuje negáciu atómu

Predikát v matematike predstavuje reláciu, alebo funkciu, o ktorej možno povedať či je pravdivá alebo nepravdivá. V predikátovej logike prvého rádu môže predikát reprezentovať vlastnosť alebo vzťah medzi entitami.

Proces je postupnosť časovo usporiadaných udalostí tak, že každá predchádzajúca udalosť sa zúčastňuje na determinácii nasledujúcej udalosti.

1 Úvod

Internet je bezpochyby jedným z najzaujímavejších a najdôležitejších objavov druhej polovice dvadsiateho storočia. Počiatok internetu možno datovať približne do 60-tych rokov 20. storočia, kedy bola predstavená počítačová sieť ARPANET (Advanced Research Projects Agency Network). Bol to projekt ministerstva obrany Spojených štátov amerických, do ktorého boli postupne zavedené štyri univerzity - University of California Los Angeles, Stanford Research Institute, University of California Santa Barbara, University of Utah. Medzi jednu z hlavných požiadaviek patrila robustnosť siete. Z toho dôvodu sieť nemala žiadny centrálny uzol, ktorý by v dôsledku zlyhania ohrozoval fungovanie celej siete. Ďalším dôležitým medzníkom je rok 1987, kedy Timothy John Berners-Lee (častejšie Tim Berners-Lee) predstavuje koncept World Wide Web (WWW, označovaný aj ako web), čo možno voľne preložiť ako „celosvetová pavučina“. Tri roky nato publikuje koncept systému hypertextom[1] prepojených dokumentov. V roku 1990 končí ARPANET, ktorý sa medzitým dostal aj do Európy a o rok nato začína svoju cestu WWW. Tim Berners-Lee nasadzuje WWW v laboratóriách Cernu. V roku 1994 dochádza ku veľkej komercionalizácii internetu. V dôsledku pomerne veľkej prístupnosti osobných počítačov do firiem a do domácností a vývojom WWW prehliadačov (Mosaic – 1993, Netscape Navigator – 1994 atď.) nastáva veľká explózia internetových používateľov, ktorá sa stále zväčšuje (1996 – 55 mil. používateľov, 2006 – viac ako 1 miliarda používateľov, 31. december 2009 - 1,802,330,457[2]).

Je zbytočné spomínať všetky výhody internetu. Od zjednodušenia komunikácie, cez výmenu dát, rýchlejšie získavanie informácií až po jednoduché nakupovanie, internet v súčasnosti ponúka nepreberné množstvo možností a svojim spôsobom uľahčil používateľom viacero činností. Postupným rastom internetu a hlavne rastom počtu internetových stránok (v súčasnosti niekoľko stoviek miliónov) bola potreba efektívneho vyhľadávania informácií, o ktoré majú používatelia záujem. Preto sa začali objavovať rôzne vyhľadávače webových stránok. Za všetky stačí spomenúť giganty, akými sú Google (www.google.com), Yahoo (www.yahoo.com) alebo Microsoft (http://www.bing.com). Sú to fulltextové vyhľadávače, ktoré pracujú s celým obsahom

[1] Theodor Holm Nelson (*1937) je americký sociológ, filozof a priekopník informačných technológií. V roku 1963 poprvý krát použil termín "hypertext".
[2] Podľa zdroja: http://www.internetworldstats.com/stats.htm

webovej stránky. Napriek tomu sa s rastúcim množstvom stránok však čoraz častejšie začína objavovať problém s nájdením relevantných informácií. Architektúra pôvodnej WWW siete sa ukazuje byť čoraz viac nedostatočná.

Jedným z problémov je, že väčšina webových stránok bola (a niekedy aj je) orientovaná výlučne na používateľa. To znamená, že daná stránka obsahuje dáta výlučne pre užívateľa (čo je logické, keď bola preňho navrhnutá). Toto ale spôsobuje problémy najmä pri snahe o automatické spracovávanie údajov z internetu. Tento problém sa snaží riešiť tzv. sémantický web. Jednou z výhod sémantického webu je, že obsahuje a pracuje s metadátami (dáta od dátach). Tieto metadáta majú slúžiť najmä pre počítačové programy (napr. programy na automatické spracovanie stránok, vyhľadávací agenti a podobne). Takéto metadáta by mali, napr. už v spomínanej doméne vyhľadávania, priniesť skvalitnenie vyhľadávania relevantných informácií.

Internet bol (ak nemáme na mysli jeho úplné začiatky) a je stále orientovaný na používateľov, ktorých spokojnosť je jedným z prvoradých mierok úspešnosti webových aplikácií. Vyvstáva však otázka, kam bude smerovať ďalší vývoj internetu. Postupným zdokonaľovaním informačných technológií, či už ich hardvérovej, alebo softvérovej stránky, je možné produkovať čoraz sofistikovanejšie produkty v oblasti WWW. V súčasnosti sa ako veľmi perspektívne javí použitie webových služieb.

Webové služby sú distribuované programy, ktoré sú umiestnené na sieti a používané pomocou štandardných protokolov. Webové služby (ďalej WS) boli navrhnuté pre distribuované výpočty, pričom pri ich návrhu sa dodržiavali presné pravidlá. Medzi hlavné pravidlá patrí používanie štandardne definovaných protokolov pri tvorbe WS a nezávislosť použitia WS od prostredia, v ktorom bola aplikácia vytvorená. Inými slovami povedané sú WS programy, ktoré sú umiestnené na sieti (najčastejšie na internete a ich umiestnenie je známe) a my ich môžeme prostredníctvom tejto siete využiť. To znamená, že ich môžeme zavolať s cieľom získania určitých informácií alebo vykonania určitých akcií.

Ako jednoduchý príklad použitia WS možno uviesť napr. problém cestovania. Predpokladáme prípad, že neponecháme celú organizáciu na cestovnú kanceláriu, ale navyše budeme zasahovať do takých parametrov ako napr. cena, miesto, ubytovanie, spôsob dopravy. V minulosti bolo treba zájsť do cestovnej kancelárie osobne, tam si potom vybrať vhodnú dovolenku (z vopred presne daných možností), ktorá by spĺňala naše parametre a zaplatiť za ňu. V súčasnosti popri spomenutej možnosti existuje aj

možnosť vybaviť danú záležitosť na internete, tým že používateľ prezrie špecializované stránky a zvolí vhodné možnosti. Potom napr. bankovým prevodom zaplatí potrebné úhrady, prípadne si môže priamo cez internet doobjednať cestovné poistenie a pri tomto všetkom nemusí ani opustiť svoju izbu. Je zrejmé, že druhá možnosť výrazne uľahčuje situáciu používateľa (napr. z časového hľadiska, zdravotného – stres, a pod.). Prestavme si teraz ale možnosť, že jedinou používateľovou starosťou by bolo zadanie požiadaviek ako napr. dátum dovolenky, miesto dovolenky, maximálna cena ubytovania, spôsob dopravy vo forme jedného dopytu. Po takomto dopyte by používateľ dostal už len relevantné možnosti ku jeho dopytu a on by si jednoducho zvolil možnosť, ktorú považuje za najprijateľnejšiu a zadal príkaz na platbu, ktorá by takisto mohla prebehnúť automaticky. Jednotlivé časti interakcie používateľa so systémom by potom mohli obstarávať práve webové služby. Napr. WS1 vracia dostupné hotely v danej lokalite a danej cenovej réžii, WS2 vyhľadáva potrebné dopravné spojenie do miesta dovolenky, WS3 zabezpečujúca platbu, a podobne. Ako bolo spomenuté WS sú programy. Ako pri klasických nedistribuovaných programoch môžu ale aj nemusia mať vstupy a výstupy. Napr. WS zabezpečujúca hotel, by mala vstupy ako dátum, cena, atď. a poskytovala by informáciu Áno/Nie, na základe toho či je voľný alebo nie je. Treba si všimnúť, že pre splnenie uvedeného cieľa je určite potrebných viac ako jedna WS. Tu sa vynára oblasť, ktorá patrí v súvislosti s webovými službami ku najviac diskutovaným, a to oblasť (semi-)automatickej kompozície WS. Práve možnostiam automatickej kompozície WS je venovaná táto práca.

1.1 Motivácia

Tak ako bolo spomenuté v predchádzajúcej kapitole, WS patria v posledných rokoch medzi často diskutované témy v oblasti informačných technológií. Hlavnou výhodou WS je to, že už od svojho uvedenia sa orientovali na sémantické technológie a štandardy. Bola snaha o štandardizáciu popisu WS, spôsobu komunikácia a podobne. Čoraz viacero výskumníkov sa začalo zaoberať WS, či už ich bezpečnosťou, spôsobom komunikácie, ale aj ich vyhľadávaním v prostredí internetu a kompozíciou. Predkladaná práca sa zaoberá kompozíciu WS. Ako jednou z najlepších volieb pre kompozíciu WS je použitie metód plánovania z oblasti umelej inteligencie. Hlavný dôvod je ten, že problém kompozície je možné relatívne priamočiaro namapovať na plánovaciu úlohu. V dnešnej dobe existuje viacero implementácií plánovačov využívajúcich spomínané metódy plánovania z umelej inteligencie (napr. [19][20][21][29]). Niektoré tieto

plánovače začali vznikať dávno predtým, ako existovali webové služby. Ale práve možnosť transformácie problému kompozície na plánovaciu úlohu dáva možnosť ich využitia pre potreby kompozície WS. Danou problematikou sa zaoberalo viacero autorov [30][36][38]. Hlavnou motiváciou predkladanej práce bolo zmapovať problematiku kompozície WS pomocou metód plánovania v umelej inteligencii, pričom dôraz práce je kladený nato, aby boli využívané dostupné znalostné technológie sémantického webu. Aj napriek tomu, že kompozíciou webových služieb pomocou metód plánovania sa zaoberalo viacero autorov, nie je veľa prác, ktoré by detailne mapovali proces transformácie problému kompozície do plánovacej úlohy. Jednou z takých je napr. práca [37], v ktorej autori popisujú spôsob transformácie OWL-S popisov webových služieb do plánovacej domény SHOP2. Na danú prácu ďalej teoreticky nadviazali autori v [52], kde teoreticky načrtli spôsob tvorby PDDL plánovacej úlohy pomocou OWL-S popisov webových služieb. V predkladanej práci sa na základe analýzy problematiky webových služieb a ich kompozície pomocou umelo inteligentných metód plánovanie prezentuje vlastný návrh, ktorý na prvotný popis problému kompozície používa znalostné prístupy. K OWL-S popisom WS prezentovaný návrh pridáva aj ontologické OWL popisy domén. Preto bolo potrebné navrhnúť metódy pre transformovanie daných ontológií (OWL a OWL-S) na plánovaciu úlohu.

1.2 Predpokladané prínosy práce

Hlavným cieľom práce je analyzovať spôsob využitia znalostných prístupov pri popise problému kompozície webových služieb a navrhnúť spôsob transformácie tohto popisu do plánovacej úlohy. Pri riešení získanej plánovacej úlohy sa využije existujúci plánovací systém. Po vyriešení danej úlohy dostávame pracovný a dátový tok webových služieb. Webové služby sú popísané pomocou jazyka OWL-S. Ako plánovací jazyk bol zvolený jazyk PDDL. PDDL rozdeľuje plánovaciu úlohu do dvoch častí, a to na plánovaciu doménu a plánovací problém. Doména obsahuje popisy predikátov, popisy typov, akcií a podobne. Vzhľadom na doménu je potom vytvorený plánovací problém, ktorý obsahuje konkrétne objekty, počiatočný a koncový stav.

Problematikou transformácie OWL-S popisov webových služieb do PDDL sa zaoberali autori viacerých prác (napr. [30][38][57]). Hlavné rozdiely medzi jednotlivými prístupmi môžeme charakterizovať podľa spôsobu tvorby plánovacej úlohy. Niektorí autori využívajú na tvorbu len OWL-S popisy webových služieb (napr.

[30][52]), ostatní okrem daných popisov navrhli svoje vlastné doplnkové popisy plánovacej úlohy [38][39].

Predkladaná práca sa podobne ako existujúce práce spomenuté vyššie orientuje na spracovanie problematiky kompozície pomocou tvorby PDDL plánovacej úlohy. Prezentuje návrh využiť pri tvorbe PDDL plánovacej úlohy OWL doménové ontológie. Pomocou daných ontológií sa popisujú počiatočné a koncové stavy problému kompozície. Na tvorbu akcií boli použité OWL-S popisy WS, pričom sa vychádzalo z existujúcich návrhov [30][52]. Hlavný prínos práce je v komplexnom popísaní procesu tvorby PDDL plánovacej úlohy s využitím znalostných prístupov. V práci je predstavených niekoľko pôvodných algoritmov na spracovanie OWL ontológií a OWL-S popisov WS pri tvorbe PDDL plánovacej úlohy.

1.3 Štruktúra práce

Predkladaná práca je rozdelená do dvoch hlavných častí. Prvá časť popisuje analýzu súčasného stavu v relevantných oblastiach výskumu, t.j. oblasť webových služieb a plánovanie metódami umelej inteligencie:

- kapitola 2 popisuje stručne štandardy a technológie súvisiace s webovými službami s dôrazom na ontologický jazyk OWL (kap. 2.5) a z neho vychádzajúci jazyk pre sémantický popis webových služieb OWL-S (kap. 2.6),
- kapitola 3 popisuje problém automatickej kompozície webových služieb. Informuje o tom čo by mal obsahovať systém pre kompozíciu a predstavuje zopár možných domén, v ktorých by mohla mať kompozícia webových služieb uplatnenie,
- kapitola 4 popisuje plánovanie, plánovací problém a najznámejšie metódy plánovania, ktoré je možné použiť v súvislosti s kompozíciou webových služieb. V danej kapitole je popísaný aj jazyk PDDL (kap. 4.3), ktorý je v doméne plánovania v súčasnosti veľmi rozšírený. V závere kapitoly je predstavených niekoľko existujúcich návrhov a systémov zaoberajúcich sa kompozíciou webových služieb (kap. 4.5).

Druhá časť práce popisuje návrh vlastného systému pre kompozíciu webových služieb so zameraním sa na proces transformácie problému kompozície popísaného pomocou znalostných prístupov do plánovacej úlohy:

- v kapitole 5 je predstavený návrh systému pre kompozíciu webových služieb. Pri návrhu boli zohľadnené požiadavky kompozície, vhodnosť použitia metód plánovania umelej inteligencie a aj existujúce prístupy ku kompozícii webových služieb predstavené v kapitole 4.
- kapitola 6 je v práci najdôležitejšia a predstavuje návrh tvorby plánovacej úlohy s využitím znalostných prístupov pri tvorbe kompozície. Problém kompozície je popísaný pomocou ontológií. Pomocou OWL ontológií je popísaná doména, v ktorej sa má vykonávať kompozícia, počiatočný aj koncový stav. Webové služby sú popísané pomocou ontológií OWL-S. V danej kapitole je popísaný spôsob transformácie ontologických popisov do PDDL plánovacej úlohy.

Predkladaná práca okrem spomínaných častí tvoriacich jej hlavnú časť obsahuje aj prílohy:

- príloha A obsahuje príklady, na ktoré sa odvoláva hlavná časť a ktoré v dôsledku svojej veľkosti neboli uvedené v hlavnej časti práce.
- príloha B je CD médium, ktoré obsahuje prácu a prílohy v elektronickej podobe. Ďalej obsahuje program ku dizertačnej práci, ktorý bol vytvorený s cieľom demonštrovať návrh kompozície webových služieb a predovšetkým algoritmy a funkcie popísané v kapitole 6. Nachádza sa tu stručná používateľská príručka, systémová príručka a Javadoc dokumenty ku programu.

Sémantický web, sémantické webové služby a plánovanie metódami umelej inteligencie

2 Štandardy a technológie webových služieb

V nasledujúcej kapitole je predstavená problematika sémantických webových služieb. Sú tu predstavené WS, popísaný jazyk WSDL, ktorý reprezentuje jazyk najčastejšie používaný na popis webových služieb a protokol SOAP, čo je najčastejšie používaný spôsob komunikácie webových služieb. Podobne je predstavený a v primeranom rozsahu popísaný ontologický jazyk OWL a s ním súvisiaci ontologický popis webových služieb, označovaný ako OWL-S. Jednotlivé časti problematiky sú vysvetlené aj na príkladoch. Vzhľadom na veľkosť príkladov sme ich umiestnili do prílohy (viď Príloha A).

V tejto práci používaný spôsob odkazovania na príklady v prílohe je nasledovný:

- **(A P)** – odkaz na príklad **P** z prílohy **A**.
- **(A P x) –** odkaz na riadok **x** z príkladu **P** v prílohe **A**.
- **(A P x:y)** –odkaz na príklad **P** z prílohy **A**, konkrétne na riadky **x** až **y** z daného príkladu.

2.1 Webové služby

Webové služby sú **distribuované programy**, ktoré sú umiestnené **na sieti** (najčastejšie na internete) a používané **pomocou štandardných protokolov** (najčastejšie pomocou protokolu HTTP). Tento koncept bol predstavený hlavnými spoločnosťami v IT oblasti, ako Microsoft, IBM a Sun. Webové služby boli navrhnuté ako alternatíva ku distribuovane objektovo orientovaným štandardom ako COBRA a Java RMI [4]. COBRA je určený pre tvorbu objektovo orientovaných distribuovaných aplikácií, kde sa dôraz kladie na používanie platformovo nezávislých, distribuovaných objektov. Rozhranie distribuovaných objektov je definované pomocou jazyka IDL[3] (Interface Description Language). Java RMI je technológia, ktorá umožňuje distribuovanú komunikáciu objektov v Java. Objekt v jednej JVM (Java Virtual Machine) je schopný volať metódu iného objektu, ktorý je umiestnený na vzdialenej JVM.

Webové služby boli navrhnuté pre distribuované výpočty, pričom sa bral ohľad na nasledovné vlastnosti webových služieb [1][2]:

[3] IDL: http://jmvidal.cse.sc.edu/csce590/spring02/corba-idl-intro.pdf, 2010

- používanie štandardizovaných protokolov pri tvorbe webových služieb a nezávislosť používania WS vzhľadom na prostredie, v ktorom boli vytvorené,
- webová služba by mala byť samo popisná, čo znamená, že súčasne s webovou službou by malo byť dostupné aj rozhranie pre webovú službu,
- malo by byť jednoduché zverejniť webovú službu s cieľom jej jednoduchej objaviteľnosti v sieti.

Webové služby komunikujú s ich používateľmi a s ostatnými WS pomocou XML správ prostredníctvom Internetu. Popis operácií, vlastností WS a formát správ WS je prístupný pomocou rozhrania WS. Na popis rozhrania WS je použitý špeciálny popisný jazyk, najčastejšie jazyk WSDL.

Medzi najzákladnejšie štandardy a protokoly, s ktorými webové služby pracujú, patria nasledovné:

- WSDL – Web Service Description Language (kapitola 2.2)
- SOAP - Simple Object Access Protocol (kapitola 2.3)
- UDDI - Universal Description, Discovery and Integration (kap. 2.4)
- OWL – Ontology Web Language (kapitola 2.5)
- OWL-S – Semantic Markup for Web Services (kapitola 2.6)

Najčastejšie používané webové služby sú služby, ktoré komunikujú pomocou SOAP správ. Tieto služby sa preto označujú ako SOAP WS. Popri SOAP WS však existujú aj iné, napr. v poslednej dobe relatívne diskutované REST (Representational State Transfer) služby [62]. REST reprezentuje štýl architektúry pre distribuované systémy. Na rozdiel od SOAP služieb, ktoré môžu byť orientované dátovo a procedurálne, je REST orientovaný výlučne dátovo. Webové služby definujú procedúry a protokol prístupu ku nim, a REST potom popisuje ako sa pristupuje ku dátam.

2.2 WSDL – Web service description language

Každá webová služba musí byť popísaná určitým popisným jazykom, ktorý nám ako používateľom, alebo softvérovým agentom, umožňuje získavať informácie

o webovej službe. Tieto informácie v sebe zahrňujú napr. ponuku informácií, parametrov informácií, spôsob komunikácie so službou a podobne. Medzi najrozšírenejší jazyk pre popis webových služieb patrí práve jazyk WSDL. WSDL (Web Service Description Language) je **popisný jazyk** založený na technológii XML. Popis služby pomocou WSDL na dva hlavne ciele:

- opis služby,
- možnosť lokalizovať službu.

Hlavným cieľom WSDL bolo zaviesť určitý stupeň štrukturalizácie popisu WS. WSDL danú štrukturalizáciu umožňuje použitím XML technológie pomocou možnosti definovať špecifické elementy a atribúty, ktoré majú za úlohu vymedziť formát pre popis vymieňaných informácií. WSDL vymedzuje webové služby ako množiny **koncových bodov**, ktoré pracujú so **správami**. Dané správy obsahujú dokumentovo alebo procedurálne orientované informácie. **Operácie** a **správy** sú vo WSDL popísané abstraktne, pričom sú ďalej viazané na konkrétny protokol a konkrétny formát správ pre definovanie jedného koncového bodu. WSDL popis WS definuje **služby** (angl. services) ako množinu koncových bodov, alebo **portov** (angl. ports). Abstraktné definície správ (angl. messages) popisujú dáta, ktoré sú spracovávané službou. Abstraktné definície operácií sú súčasťou tzv. **typu portu** (angl. port type). Po abstraktných definíciách je potrebné pre ne získať konkrétny protokol a špecifikáciu formátu dát. Toto sa zabezpečuje prostredníctvom **viazania** týchto abstraktných definícií na konkrétne protokoly a formáty dát v časti viazanie (angl. binding).

Pre definovanie webových služieb pomocou WSDL sú používané elementy uvedené nižšie [5]. Príklady použitia jednotlivých elementov je možné vidieť v (A 1).

- **types** (A 1.1 14:18) – popisuje definície typov dát, ktoré prislúchajú správam vymieňaným službou. Pre popis typov dát je možné použiť ľubovoľný systém dát. Odporúča sa však používať XSD (XML Schema Definition) (viď príklad A 1.2).
- **messages** (A 1.1 20:43) – predstavujú abstraktné definície spracovávaných dát. Skladajú sa z jednej alebo viacerých častí, kde každej časti prislúcha typ z typového systému (napr. z XSD). Pre toto priradenie WSDL v správach používa dva typy atribútov, atribút element, ktorý odkazuje na XSD element a atribút type, ktorý odkazuje na XSD typ.

- **operation** (A 1.1 45:68) – predstavujú abstraktné definície operácií (funkcií) podporovaných službou. Využívajú atribút `name`, ktorým sa operácii definuje v rámci popisu jedinečné meno. V prípade operácie môže každý koncový bod používať jeden zo štyroch spôsobov komunikácie:
  1. *jedno cestná komunikácie* (angl. One - way) – koncový bod prijíma správu.
  2. *požiadavka – odpoveď* (angl. Request - response) – koncový bod prijme správu a odošle odpoveď.
  3. *vynútená odpoveď* (angl. Solicit - response) – koncový bod odošle správu a prijme odpoveď.
  4. *notifikácia* (angl. Notification) – koncový bod posiela správu.
- **portType** (A 1.1 44:69)– predstavuje množiny zahrňujúcu abstraktné popísané operácie a správy.
- **binding** (A 1.1 70:109)– popisuje formát dát pre správy a protokol pre operácie definovaných v type portu. Podobne ako aj pri ostatných elementoch musí mať aj viazanie jedinečné meno v rámci popisu WS. Je to zabezpečené použitím atribútu name.
- **ports** (A 1.1 111:113)– predstavujú jedinečný koncový bod, ktorý je reprezentovaný konkrétnou sieťovom adresou
- **service** (A 1 110:114)– obsahuje množinu súvisiacich koncových bodov.

V (A 1) je uvedený príklad WSDL popisu webovej služby. Daná služba poskytuje štyri operácie: drive (45:50), cross (57:62), board (51:56) a disembark (63:68). Služba je zameraná na doménu cestovanie. Jednoduchý opis operácií môže byť nasledovný: operácia `drive`, premiestni určitú vec (napr. osobu, vozidlo a pod.) z miesta A do miesta B po ceste. Operácia `cross` je podobná operácii `drive`, s rozdielom, že sa premiestňuje po moste. Operácia `board` slúži osobe na nastúpenie do vozidla a `disembark` naopak na vystúpenie z vozidla.

V prípade WS bolo potreba štrukturalizovať aj spôsob výmeny dát. V úvode kapitoly 2 bolo uvedené, že najčastejšie používaný protokol pre štrukturalizovanú výmenu dát je protokol SOAP. WSDL umožňuje priame prepojenie na SOAP protokol. Pomocou špecifických elementov je možné zapísať nasledovné informácie:

- informácia, že dané WSDL využíva protokol SOAP 1.1,
- definícia adries koncových bodov,

- URI[4] pre SOAP viazanie,
- zoznam definícií hlavičiek nachádzajúcich sa v SOAP správe.

2.3 SOAP - Simple Object Access Protocol

SOAP (Simple Object Access Protocol) je protokol založený na XML. Slúži aplikáciám (webovým službám) na výmenu informácií cez sieťové rozhranie, najčastejšie pomocou HTTP. SOAP je komunikačný protokol, ktorý slúži na komunikáciu cez internet. Je platformovo a jazykovo nezávislý, tzn. že je navrhnutý tak, aby bol nezávislý na konkrétnom programovacom modeli. Je vyvíjaný ako W3C[5] štandard.

Hlavný dôvod vývoja SOAP bolo zabezpečiť aplikáciám komunikáciu cez sieťové rozhranie, hlavne cez internet, pomocou XML správ. V minulosti bola komunikácia cez sieťové rozhranie prevádzaná najmä pomocou RPC (Remote Procedure Calls) medzi objektmi ako DCOM alebo COBRA. V prípade RPC vystupuje do popredia problém kompatibility a bezpečnosti, pričom firewall-y a proxy servery blokujú tento spôsob komunikácie. Výhodnejšie na komunikáciu medzi aplikáciami je využiť HTTP, pretože je podporovaný všetkými internetovými prehliadačmi a servermi. SOAP poskytuje cestu na komunikáciu medzi aplikáciami bežiacimi na rôznych operačných systémoch, s rôznymi technológiami a programovacími jazykmi.

Základné stavebné bloky SOAP a jeho štruktúra je vysvetlená na nasledovnej štruktúre SOAP správy [7]:

```
<?xml version="1.0"?>
  <soap: Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   soap:encodingStyle="http://www.w3.org/2003/05/soapencoding">
    <soap:Header>
      ...
    </soap:Header>
    <soap:Body>
      ...
      <soap:Fault>
        ...
      </soap:Fault>
    </soap:Body>
  </soap:Envelope>
```

[4] URI Uniform Resource Identifier - http://tools.ietf.org/html/rfc3986
[5] World Wide Web Consortium: http://www.w3.org/

SOAP správa začína hlavičkou XML dokumentu **<?xml version="1.0"?>**. Za ňou nasleduje element **Envelope**, ktorý je koreňovým elementom a vyskytuje sa v dokumente len raz. **Envelope** (angl. obálka, obal), v sebe zahrňuje celú SOAP správu. Obsahuje atribúty ako napr. menný priestor **xmlns:soap="http://www.w3.org /2003/05/soap-envelope"**. Môže obsahovať element **Header** a musí obsahovať element **Body**. **Header** (hlavička) umožňuje rozšíriť funkčnosť SOAP správy. **Body** element obsahuje jadro SOAP správy. Ďalej správa môže obsahovať element **Fault** (chyba) ktorý poskytuje informácie o chybách, ktoré nastali počas vykonávania správy.

V (A 2) je uvedený príklad SOAP požiadavky a odozvy na danú požiadavku. V SOAP požiadavke (A 2.1) je uvedená správa, ktorá je odosielaná ako požiadavka na server. Naša konkrétna správa volá metódu **PrekladSlova** (A 2.1 8). Daná metóda má jeden parameter **anglicky**, ktorý sa nachádza ako element vo vnútri elementu **PrekladSlova** (A 2.1 9). Pomocou danej metódy chceme preložiť slovo **„hello“**, ktoré sa nachádza v elemente **anglicky**. Ako odpoveď na našu požiadavku je zo strany servera poslaná správa (A 2.2) obsahujúca element **PrekladSlovaResponse**. Element **PrekladSlovaResponse** je odpoveďou na našu požiadavku a požadovaná hodnota sa nachádza v obsahu elementu **return** (**„ahoj“**).

2.4 UDDI - Universal Description, Discovery and Integration

UDDI (Universal Description, Discovery and Integration) je štandard na registráciu, kategorizáciu a vyhľadávanie webových služieb. Spôsobom práce pripomína katalóg, v ktorom sú uložené informácie o poskytovateľoch WS a poskytovaných webových službách. Služby registrované v UDDI musia pracovať so štandardami ako WSDL a SOAP a ku každej webovej službe by mal byť príslušný WSDL dokument. Z daného popisu je potom možné vygenerovať SOAP požiadavku na WS. Štandard UDDI zastrešuje organizácia OASIS. Organizácia OASIS je zoskupenie popredných IT firiem, ako napríklad IBM, Microsoft, HP, Oracle, Sun Microsystems, Intel a ďalších, čo zaisťuje jeho silné zázemie.

V UDDI sa dá vyhľadávať troma spôsobmi [8][15]:

- **White pages** (biele stránky): v bielych stránkach sa vyhľadáva meno spoločnosti, popis spoločnosti, kontaktné údaje, pripadne identifikátory z iných systémov. Vyhľadáva sa ten, *kto* poskytuje službu.

- **Yellow pages** (žlté stránky): v žltých stránkach sa vyhľadávajú služby podľa rôznych kategórií, do ktorých sú spoločnosti zaradené (priemyselné kategórie, kategórie služieb, geografické kategórie, ale je možnosť zadefinovať si aj vlastné kategórie). Vyhľadáva sa *čo* je poskytované.
- **Green pages** (zelené stránky): v zelených stránkach sa vyhľadávajú webové služby podľa funkčnej (business process – napr. WSCI) a technickej špecifikácie (service specification – WSDL), prípadne rôzne kategórie webových služieb. Vyhľadáva sa *ako* je služba poskytnutá.

V súvislosti s UDDI existujú aj návrhy pre využívanie iných technológií na popis a vyhľadávanie služieb. Napr. pre sémantických popis služieb môžu byť využité priamo OWL-S ontológie (kapitola 2.6). Možnosť kombinovať UDDI a OWL-S je popísané napr. v [55].

2.5 OWL – Ontology Web Language

OWL je jazyk určený pre popis ontológií na internete. Patrí do skupiny jazykov umožňujúcich reprezentáciu znalostí a je schválený konzorciom W3C. V počítačovej vede ontológia predstavuje formálnu reprezentáciu znalostí pomocou množiny konceptov v určitej doméne a pomocou vzťahov medzi týmito konceptmi. OWL je sémantický jazyk pre publikovanie a zdieľanie ontológií. Bol vyvíjaný ako rozšírenie RDF [59](Resource Framework Description) a je odvodený zo starších typov ontologických jazykov ako DAML+OIL[6].

Jazyk OWL sa z pohľadu expresívnosti delí na tri podjazyky [10][60]:

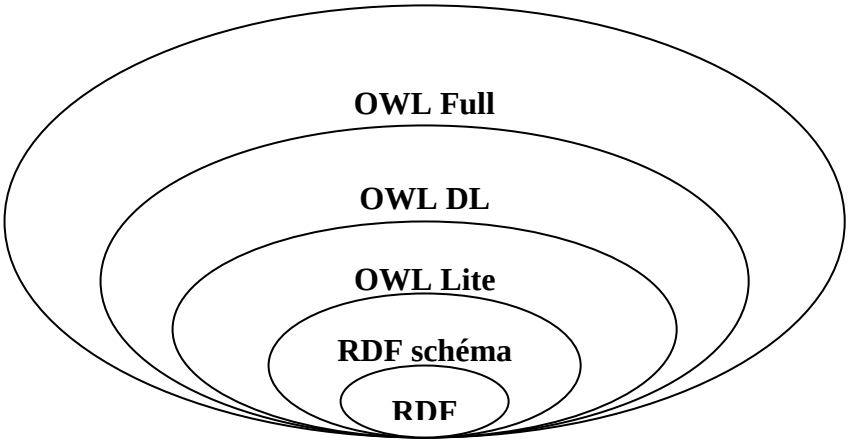
1. **OWL Lite** predstavuje základný podjazyk OWL. Poskytuje základné možnosti tvorby hierarchie tried spolu so základnými možnosťami tvorby ohraničujúcich podmienok (napr. kardinalita tried (0 - 1) a podobne). Jeho hlavná výhoda je tvorba relatívne jednoduchých a zároveň robustných nástrojom pre spracovanie takýchto ontológií.
2. **OWL DL** je podjazykom OWL, ktorý poskytuje maximálnu expresivitu jazyka pri zachovaní výpočtovej úplnosti pri spracovaní ontológií. Skratka DL znamená deskripčná logika (angl. Description Logic) a jazyk sa opiera hlavne o deskripčnú logiku prvého rádu.

[6] DAML (Darpa Agent MarkUp Language) + OIL (Ontology Interface Language): http://www.daml.org/2001/03/daml+oil-index

3. **OWL Full** je svojou expresivitou podobný OWL DL. V prípade syntaktického vyjadrovania však jazyk zachádza oveľa ďalej a umožňuje používať syntaktickú voľnosť RDF. Z tohto dôvodu pri spracovávaní ontológie nie je garantovaná výpočtová úplnosť.

Medzi týmito troma podjazykmi, RDF a RDF schémou existujú určité vzťahy (Obr.1):

- každá validná OWL Lite ontológia je validná OWL DL ontológia
- každá validná OWL DL ontológia je validná OWL Full ontológia.



Obr. 1 Vzťahy podjazykov OWL na základe expresívnosti s RDF a RDF schémou

OWL ontológie využívajú pre svoju reprezentáciu RDF a RDF schému. Ontológia je preto reprezentovaná ako RDF graf, ktorý je tvorený z tripletov. Ontológie môžu byť zapísané rôznou syntaktickou formou. Medzi dve najčastejšie používané patria RDF/XML (napr. (A 3)) a OWL/XML. Ako príklad môže poslúžiť zápis toho istého RDF tripletu pomocou oboch foriem (Príklad 1.)

RDF/XML:

```
<owl:Class rdf:ID=”Automobil”/>
```

OWL/XML:

```
<rdf:Description rdf:about:”#Automobil”>
      <rdf:type
rdf:resource=”http://www.w3.org/2002/07/owl#thing/>
</rdf:Description>
```

Pr. 1 RDF/XML a OWL/XML zápis triedy ontológie

Štruktúra OWL ontológie sa skladá z troch častí [9]:

1. **definovanie menných priestorov**,

2. **hlavička ontológie**,

3. **elementy ontológie**.

Problematika menných priestorov [6] spočíva v tom, že každý element dokumentu (v našom prípade ontológie) by mal mať jednoznačne pridelený menný priestor. Menný priestor predstavuje priestor, v ktorom sú všetky názvy jednotlivých termov unikátne. Tieto termy sú potom charakterizované vlastným menom a menným priestorom, ktorý používajú. Preto ak v ontológii používame term, musí byť presne špecifikované odkiaľ daný term pochádza. Toto je dané pomocou deklarácie menných priestorov nachádzajúcej sa v tagu **`rdf:RDF`** (napr. (A 3.1 10:16)).

Hlavička ontológie sa nachádza v tagu **`owl:Ontology`**. Hlavička môže obsahovať informácie ako napr. popis ontológie pre ľudských používateľov (**`rdfs:comment`**), verzie ontológie (**`owl:priorVersion`**), názov ontológie (**`owl:Label`**), importy ďalších ontológií (**`owl:import`**) a podobne (napr. (A 3.1 17)).

Po definovaní menných priestorov a hlavičky ontológie je možné definovať samotné elementy ontológie (napr. (A 3.1 20:112)). Medzi hlavné časti ontológie patria:

- **triedy**,
- **vlastnosti**,
- **inštancie tried** a **vzťahy medzi týmito inštanciami**.

Základnou vlastnosťou ontológií je schopnosť definovať triedy. Triedy si môžeme predstaviť ako množiny združujúce objekty s rovnakými vlastnosťami. V OWL ontológiách súvisia s triedami dva hlavné elementy. Prvý je **`owl:Class`** a umožňuje definovanie triedy v ontológii. Druhým je **`rdfs:subClassOf`**, ktorý informuje o tom, že daná trieda je podtriedou nejakej inej triedy. (napr. (A 3.1 86:112)).

S triedami súvisia inštancie tried (označované aj ako OWL Individual). Inštancie triedy predstavujú jej členov (napr. (A 3.2 24:95)). Napr.:

```
<owl:NamedIndividual rdf:about="karol">
      <rdf:type rdf:resource="person"/>
</owl:NamedIndividual>
```

hovorí o tom, že **`karol`** je inštanciou triedy **`person`**.

V prípade že sú definované triedy a inštancie tried, možno definovať všeobecné fakty o triedach a špecifické fakty o inštanciách. Vlastnosti v OWL ontológiách sú binárne relácie a môžu byť dvojakého typu:

- dátové vlastnosti (v ontológii **`owl:DatatypeProperty`**)
- objektové vlastnosti (**`owl:ObjectProperty`**)

Dátové vlastnosti predstavujú relácie medzi inštanciami tried a dátovými typmi z XML schémy (napr. (A 3.1 69:83)). Objektové vlastnosti predstavujú relácie medzi dvoma inštanciami tried (napr. (A 3.1 20:66)). Oba typy majú podobný spôsob deklarácie. Pri oboch typoch je potrebné zadefinovať doménu (**`rdfs:domain`**), na ktorú sa daná vlastnosť viaže (definičný obor vlastnosti) a rozsah (**`rdfs:range`**), z ktorého hodnoty môže vlastnosť nadobúdať (obor hodnôt). Doména vlastnosti je obyčajne reprezentovaná OWL triedou, na ktorú sa daná vlastnosť viaže. Rozsah je v prípade objektových vlastností reprezentovaný obvykle tiež OWL triedou. Pri dátových vlastnostiach je rozsah reprezentovaný odkazom na príslušný dátový typ z XML schémy. Príkladmi dátových typov z XML schémy sú: **`xsd`**[7]**`:float`** – reálne čísla, **`xsd:boolean`** – pravdivostná hodnota, **`xsd:integer`**, **`xsd:date`** – dátum, a podobne.

2.6 OWL-S (Semantic Markup for Web Services)

OWL-S môžeme charakterizovať ako sémantický popisný jazyk pre webové služby. Ako už bolo spomenuté v kapitole 2.1, webové služby sú programy umiestnené na sieti (najčastejšie na internete) a používané pomocou štandardných komunikačných protokolov (najčastejšie HTTP). Každá webová služba by mala byť popísaná určitým popisným jazykom, ako napr. najčastejšie používaný WSDL jazyk (kap. 2.2). Tento umožňuje poskytnúť používateľovi dôležité informácie o službe, ako sú vstupy, výstupy, protokoly komunikácie, umiestnenie služby a podobne. Tieto informácie však začali byť z pohľadu sémantického webu, hlavne z pohľadu automatizácie určitých činností, nepostačujúce. Daný popis nám napr. umožňuje povedať, že vstupom do služby je reťazec (z XML schémy `xsd:string`), neumožňuje však povedať, čo daný reťazec reprezentuje (napr. či je to názov knihy alebo meno auta). S príchodom ontológií sa ukázalo byť vhodné ontologicky popísať webové služby a prepojiť tieto ontológie s už existujúcimi ontológiami. Jedna z najčastejšie používaných ontológií na sémantický popis webových služieb je OWL-S (staršie verzie označované ako DAML-S). Hlavnými motivačnými úlohami, ktoré podnietili vznik OWL-S, boli nasledovné [11]:

[7] XML Scheme Definition: http://www.w3.org/XML/Schema.html

- **automatické získavanie WS** – predstavuje proces automatického objavovania služieb, ktoré spĺňajú určité požiadavky klienta.
- **automatická invokácia služieb** – je automatické vyvolanie procesu, ktorý služba poskytuje, prostredníctvom klienta (počítačového programu).
- **automatická kompozícia webových služieb** – spočíva v automatickom výbere WS a ich kompozícii do pracovného a dátového toku s cieľom splnenia určitej komplexnejšej úlohy.

OWL-S popis WS sa skladá z troch hlavných častí:

- **profil služby** – opisuje čo služba vykonáva a akú funkčnosť poskytuje (napr. (A 5 95)),
- **grounding služby** – popisuje ako služba komunikuje (napr. (A 4 246:295)),
- **procesný model služby** - popisuje ako služba pracuje (napr. (A 4 109)).

Treba poznamenať, že dané tri časti môžu byť reprezentované prostredníctvo troch samostatných ontológií uložených v troch samostatných súboroch a spoločne importovaných v hlavnej ontológii určenej pre popis webovej služby. Môžu sa však nachádzať aj v spoločnej ontológii (napr. príklad (A 4)).

Profil služby poskytuje špeciálnu reprezentáciu služby pomocou OWL tried a podtried, pričom táto reprezentácia je poskytnutá prostredníctvom triedy **Profile**. OWL-S profil popisuje služby pomocou troch základných druhov informácií:

- aká organizácia poskytuje službu,
- akú funkcionalitu má daná služba,
- a pomocou špecifických charakteristík služby (vstupy, výstupy, predpoklady a efekty).

Vzťah medzi službou a profilom služby je daný pomocou vlastností **`presents`** (prezentuje) a **`presentedBy`** (prezentovaný). **`Presents`** vyjadruje vzťah medzi službou a profilom služby. **`PresentedBy`** je inverzná vlastnosť ku predchádzajúcej a poskytuje informácie v profile o tom, ktorú službu daný profil predstavuje.

Ďalej sa v profile môžu nachádzať aj informácie pre ľudského používateľa:

- **`serviceName`** - názov služby
- **`textDescription`** - stručný popis služby

- **`contactInformation`** - kontaktné informácie pre používateľa, napr. pomocou FOAF[8] alebo VCard[9], a podobne.

Medzi informácie, ktoré so sebou nesie profil služby, patria tzv. špecifikácie funkcionality služby (popisy vstupov, výstupov služby) a popis podmienok, ktoré musia byť splnené aby sme dosiahli výsledok (prepisy predpokladov a efektov). Súhrnne sa tieto informácie označujú ako **IOPE** (z angl. **Inputs Outputs Preconditions Effects**). Pre IOPE sú v profile služby definované nasledovné vlastnosti:

- **`hasParameter`** – odkazuje na parameter z procesnej ontológie,
- **`hasInput`** – reprezentuje inštancie vstupov z procesnej ontológie,
- **`hasOutput`** – reprezentuje inštancie výstupov z procesnej ontológie,
- **`hasPrecondition`** – špecifikácia predpokladov z procesnej ontológie,
- **`hasResult`** – reprezentuje jeden výsledok definovaný pomocou triedy result v procesnej ontológii.

Všetky tieto vlastnosti súvisia s procesným modelom služby. Je to dané tým, že profil služby nevie priamo popísať IOPE a odkazuje sa na procesný model, ktorý má vytvorené konštrukty pre definovanie týchto informácií. Profil služby na ne referuje práve prostredníctvom spomínaných vlastností.

V OWL-S ontológii sú operácie modelované ako **procesy**. Dané procesy sa nachádzajú v procesnej ontológii, ktorá reprezentuje procesný model služby. Treba poznamenať, že proces nie je program, ktorý môže byť priamo vykonaný. Je to len špecifikácia toho, ako môže klient interagovať so službou, čo musí byť splnené aby mohla byť služba vykonaná a čo nastane po vykonaní danej služby. V OWL-S ontológii poznáme tri typy procesov [11]:

- **atomický proces** – z pohľadu klienta predstavuje model jedného kroku webovej služby, ktorá je priamo vykonaná pre dosiahnutie určitého cieľa. Vykonanie atomického procesu spočíva vo volaní odpovedajúceho webového programu (služby) s hodnotami viazanými na jednotlivé parametre (IOPE).
- **zložený process** predstavuje proces, ktorý môže byť dekomponovaný do niekoľkých atomických, jednoduchých alebo aj ďalších zložených procesov. Kompozícia zloženého procesu sa uskutočňuje prostredníctvom riadiacich

[8] FOAF - FOAF Vocabulary Specification 0.97: http://xmlns.com/foaf/spec/
[9] VCard - http://www.w3.org/TR/vcard-rdf/

štruktúr. Množina riadiacich štruktúr zahŕňa: *sequence* (sekvencia), *choice* (voľba), *if-then-else* (ak-potom-inak), *repeat-while* (opakuj pokiaľ), *repeat-until* (opakuj až do), *split* (delenie), *split-joint* (delenie a spájanie) a *any-order* (ľubovoľný výber).

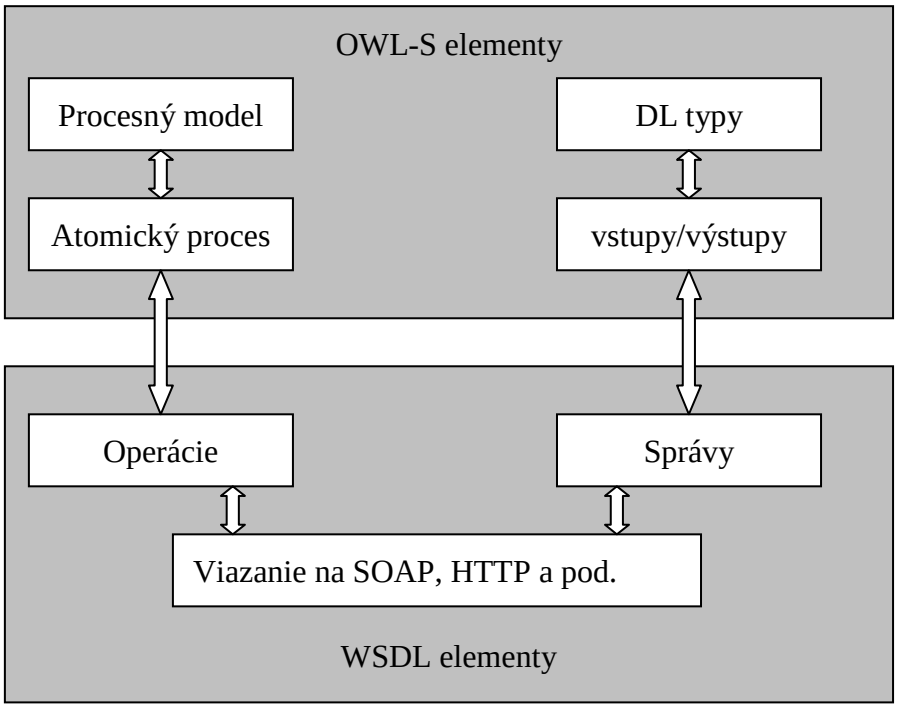
- **jednoduchý proces** poskytuje abstraktné videnie akcií. To znamená, že na rozdiel od atomických procesov sa je možné pozrieť do internej štruktúry jednoduchého procesu.

Proces popisuje IOPE pomocou nasledovných vlastností:

- **`hasParticipant`** – popisuje účastníkov procesu. Sú minimálne dvaja, a to klient na jednej strane a server na strane druhej.
- **`hasInput`** – spolu s ďalšou vlastnosťou `hasOutput` reprezentuje tzv. dátové transformácie. Popisuje vstupné dáta do procesu. V prípade atomického procesu tieto dáta poskytuje klient. V prípade zloženého procesu môžu dáta pochádzať z iného procesu.
- **`hasOutput`** – popisuje výstupné dáta procesu. V prípade atomického procesu sú tieto dáta prezentované klientovi. Pri zloženom procese môžu byť brané ako vstupné dáta do iného procesu.
- **`hasLocal`** – popisuje lokálne premenné procesu.
- **`hasPrecondition`** – predstavuje predpoklady, ktoré musia byť splnené aby mohlo dôjsť ku vykonaniu procesu.
- **`hasResult`** – predstavujú výsledok po vykonaní procesu. Zahrňujú v sebe podmienkové výstupy a efekty.

OWL-S umožňuje zapísať podmienky pre predpoklady a efekty viacerými spôsobmi. Medzi najčastejšie používané patrí popis podmienok pomocou jazyka SWRL (Semantic Web Rule Language) [12] a KIF (Knowledge Interchange Format) [13].

Grounding služby detailne popisuje ako pristupovať ku službe. Popisuje protokoly komunikácie, formáty správ a podobne. OWL-S môžeme v groundingu služby priamo naviazať na popis služby, ktorý využíva určitý popisný jazyk [14]. V prípade WSDL je vzťah medzi OWL-S a WSDL vyjadrený na obrázku 1:



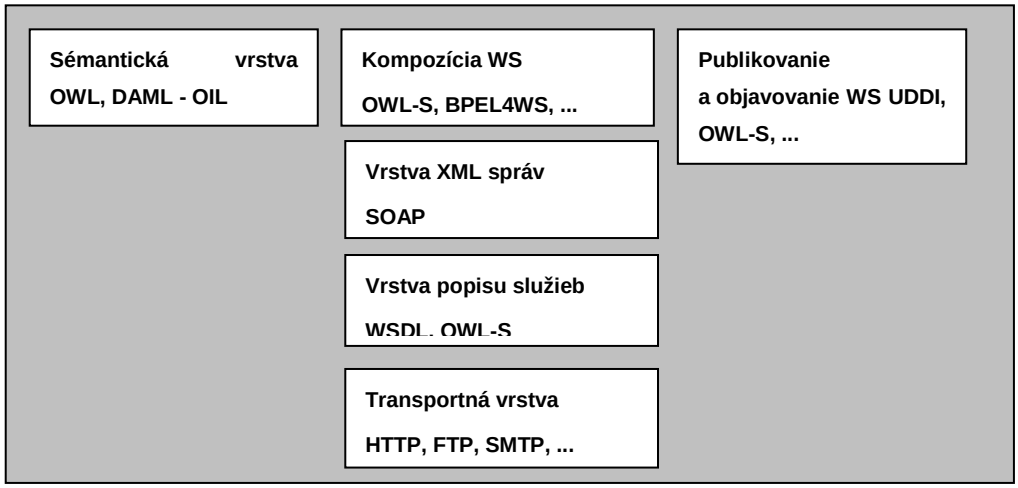
Obr. 2 Vzťah medzi OWL-S a WSDL

3 Automatická kompozícia webových služieb

Webové služby (kap. 2.1) sú považované za samostatné, samopopisujúce aplikácie, ktoré sú umiestnené a vyvolávané prostredníctvom siete, najčastejšie internetu. V súčasnosti čoraz viac organizácií, spoločností ale aj jednotlivcov implementuje svoje aplikácie alebo ponúka svoje služby práve pomocou technológie webových služieb. Webové služby majú väčšinou určité vstupy a výstupy. To znamená, že po dopyte zvonku poskytnú určitú informáciu na základe hodnôt, ktoré im boli v dopyte poskytnuté. Vo všeobecnosti platí, že ak pre daný dopyt nevyhovuje jedna služba, je možné ju splniť kombináciou (kompozíciou) jednotlivých elementárnych webových služieb (WS).

V súvislosti s tým, že WS sú produkované rôznymi spoločnosťami a jednotlivcami, bolo potrebné vytvoriť štandardy a jazyky, ktoré by obmedzili v určitej miere vznik veľkej heterogenity v tejto oblasti. Medzi v súčasnosti najviac diskutované a používané jazyky a štandardy patria WSDL (kap. 2.2), SOAP (kap. 2.3) a OWL-S (kap. 2.5).

Okrem spomínaných jazykov existujú aj ďalšie, napr. BPEL4WS (Business Proces Execution Language for Web Service), ktorý sa zameriava na reprezentáciu kompozície služieb. V tomto prípade je ale pracovný tok a vzťahy medzi službami dopredu známy.



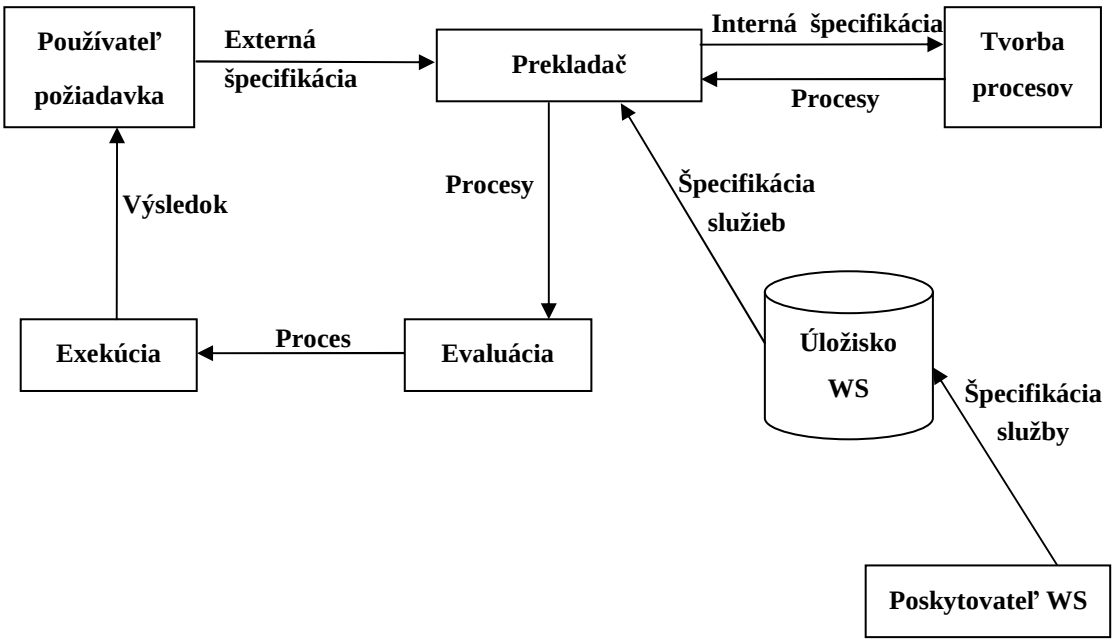
Obr. 3 Prehľad zaradenia prezentovaných technológií webových služieb [16]

Na Obr. 3 sú znázornené funkcie jednotlivých štandardov súvisiacich s webovými službami z pohľadu sémantického webu.

Medzi hlavné, veľmi globálne vzaté problémy súvisiace s kompozíciou WS, možno zaradiť [17]:

- rapídne narastanie počtu WS, dostupných na internete,
- WS môžu byť v ľubovoľnom okamžiku zmenené, preto je potrebné pred samotným použitím služby skontrolovať, či bude aj zmenená služba vykonávať to, čo sa od nej očakáva,
- aj napriek snahe používať určité pravidlá a štandardy pri tvorbe WS a pri ich popise môže medzi rôznymi WS existovať značná miera heterogenity, ktorá môže vychádzať napr. z toho, ako si jednotliví tvorcovia WS pomenujú jednotlivé premenné.

Ako už bolo spomenuté, WS sú programy, ktoré môžu mať určité vstupy a výstupy. Ak by bola kompozícia uskutočňovaná len vzhľadom na vstupy a výstupy, bola by relatívne jednoduchá. Jednoducho by mohlo dochádzať napríklad ku spätnému reťazeniu služieb, kde by sa hľadali služby, ktoré môžu splniť daný cieľ. Avšak WS sú často súčasťou určitého modelu sveta a ich exekúcia môže meniť podmienky, v akých sa daný model aktuálne nachádza. Preto sa pri WS službách často nachádzajú aj predpoklady (angl. preconditions) a efekty (angl. effects). Predpoklady sú podmienky, ktoré musia byť splnené, aby mohla byť daná služba vykonaná. Efekty sú zase dopady WS na celkový model po jej vykonaní.



Obr. 4 Prostredie pre kompozíciu WS [17]

Na Obr. 4 je znázornená jednoduchá architektúra, ktorá približuje proces kompozície WS. V nasledujúcich odsekoch sú popísané jej jednotlivé prvky.

- **Úložisko WS** (angl. Service repository) – slúži na uloženie atomických WS od poskytovateľov WS (angl. Service Provider).
- **Prekladač** (angl. Translator) je komponent, ktorý slúži na spracovanie informácií získaných od používateľa a z úložiska WS. V mnohých systémoch popisujúcich kompozíciu WS je špecifikácia jazyka, ktorým používateľ zadá svoju požiadavku a špecifikácia jazyka, ktorý je používaný samotným programom na kompozíciu, rozdielna. Napr. používateľ môže používať spomínané štandardy ako WSDL, OWL a OWL-S, zatiaľ čo samotný program môže pracovať napr. na princípe logického programovania (kapitola 4.2.4) a používať jazyk ako napr. PROLOG.
- **Tvorba procesov** (angl. Process Generator) – hlavný komponent celého systému. Je to vlastne aplikácia algoritmov, ktoré na základe používateľovej požiadavky (často označované aj ako dopyt) vyberú atomické WS, ktoré ju splnia. Na konci procesu je poskytnutá množina atomických WS s pracovným a dátovým tokom medzi týmito službami.
- **Evaluácia** (angl. Evaluator) nastáva v prípade, že je vygenerovaných viacero plánov. Príčinou existencie viacerých plánov pre jeden problém môže byť existencia viacerých ekvivaletných služieb s podobnou funkcionalitou, ako aj viacero spôsobou ako tieto služby komponovať pre dosiahnutie požadovaného cieľa. Preto v tomto prípade, ak je navrhnutých viacero plánov, rozhoduje evaluátor na základe nezávislých hodnôt (napr. používateľových podmienok), ktorý plán sa zvolí.
- **Exekúcia** (angl. Execution engine, t.j. vykonanie plánu) zvolených webových služieb v poradí určenom plánovačom. Výsledok je poskytnutý používateľovi.

3.1 Potenciálne domény a scenáre využitia webových služieb a ich kompozície

V tejto časti sú stručne popísané niektoré vybrané domény, v ktorých sa ukazujú dobré možnosti využitia webových služieb a ich kompozície.

- **Nákup** – množina webových služieb bude poskytovať možnosti prezerania on-line katalógov predajcov a nákup tovarov z týchto katalógov. Možným cieľom kompozície WS by bolo nájdenie vhodného produktu a vybavenie kúpy tohto produktu na základe požiadaviek používateľa.
- **Správa dokumentov** – služby budú slúžiť na manipuláciu s dokumentmi, konvertovanie, kompresiu a kódovanie určitých dokumentov. Možné ciele predstavujú napr. transformáciu dokumentov na základe používateľovej požiadavky, t.j. napr. konverzia a kompresia určitej množiny dokumentov, pričom táto množina môže byť heterogénna (jednotlivé dokumenty sa môžu od seba líšiť, napr. formátom súborov).
- **Odpovedanie na emaily** – typické ciele pre túto doménu predstavujú odpovedanie na správy prostredníctvom emailov a adaptáciu daného systému, t.j. kontextovo a užívateľsky závislé zobrazenie, preklad textu, konverzia obrázkov a pod.
- **Cestovanie** [4]– množina webových služieb ohľadom cestovania, napr. WS pre výber trasy z miesta A do miesta B, WS zabezpečujúce letecké spojenie a s ňou súvisiace služby ako napr. rezervácia leteniek, ďalej napr. WS pre rezerváciu hotela a podobne. Pre rozsiahlosť danej domény je práve táto oblasť najčastejšie diskutovaná v súvislosti s kompozíciou WS.
- **Analýza textov** (angl. Text mining) [51]– množina WS pre prácu s textovými dokumentmi s cieľom ich predspracovania (napr. indexácia, odstraňovanie určitých slov) a následného dolovania zaujímavých znalostí (napr. klasifikácia, zhlukovanie, extrakcia informácií a pod.)
- **Preklad medzi rôznymi jazykmi** [36]– WS majú možnosť prekladať medzi rôznymi jazykmi. Automatická kompozícia by tu mohla byť využitá napr. v prípade, ak nie je možné priamo nájsť službu na preklad z daného jazyka do iného jazyka, ale existujú služby, ktorých kompozíciou by sme dosiahli požadovaný výsledok (napr. je potrebné preložiť slovo zo slovenčiny do

čínštiny, ale nie je k dispozícii služba, ktorá to je schopná vykonať. Existujú ale dve iné služby: prvá WS prekladá zo slovenčiny do angličtiny a druhá WS prekladá z angličtiny do čínštiny. Ich kompozíciou je možné dosiahnuť požadovaný výsledok)

- **Zdravotníctvo** [37]– napr. WS budú slúžiť na plánovanie a koordináciu potrebných vyšetrení, stretnutí s obvodným lekárom po vyšetreniach, zisťovanie dostupnosti liekov v jednotlivých lekárňach a podobne.

4 Kompozícia webových služieb pomocou metód plánovania

Kompozíciu webových služieb možno postaviť na viacerých metódach. Ako bolo uvedené napr. v [32][56], je možné použitie situačného kalkulu, Petriho sietí, ako aj použitie metód plánovania. Ako jedna z najvhodnejších volieb sa ukazuje byť práve použitie metód plánovania, známych z oblasti umelej inteligencie (UI). Plánovanie predstavuje riešenie plánovacieho problému hľadaním riešenia nad množinou dostupných plánov. Úlohou určiť riešenie vo forme vhodnej postupnosti akcií (plánu), ktorá vedie k dosiahnutiu stanovených cieľov. Plánovaciu úlohu možno reprezentovať ako určitý model sveta a je možné ju popísať ako päticu **<S,S_0,G,A,Γ>** [54]:

- **S** reprezentuje množinu všetkých možných stavov v danom modeli,
- **S_0** je podmnožinou **S** a označuje počiatočný stav,
- **G** označuje cieľový stav,
- **A** je množina dostupných akcií, z ktorých každá mení stav sveta jeho prechodom z jedného stavu do iného,
- relácia **Γ** je podmnožinou **S x A x S** a definuje predpoklady a efekty pre každú akciu.

Pri webových službách sa pracuje so sémantickými technológiami. Ako bolo uvedené v kapitole 2, medzi tieto môžu patriť napr. WSDL, OWL, OWL-S a podobne. Súvislosť plánovania a automatickej kompozície WS vzhľadom na použitie sémantických technológií môže byť nasledovná:

- **S_0** a **G** množiny reprezentujú počiatočný a koncový stav, ktoré môžu byť reprezentované sémanticky pomocou ontológií, napr. v OWL.
- **A** je množina akcií a je reprezentovaná množinou dostupných atomických WS.
- **Γ** nám reprezentuje funkcie zmeny stavu pre každú službu. Ako najvhodnejší jazyk pre popis WS sa na priame prepojenie s UI plánovaním javí OWL-S popis služieb. Pomocou OWL-S je možné okrem vstupov a výstupov priamo popísať aj predpoklady a efekty.

4.1 Plánovacia úloha

Plánovacia úloha vo všeobecnosti pozostáva z nasledovných častí [42]:

- **popis dostupných akcií**, ktoré môžu byť vykonané (v určitom formálnom jazyku - doménová teória, resp. plánovacia doména),
- **popis počiatočného stavu** sveta,
- **popis** požadovaného, **cieľového stavu** sveta.

4.1.1 Definovanie plánovacej domény

Cieľom definovania plánovacej domény je poskytnúť plánovaciu teóriu, čo predstavuje poskytnutie určitých plánovacích pravidiel a sémantiky pre popisované operácie. Plánovacie operácie (označované aj ako akcie) môžu reprezentovať fyzické operácie (napr. pri riadení robota manipuláciu s ramenom a pod.) alebo abstraktné operácie (napr. manipulácia so súbormi, alebo rezervácia hotela atď.)[37]. Obvykle doménová teória predstavuje stavovo priestorový model, ktorý slúži na popis stavu sveta. Cieľom je poskytnúť prostriedky na zaznamenania stavu sveta (alebo situácie) v danom časovom okamžiku. Ďalej je potrebné popísať operácie, ktoré môžu tento stav meniť a vytvárať iný stav.

Formalizmy, ktoré sa používajú na popísanie plánovacej domény v prípade plánovania metódami UI, väčšinou vychádzajú zo **STRIPS**-u [18](kap. 4.2.1). STRIPS bol prvý krát použitý v 70. rokoch na popis plánovacej domény pre robotický systém Shakey. STRIPS definuje operácie pomocou definovania troch základných zoznamov. Prvý zoznam **PRE** predstavuje podmienky, ktoré musia byť splnené pre vykonanie danej operácie. Ďalej je to zoznam **ADD**, v ktorom sú definované atómy, ktoré budú po vykonaní operácie pridané do aktuálneho stavu sveta a zoznam **DEL**, ktorý obsahuje atómy, ktoré budú odstránené z aktuálneho stavu sveta po vykonaní akcie.

Ďalšia snaha pre viac expresívny popis plánovacej úlohy bol jazyk **ADL** (Action description language). Umožňuje definovať závislosti medzi efektmi (zoznamy ADD), poskytuje negácie a disjunkcie atómov. Zoznamy v STRIPS-e boli reprezentované ako konjunkcie atómov.

Ďalší, v posledných rokoch najviac používaných jazyk pre popis plánovacích domén, je jazyk **PDDL** (Planning Domain Definition Language) (kapitola 4.3), ktorý vychádza z jazyka ADL a formalizmu STRIPS-u. Jeho vysoké používanie je dôsledkom

toho, že bol použitý na IPC[10] súťaži. Umožňuje definovanie plánovacích domén expresivitou ADL, pričom zahŕňa nové prvky, ako napr. metrické alebo časové premenné (PDDL 2.1).

4.1.2 Definovanie počiatočného stavu

V prípade klasického plánovania potrebujeme definovať počiatočný stav (stav, v ktorom sa systém nachádza) a cieľový stav (stav, v ktorom by sme chceli aby sa systém nachádzal). Cieľom nášho plánovania je poskytnúť takú postupnosť operácií, ktoré systém dovedú z počiatočného stavu do cieľového stavu.

Hlavné prvky, pomocou ktorých je plánovacej úlohe obyčajne reprezentovaný stav, sú atómy. Atómy predstavujú množinu výrokových predpisov, ktoré môžu nadobúdať pravdivostné hodnoty a takto definovať stavy sveta. Pri plánovaní sa obyčajne pracuje z uzavretým modelom sveta a predpokladá sa, že akýkoľvek fakt, ktorý nie je zahrnutý, je nepravdivý. Keď sa však plánuje v reálnom svete, je tento predpoklad veľmi ťažko splniteľný a stretávame sa s nasledovnými problémami:

- **nekompletné informácie** – nemáme všetky relevantné informácie ku plánovaciemu problému,
- **nesprávne informácie** – niektoré informácie plánovacieho problému môžu byť zavádzajúce,
- **nejasné informácie** – niektoré informácie môžeme mať dané s určitou pravdepodobnosťou ich výskytu.

V súvislosti s danými problémami sa začal používať termín lokálne uzatvoreného sveta. Znalosti ohľadom lokálne uzatvoreného sveta sú uložené vo dvoch databázach, a to **M** a **N**. Databáza **M** obsahuje známe fakty a databáza **N** obsahuje popisy týchto faktov a rozsah ich platnosti vzhľadom na daný plánovací problém.

4.1.3 Definovanie cieľového stavu

V prípade klasických prístupov ku AI plánovaniu sú ciele plánovania obvykle vyjadrené ako vlastnosti, ktoré musia byť splnené v požadovanom stave sveta. Toto vyjadrenie je reprezentované najčastejšie formou konjunkcie a/alebo disjunkcie literálov

[10] IPC – International Planning Competition, http://planning.cis.strath.ac.uk/competition/

(t.j. pozitívnych alebo negatívnych atómov). Cieľom plánovania je získať plán, ktorého vykonaním v počiatočnom stave sa dosiahne požadovaný cieľový stav.

Požiadavky v prípade automatickej kompozície webových služieb (ale aj pre iné domény založené na plánovaní) môžu byť napr. tieto [32].

- **Dočasné štruktúry** – v určitých metódach plánovania môže nastať problém ak sa cieľ, ktorý chceme dosiahnuť nachádza aj vo vnútri samotného plánu (napr. chceme si naplánovať cestu z bodu A do bodu B a naspäť do bodu A – nemôžeme zadefinovať bod A jednoducho ako cieľ, lebo by bol rovný počiatočnému stavu). Dočasné štruktúry potom slúžia na spresnenie definovania cieľa, popr. rozdelenie cieľa na niekoľko podcieľov.
- **Nedeterminizmus** – v prípade, že operácia nedosiahne požadovaný výsledok, musí byť systém schopný sa s takouto situáciou vysporiadať.
- **Zaručené vlastnosti niektorých objektov** – je treba zabezpečiť, že hodnoty niektorých objektov sa nemôžu počas plánovania meniť, popr. sa môžu meniť len v určitom intervale (napr. pri práci s bankovým kontom).
- **Rozlišovanie cieľov** – rozlišovanie akcií na tie, ktoré môžu alebo nemôžu ovplyvniť požadovaný výsledok (napr. akcia ktorá nám vráti farbu určitého objektu od akcie ktorá je schopná meniť farbu daného objektu). Akcie, ktoré nemôžu meniť stav sveta sa označujú ako snímacie akcie (čisto informačné).
- **Preferencie používateľa** – zohľadňovanie preferencií používateľa počas tvorby plánu (napr. preferovať cestovanie vlakom pred lietadlom a podobne).

4.2 Plánovanie metódami umelej inteligencie

V danej kapitole si veľmi stručne popíšeme techniky UI plánovania, ktoré sú vhodné pre kompozíciu WS. Následne kapitola 4.5 obsahuje popis konkrétnych implementácií systémov pre kompozíciu WS, pričom tieto sú vzťahované k metódam popísaným v tejto kapitole.

4.2.1 Stavovo – priestorové plánovanie

Stavový priestor [18] pozostáva z konečnej množiny stavov **S**, z konečnej množiny akcií **A**, z prechodovej funkcie **f**, ktorá popisuje, ako je možne prechádzať medzi jednotlivými stavmi, a z cenovej funkcie **c(a,s)>0**, ktorá určuje „cenu“ použitia akcie **a** v stave **s**. Stavový priestor, ktorý je zložený z počiatočného stavu $\mathbf{S_0}$ a množiny koncových cieľov $\mathbf{S_G}$, sa zvykne označovať **stavový model**.

Stavovo orientované plánovače sa zaoberajú riešením plánovacieho problému pomocou hľadania vhodných operátorov (akcií), pomocou ktorých je možné dosiahnuť želaný stav. V závislosti na smere prehľadávania stavového priestoru rozoznávame dva základné typy:

- **dopredné plánovanie** – štartuje v počiatočnom stave a smeruje smerom ku cieľovému stavu,
- **spätné plánovanie** – štartuje v cieľovom stave a plánuje smerom ku počiatočnému stavu stavového modelu.

V oboch prípadoch je cieľom nájsť postupnosť akcií, ktorej použitím v počiatočnom stave stavového modelu, postupnou aplikáciou akcií docielime stav cieľový. Formálny zápis riešenia je sekvencia akcií $\mathbf{\{a_0,a_1,\ldots,a_n\}}$, ktoré generujú postupnosť stavov $\mathbf{s_0, s_1=f(s_0, a_0),\ldots,s_{n+1}=f(s_n, a_n),}$ kde platí, že akciu $\mathbf{a_i}$ je možné použiť v stave $\mathbf{s_i, a_i \epsilon A(s_i),}$ kde $\mathbf{A(s_i)}$ je konečná množina dostupných akcií v stave $\mathbf{s_i}$ a stav $\mathbf{s_{n+1} \epsilon S_G}$ je cieľový stav.

V princípe môže byť použitý na stavovo orientované prehľadávanie ľubovoľný prehľadávací algoritmus. Stavový priestor zachytávajúci situácie z reálneho sveta však môže byť veľmi rozsiahly. Preto vo výbere prehľadávajúceho algoritmu treba dbať na jeho výkonnosť a hlavne nato, ako si vie poradiť s rozsiahlosťou prehľadávaného priestoru. Dobrá voľba je použitie algoritmov, ktoré sú schopné redukovať stavový priestor. Napr. v prípade, že je priestor zapísaný vo forme stromu, dané algoritmy sú schopné rozlišovať medzi vhodnými a menej nevhodnými (až zbytočnými) vetvami stromu. Vhodné vetvy sú tie ktoré vedú ku želanému cieľu, ktorý môže navyše vykazovať určitú kvalitu. Nevhodné vetvy sú tzv. slepé vetvy, ktoré nikdy nedosiahnu cieľový stav, alebo vetvy, ktoré síce dosiahnu cieľový stav, ale ten nemusí zaručovať požadovanú kvalitu.

Medzi prvé pokusy redukovať stavový prehľadávací priestor patril **STRIPS** (STanford Research Institute Problem Solver)[18]. STRIPS používa spätné reťazenie.

To znamená, že štartuje cieľovým stavom, pričom hľadá akciu ktorej vykonaním je možné dosiahnuť práve daný stav (v prvom kroku hľadania teda vyberá takú akciu, ktorá vedie k cieľovému stavu). V druhom kroku hľadá akciu (popr. akcie), ktorej vykonaním je možné dosiahnuť vstupné predpoklady akcie vybranej v predchádzajúcom kroku atď. Takto sa pokračuje až pokiaľ nie je dosiahnutý počiatočný stav modelu. STRIPS pracuje s nasledovnými prvkami.

- **počiatočný stav,**
- **cieľový stav,**
- **množina akcií,** kde každá akcia obsahuje:
  - **predpoklady** (angl. preconditions) – informujú o tom, čo musí byť splnené pre vykonanie danej akcie,
  - **závery** (označované aj ako **efekty**, angl. postconditions, resp. effects) – informujú o tom, čo sa v modeli zmení vykonaním danej akcie.

Matematicky možno STRIPS definovať ako množinu **<A,O,I,G>**, pre ktorú platí:

- **A** je množina podmienok, obyčajne reprezentovaných formou atómov.
- **O** je množina operátorov (t.j. akcií), z ktorých každý je daný štvoricou **<α,β,γ,δ>:**
  - **α** hovorí o tom, ktoré podmienky musia byť splnené pre vykonanie akcie,
  - **β** označuje podmienky, ktoré musia byť nepravdivé predtým ako sa môže vykonať daná akcia,
  - **γ** sú podmienky, ktoré budú pravdivé po vykonaní akcie,
  - **δ** sú zase tie, ktoré budú po vykonaní akcie nepravdivé.
- **I** je počiatočný stav modelu, ktorý je daný množinou podmienok, ktoré sú v tomto stave pravdivé a zároveň všetky ostatné sú považované za nepravdivé. Platí, že **I** podmnožinou **A**.
- **G** je cieľový stav modelu, ktorý je zadaný ako dvojica **<N, M>**, ktorá špecifikuje, ktoré podmienky sú pravdivé, a ktoré nepravdivé, v danom stave. Platí, že **G** je podmnožinou **A**.

Redukcia prehľadávacieho priestoru spočíva v tom, že v každom nasledovnom kroku sa akcie vyberajú na základe predpokladov kroku predchádzajúceho, pričom tieto predpoklady sú porovnávané s efektmi akcií z predchádzajúceho kroku (STRIPS pracuje na základe spätného reťazenia).

Ďalšia cesta smerujúca ku redukcii prehľadávacieho priestoru je použitie **heuristických funkcií**, ktoré budú odhadovať užitočnosti alternatívnych akcií. Plánovač sa na základe týchto hodnôt bude rozhodovať, ktoré vetvy budú užitočné a ktoré budú viesť do slepých uličiek, popr. budú viesť ku menej kvalitným riešeniam. Skutočne automatické doménovo nezávislé plánovače (napr. aj systémy popísané v kapitole 4.5) musia byť vybavené práve takýmito heuristickými funkciami, pre zabezpečenie rýchlosti a efektívnosti plánovania v rozsiahlych doménach.

Heuristiku vo svojom plánovaní používa napr. **plánovač HSP** (Heuristic Search Planner – Heuristicky prehľadávací plánovač) [19]. Je to plánovač založený na princípe STRIPS. Plánovací problém P v HSP reprezentuje množina **P1=<S, s_0, S_G, Ac, f, c>**, kde:

- stavy **s ϵ S** sú reprezentované množinami atómov z **A,**
- počiatočný stav **s_0** je **I** (I reprezentuje počiatočný stav plánovacej úlohy, s_0 počiatočný stav plánovania)**,**
- pre cieľové stavy **s ϵ S_G** platí, že **G** je podmnožinou **S,**
- akcie **a ϵ Ac(s)** sú operátory **op**, pričom **predpoklady op** sú podmnožinou **s**
- prechodová funkcia **f** mapuje stav **s** do stavu **s'**, pričom platí **s'=s-Del(a)+Add(a)** pre **a ϵ Ac** (Del a Add reprezentujú funkcie, ktoré odstraňujú, resp. pridávajú atómy a vytvárajú tak nový stav sveta),
- ceny akcií **c(a)** sú rovné 1.

V prípade predchádzajúceho prehľadávania bol stavový priestor v tzv. progresnom stave (tzn. **s_0 je I**). Regresný stavový (**s_0 je G**) priestor s tým istým plánovacím problémom je možné vzhľadom na STRIPS zadefinovať ako množinu **P2=<S,s_0,S_G,Ac,f,c>**:

- stavy **s ϵ S** sú reprezentované množinami atómov z **A** (A je množina atómov, viď. definíciu STRIPS-u),
- počiatočný stav **s_0** je **G** (plánovanie začína v cieľovom stave),

- pre cieľové stavy **s** ϵ $\mathbf{S_G}$ platí, že **s** je podmnožinou **I**,
- množinu akcií **Ac(s)** vykonáme v stave **s** v tom prípade, že pre operátory **op** platí: **Add(op)** (reprezentujúci atómy operátora, ktoré budú po jeho vykonaní pridané) prienik **s** je rôzne od nuly a **Del(op)** prienik **s** je rôzne od nuly,
- stav **s'=f(s,a)**, ktorý nastáva aplikáciou akcií **a = op** v stave *s*, pre **a ϵ A(s)**, je taký že platí **s'=s-Del(a)+Add(a)**, pre **a ϵ Ac**
- ceny akcií **c(a)** sú rovné 1

Spätný model má výhodu v tom, že heuristická funkcia sa počíta len raz, zatiaľ čo pri doprednom modeli sa musí vypočítať pre každý nový stav od začiatku. Toto je relatívne nevýhodné, avšak ukázalo sa, že v niektorých prípadoch to môže so sebou prinášať nové vhodné informácie.

Medzi stavovo orientované plánovače patrí aj **dopredný plánovač FF** (FF – Fast Forward Planner) [21]. Je založený na doprednom prehľadávaní stavového priestoru, ktoré je riadené heuristicky. Na rozdiel od predchádzajúceho plánovača HSP má viacero výhod:

- FF plánovač používa sofistikovanejšiu metódu získavania heuristických hodnôt, ktorá je podobná GraphPlanu (kap. 4.2.2).
- Používa novú metódu lokálnej stratégie prehľadávanie, ktorá umožňuje rýchlejšie opustiť tzv. roviny v prehľadávacom priestore a vyhnúť sa lokálnym maximám. Metóda sa označuje ako *vynútená gradientová metóda* (angl. enforced hill-climbing)
- Lepší odhad nasledovníkov daného uzla vzhľadom na celkový cieľ.

FF plánovač sa zúčastnil aj medzinárodnej súťaži v plánovaní ICP-2000[11], kde porazil plánovač HSP 2.0.

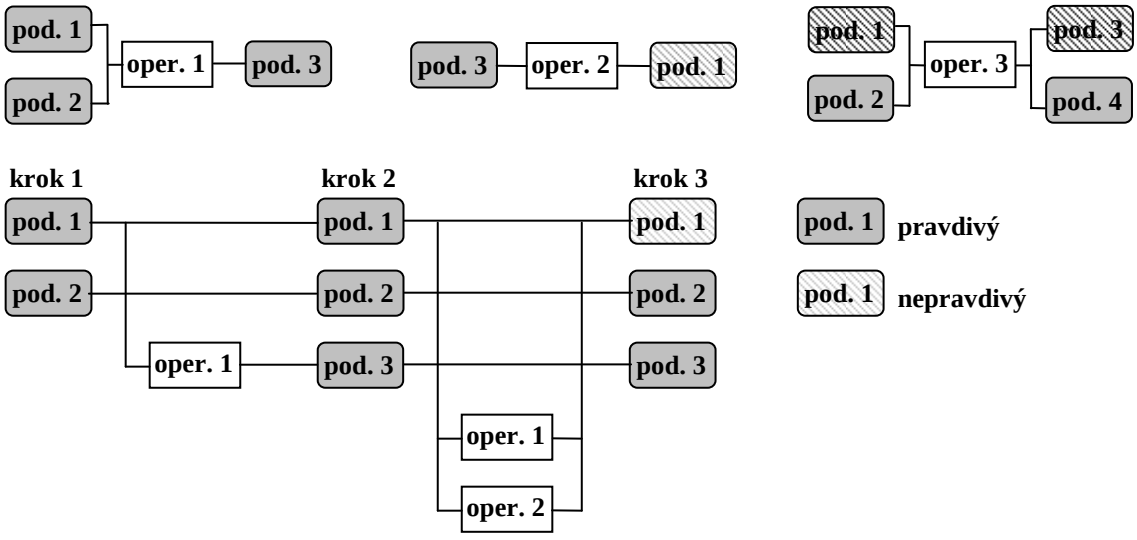
Vzťah medzi kompozíciou Webových služieb a STRIPS-om, od ktorého sa vyvíja väčšina stavovo orientovaných plánovačov je popísaný napr. v [22].

[11] AIPS-00 Planning Competition, The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems, Breckenridge, CO, April 15-19, 2000. Dostupné na: http://www.cs.toronto.edu/aips2000/

4.2.2 Grafovo orientované plánovanie

Grafovo orientované plánovania pre svoju činnosť používa **grafové štruktúry**. Tieto grafové štruktúry sa označujú ako **plánovacie grafy**. Plánovací graf sa odlišuje od stavovo priestorového grafu, ktorý reprezentuje stavy ako uzly grafu a hrany ako prechody medzi jednotlivými stavmi. Plánovací graf pozostáva z dvoch typov uzlov, a to **uzly akcií** a **podmienkové uzly** (označované aj ako uzly výrokové). Tieto uzly sú rozmiestnené v striedajúcich sa vrstvách. Vrstva výrokových uzlov je nasledovaná vrstvou uzlov akcií atď. Každému uzlu je priradený určitý časový okamžik. Prvá úroveň plánovacieho grafu je výroková a obsahuje **jeden uzol** pre každé tvrdenie z počiatočnej situácie. Druhá úroveň je úroveň akcií a obsahuje všetky akcie, ktorých predpoklady sú splnené na základe predchádzajúcej výrokovej vrstvy. Tretia vrstva je znova výroková a nachádzajú sa tu uzly z prvej vrstvy a uzly reprezentujúce efekty akcií z druhej vrstvy. Konštrukcia plánovacieho grafu pokračuje až pokiaľ nie sú dve po sebe idúce výrokové vrstvy totožné.

Medzi prvý plánovací systém používajúci plánovanie na základe grafov patrí **GraphPlan** (Avrim Blum a Merrick Furst, 1995) [31][32]. GraphPlan plánuje v stavovom priestore a na vstupe má vstup rovnaký ako STRIPS (kap. 4.2.1, str. 35). Pracuje so stavovým priestorom, kde jednotlivé uzly grafu predstavujú stavy a operácie a hrany predstavujú dosiahnuteľnosť jednotlivých stavov. Ako bolo uvedené vyššie, uzly grafu sú striedavo operácie a podmienky, tzn. že hrany vedú od podmienok ku operáciám a naopak.



Obr. 5 Rozvíjanie Grafu z GraphPlane [27]

GraphPlan využíva tzv. systém vzájomných vylúčení (angl. mutual exclusions („mutex“) relations), operácií, ktoré nemôžu byť vykonávané paralelne. Vzájomné vylúčenia musia byť vykonávané pre každý krok algoritmu zvlášť. Poznáme tri druhy vzájomného vylúčenia pre akcie [27][32]:

- nekonzistentné efekty: efekt jednej akcie neguje efekt druhej akcie,
- interferencia: efekt jednej akcie je negácia predpokladu druhej akcie,
- nekonzistentné predpoklady: predpoklad jednej akcie sa vzájomne vylučuje s predpokladom druhej akcie.

Najčastejšie použitou metódou pre vyhľadávanie plánu v grafe je metóda spätného reťazenia. Používajú sa parciálne grafy, čiže jednotlivé kroky potenciálnych riešení, kde v prvom kroku sú ako cieľ zvolené cieľové stavy systému. Pre každý stav sú nájdené všetky operácie, ktorými sa dá daný stav dosiahnuť (graf nám vylúči tie operácie, ktoré síce ku cieľovému stavu vedú, ale môžu byť použité až v ďalších krokoch). S pomocou tabuľky vzájomných vylúčení zistíme, koľkými rôznymi spôsobmi môže systém získať cieľové stavy z predchádzajúceho kroku. Každý spôsob bude označený ako jeden krok potenciálneho plánu. Pokiaľ pre daný stav neexistuje riešenie, môžeme z potenciálneho plánu odstrániť všetky kroky, ktoré vedú ku danému cieľu a všetky kroky pred ním, ktoré prídu takto o všetkých nasledovníkov. Algoritmus GraphPlanu je popísaný napr. v [27].

S GraphPlanom je často porovnávaný systém *SATPlan* (SAT – propositional satisfability) [32][27]. Na rozdiel od GraphPlanu funguje na úplne inom princípe. SATPlan konvertuje plánovací problém do tvaru, ktorý je riešiteľný výrokovou logikou. Výroková logika sa skladá z množiny premenných a základných operátorov AND ($\wedge$, konjunkcia, logicky súčin), OR (v, disjunkcia, logicky súčet) a NOT (~, negácia). Jednotlivé výrazy sa skladajú z literálov (čo sú premenné alebo ich negácie) a základných operátorov. Zvyčajne sa ešte používajú operátory implikácie ($\Rightarrow$) a ekvivalencie (<=>) .

4.2.3 Plánovanie využitím hierarchických sietí

Hierarchické siete úloh (Hierarchical Task Network – HTN) [29][42][53] predstavujú ďalší prístup ku automatickému plánovaniu. Vzťahy medzi jednotlivými

akciami plánovacieho problému sú vyjadrené pomocou sietí. Plánovací problém je špecifikovaný v hierarchickej sieti úloh pomocou nasledovnej množiny povolených úloh:

- **primitívne úlohy**, ktoré odpovedajú akciám v STRIPS-e (kap. 4.2.1),
- **zložené úlohy**, ktoré môžu byť zložené z viacerých jednoduchších úloh (čiže z primitívnych alebo aj ďalších zložených úloh),
- **cieľové úlohy**, ktoré odpovedajú cieľom v STRIPS-e.

Primitívne úlohy predstavujú akcie, ktoré môžu byť priamo vykonané. Zložené úlohy predstavujú sekvenciu akcií a cieľové úlohy reprezentujú podmienky. Zložené úlohy pre svoje vykonanie potrebujú poznať množinu primitívnych akcií, z ktorých sú zložené. Cieľové úlohy reprezentujú podmienky, ktoré musia nadobúdať v cieľovom stave pravdivostnú hodnotu pravda. HTN plánovanie poskytuje hierarchickú abstrakciu a výkonnú stratégiu pre prácu s komplexnými a komplikovanými doménami pochádzajúcimi väčšinou z „reálneho sveta“. Podobne ako pri iných plánovacích technikách, aj v HTN sa predpokladá existencia operátorov, ktoré produkujú určité efekty, a to len v prípade ak sú ich predpoklady splnené pred ich vykonaním.

Plánovanie pomocou HTN bolo prvý krát prezentované v ABSTRIPS plánovacom systéme [32], neskôr použité v plánovači NOAH. Sémanticky bolo popísané napr. v [29] a [34].

Medzi najznámejší variant HTN plánovania patrí plánovanie pomocou **dekompozície usporiadaných úloh**. Plánovače založené na tomto prístupe, ako napr. **SHOP** (Simple hierarchical ordered planner) [34], akceptujú ciele ako zoznam úloh, kde zložené úlohy môžu pozostávať z ďalších zložených úloh alebo z primitívnych úloh. Systém dekompozície usporiadaných úloh neplánuje priamo dosiahnutie definovaného cieľa, ale plánuje vykonanie (komplexnej alebo primitívnej) akcie.

HTN plánovací systém dekomponuje plánovanú úlohu do množiny podúloh, a tieto podúlohy potom do ďalších podúloh, až pokiaľ množina úloh neobsahuje iba primitívne úlohy, ktoré je možné priamo vykonať pomocou volania atomických operácií. Počas každého cyklu dekompozície sa testuje či nie je porušená žiadna z daných podmienok. Plánovací problém je úspešne vyriešený, ak je cieľová komplexná úloha dekomponovaná do množiny primitívnych úloh, a to bez porušenia akejkoľvek podmienky.

Použitie HTN plánovania bolo ukázané napr. v [28][29] a v oblasti WS v SHOP2 systéme [37], ktoré patria do rodiny plánovačov založených na dekompozícii usporiadaných úloh. Prezentuje sa tu metóda transformácie OWL-S procesov do hierarchickej siete úloh.

4.2.4 Plánovanie pomocou logického programovania

Ďalším spôsobom reprezentácie plánovacieho problému a z toho vyplývajúcu možnosť riešenia kompozície webových služieb, predstavuje využitie **logického programovania**. Pri logickom programovaní uvažujeme o programe, ktorý možno reprezentovať ako množinu tzv. **Hornových klauzul** vo forme implikácie $\mathbf{A \leftarrow (B_1 \text{ a } B_2 \text{ a } ... \text{ a } B_n)}$. Každá takáto Hornova klauzula môže byť reprezentovaná ako disjunkcia literálov (v matematike literál označuje atomickú formulu, teda atóm, pričom pozitívny literál predstavuje atóm a negatívny literál predstavuje negáciu atómu), kde môže byť pozitívny najviac jeden literál, t.j. $A \vee \neg B_1 \vee \neg B_2 \vee ... \neg B_n$. Negácia je v logickom programe vyjadrená ako negácia na základe zlyhania (angl. *NAF – negation as failure*).

V prípade logického programovania sa ako vhodné metódy pre plánovanie ukazujú metódy založené na deduktívnom uvažovaní, ako je tomu napr. v prípade PROLOG-u. Viacero autorov sa zameralo však aj na použitie alternatívnych metód (napr. [41]). Autori ukázali, že plánovací problém môže byť konvertovaný do programu logického programovania v lineárnom čase, pričom pôvodný cieľ je dosiahnuteľný, iba ak súvisiaci cieľ G^* je dosiahnuteľný v niektorom stabilnom modeli logického programu získaného transformáciou. Takto to bolo napr. implementované v systéme SMODELS [23].

V [24] navrhli systém, ktorý pracuje s plánovacím problémom pomocou programu logického programovania a dodatočného grafového plánovania. Tento systém využíva výhody logického programovania kombinované s výhodami plánovacích systémov ako napr. GraphPlan (kap 4.2.2). Autori poukazujú na potreby tzv. „nemonotónneho“ uvažovania v systéme. Nemonotónne uvažovanie bolo motivované potrebou zachytiť do logického systému aspekty ľudského uvažovania, ktoré umožňujú odvolať predchádzajúci úsudok (napr. o určitej situácii, v ktorej sa nachádza) na základe nových informácií, ktoré sa dozvedel dodatočne. Systémy logického programovania pracujú s nemonotónnym uvažovaním pomocou negácie NAF. V ich najjednoduchšej forme, nemonotónne logické programy (tiež nazývané normálne logické programy) sú množiny

pravidiel vo forme **L ← A_1, A_2,..., A_n, notB_1, notB_2,..., notB_m**, kde **n,m ≥0** a **L, Ai** a **Bj** sú atómy. Atómy z prefixom **not** sú nazývané NAF literály, pričom platí, že **notB** má pravdivostnú hodnotu pravda, ak všetky dostupné cesty pre dokázanie B zlyhajú (negácia na základe zlyhania).

Medzi iné aplikácie logického programovania patrí napr. Reinerova implementácia **GOLOG**-u [25]. Autor sa zameriava na znalostne orientované GOLOG-ovské programy, ktoré môžu obsahovať snímacie akcie. Tieto programy sa odkazujú na znalosti agenta a sú navrhnuté pre online vykonávanie pod predpokladom dynamicky uzatvoreného sveta. Počas vykonávania sa automaticky upravujú základné axiómy na základe snímacích akcií.

Ako príklad pre použitie GOLOGu je možné spomenúť aj [35], kde používajú GOLOG v súvislosti so situačným kalkulom. GOLOG je logický programovací jazyk vyvinutý na Univerzite v Toronte pre špecifikáciu a exekúciu komplexných akcií v dynamických doménach. GOLOG je postavený na situačnom kalkule, pričom sú tu doplnené konštrukty na skladanie primitívnych akcií, definovaných v situačnom kalkule, do komplexných akcií, ktoré sa zvyknú označovať ako program S.

Koštrukty sú nasledovné:

- a – primitívne akcie
- S1;S2 – sekvencie akcií
- Φ? – testovanie podmienok
- δ1|δ2 – nedeterministický výber akcií
- (πx)δ(x) – nedeterministický výber argumentov
- δ* - nedeterministická iterácia
- If Φ then δ1 else δ2 endif – podmienky
- While Φ do δ endWhile – cykly

Dané konštrukty je možné použiť na tvorbu programu v jazyku pre doménovú teóriu, napr.:

```
buyAirTicket(x);
      if far then    rentCar(y) else
                     bookCar(y) endIf
```

V prepojení na kompozíciu WS sú dotaz používateľa a obmedzujúce podmienky prevedené a prezentované pomocou situačného kalkulu. Každá webová služba (primitívna aj komplexná) sa chápe ako akcia. Podobne aj predpoklady a efekty sú vyjadrené pomocou konštruktov situačného kalkulu. Potom zložené služby predstavujú množiny atomických služieb, ktoré sú pospájané pomocou prostriedkov procedurálneho programovania (podmienky If-then-else, cykly while a pod.).

Logické programovanie v súvislosti s webovými službami používa napr. nástroj SWORD prezentovaný v [26]. Plán je priamo extrahovaný z vykonania programu v prologovskom interpreteri.

4.3 PDDL – Problem Domain Definition Language

Jazyk PDDL vznikol ako snaha o štandardizáciu popisu plánovacích domén a problémov. Jazyk bol vyvíjaný Drewom McDermottom pre medzinárodnú súťaž v plánovaní ICP[12] (International Planning Competition), ktorá sa konala v roku 1998. PDDL sa ako popisný jazyk inšpiroval syntaxou jazyka LISP. Vychádza z formalizmu STRIPS-u [18] (Stanford Research Institute Problem Solver) a z jazyka ADL [61](Action Description Language). Ďalej sa jazyk inšpiroval formalizmami viacerých plánovacích systémov, ako napr. SIPE-2, UMCP, Unpop [43]. Plánovacia úloha je v jazyku PDDL rozdelená do dvoch súborov:

- definovanie plánovacej domény a
- definovanie plánovacieho problému vzhľadom na plánovaciu doménu.

Ako bolo spomenuté, jazyk PDDL sa dostal do povedomia komunity ľudí zaoberajúcich sa plánovaním pri príležitosti prvej medzinárodnej súťaži v plánovaní ICP. Konkrétne to bola verzia PDDL v 1.2, ktorá sa vyznačovala nasledovnými hlavnými črtami: akcie typu STRIPS, podmienené efekty, dynamické spracovanie objektov (t.j. tvorba a zánik objektov počas plánovacieho procesu), definície doménových axióm, definovanie typov objektov, definovanie rovnosti objektov, špecifikácie akcií zložených z jednoduchších akcií, definovanie tzv. fluent typov, ktoré sa môžu meniť v čase.

[12] 1st International planning competition - 1998: ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html

Potom ako bol jazyk PDDL pozitívne prijatý plánovacou komunitou, sa začal prirodzený vývoj jazyka. V roku 2002 Derek Long a Maria Foxová pri príležitosti konania tretieho ročníka ICP[13] uviedli rozšírenú verziu jazyka PDDL v. 2.1. Medzi hlavné nové črty jazyka patrili [45][46]:

- modifikácia typov „fluents" a pridanie syntaxe pre zápis číselných „fluents" typov.
- možnosť definovania tzv. duratívnych (trvajúcich) akcií. Tieto akcie môžu byť dvojakého typu, a to diskrétne a plynulé. Obidva typy sa viažu na logické zmeny, ktoré sú vyvolané aplikovaním danej akcie.

V roku 2004 S. EdelKamp a J. Hoffmann pre ICP-4[14] uvádzajú verziu 2.2 [47]. Daná verzia prináša dve hlavné novinky:

- možnosť definovania odvodených predikátov a
- definovania časovo inicializovateľných literálov.

Odvodené predikáty sú predikáty, ktoré ku svojej definícii používajú už existujúce predikáty. Časovo inicializovateľné literály sú literály, ktoré je možné inicializovať v určitom okamžiku behu programu. Obyčajne sa dané literály viažu na trvajúce (duratívne) akcie.

V roku 2005 pre ICP-5[15] uviedli PDDL v 3.0 [48]. V danej verzii zaviedli tzv. preferovanie cieľa a rôzne obmedzenia súvisiace s cieľmi. Ako príklad možno uviesť napr. prípad keď je potrebné zabezpečiť, aby určitý cieľ bol splnený počas celej doby plánovania. V danej verzii PDDL je to možné dosiahnuť pomocou obmedzení cieľa, konkrétne pomocou kľúčového slova **`always <GD>`** [49].

Posledná verzia jazyka PDDL je verzia 3.1 a bola uvedená pre ICP 2008.

V tejto práci je používaná základná verzia PDDL v 1.2. V prípade použitia rozšírených funkcií z vyšších verzií to bude vždy dopredu uvedené. Pre formálny popis štruktúry plánovacej domény a problému je používaná EBNF forma (Extended Bactus – Naur Form) [44].

Pred samotnou definíciou EBNF formy štruktúry PDDL domény sú najprv uvedené jednoduché príklady.

[13] 3rd International planning competition - 2002: http://planning.cis.strath.ac.uk/competition/
[14] 4th International planning competition - 2004: http://www.tzi.de/~edelkamp/ipc-4/
[15] 5 th International planning competition - 2005: http://zeus.ing.unibs.it/ipc-5/

```
(define (domain cestovanie)
  (:requirements :strips)
  (:predicates (cesta ?A ?B)
               (na ?Osoba ?Miesto)
               (osoba ?O))

  (:action Chod
    :parameters (?Osoba ?Z ?Do)
    :precondition
      (and (cesta ?Z ?Do)
          (na ?Osoba ?Z)
          (osoba ?Osoba))
    :effect
      (and (na ?Osoba ?Do)
        (not (na ?Osoba ?Z))))
 )
```

Pr. 2 PDDL doména cestovanie

```
(define (problem prob1)
  (:domain cestovanie)
  (:requirements :strips )
  (:objects a b c d palo)

  (:init (na palo a)
    (cesta a b)
    (cesta b a)
    (cesta b c)
    (cesta c b)
    (cesta c d)
    (cesta d c)
    (osoba palo))

  (:goal (and (na palo d)))
)
```

Pr. 3 PDDL problém cestovanie

Na Pr. 2 je znázornená plánovacia doména cestovanie popísaná v jazyku PDDL. V danej doméne sa uvažuje o osobách, ktoré sa môžu presúvať z bodu A do bodu B po ceste. V doméne sú tri predikáty. Predikát **`cesta`**, ktorý má dva parametre **`?Z`** a **`?Do`**, slúži na definovanie cesty vychádzajúcej z miesta A do miesta B. Predikát **`na`** slúži na definovanie informácie, že určitá osoba **`?Osoba`** na nachádza na danom mieste **`?Miesto`**. Predikát **`osoba`** slúži na poskytnutie informácie, kto je osoba v našej doméne. Ďalej máme v doméne definovanú jednu akciu **`Chod`**. Daná akcia má tri parametre **`?Osoba`**, **`?Z`** a **`?Do`** a slúži na premiestnenie osoby **`?Osoba`** z miesta **`?Z`** do miesta **`?Do`**.

Na Pr. 3 je znázornený plánovací problém vzhľadom na plánovaciu doménu definovanú vyššie. Daný plánovací problém má definovaných päť objektov **`a`**, **`b`**, **`c`**, **`d`** a **`palo`**. Z počiatočného stavu vidíme, že objekt **`palo`** je osoba a že sa na začiatku nachádza na mieste **`a`**. Našim cieľom je dostať objekt **`palo`** (osobu Paľo) na miesto **`d`**.

V prípade ak riešenie existuje, výsledkom bude pracovný a dátový tok. Konkrétne riešenie plánovacej úlohy definovanej plánovacou doménou z Pr. 2 a plánovacím problémom Pr. 3 je zobrazené na Pr. 4 nižšie.

```
1: (Chod palo a b)
2: (Chod palo b c)
3: (Chod palo c d)
```

Pr. 4 Výsledok PDDL plánovacej úlohy.

Po použití nasledovného plánu z Pr. 4(trikrát akcia **chod**) dosiahneme požadovaný cieľový stav. Rozsiahlejší príklad plánovacej úlohy je uvedený v prílohe A, príklad 5.

Pre ďalšiu prácu budeme potrebovať syntakticky popísať PDDL doménu a riešený problém. Syntaktický popis (podobný tomu v [43]):

- každé pravidlo má formu **<syntaktický element> ::= expanzia elementu**
- mená syntaktických elementov sú zapisované v ostrých zátvorkách **<>**
- voliteľné časti popisu sa nachádzajú v hranatých zátvorkách **[]**
- ak má nejaká časť syntaktického popisu horný index, napr. **[<types>]**$^{:typing}$, znamená to, že táto môže byť zahrnutá v popise len v prípade, ak má doména definovanú danú požiadavku (konkrétny príklad požaduje mať definovanú požiadavku **:typing**, čo nám potom umožňuje definovanie typov).
- podobne aj symbol **::=** môže byť indexovaný. V tomto prípade expanzia daného elementu môže nastať len v prípade, ak daná požiadavka je definovaná či už v popise domény, alebo v popise problému.
- znak ***** určuje, že daný prvok, element, sa na danom mieste môže nachádzať 0 a viac krát. Znak **+** určuje, že daný prvok sa tam môže nachádzať jeden a viac krát.

EBNF forma pre **definovanie štruktúry PDDL domény** je podľa špecifikácie jazyka nasledovná:

```
<domain> ::= (define (domain <name>)
                [<extension-def>]
                [<require-def>]
                [<types-def>]:typing
                [<constants-def>]
                [<domain-vars-def>]
```

[timless-def>]
[<safety-def>]:safety-constraints
<structure-def>*)

Z definície je zrejmé, že jediný povinný element je názov domény (**domain <name>**). Takáto definícia domény by však bola viac menej zbytočná. Z pohľadu tejto práce sú podstatné tieto časti definície domény:

- **[<require-def>]** – definovanie požiadaviek domény. Každá doména má minimálne jednu a tou je STRIPS (napr. Pr. 2). Medzi ostatné patria napr. **:typing**, v prípade ak je potrebné definovať typy v doméne, alebo **:equality**, v prípade ak je potrebná možnosť definovania rovnosti parametrov.
- **[<types-def>]**:typing – umožňuje definovanie typov
- <**structure-def> -** reprezentuje štruktúry (akcie, axiómy, metódy) a môže sa v definícii domény nachádzať viackrát.
- V prípade PDDL akcií ich môžeme rozdeliť do dvoch častí:
- **primitívne akcie** – najjednoduchšie akcie bez ďalšej vnútornej štruktúry.
- **zložené akcie** – môžu byť ďalej vnútorne delené na menšie celky pomocou riadiacich štruktúry. Pre definovanie takýchto akcií musíme mať v doméne definovanú požiadavku **:action-expansion**.

EBNF forma pre **definovanie štruktúry PDDL problému** je nasledovná:

<problem> ::= (define (problem <name>)
(:domain <name>)
[<require-def>]
[<situation>]
[<object declaration>]
[<init>]
<goal>+
[<length-spec>])

Z danej definície je zrejmé, že popri definovaní mena problému a mena domény, ktorej daný problém prislúcha, definícia problému obsahuje minimálne jeden cieľ. Z pohľadu tejto práce sú podstatné hlavne tieto časti:

- **[<object declaration>]** – deklarácie objektov

- **[<init>]** – počiatočný stav využíva predikáty definované v doméne na popis počiatočného stavu pomocou objektov deklarovaných v plánovacom probléme.
- **<goal>+** - cieľový stav plánovacej úlohy.

Podrobnejší popis jednotlivých častí štruktúr PDDL domény a problému je popísaný v kapitole 6, zaoberajúcou sa mapovaním OWL ontológií a OWL-S ontologických popisov služieb na PDDL plánovaciu úlohu.

4.4 Požiadavky kompozície webových služieb vs. plánovanie metódami umelej inteligencie

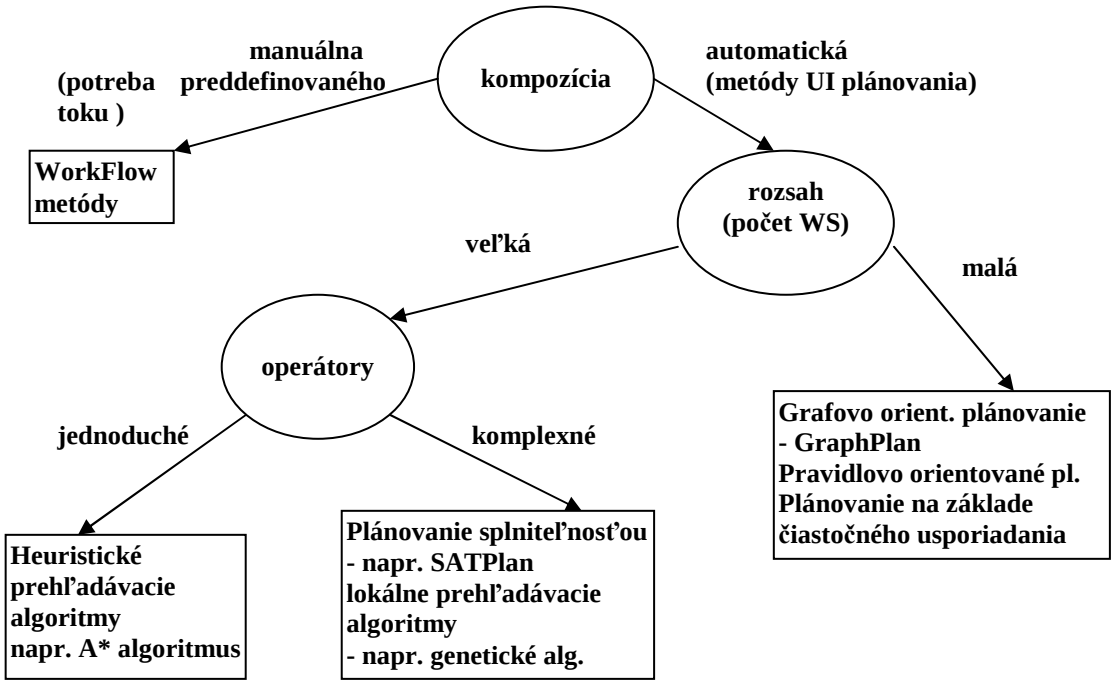
V nasledujúcej Tab. 1 [32] je výsledok analýzy prezentujúci vhodnosť plánovačov a popisovaných metód z pohľadu požiadaviek na kompozíciu WS.

Plánovač	**Plánovacia technika**	**Doménová komplexita**	**Snímacie akcie**	**Nedetermin. akcie**	**Rozšíriteľnosť cieľov**
FF	Stavovo orientované plánovanie	PDDL 2.1	Nie	Nie	Nie
HSP 2.0	Stavovo orientovaný	PDDL/ADL	Nie	Nie	Nie
SGP	GraphPlan	PDDL/ADL	Podpora snímacích akcií	Nie	Nie
SHOP2	Založený na HTN	PDDL/ADL	Áno	Áno	Áno, pomocou HTN metód
ConGolog	Založený na vykonávaní GOLOG-ovského programu	Situačný kalkulus	Áno	Áno	Áno
XPlan	Založený na HTN a grafovo orientovanom plánovanie	PDDL 2.2	Áno	Áno	Áno

Tab. 1 Požiadavky kompozície WS vs. Podpora vybraných plánovacích systémov

Na Obr. 6 máme znázornený rozhodovací strom, ktorý môže byť nápomocný pri voľbe správneho prístupu ku kompozícii webových služieb na základe troch rozhodnutí:

- **typ kompozície**
  - V prípade manuálnej kompozície zvolíme workflow metódy, kde je kompozícia vytváraná manuálne, pričom sa definuje poradie služieb a dátový tok medzi službami. Tieto techniky je možné aplikovať len pri malom počte služieb.
  - V prípade práce s väčším počtom služieb a potreby automatickej kompozície volíme metódy UI plánovania.
- **veľkosť plánovacej domény**
  - **veľký počet operátorov**
    - **Jednoduché operátory** – použitie heuristických prehľadávacích algoritmov
    - **Zložité operátory** – použitie plánovania splniteľnosťou (satisfiability)
  - **malý počet operátorov** – použitie grafovo orientovaných plánovacích techník, pravidlovo orientovaného plánovania, alebo plánovania na základe čiastočného usporiadania.



Obr. 6 Rozhodovací strom UI riešení pre kompozíciu WS [40]

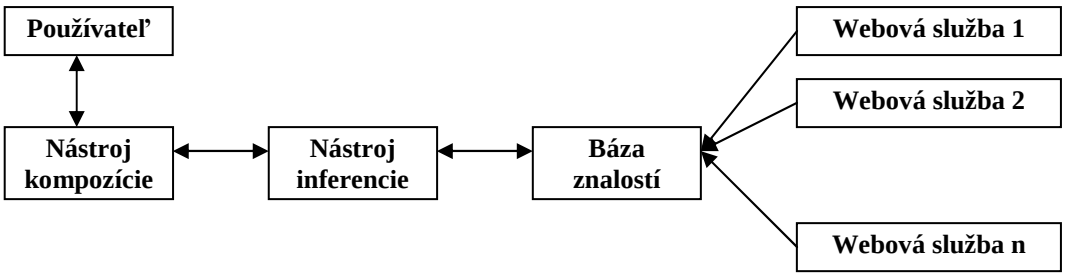
4.5 Implementácie systémov pre kompozíciu webových služieb

4.5.1 (Semi)automatická kompozícia webových služieb využívajúca PROLOG

Autori v [36] navrhujú (semiautomatickú) kompozíciu WS využívajúcu prvky logického programovania, konkrétne jazyka PROLOG. Prototyp ich systému kompozície WS má dve hlavné komponenty (Obr. 7):

- nástroj kompozície (NK) a
- nástroj inferencie (NI).

Nástroj inferencie uchováva informácie o známych službách v báze znalostí a v prípade potreby je schopný nájsť vhodné služby. Nástroj kompozície zabezpečuje okrem kompozície aj komunikáciu medzi ľudským operátorom a nástrojom inferencie a kompozície.



Obr. 7 Architektúra systému pre kompozíciu WS založenom na logickom programovaní

Nástroj inferencie slúži na OWL odvodzovanie a je postavený na PROLOGU. Informácie popísané v OWL sú konvertované do RDF trojíc (tripletov) a načítané do bázy znalostí (BZ). NI má vstavané odvodzovacie pravidlá pre OWL. Tieto pravidlá sú aplikované na fakty v BZ za účelom nájdenia všetkých závislostí, takými je napr. objavenie dedičnosti medzi dvoma triedami, ktorá nie je priamo zakódovaná vo vzťahoch podtried. NK podporuje používateľa pri tvorbe pracovných tokov (workflows) ponúkaním dostupných služieb v každom kroku. Používateľ začína proces kompozície zvolením jednej zaregistrovanej služby. Požiadavka je odoslaná do BZ s cieľom získania informácií o vstupoch do služby a pre každý vstup sa produkuje nová požiadavka s cieľom získania zoznamu služieb, ktoré vedia zabezpečiť dané vstupy. NK tiež ukáže odlišné triedy služieb dostupné v systéme a odfiltruje výsledok na základe obmedzení, ktoré používateľ špecifikoval na základe atribútov služieb.

- **Matching funkcionálnych vlastností** - NK poskytuje ako možnosti pre kompozíciu iba tie služby, ktorých výstupy by mohli byť vhodné ako vstupy zvolenej služby. Matching (párovanie dvoch služieb na základe podobnosti) dvoch služieb sa uskutočňuje na základe informácií z ich profilov. Každý profil popisuje jeho vstupy, výstupy a rozsah jeho parametrov.
- **Filtrovanie pomocou nefunkcionálnych atribútov** - Počet zobrazených služieb v zozname možných zhôd (matches) môže byť v niektorých prípadoch veľmi veľký. Ak názov služby nepomôže vyznačiť vhodnú službu, je treba použiť nefunkcionálne atribúty, ako napr. umiestnenie služby. Teda je potreba doplnkových (snímacích) akcií, ktoré nám budú poskytovať napr. informácie o lokalizácii webovej služby, typy webových služieb, čas umiestnenia na sieť atď.
- **Generovanie zložených služieb** - Každá kompozícia generovaná používateľom môže byť sama o sebe realizovaná ako OWL-S zložený proces. To znamená, že môže byť publikovaná, objavovaná a komponovaná s ďalšími službami. V NK generujeme presne taký popis *Zloženého Procesu* a tiež vytvárame odpovedajúce *Profily Služieb* s pridanými nefunkcionálnymi vlastnosťami používateľa. Tento popis je okamžite dostupný systému ako menovaná služba, ktorá môže byť filtrovaná a skladaná normálnou cestou. Touto cestou môže používateľ vytvárať komplexnú kompozíciu.
- **Vykonanie zložených služieb** - Aktuálna implementácia systému vykonáva výsledok kompozície pomocou volania každej individuálnej služby. Tento spôsob je primárne daný pomocou OWL-S a WSDL špecifikácie. Klientsky program slúži ako prvok, ktorá má na starosti všetky volanie jednotlivých služieb.

Workflow je zápísaný pomocou XML popisu a môže byť vykonávaný výmenou SOAP správ medzi službami. Použitím štandardov je zabezpečená systémová adaptovateľnosť a možnosť rozširovať rozhrania systému.

4.5.2 Kompozícia webových služieb plánovača SHOP2

SHOP2 je doménovo nezávislý HTN plánovací systém, ktorý vyhral jednu zo štyroch hlavných cien spomedzi 14 plánovačov na medzinárodnej súťaži v plánovaní v roku 2002 (IPC-2002). HTN je metóda UI plánovania, ktorá je zameraná na tvorbu plánu pomocou dekompozície úloh. Dekompozícia úloh je uskutočňovaná až pokiaľ nie

sú všetky úlohy rozložené na primitívne úlohy. Definícia plánovacieho problému v SHOP2 je nasledovná:

Plánovací problém podľa [30][50] pre SHOP2 je trojica **(S,T,D)**, kde **S** je počiatočný stav, **T** je množina úloh a **D** je popis domény. Dodaním danej trojice (S,T,D) ako vstup bude vrátený plán **P = (p1p2...pn)**, čo je množina operátorov, pomocou ktorých je možné dosiahnuť **T** z **S** v doméne **D**.

Operátor je výraz **(h(v) Pre Del Add)**, kde

- **h(v)** reprezentuje primitívnu úlohu so zoznamom vstupných parametrov *v*,
- **Pre** reprezentuje množinu podmienok (predpoklady), ktoré musia byť splnené, aby mohol byť daný operátor aplikovaný,
- **Del** reprezentuje zoznam atómov, ktoré nadobudnú hodnotu **false** po vykonaní daného operátora,
- **Add** reprezentuje zoznam atómov, ktoré nadobudnú hodnotu **true**.

Metóda je výraz **(h(v) Pre T)**, kde:

- **h(v)** reprezentuje zloženú úlohu so zoznamom vstupných parametrov **v**.
- **Pre** reprezentuje predpoklady (preconditions).
- **T** reprezentuje čiastočne usporiadaný zoznam podúloh, ktorý by vznikol dekompozíciou **h(v)**.

Koncept dekompozície úloh v HTN je možné prirovnať ku konceptu dekompozície procesu v OWL-S. A preto je HTN plánovanie ďalším vhodným kandidátom na automatickú kompozíciu WS.

Príklad [30]: Transformácia atomického procesu (Q)

Vstup: OWL-S popis Q atomického procesu A

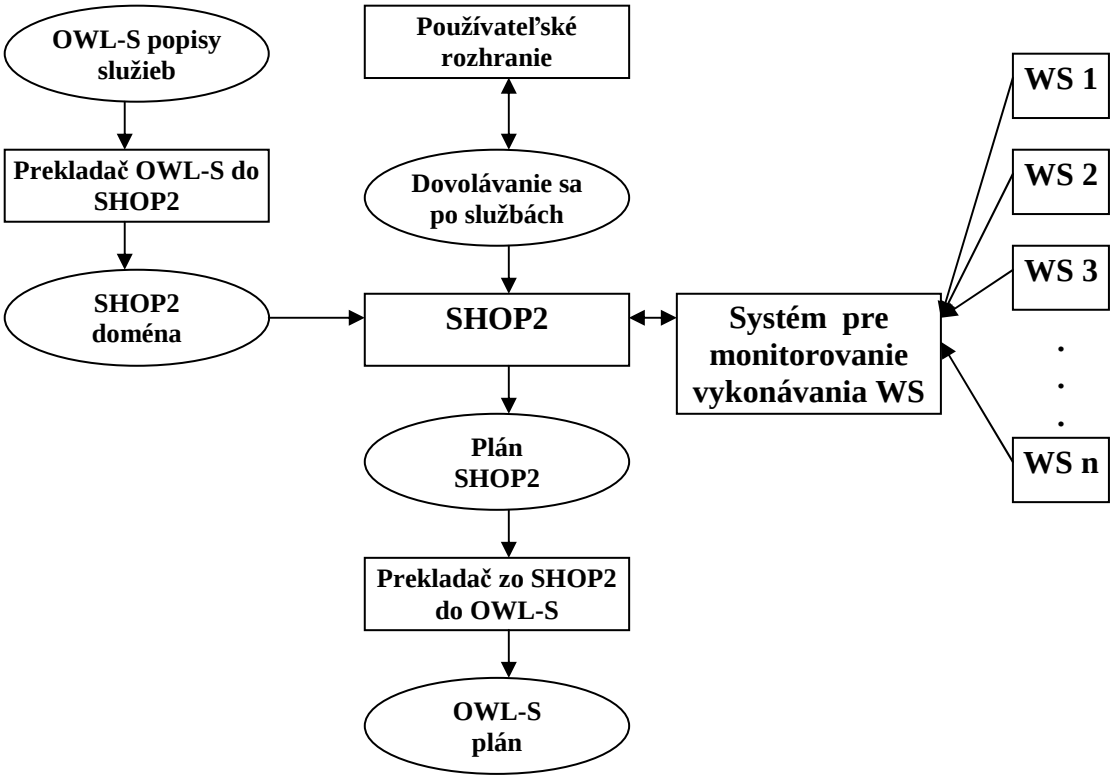
Výstup: SHOP2 operátor

Procedúra:

1. v – zoznam *vstupných* parametrov pre proces A v Q
2. Pre – konjunkcia všetkých *predpokladov* pre A v Q
3. Add – zoznam *pozitívnych efektov* z A
4. Del – zoznam *negatívnych efektov* z A

Return O = (A(v) Pre Add Del)

Na danom príklade je veľmi dobre viditeľné, ako jednoducho je možné prispôsobiť atomický proces popísaný pomocou OWL-S do HTN procedúry. Pre viac príkladov viď. [37].



Obr. 8 Architektúra systému pre kompozíciu WS využívajúceho SHOP2 plánovač

Na Obr. 8 je znázornené architektúra systému pre kompozíciu WS použitím plánovača SHOP2. Systém sa skladá z nasledovných hlavných častí.

- **Plánovač** SHOP2 – na základe počiatočného, koncového stavu a dostupných akcií tvorí plán, popísaný v plánovacej doméne SHOP2.
- **Nástroj pre správu WS** – má za úlohu uchovávať informácie o dostupnosti WS a vykonávanie plánu.
- Nástroj pre **reprezentáciu výsledného plánu** *v OWL-S* ontológii – poskytuje reprezentácie plánu v štandarde OWL-S.
- Nástroj pre **preklad OWL-S popisu služieb do SHOP2** plánovacej domény – prekladá nám popisy služieb OWL-S do SHOP2 domény (napr. atomické procesy z OWL-S do procedúr HTN plánovania).

Treba pripomenúť, že za oboma zatiaľ spomínanými systémami stojí skupina ľudí okolo prof. Jamesa Hendlera[16].

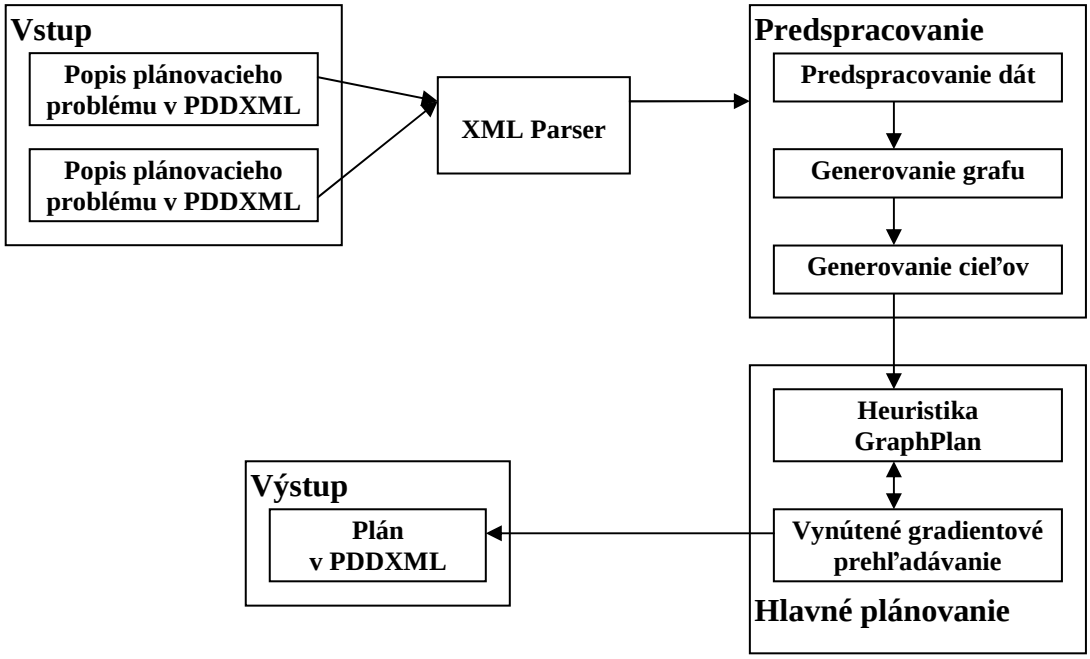
4.5.3 OWL-SXPlan

OWLS-XPlan [38][39] je nástroj vyvinutý na realizáciu kompozície sémantických webových služieb pomocou UI plánovania. Realizuje konvertovanie webových služieb popísaných pomocou OWL-S 1.1 štandardu do ekvivalentného problému realizovaného v jazyku PDDL 2.1. Po tomto prepísaní problému do plánovacej domény sa realizuje samotné plánovanie pomocou UI plánovača Xplan (Obr. 9). Výstupom z daného plánovača je sekvencia služieb (akcií) označovaná aj ako plán. Xplan je založený na rozšírení rýchleho dopredného plánovača (Fast-Forward planner) pomocou HTN plánovania.

HTN (Hierarchical task network) plánovače (napr. SHOP2) dosahujú pomerne kvalitné výsledky v doménach, pri ktorých sú dostupné komplexné znalosti a sú k dispozícii čiastočne štruktúrované vykonávacie vzory (napr. scenár plánovania záchrannej akcie). V doménach kde toto neplatí, t.j. nie je poskytnutý žiadny konkrétny súbor metód a pravidiel, HTN plánovač nedokáže nájsť riešenie kvôli pevne danej štruktúre hierarchických dekompozičných akcií uložených v jeho databáze. Toto samozrejme limituje použiteľnosť HTN plánovača.

Xplan [39] kombinuje výhody založené na FastForward plánovači s HTN plánovaním. Pre použitie XPlanu na kompozíciu webových služieb bol XPlan doplnený nástrojom OWLS2PDDL na preklad OWLS 1.1 popisov služieb do odpovedajúceho popisu pomocou PDDL 2.1, ktoré sú potom používané ako vstup do plánovača. Na rozdiel od HTN plánovačov, XPlan stále nájde riešenie nad priestorom možných plánov, samozrejme ak také existuje. XPlan v sebe implementuje možnosť dynamického zasahovania do plánovania. XPlan a OWLS2PDDL predstavujú spolu plánovač *OWLS-XPlan*.

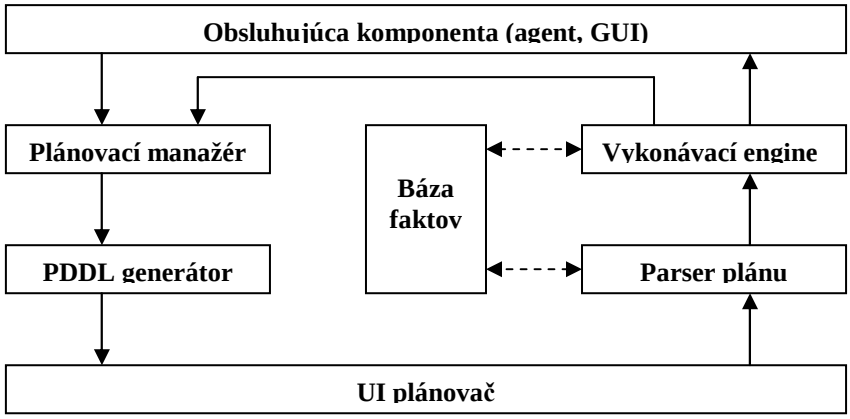
[16] James Hendler (narodený 2. apríla 1957) je výskumník v oblasti umelej inteligencie na Polytechnickom inštitúte v Rensselaer. Je jedným z tvorcov Sémantického webu.



Obr. 9 XPlan plánovač

4.5.4 WSPlan

V [33] sa autori zaoberali kompozíciou webových služieb len s využitím WSDL popisu služieb. Navrhli systém WSPlan, ktorého architektúru je možné vidieť na obrázku Obr. 10. Cieľom systému WSPlan je zabezpečiť kompozíciu pomocou UI plánovača. Úloha kompozície webových služieb sa začína obsluhujúcou časťou, ktorá v sebe zahŕňa grafické používateľské rozhranie. Pomocou tejto komponenty je zavolaná časť **plánovací manažér**, ktorá má na starosti spustenie procesu kompozície pomocou tvorby množiny relevantných webových služieb k danému problému kompozície. Danú množinu relevantných webových služieb autori preddefinovali manuálne, aj keď poukázali na vhodnosť, aby bola generovaná automaticky pomocou vhodného inteligentného systému. Z tejto množiny služieb je potom vygenerovaná PDDL plánovacia úloha pomocou **PDDL generátora**. Následne je použitý zvolený UI **plánovač**, ktorý rieši plánovaciu úlohu a vygeneruje plán. Vygenerovaný plán je spracovaný pomocou **parsera plánu** a **vykonávací engine** následne zabezpečuje volanie jednotlivých služieb podľa dostupného poradia. Centrálnou časťou celého systému je **báza faktov**, ktorá obsahuje používateľské dáta, dáta získané počas plánovania a aj počas vykonávania webových služieb.



Obr. 10 Architektúra systému WSPlan

4.5.5 SEMCO-WS

Projekt SEMCO-WS [58] sa zaoberal spracovaním problematiky kompozície webových a gridových služieb. Na jeho riešení sa spolupodieľal aj autor tejto práce. Kompozícia služieb je uskutočňovaná cez sémantickú anotáciu webových služieb a dát a následnú tvorbu vykonateľných pracovných tokov. Ako metóda pre kompozíciu služieb je použitá metóda **Petriho sietí**. Projekt preberá viacero nástrojov projektu K-Wf Grid[17] (Knowledge Based Workflow System for Grid Applications), ktorý sa zaoberal automatickou kompozíciou pracovných tokov gridových služieb použitím sémantických anotácií. Oproti K-Wf gridu sa zmenili hlavne dve časti [58]:

- použitie ontológie - znalostné úložisko
- obsluha webových služieb (napr. vykonávací engine)

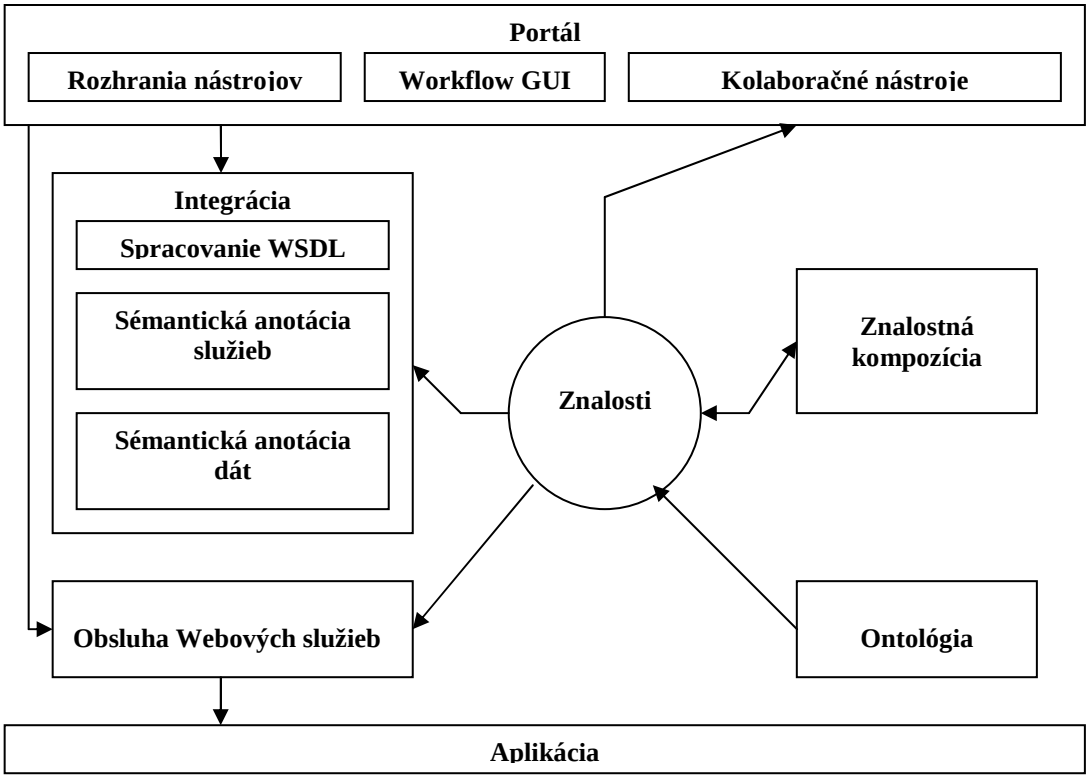
Architektúra systému je zobrazená na obrázku Obr. 11.

Platforma SEMCO-WS obsahuje úložisko sémantických údajov a ich štruktúru vo forme modulárnej a rozšíriteľnej ontológie, nástroje na vyhodnocovanie kvality webových služieb, kompozíciu abstraktných tokov práce, ich konkretizáciu a vykonávanie, ako i komfortné používateľské rozhranie. Bola otestovaná na pilotnej aplikácii z oblasti krízového manažmentu v životnom prostredí, je však ľahko aplikovateľná na akúkoľvek inú architektonicky príbuznú aplikáciu.

Modul riadenia tokov práce podporuje na úrovni transformácie metaúdajov využiteľných pre automatický proces tvorby vykonateľného toku práce hybridný prístup kombinujúci programové funkčné závislosti spolu so závislosťami popísanými proprietárnou ontológiou. Všetky bežné funkčné závislosti je možné popísať v ontologickom jazyku, na zložitejšie je potom možné využiť programovací jazyk. Jeho

[17] K-Wf Grid consortium: http://www.kwfgrid.eu

ďalším zásadným prínosom je schopnosť znovupoužitia existujúcich údajov v procese konkretizácie toku práce, vychádzajúc z použitia reprezentácie toku práce ako Petriho siete.



Obr. 11 Architektúra systému SEMCO-WS

Bola vytvorená a pilotnou aplikáciou overená modulárna ontológia pre popisovanie komponentov a vlastností distribuovaných aplikácií založených na webových službách. Táto ontológia je modulárna a rozšíriteľná i na iné domény.

4.5.6 Porovnanie jednotlivých systémov

V Tab. 2 je porovnanie jednotlivých systémov popísaných v predchádzajúcich kapitolách.

Systém	Plánovacia technika	Podpora štandardov WS	Vnútorná reprezentácia problému	Typ systému	Dynamické zasahovanie do plánovania
Kompozícia WS – PROLOG	Logické programovanie	OWL-S, WSDL	Program logického programovania	Semiautomatická kompozícia, kolaborácia s používateľom	Nie
SHOP2 systém	HTN plánovanie	OWL-S, WSDL	PDDL 2.1	Automatická kompozícia	Nie
OWLSXplan	Rozšírené dopredné plánovanie FF a HTN plánovanie	OWL-S, WSDL a OWL	PDDL 2.1	Automatická kompozícia	Áno OWLSXplan verzia 2
WSPlan	Nešpecifikovaný UI plánovač	WSDL	PDDL 1.2	Automatická kompozícia s manuálnym výberom WS	Nie
SEMCO-WS	Petriho siete	WSDL, OWL-S, OWL	GWorkflowDL	Semiautomatická kompozícia pomocou pracovných tokov	Nie

Tab. 2 Porovnanie vybraných systémov pre kompozíciu WS

Analýzou existujúcich systémov sa zistilo, že štruktúra prezentovaných systémov vo všeobecnosti odpovedá štruktúre všeobecného návrhu pre AKWS zobrazenú na Obr. 4. Každý systém obsahuje minimálne používateľskú časť, prekladač, generátor procesov a časť pre obsluhu webových služieb.

Medzi najčastejšie používaný spôsob popisu WS patrí OWL-S v kombinácií s WSDL popisom. Ako generátor procesov je najčastejšie použitý plánovač umelej inteligencie v kombinácii s plánovacím jazykom PDDL.

Použitie jazyka PDDL ako jazyka pre internú špecifikáciu je vhodné z viacerých dôvodov:

- expresivita jazyka PDDL je z pohľadu kompozície WS postačujúca,
- jazyk PDDL podlieha vývoju, z čoho vyplýva, že možnosti daného jazyka z pohľadu kompozície WS budú narastať,

- existuje viacero plánovačov, ktoré využívajú jazyk PDDL ako plánovací jazyk a je tu pravdepodobnosť možnosti ich využitia pre potreby AKWS,
- relatívne priamočiare mapovanie OWL-S procesov na PDDL akcie.

Pre popis WS sú použité OWL-S popisy WS. Vhodnosť týchto popisov vyplýva z definície OWL-S (viď kapitola 2.6). Hlavný dôvod je popis procesov pomocou IOPE – vstupov (Inputs), výstupov (Outputs), predpokladov (Preconditions) a efektov (Effects). Tieto môžu byť prípade použitia UI plánovača s PDDL plánovacím jazykom transformované na PDDL akcie.

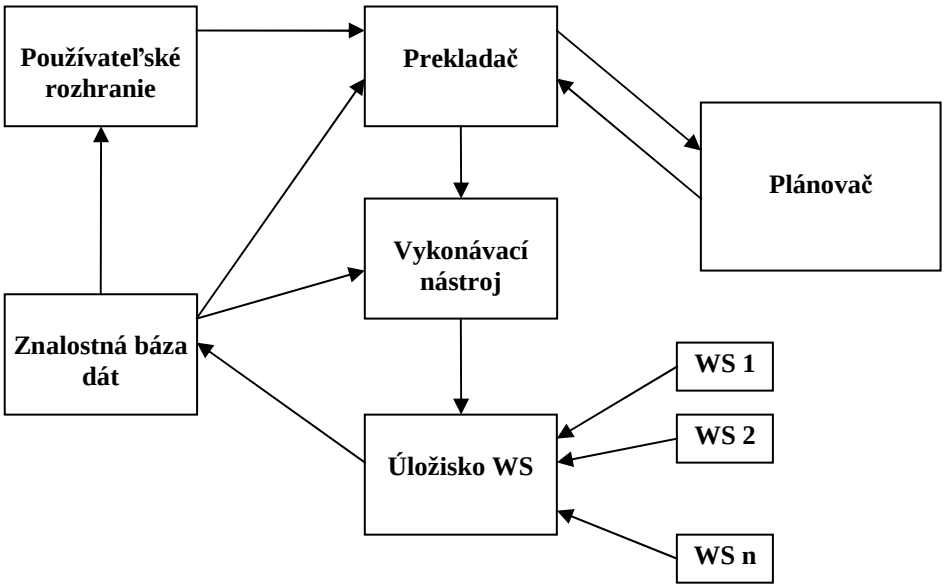
Vzhľadom na analýzu existujúcich systémov, a potreby kompozície WS, je v nasledujúcej kapitole predstavený vlastných návrh systému pre AKWS. Systém podobne ako niektoré existujúcej systémy využíva OWL-S popisy WS a PDDL plánovacím jazykom. Navrhovaný systém však počíta aj s použitím OWL doménových ontológií používaných pri tvorbe počiatočných a koncových stavov problému kompozície (kapitola 6).

Systém pre automatickú kompozíciu WS a tvorba PDDL plánovacej úlohy

5 Návrh vlastného systému pre automatickú kompozíciu WS

V tejto kapitole je prezentovaný návrh systému pre automatickú kompozíciu WS, ktorý podľa názoru autora najlepšie odráža požiadavky používateľov. V predkladanom návrhu boli zohľadnené poznatky prezentované v kapitole 3 ako aj poznatky získané štúdiom existujúcich návrhov (kapitola 4.5).

Kompozícia WS predstavuje komplexný problém. Systém, ktorý má slúžiť na realizáciu kompozície je preto potrebné rozdeliť na niekoľko samostatných častí. Všeobecný systém pre kompozíciu WS bol prezentovaný v kapitole 3. Vlastný návrh systému vychádzajúceho zo všeobecného je zobrazený na obrázku Obr. 12. Systém je označovaný skratkou ZKWS - Znalostná kompozícia webových služieb.



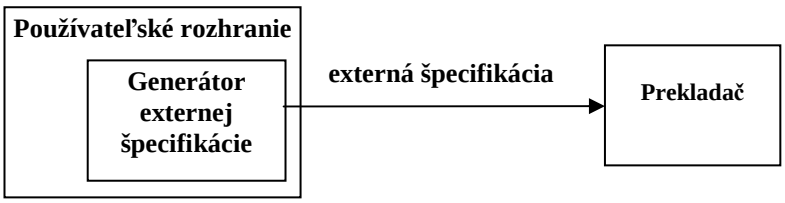
Obr. 12 Návrh systému ZKWS

5.1.1 Externá špecifikácia problému kompozície WS

Problém kompozície webových služieb spočíva v definovaní požiadaviek, ktoré je potrebné splniť pomocou dostupných webových služieb. Je potrebné získať dátový a pracovný tok tvorený webovými službami. Vykonanie daného pracovného toku (množiny WS) spolu s odpovedajúcimi dátami pre jednotlivé služby je riešením problému kompozície. Z pôvodného návrhu systému, ktorý bol prezentovaný na Obr. 4 je zrejmé, že systém pracuje minimálne z dvoma jazykmi pre špecifikáciu problému:

- **interná špecifikácia problému** a
- **externá špecifikácia problému**.

Externá špecifikácia problému je reprezentovaná špecifikáciou, pomocou ktorej definujeme problém kompozície webových služieb. Externá špecifikácia predstavuje aj špecifikáciu komunikácie používateľa so systémom. Z pohľadu bežného používateľa používa väčšina existujúcich systémov danú externú špecifikáciu značne ťažkopádnu. Vyplýva to z toho, že dané systémy boli nasadené zatiaľ len vo výskumnej sfére. Zápis OWL-S popisov WS, OWL ontológie, špeciálnych XML súborov popisujúcich stavy ontológie, popr. iné doplnkové informácie pre problém kompozície nemusí byť pre bežného používateľa jednoduché pochopiť. Ak by sa však malo jednať o systémy, ktorých nasadanie sa predpokladá v sfére, kde by ku systému mali pristupovať aj bežní užívatelia, je potrebné aby táto externá špecifikácia bola zjednodušená pre bežných používateľov (Obr. 13).



Obr. 13 Externá špecifikácia problému kompozície

Existujúce systémy používajú ako externú špecifikáciu napr. OWL-S ontológie pre popisy služieb (napr. kap. 4.5.2), popr. špecifikujú problém len pomocou preddefinovanej množiny vhodných služieb popísaných WSDL popismi (napr. kap. 4.5.4). Je vhodné a potrebné zabezpečiť nástroj, ktorý bude používateľovi uľahčovať tvorbu externej špecifikácie (Obr. 13):

Externá špecifikácia bude v prípade ZKWS pozostávať zo špecifikovania problému kompozície s využitím znalostných prístupov. V danej práci pracujeme so SOAP webovými službami, ktoré sú popísané pomocou WSDL (kapitola 2.2). Pre sémantickú kompozíciu WS však WSDL popis nie je postačujúci, a preto pri popise jednotlivých služieb systém používa OWL-S ontologické popisy služieb (kapitola 2.6). Dané popisy umožňujú viazanie na WSDL a popísať jednotlivé prvky WSDL, ako napr. operácie, vstupy a výstupy, pomocou ontológií. OWL-S okrem popisu vstupov a výstupov služieb umožňuje aj popis predpokladov a efektov pomocou podmienok. Na

definovanie daných podmienok je použitý jazyk SWRL [12]. Pre definovanie stavov prostredia, v ktorom sa kompozícia uskutočňuje, sa používajú OWL ontológie.

Generátor externej špecifikácie je nástroj**,** ktorý uľahčuje používateľom tvorbu špecifikácie problému kompozície. V ideálnom prípade by používateľ komunikoval so systémom v hovorenej reči blízkej používateľovi. Používateľ by jednoducho zadal dopyt napr. vo forme reťazca. Keďže daná možnosť je momentálne ťažko realizovateľná, je potrebné hľadať iné alternatívy. Tieto predstavujú tvorbu nástroja, ktorý bude používateľovi uľahčovať tvorbu externej špecifikácie. V prípade, že externá špecifikácia problému kompozície bude reprezentovaná pomocou OWL ontológie a OWL-S popisov služieb, môže platiť :

- OWL-S popisy služieb sú pre bežného používateľa neznáme,
- je vhodné mať nástroj, ktorý zachytáva aktuálny stav v doméne kompozície WS a reprezentuje ho vo forme ontológií (v ideálnom prípade automatická tvorba počiatočného stavu)
- používateľovi sú ponúknuté dostupné možnosti tvorby koncového stavu (napr. zoznam OWL tried, objektových a dátových vlastností pre vytvorenie faktov v ontológii)

Informácie získané od používateľa sú následne použité generátorom externej špecifikácie pre popis problému kompozície pomocou externej špecifikácie (v systéme ZKWS pomocou OWL a OWL-S).

5.1.2 Plánovač

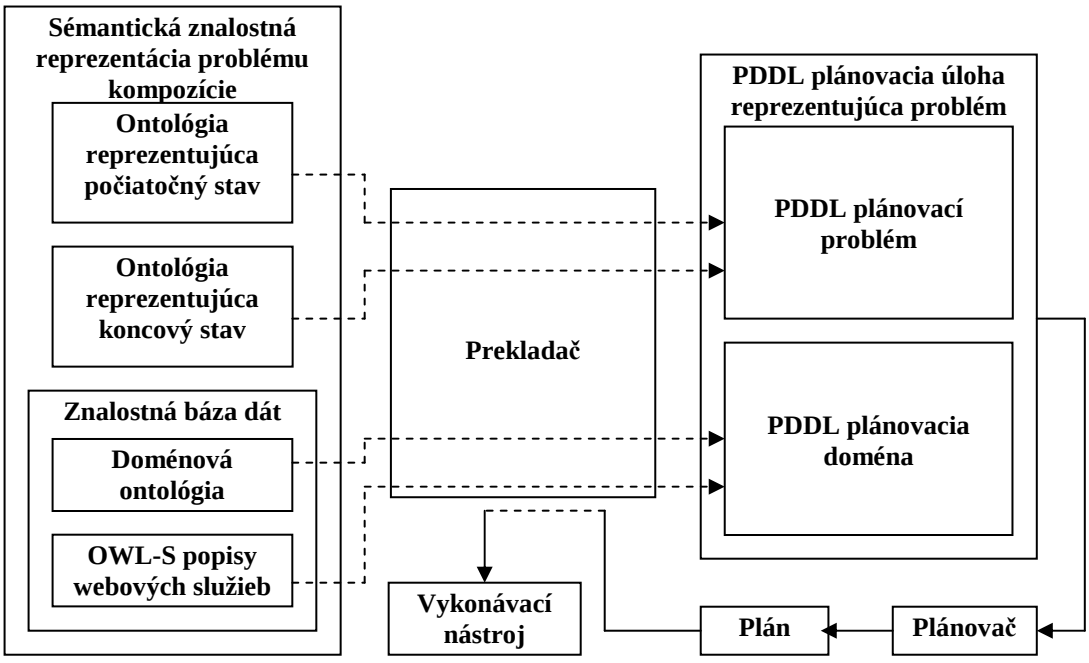
Na riešenie problému kompozície WS musí byť vytvorený vhodný nástroj. Za najlepšiu voľbu je zvoliť plánovač využívajúci metódy umelej inteligencie (dôvody viď kap. 4). Každý plánovač pracuje s určitou špecifikáciou plánovacej úlohy. V systéme bol pre definíciu plánovacej úlohy zvolený jazyk PDDL (kapitola 4.3). Jazyk PDDL je relatívne nový jazyk (1998), mapuje však väčšinu požiadaviek plánovacej domény a stále sa vyvíja. Aj základná verzia daného jazyka poskytuje dostatočnú expresivitu a možnosti pre popis problému kompozície pomocou daného jazyka. Zároveň však vývoj jazyka (v súčasnosti verzia 3.0 [48]) umožňuje predpokladať, že jeho vhodnosť pre problém kompozície WS nebude klesať.

5.1.3 Prekladač

Ako bolo spomenuté v predchádzajúcich dvoch kapitolách, systém pre kompozíciu webových služieb pracuje minimálne s dvoma špecifikáciami problému kompozície. V prípade systému ZKWS externá špecifikácia popisuje problém s využitím znalostného prístupu ontologicky, zatiaľ čo pre riešenie problému kompozície je použitý plánovač umelej inteligencie využívajúci špecifikáciu pomocou PDDL jazyka. Preto systém potrebuje nástroj pre preklad externej špecifikácie do internej špecifikácie problému kompozície (Obr. 14).

Jazyk PDDL reprezentuje plánovaciu úlohu (transformovaný problém kompozície) pomocou dvoch častí: plánovacia doména a plánovací problém. Prekladač musí zabezpečiť tvorbu týchto dvoch prvkov plánovacej úlohy. Pre danú transformáciu boli v kapitole 6 navrhnuté algoritmy. Po vyriešení plánovacej úlohy pomocou PDDL plánovača máme k dispozícii PDDL plán reprezentovaný PDDL akciami a dátovým tokom medzi týmito akciami. Prekladač preto ešte musí zabezpečiť spracovanie daného plánu. Spracovanie plánu môže byť nasledovne:

1. PDDL plán je transformovaný do OWL-S zloženého procesu, ktorý môže byť vykonaný, spolu s dátovým tokom, alebo,
2. PDDL akcie sú mapované priamo na odpovedajúce atomické procesy definované v OWL-S a vykonávané s odpovedajúcimi vstupmi, ktoré sme získali plánom a ktorým vieme priradiť konkrétne vstupy pre OWL-S proces.



Obr. 14 Prekladač –tvorba PDDL plánovacej úlohy

5.1.4 Vykonávací nástroj

Vykonávanie spočíva vo vykonávaní jednotlivých služieb tvoriacich vygenerovaných pracovný tok (PDDL plán). Pre vykonávanie služieb musí byť vytvorený vhodný nástroj, ktorý umožňuje volať potrebné WSDL operácie s odpovedajúcimi vstupmi a samozrejme zachytávať výstupy zo služieb, ktoré sú súčasťou pracovného toku. V systéme ZKWS sú jednotlivé WS reprezentované pomocou OWL-S. OWL-S popisy umožňujú reprezentovať služby pomocou procesov. Preto bude vykonávací nástroj zameraný na vykonávanie OWL-S procesov. Na danú exekúciu procesov je možné použiť napr. OWL-S API[18], ktoré podobný nástroj vytvorený v jazyku Java[19] v sebe implementuje.

5.1.5 Znalostná báza dát

Znalostná báza dát v sebe zahŕňa sémantické dáta, pomocou ktorých je možné špecifikovať problém kompozície. Tieto dáta sú reprezentované pomocou ontologických OWL-S popisov webových služieb. Webové služby sú dostupné z úložiska webových služieb (kapitola 5.1.6). Ďalej báza dát obsahuje OWL ontológie reprezentujúce domény, v ktorých sa uskutočňuje kompozícia webových služieb. Použitie daných ontológií je nasledovné:

- ontológie sa používajú pri definovaní PDDL plánovacej domény spolu s OWL-S popismi služieb (kapitola 6.1),
- umožňujú definovať počiatočné a koncové stavy problému kompozície, ktoré sa ďalej používajú na definovanie plánovacieho problému plánovacej úlohy v PDDL (kapitola 6.2).
- ontológie sú používané v OWL-S popisoch služieb na definovanie typov vstupov, výstupov, ako aj v SWRL podmienkach.

5.1.6 Úložisko webových služieb

Úložisko webových služieb má za úlohu poskytovať zaručené referencie na webové služby, ktoré sú popísané pomocou WSDL jazyka. Na dané WSDL súbory sa viažu OWL-S popisy webových služieb v znalostnej báze dát.

[18] OWL-S API - http://on.cs.unibas.ch/owls-api/
[19] Programovací jazyk Java - http://www.java.com/

6 Transformácia OWL ontológií a OWL-S ontologického popisu služieb do PDDL plánovacej úlohy

V tejto kapitole je popísaný navrhnutý spôsob pre získavanie PDDL plánovacej úlohy z ontológií. Pre popis počiatočného a koncového stavu sú použité OWL ontológie. Pre popis akcií sú využité OWL-S popisy webových služieb. Ako bolo spomenuté v kapitole 4 o PDDL, plánovacia úloha v danom jazyku pozostáva z plánovacej domény a plánovacieho problému.

6.1 Tvorba PDDL plánovacej domény

Kompletná štruktúra plánovacej domény pre PDDL verziu 1.2 bola v EBNF forme zobrazená v kapitole 4. Z pohľadu tvorby plánovacej domény z OWL ontológií je však možné ju trocha zjednodušiť vypustením niektorých, pre účely plánovania v automatickej kompozícii služieb, nepotrebných častí. Zjednodušená EBNF forma štruktúry PDDL domény vyzerá potom nasledovne:

```
<domain> ::= (define (domain <name>)
             [<require-def>]
             [<types-def>]:typing
             [predicates-def>]
             <structure-def>*)
```

Štruktúra PDDL domény sa teda v našom prípade bude skladať z piatich častí:

- definícia mena domény,
- definícia požiadaviek domény,
- typy domény,
- predikáty,
- definícia akcií.

Tvorba každej jednej z uvedených častí je popísaná osobitne v samostatnej podkapitole.

6.1.1 PDDL doména – tvorba mena domény

Ku tvorbe mena plánovacej domény je potrebná základná (doménová) ontológia. Ako vyplýva z kapitoly o OWL (kapitola 2.5), ontológia obsahuje tri hlavné časti:

- definíciu menných priestorov,

- hlavičku ontológie a
- definície samotných elementov ontológie.

Pre definíciu mena domény je potrebné pracovať s hlavičkou ontológie, kde je možnosť definovať názov ontológie (napr. pomocou **<dc:title>,** kde dc je menný priestor **dc=http://purl.org/dc/elements/1.1/**. DC[20] (Dublin Core) predstavuje štandard pre metadátový popis objektov). Daný názov ontológie je možné považovať za meno domény.

Meno domény je v EBDF forme definované ako:

(domain <name>)

Časť **<name>** predstavuje syntaktické pravidlo pre definovanie ľubovoľného mena v PDDL plánovacej úlohe. Meno v PDDL predstavuje reťazec, ktorý môže obsahovať len nasledovné znaky: veľké a malé písmená, číslice, podčiarnik („_“) a pomlčku („-“). Všetky ostané znaky sú zakázané. Navrhnutý algoritmus pre získanie doménového mena z ontológie je uvedený nižšie:

```
-----------------------------------------------------------------------------------------------
FUNCTION owl2pddlDomainName(O : OWL Ontológia) : PDDL Name
  VAR menoDomeny : PDDL Name;
  BEGIN
    menoDomeny := ziskajMenoZOntologie(O);
    IF (menoDomeny spĺňa podmienky PDDL Name) THEN BEGIN
      continue;
    ELSE BEGIN
      menoDomeny := upravMeno(menoDomeny);
    END;
    owl2pddlDomainName : menoDomeny;
  END;
-----------------------------------------------------------------------------------------------
```

Alg. 1 Spôsob tvorbu PDDL doménového mena z OWL ontológie

Algoritmus Alg. 1 je reprezentovaný vo forme funkcie **owl2pdllDomainName**, kde na vstupe je OWL ontológia a na výstupe je reťazec, spĺňajúci obmedzenia definovania mena v PDDL špecifikácii. Najprv je potrebné získať názov ontológie. Daný názov možno v ontológii reprezentovať viacerými spôsobmi. Napr. ako bolo uvedené vyššie pomocou **<dc:title>**, alebo napr. pomocou komentára z rdfs schémy:

<rdfs:comment>Menovka ontológie</rdfs:comment>

[20] The Dublin Core Metadata Initiative - http://dublincore.org/

Táto menovka môže samozrejme obsahovať aj iné ako povolené znaky v PDDL. Preto túto skutočnosť je nutné ošetriť. Pomocou funkcie **splnaPodmienky(X:Reťazec):Boolean** sa zistí, či vstupný reťazec spĺňa podmienky. Ak nie tak je následne upravený pomocou funkcie **upravMeno(X:Reťazec):Reťazec** tým, že napr. nepovolené znaky jednoducho odignorujeme.

6.1.2 PDDL doména – požiadavky

PDDL požiadavky sú v doméne definované v časti **[<require-def>]**. Ide o voliteľný element štruktúry domény. V prípade, že nie je uvedený, automaticky sa predpokladá jedna požiadavka domény a tou je STRIPS (**:strips**). V prípade tvorby PDDL plánovacej domény z ontológií sú potrebné aj ďalšie požiadavky, napr. možnosť definovania typov (**:typing**). Syntaktické pravidlá pre definovanie požiadaviek v doméne sú v EBNF forme definované nasledovne:

```
<require-def> ::= (:requirements <require-key>+)
<require-key> - :strips, :typing, :equality, :fluents, ...
```

<require-key> predstavuje teda syntaktické pravidlo, za ktoré je potrebné dosadiť konkrétnu požiadavku. Zoznam všetkých je možné vidieť napr. v [43]. Popri požiadavke **:strips**, budú potrebné aj iné, napr. pre možnosť definovania typov v doméne. Preto v našom prípade vyzerá táto časť nasledovne:

```
(:requirements :strips :typing)
```

Dané požiadavky sa pridávajú do definície PDDL domény manuálne. V prípade, ak by bola potreba automatického pridávania požiadaviek, mohli by byť umiestnené napr. v hlavičke ontológie.

6.1.3 PDDL doména – typy v doméne

Definovanie typov v PDDL plánovacej úlohe je možné pomocou syntaktických pravidiel týkajúcich sa časti **[<types-def>]**$^{:typing}$. Horný index pri danej časti znamená, že použitie daného elementu v štruktúre domény je možné len v prípade, ak je daná požiadavka (v tomto prípade **:typing**) definovaná medzi požiadavkami. Bez ujmy na všeobecnosti môžeme predpokladať, že je definovaná (kapitola 6.1.2). EBNF forma pre definovanie typov je nasledovná:

```
<types-def> ::= (:types <typed list (name)>
```

Typy sú v štruktúre PDDL domény reprezentované v špeciálnom zozname označovanom **<typed list (name)>**. EBNF forma tohto zoznamu je uvedená nižšie:

<typed list (x)> ::= x*;

<typed list (x)> ::= :typing x+ - <type><typed list(x)>

<type> ::= <name>

Táto definícia znamená, že daný zoznam môže byť expandovaný podľa dvoch pravidiel. Zoznam sa okrem definovania typov používa aj v iných častiach PDDL domény alebo problému (napr. definovanie parametrov akcií). Prvý prípad predstavuje klasický zoznam prvkov **x**. Dané **x** je možné nahradiť ľubovoľným elementom. V prípade typov bol nahradený elementom **name**, ktorého definícia bola spomenutá v kapitole 6.1.1. Druhý spôsob je možné použiť len v prípade keď je definovaná požiadavka **:typing**. V tomto prípade je v zozname určitý počet prvkom nasledovaný ich typom a za nimi nasleduje ďalší zoznam, napríklad:

(:types auto autobus traktor – vozidlo vodic chodec cestujuci - osoba)

V príklade vyššie je definovaných 8 typov. Prvé tri typy (**auto autobus traktor**) sú podtypy typu **vozidlo**, ďalšie tri typy sú typu **osoba**.

Analógia PDDL typov v OWL ontológiách môže predstavovať **triedy** a vzťahy **trieda-supertrieda**. Táto transformácia je viac menej priamočiara.

Predpokladajme, že množina **C** reprezentuje všetky triedy z ontológie **O**. Ďalej predpokladajme, že každá trieda **c** patriaca do množiny **C** je podtriedou maximálne jednej triedy **s** (trieda **s** sa v OWL označuje aj **supertrieda**). V prípade, že trieda nie je podtriedou žiadnej triedy predpokladáme, že supertrieda triedy **c** je **object** (viď. upravený príklad nižšie).

(:types vozidlo osoba – object auto autobus traktor – vozidlo vodic chodec cestujuci - osoba)

Algoritmus získavania PDDL typov z ontológie:

```
FUNCTION owl2pddlTypes(O : OWL Ontológia) : PDDL Typed List
  VAR
    TL : PDDL Typed List - zoznam typov;
    CS : množina usporiadaných dvojíc - trieda supertrieda;
    C  : množina reťazcov - triedy;
    S  : množina reťazcov – rôzne typy;
  BEGIN
    C := ziskajVsetkyTriedy(O);
    FOR c TO C DO BEGIN
```

```
      VAR
        pom : dvojica trieda-supertrieda;
      pom.trieda := c;
      pom.supertrieda := ziskajSupertriedu(O,c);
      IF (supertrieda neexistuje) THEN BEGIN
        supertrieda := "object";
      END;
      IF (S neobsahuje supertriedu) THEN BEGIN
        pridaj supertriedu do S;
      END;
      pridaj pom do množiny CS;
    END;
    FOR s TO S DO BEGIN
      VAR
        type : reťazec;
      X : množina tried so supertriedou s;
      type := s;
      FOR cs TO CS DO BEGIN
        IF (cs.supertrieda = s) THEN BEGIN
          pridaj do množiny X cs.trieda;
        END;
      END;
      pridaj do TL X s príslušným typom type;
    END;
    owl2pddlTypes : TL;
  END;
```

Alg. 2 Získavanie PDDL typov z OWL ontológie

Algoritmus funkcie v prvom kroku získa všetky triedy z ontológie **O.** Triedy si uložíme do množiny **C**. Ďalej pre každú triedu z množiny **C** sa vyhľadá **supertrieda**. V prípade že neexistuje, za **supertriedu** sa považuje najvšeobecnejšia trieda **object**. Každá trieda so svojou supertriedou sa uloží ako usporiadaná dvojica do množiny **CS**. Potom sa vytvorí množina **S**, ktorá bude obsahovať všetky rôzne supertriedy. (v predchádzajúcom príklade by množina S obsahovala tri typy: object, vozidlo, osoba). Pre každý typ z množiny S sa nájdu triedy, ktoré ho majú ako supertriedu. Všetky tieto triedy spolu s daným typom budú vytvárať jeden zoznam typov ($\mathbf{x_{11}\ x_{12}\ x_{13}\ ...\ x_{1n}\ -\ type_1}$). Pre každý typ z **S** je tento zoznam vložený do celkového zoznamu typov ($\mathbf{x_{11}\ x_{12}\ x_{13}\ ...\ x_{1n}\ -\ type_1\ x_{21}\ x_{22}\ x_{23}\ ...\ x_{2m}\ -\ type_2\ x_{w1}\ x_{w2}\ x_{w3}\ ...\ x_{wt}\ -\ type_w}$). Definícia typov v štruktúre PDDL domény potom bude vyzerať nasledovne:

(:types $\mathbf{x_{11}\ x_{12}\ ...\ x_{1n}\ -\ type_1\ x_{21}\ x_{22}\ ...\ x_{2m}\ -\ type_2\ x_{w1}\ ...\ x_{wt}\ -\ type_w}$**)**,

kde bude **w** supertypov, pričom každému sú priradené príslušné podtypy. Príklad PDDL typov vytvorený z ontológie je zobrazený na Pr. 5.

Fragment OWL ontológie:

```
<!-- OWL triedy -->
<owl:Class rdf:about="&travel;#at"/>
<owl:Class rdf:about="&travel;#bridge"/>
<owl:Class rdf:about="&travel;#driving"/>
<owl:Class rdf:about="&travel;#mobile"/>
<owl:Class rdf:about="&travel;#person">
    <rdfs:subClassOf rdf:resource="&travel;#thing"/>
</owl:Class>
<owl:Class rdf:about="&travel;#place"/>
<owl:Class rdf:about="&travel;#road"/>
<owl:Class rdf:about="&travel;#thing"/>
<owl:Class rdf:about="&travel;#vehicle">
    <rdfs:subClassOf rdf:resource="&travel;#thing"/>
</owl:Class>
```

Získané PDDL typy:

```
(:types
       driving road thing place at bridge mobile - object
       person vehicle - thing
       )
```

Pr. 5 Tvorba PDDL typov z OWL tried

6.1.4 PDDL doména – predikáty

Predikát môže reprezentovať vlastnosť alebo vzťah medzi entitami. Napr. ak je potrebné v plánovacej doméne definovať fakt, že „Paľo je osoba“. V tomto prípade je potrebné mať definovaný predikát v plánovacej doméne, ktorý to umožní. Predikát bude mať napr. formu **`(osoba ?x - person)`**. Daný predikát bude mať názov **`osoba`** a bude obsahovať jeden parameter typu **`person`**. Definícia faktu je potom nasledovná: **`(osoba Palo)`**.

EBDN forma pre definovanie predikátov má tvar:

`<predicates-def> ::= (predicates <atomic formula skeleton>+)`

`<atomic formula skeleton> ::= (<predicate> <typed list(variable)>)`

`<predicate> ::= <name>`

`<variable> ::= ?<name>`

Predikáty sú v štruktúre domény reprezentované ako zoznam kostier atomických formúl. Každá kostra atomickej formule predstavuje jeden predikát a ten sa skladá z predikátového mena a zoznamu premenných. Na zoznam premenných je použitý zoznam typov podobne ako v predchádzajúcej kapitole. Ak nie je potrebné otypovanie premenných (**`:typing`** nie je medzi požiadavkami domény), spomínaný zoznam je len

jednoduchý zoznam premenných. Ak je potrebné definovanie typov (je tu požiadavka **`:typing`**), pri každej premennej potrebujeme určiť jej typ. Treba poznamenať, že usporiadanie premenných je pevne dané.

Predikáty môžeme považovať za vzory, pomocou ktorých sme schopný vytvárať fakty dosadením konkrétnych objektov za premenné predikátu. Z kapitoly 2.5 vyplýva, že v ontológii môžeme definovať všeobecné fakty o triedach a špecifické fakty o inštanciách. V našom prípade potrebujeme získať všeobecné vlastnosti tried. Ďalej sú zaujímavé samotné triedy, hlavne možnosť tvorby inštancie tried. Napr. nech existuje trieda person s inštanciou Palo. V PDDL je to možné zapísať (viď vyššie) : **`(osoba palo)`** a zodpovedajúci predikát môže mať nasledovnú podobu: **`(osoba ?x - person)`**.

V ontológii možno definovať dva typy vlastností: objektové vlastnosti a dátové vlastnosti. Obidva typu predstavujú binárne relácie. V prípade objektových vlastností je to relácia medzi objektmi (triedami). Definičný obor relácie tvoria inštancie určitej triedy, podobne aj obor hodnôt je tvorený množinou inštancií určitej triedy.

V prípade dátových vlastností je to relácia medzi objektom (triedou) a dátovými typmi (obyčajne z XML schémy, napr. reťazec, celé číslo, dátum a pod.). Označme **V** množinu všetkých objektových a dátových vlastností, pričom platí, že ku každej vlastnosti vieme určiť doménu **d** a rozsah **r**. Teraz určime všetky rôzne triedy, ktoré tvoria doménu ľubovoľnej vlastnosti. Množina týchto tried bude reprezentovať predikáty a označíme ju **P**. Potrebujeme získať parametre pre predikáty. Pre každú triedu **t** z množiny **P** získame vlastnosť, kde daná trieda tvorí doménu. Dané vlastnosti nám budú reprezentovať parametre predikátu a rozsahy jednotlivých vlastností typy týchto parametrov. Takto sa vytvoria predikáty z vlastností definovaných v OWL ontológii. Druhá časť predikátov bude tvorená z tried, ktoré sa nenachádzajú v doméne žiadnej vlastnosti. Názov predikátu, parameter aj typ parametra budú tvorené zhodným názvom, napr. pomocou predikátu **`(person ?person - person)`** môžeme definovať fakt reprezentujúci osobu.

Algoritmus pre tvorbu PDDL predikátov z OWL ontológie je zobrazený na Alg. 3. Za definíciou algoritmu nasleduje príklad Pr. 6, ktorý predstavuje tri PDDL predikáty vzniknuté z OWL ontológie.

```
FUNCTION owl2pddlPredicates( O : OWL Ontológia) : Zoznam Predikátov
  VAR
    V : množina objektových a dátových vlastnosti;
    P : množina tried tvoriacich domény vlastnosti;
    C : množina všetkých tried ontológie;
    Preds  : Zoznam Predikátov;
  BEGIN
    V := ziskajVsetkyVlastnosti(O);
    FOR v TO V DO BEGIN
      VAR
        d : je doména vlastnosti v;
      IF (P neobsahuje d) THEN BEGIN
        vlož d do P;
      END;
    END;
    FOR p TO P DO BEGIN
      VAR
        pred : PDDL predikát;
      meno predikátu pred := p;
      FOR v TO V DO BEGIN
        IF (v má doménu p) DO BEGIN
          v je parameter predikátu pred;
          rozsah vlastnosti v je typom parametru predikátu pred;
          v aj s typom pridáme zo zoznamu param. predikátu pred;
        END;
      END;
      usporiadaj parametre predikátu podľa typu;
      predikát pred vložíme do zoznamu preds;
    END;
    FOR c TO (C-P) DO BEGIN
      VAR
        pred : predikát;
      meno predikátu, parameter aj type := c;
      predikát pred vložíme do zoznamu preds;
    END;
    owl2pddlTypes : preds;
  END;
```

Alg. 3 Tvorba PDDL predikátov z OWL ontológie

Fragment OWL ontológie:

```
<!-- Object Properties -->
 <owl:ObjectProperty rdf:about="&travel;#has_person">
     <rdfs:domain rdf:resource="&travel;#driving"/>
     <rdfs:range rdf:resource="&travel;#person"/>
     <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
 </owl:ObjectProperty>
```

```
    <owl:ObjectProperty rdf:about="&travel;#has_vehicle">
        <rdfs:domain rdf:resource="&travel;#driving"/>
        <rdfs:range rdf:resource="&travel;#vehicle"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="&owl;topObjectProperty"/>

    <!-- Classes -->
    <owl:Class rdf:about="&travel;#driving"/>
    <owl:Class rdf:about="&travel;#person"/>
    <owl:Class rdf:about="&travel;#vehicle"/>
```

PDDL predikáty:

```
(:predicates
        (driving   ?has_person - person  ?has_vehicle - vehicle)
        (person   ?person - person)
        (vehicle   ?vehicle - vehicle)
)
```

Pr. 6 Tvorba PDDL predikátov z OWL ontológie

6.1.5 PDDL doména – definícia akcií

V tejto podkapitole je ukázaný spôsob ako získať PDDL popis akcií z OWL-S ontológií.

Akcie na nachádzajú v štruktúre domény PDDL pod syntaktickým elementom **<structures-def>***. Sú reprezentované ako štruktúry a ich EBNF forma je nasledovná:

<structure-def> ::= <action-def>

<structure-def> ::= :domain-axioms <axiom-def>

<structure-def> ::= :action-expansion <method-def>

V prípade spracovania OWL-S procesov nás bude zaujímať prvý riadok danej štruktúry, a to štruktúra **<action-def>**. EBNF forma syntaktického elementu action-def je nasledovná:

```
<action-def> ::= (:action <action functor>
                    :parameters ( <typed list (variable)> )
                    <action-def body>)
<action functor> ::= <name>
<action-def body> ::=
                    [:precondition <GD>]
                    [:effect <effect>]
```

Každá akcia v PDDL sa bude skladať z mena akcie (**<action functor>**), parametrov akcie (**<typed list (variable)>**) a z tela akcie (**<action-def body>**). Treba poznamenať, že definícia EBNF formy pre telo akcie bola zjednodušená len

na definíciu predpokladov a efektov akcie. Kompletnú definíciu je možné vidieť napr. v [43]. Definície syntaktické elementy **<GD>** a **<effect>** sú uvedené nižšie:

```
<GD> ::= <atomic formula (term)>
<GD> ::= (and <GD>*)
<GD> ::= <literal (term)>

<literal (t)> ::= <atomic formula (t)>
<literal (t)> ::= (not (atomic formula (t))

<atomic formula (t)> ::= (<predicate> t*)
<term> ::= <name>

<effect> ::= (and <effect>*)
<effect> ::= (not <atomic formula (term)>
<effect> ::= <atomic formula(term)>
```

Element **GD** (Goal Descriptions) slúži na definovanie požadovaného cieľa (viď. kapitola 6.2.5), ale aj na definovanie predpokladov pre akcie. Môže obsahovať v sebe ďalšie definície GD spojené logickou spojkou **a** (**and**), alebo literál poprípade atomickú formulu. Atomická formula v tomto prípade vzniká dosadením konkrétnych termov za parametre určitého predikátu. Literál reprezentuje buď samotnú atomickú formulu, alebo jej negáciu. Efekt akcie v sebe môže obsahovať ďalšie efekty, spojené logickou spojkou **a** (**and**), alebo atomickú formulu termov, popr. negáciu tejto formuly.

Pre lepšiu predstavivosť je uvedený jednoduchý príklad PDDL akcie. Akcia má názov **Drive** a jej úlohou je premiestniť vec (**?Thing**) z jedného miesta (**?from**) na iné miesto (**?Thing**):

```
(:action Drive
       :parameters ( ?from ?to - place ?Thing - thing)
       :precondition (and (road ?from ?to)
                          (at ?Thing ?from)
       :effect (and (at ?Thing ?to) (not (at ?Thing ?from))))
```

- názov akcie – **Drive**

- parametre akcie – zoznam typov (**`?from ?to - place ?Thing - thing`**), kde každá premenná musí mať pridelený typ (**`?from ?to`** sú typu **`place`**, **`?Thing`** je typu **`thing`**).
- telo akcie:
  - predpoklady: **`(and (road ?from ?to) (at ?Thing ?from))`**

    - vychádza sa z predikátov, ktorých tvorba bola vysvetlená v kapitole 6.1.4.

    Pre vykonanie akcie musia byť splnené nasledovné predpoklady:

    - musí existovať cesta z miesta **`?from`** do miesta - **`?to`**

    - vec **`?Thing`** sa musí nachádzať na mieste **`?from`**
  - efekty: **`(and (at ?Thing ?to) (not (at ?Thing ?from))))`**

    - po vykonaní akcie budú aplikované nasledovné efekty:

    - vec **`?Thing`** bude na mieste **`?to`**

    - vec **`?Thing`** sa už nemôže nachádzať na mieste **`?from`**.

Príklady viacerých PDDL akcií sú prezentované v prílohe A príklad 5.1.

V prípade OWL-S ontológií v súvislosti s možnosťou tvorby PDDL akcií je potrebný procesný model OWL-S popisu. Daný model v sebe nesie procesy, ktoré reprezentujú operácie webových služieb. Ako bolo uvedené v kapitole 2.6, v OWL-S ontológii môžeme prezentovať tri druhy procesov:

- **atomický proces**,
- **zložený proces**,
- **jednoduchý proces**.

Spôsob transformácie každého z týchto procesov na PDDL akciu je rozdielny, preto pre každý typ procesu budeme musieť navrhnúť osobitnú funkciu. Pri návrhu funkcií sme sa inšpirovali čiastočne v [52]. Nižšie je uvedený algoritmus na získavanie procesov z ontológie a volanie príslušnej funkcie transformácie podľa typu procesu.

```
FUNCTION owls2pddlActions(O : OWL-S Ontológia) : PDDL akcie
  VAR
    procesy : množina OWL-S procesov;
    structures : množina PDDL akcii;
  BEGIN
    FOR p TO procesy DO BEGIN
      VAR
```

```
        a : PDDL akcia - actionDef;
      IF (p je atomický proces) THEN BEGIN
        a := owls2pddlActionAtomicProcess(p);
      END;
      IF (p je zlozený proces) THEN BEGIN
        a := owls2pddlActionCompositeProcess(p);
      END;
      IF (p je jednoduchý proces) THEN BEGIN
        a := owls2pddlActionSimpleProcess(p);
      END;
        pridaj akciu a do množiny akcií structures;
    END;
    owls2pddlActions:=structures;
  END;
```

Alg. 4 Tvorba PDDL akcií z OWL-S procesov

Pre každý typ procesu bola navrhnutá osobitná funkcia na jeho spracovanie. Tieto sú popísané v nasledujúcich podkapitolách.

6.1.5.1 Atomický proces

Atomický proces je najjednoduchší proces. Z pohľadu používateľa predstavuje proces jedného kroku a spočíva v priamom volaní webovej služby, ktorú popisuje. Atomický proces je reprezentovaný pomocou **IOPE** – vstupov (angl. **Inputs**), výstupov (**Outputs**), predpokladov (**Preconditions**) a efektov (**Effects**). Dané informácie proces popisuje pomocou vlastností z príslušnej ontológie. Ktoré to sú, bolo popísané v kapitole 2.6. V prípade transformácie OWL-S procesu na PDDL akciu sú potrebné nasledovné:

- **`hasInput`** – reprezentujú vstupné dáta do procesu,
- **`hasOutput`** – reprezentujú výstupné dáta z procesu,
- **`hasPrecondition`** – predstavujú podmienky, ktoré musia byť splnené, aby mohol byť proces vykonaný,
- **`hasResult`** – výsledok vykonania procesu. Reprezentuje efekty procesu.

V prípade atomických procesov je nutné zaviesť podmienku, že žiadny atomický proces, ktorý je využívaný na získavanie PDDL akcií, nebude obsahovať výstupy a efekty zároveň. Vychádzame z [50][52], kde je spomenuté, že po vykonaní odpovedajúcej webovej služby získavame výstupy a splnia sa efekty služby (reprezentované v OWL-S ontológii). Pritom výstupy majú čisto informačný charakter,

zatiaľ čo efekty predstavujú fyzické zmeny stavov v svete, v ktorom webová služba pôsobí. Preto atomický proces len s výstupmi modeluje čisto informačnú webovú službu zatiaľ čo atomický proces len s efektmi modeluje čisto fyzickú webovú službu. Počas plánovania vo všeobecnosti nie je potrebné meniť stav sveta, ale naopak potrebujeme získavať informácie o tom, ako sa budú správať akcie počas vykonávania plánu. Z toho dôvodu je potrebné mať k dispozícii počas plánovania atomické procesy buď len s výstupmi alebo len s efektmi.

V prípade spracovania atomického procesu budeme teda potrebovať dve funkcie, osobitne pre atomický proces s výstupmi a osobitne pre proces s efektmi (Alg. 5).

```
-------------------------------------------------------------------------------------------------
FUNCTION owls2pddlActionAtomicProcess(P : OWL-S Atomický Proces)
                                                          : PDDL Akcia
  VAR
    A : PDDL akcia;
  BEGIN
    IF (P je atomický proces len s výstupmi) DO BEGIN
      A := owls2pddlAction_AtomicProcessOutputs(P);
    END;
    IF (P je atomický proces len s efektmi) DO BEGIN
      A := owls2pddlAction_AtomicProcessEffects(P);
    END;
    owls2pddlActionAtomicProcess : A;
  END;
-------------------------------------------------------------------------------------------------
```

Alg. 5 Predklad atomického proces do PDDL akcie

V prípade spracovania atomického procesu len s efektmi je situácia najjednoduchšia, lebo mapovanie informácií, ktoré poskytuje takýto proces na PDDL akciu je priamočiare. Každá PDDL akcia obsahuje názov (**<action functor>**), parametre (**:parameters (<typed list (variable)>)**) a telo akcie (**<action-def body>**). V tele funkcie budeme predpokladať, že sa skladá len z efektov a predpokladov:

```
<action-def body> ::=
                    [:precondition <GD>]
                    [:effect <effect>]
```

Potom funkcia pre transformáciu OWL-S atomického popisu služby do odpovedajúcej PDDL akcie môže mať tvar ako je uvedené nižšie (Alg. 6).

```
-----------------------------------------------------------------------------------------------------

FUNCTION owls2pddlAction_AtomicProcessEffects(P :
                                OWL-S Atomický Proces) : PDDL Akcia
  VAR
    A : PDDL akcia;
    Pr : predikáty z PDDL domény;
    Pred : predpoklady;
  BEGIN
    názov akcie A := názov procesu P;
    FOR i TO všetky vstupy P DO BEGIN
      pridaj i so svojim typom medzi parametre akcie A;
    END;
    usporiadaj parametre A podľa typov;

    FOR p TO všetky predpoklady procesu P DO BEGIN
      namapuj daný predpoklad p na odpovedajúci predikát z Pr;
      vlož predpoklad do množiny Pred;
    END;
    vlož predpoklady z Pred ako konjunkciu do predpokladov akcie A;
    FOR e TO všetky efekty procesu P DO BEGIN
      namapuj daný efekt e na odpovedajúci predikát z Pr.;
      vlož upravený efekt do množiny efektov akcie A;
    END;
    owls2pddlAction_AtomicProcessEffects : A;
  END;
-----------------------------------------------------------------------------------------------------
```

Alg. 6 Atomický proces, ktorý obsahuje len efekty

V prípade spracovanie atomického procesu obsahujúceho len výstupy je situácia trošku zložitejšia. PDDL akcia bude mať podobný tvar bolo definované vyššie. Proces bude okrem vstupov a výstupov obsahovať predpoklady. Efekty sa tu nachádzať nebudú. Vstupy a predpoklady sa budú transformovať podobne ako v prípade procesu obsahujúceho efekty. Problém nastáva v prípade tvorby efektov PDDL akcie. Efekty je potrebné vytvoriť pomocou výstupov z procesu. Preto potrebujeme medzi predpoklady pridať ešte predpoklad, že daný proces je vykonateľný. Na overenie vykonateľnosti daného procesu bola navrhnutá funkcia **`vykonateľný`**, ktorá informuje o vykonateľnosti daného procesu [30][50][52]. Efekty PDDL akcie je potom možné dostať vhodnou transformáciu výstupov, ktoré boli získané vykonaním procesu. Vykonanie procesu je zabezpečené funkciou **`vykonaj`**.

```
-----------------------------------------------------------------------------------------------------

FUNCTION owls2pddlAction_AtomicProcessOutputs(P :
                        OWL-S Atomický Proces) : PDDL Akcia
  VAR
    A : PDDL Akcia;
    Pr : Predikáty z PDDL domény;
    Pred : Predpoklady;
    V : množina výstupov;
  BEGIN
    názov akcie A := názov procesu P;
    FOR i TO všetky vstupy P DO BEGIN
      pridaj i so svojim typom medzi parametre A;
    END;
    usporiadaj parametre A podľa typov;

    FOR p TO všetky predpoklady procesu P DO BEGIN
      namapuj Dany predpoklad p na odpovedajúci predikát z Pr;
      vlož upravený predpoklad do množiny Pred;
    END;
    vlož predpoklady z Pred ako konjunkciu do predpokladov akcie A;
    medzi predpoklady A pridaj vykonateľný(P);

    V := vykonaj(P)
    FOR v TO V DO BEGIN
      vlož V medzi efekty akcie A;
    END;
    owls2pddlAction_AtomicProcessEffects : A;
  END;
-----------------------------------------------------------------------------------------------------
```

Alg. 7 Atomický proces, ktorý obsahuje len výstupy

Predpoklady a efekty môžu byť v OWL-S popisoch reprezentované rôznymi definíciami (kapitola 2.6), napr. najčastejšie pomocou SWRL alebo KIF podmienok. Z toho dôvodu pri definíciách jednotlivých algoritmov na transformovanie OWL-S procesov do PDDL nešpecifikujeme popis podmienok, ktoré definujú efekty a predpoklady procesov.

Predpokladajme však, že pre definíciu predpokladov a efektov sa bude využívať jazyk SWRL (Semantic Web Rule Language) [12]. SWRL je jazyk založený na kombinácii OWL jazyka a jazyka RuleML[21] (Rule Markup Language). Jadro RukeML jazyka umožňuje vytvárať ľubovoľné n-áre relácie. To znamená, že počet argumentov relácie nie je obmedzený. V prípade SWRL sa však jedná o podmnožinu RuleML, ktorá

[21] The Rule Markup Initiative - http://ruleml.org/

umožňuje definovať iba unárne a binárne relácie pre kombináciu s OWL. Jazyk SWRL umožňuje (okrem iného) vytvárať tvrdenia o vstupoch a výstupoch služieb, ktoré dané procesy obsahujú. Tieto tvrdenia potom môžu byť použité pre definovanie predpokladov a efektov pri PDDL akciách. SWRL podmienky sa skladajú z atómov. Atómy môžu byť vytvorené pomocou unárnych predikátov, binárných predikátov, rovnosti alebo rozdielnosti. Pri definícii predpokladov a efektov sa bude predpokladať, že v SWRL podmienkach sa zatiaľ nachádzajú len nasledovné atómy:

- unárne atómy – reprezentujúce triedy (`classAtom`)
- binárne atómy – reprezentujúce vlastnosti (`individualPropertyAtom`, `datavaluedPropertyAtom`)
- buildin atómy – v práci sa využívajú atómy pre definovanie pravdivostných hodnôt (`booleanNot`)

Predpokladáme, že pomocou SWRL podmienok ideme definovanie predikáty a efekty mapovaním na odpovedajúce predikáty, ktoré sme získali z OWL ontológie (kap. 6.1.4). Definície SWRL podmienok si preto obmedzíme nasledovne:

- každá SWRL podmienka definuje jednu atomickú formulu
- daná atomická formula môže byť negatívna alebo pozitívna. Preto sa v SWRL podmienke nachádza maximálne jeden buildin atóm (`not`), ktorý nám definuje negatívnu atomickú formulu (literál)
- v prípade ak medzi atómami zo SWRL podmienky sa bude nachádzať len unárny atóm reprezentujúci triedu, predpokladáme, že daná trieda v argumente je namapovateľná na predikát, ktorý vznikol z faktu o príslušnosti ku triede (viď kap. 6.1.4)

Algoritmy Alg. 8 a Alg. 9 definujú funkcie pre tvorbu predpokladov a efektov PDDL akcie. Obidve funkcie využívajú funkciu **`swrl2atomicFormula`** (Alg. 10), ktorá spracúva jednu SWRL podmienku a vracia atomickú formulu.

```
FUNCTION swrl2pddlPreconditions(P : OWL-S Atomický Proces,
                                bO: OWL Doménová Ontológia): PDDL GD
  VAR:
    predpoklady : PDDL GD;
    pred : PDDL predikáty;
  BEGIN
    pred := owl2pddlPredicates(bO);
```

```
    FOR p TO všetky predpoklady procesu P DO BEGIN
      pridaj medzi predpoklady := swrl2atomicFormula(p, pred);
    END;
    swrl2pddlPreconditions : predpoklady;
  END;
```

Alg. 8 Tvorba PDDL predpokladov akcie zo SWRL podmienok

```
FUNCTION swrl2pddlEffects(P : OWL-S Atomický Proces,
                          bO: ontológia) : PDDL efekt
  VAR:
    efekty : PDDL efekt;
    pred : PDDL predikáty;
  BEGIN
    pred := owl2pddlPredicates(bO);
    FOR e TO všetky efekty procesu P DO BEGIN
      pridaj medzi efekty := swrl2atomicFormula(e, pred);
    END;
    swrl2pddlEffects : efekty;
  END;
```

Alg. 9 Tvorba PDDL efektov akcie zo SWRL podmienok

```
FUNCTION swrl2atomicFormula (S : SWRL Podmienka,
                  P : PDDL Predikáty) : PDDL Atomická Formula
  VAR:
    CA : množina atómov tried;
    PA : množina atómov vlastností;
    BA : množina vstavaných atómov (not);
    af : atomická formula;
  BEGIN
    FOR a TO všetky atómy z podmienky S DO BEGIN
      IF (a je atóm pre triedu) DO BEGIN
        do CA pridaj a;
      END;
      IF (a je atóm pre vlastnosť) DO BEGIN
        do PA pridaj a;
      END;
      IF (a je atóm pre not) DO BEGIN
        do BA pridaj a;
      END;
    END;
    IF (množina PA je prázdna) DO BEGIN
      namapuj triedu z CA na odpovedajúci predikáty z P;
```

```
      skontroluj, či názov danej triedy nie je v množine BA a nie je
      definovaná negatívna atom. formula pomocou not;
      vytvor atomickú formulu af;
    ELSE BEGIN
      vytvor z atómov z PA a CA atomickú formulu a namapuj ju na
      odpovedajúci predikát z P;
      skontroluj, či názov triedy reprezentujúcej doménu vlastnosti
      nie je v množine BA a nie je definovaná negatívna atomická
      formula pomocou not;
      vytvor atomickú formulu af;
    END;
    swrl2gd : af;
  END;
```

Alg. 10 Tvorba PDDL atomických formúl zo SWRL podmienok

Na príklade Pr. 7 je zobrazený príklad SWRL podmienky definovanej v OWL-S popise služby. Podmienka popisuje, že určitá vec (**`?Thing`**) sa musí nachádzať na danom mieste (**`?FromPlace`**). Po transformovaní SWRL podmienky do PDDL predpokladu nám pribudne pre akciu nasledovný predpoklad **`(at ?FromPlace ?Thing)`**.

SWRL podmienka použitá v OWL-S popise služby (priloha A):

```
<expr:SWRL-Condition rdf:ID="PrecAt">
     <expr:expressionObject>
         <swrl:AtomList>
             <rdf:first>
                 <swrl:ClassAtom>
                     <swrl:classPredicate rdf:resource="&travel;#at"/>
                     <swrl:argument1>
                         <swrl:Variable rdf:ID="a1"/>
                     </swrl:argument1>
                 </swrl:ClassAtom>
             </rdf:first>
             <rdf:rest>
                 <swrl:AtomList>
                     <rdf:first>
                         <swrl:IndividualPropertyAtom>
                             <swrl:propertyPredicate
                                         rdf:resource="&travel;#has_thing"/>
                             <swrl:argument1 rdf:resource="#a1"/>
                             <swrl:argument2 rdf:resource="#Thing"/>
                         </swrl:IndividualPropertyAtom>
                     </rdf:first>
                     <rdf:rest>
                         <swrl:AtomList>
                             <rdf:first>
                                 <swrl:IndividualPropertyAtom>
                                     <swrl:propertyPredicate
                                                 rdf:resource="&travel;#has_place"/>
                                     <swrl:argument1 rdf:resource="#a1"/>
                                     <swrl:argument2 rdf:resource="#FromPlace"/>
                                 </swrl:IndividualPropertyAtom>
                             </rdf:first>
                             <rdf:rest rdf:resource="&rdf;#nil"/>
                         </swrl:AtomList>
                     </rdf:rest>
                 </swrl:AtomList>
             </rdf:rest>
         </swrl:AtomList>
     </expr:expressionObject>
```

```
</expr:SWRL-Condition>
```

PDDL predpoklad:

```
:precondition
       (and
               (at ?FromPlace ?Thing)
                    )
```

Pr. 7 Tvorba PDDL predpokladu zo SWRL podmienky

Na príklade Pr. 8 je zobrazený príklad SWRL podmienky definovanej v OWL-S popise služby, ktorá je použitá na definovanie efektu procesu. Podmienka popisuje, že určitá vec (**`?Thing`**) sa po vykonaní akcie nemôže nachádzať na mieste (**`?FromPlace`**). Po transformovaní SWRL podmienky do PDDL efektu nám pribudne pre akciu nasledovný efekt **`(not (at ?FromPlace ?Thing))`**.

OWL-S Result služby (priloha A):

```
<process:Result rdf:ID="EffectAt">
    <process:hasEffect>
        <expr:SWRL-Expression rdf:ID="EffAt2">
            <expr:expressionObject>
                <swrl:AtomList>
                    <rdf:first>
                        <swrl:ClassAtom>
                            <swrl:classPredicate rdf:resource="&travel;#at"/>
                            <swrl:argument1>
                                <swrl:Variable rdf:ID="a3"/>
                            </swrl:argument1>
                        </swrl:ClassAtom>
                    </rdf:first>
                    <rdf:rest>
                        <swrl:AtomList>
                            <rdf:first>
                                <swrl:IndividualPropertyAtom>
                                    <swrl:propertyPredicate
                                                rdf:resource="&travel;#has_thing"/>
                                    <swrl:argument1 rdf:resource="#a3"/>
                                    <swrl:argument2 rdf:resource="#Thing"/>
                                </swrl:IndividualPropertyAtom>
                            </rdf:first>
                            <rdf:rest>
                                <swrl:AtomList>
                                    <rdf:first>
                                        <swrl:IndividualPropertyAtom>
                                            <swrl:propertyPredicate
                                                rdf:resource="&travel;#has_place"/>
                                            <swrl:argument1 rdf:resource="#a3"/>
                                            <swrl:argument2
                                                rdf:resource="#FromPlace"/>
                                        </swrl:IndividualPropertyAtom>
                                    </rdf:first>
                                    <rdf:rest>
                                        <swrl:AtomList>
                                            <rdf:first>
                                                <swrl:BuiltinAtom>
                                                    <swrl:builtin
                                     rdf:resource="&swrlb;#booleanNot"/>
                                                    <swrl:arguments>
                                                        <rdf:List>
                                                            <rdf:first
                                                               rdf:resource="#a3"/>
                                                            <rdf:rest
                                                        rdf:resource="&rdf;#nil"/>
```

```
                                                                    </rdf:List>
                                                                </swrl:arguments>
                                                            </swrl:BuiltinAtom>
                                                        </rdf:first>
                                                        <rdf:rest rdf:resource="&rdf;#nil"/>
                                                    </swrl:AtomList>
                                                </rdf:rest>
                                            </swrl:AtomList>
                                        </rdf:rest>
                                    </swrl:AtomList>
                                </rdf:rest>
                            </swrl:AtomList>
                        </expr:expressionObject>
                    </expr:SWRL-Expression>
                </process:hasEffect>
            </process:Result>
```

PDDL efekt:

```
:effect
       (and
              (not
                     (at ?FromPlace ?Thing)
                     )
              )
```

Pr. 8 Tvorba PDDL efektu z popisu efektu služby pomocou SWRL podmienky

Príklad Pr. 9 prezentuje tvorbu PDDL akcie z OWL-S atomického procesu obsahujúceho predpoklady a efekty. Na tvorbu akcie je použitá funkcia definovaná v Alg. 6, ktorá používa funkcie definované v Alg. 8 a Alg. 9 pre tvorbu PDDL predpokladov a efektov akcie.

Fragment OWL-S popisu služby (+ Pr. 7 a Pr. 8 uvedené vyššie):

```
<!-- Process description -->
<process:AtomicProcess rdf:ID="DriveProcess">
    <service:describes rdf:resource="#DriveService"/>
    <process:hasInput rdf:resource="#Thing"/>
    <process:hasInput rdf:resource="#FromPlace"/>
    <process:hasInput rdf:resource="#ToPlace"/>

    <process:hasPrecondition rdf:resource="#PrecAt"/>
    <process:hasResult rdf:resource="#EffectAt"/>
</process:AtomicProcess>

<process:Input rdf:ID="Thing">
    <process:parameterType rdf:datatype="&xsd;#anyURI">
           &travel;#thing
    </process:parameterType>
    <rdfs:label>osoba alebo vozidlo, ktore chceme premiestnit</rdfs:label>
</process:Input>
<process:Input rdf:ID="FromPlace">
    <process:parameterType rdf:datatype="&xsd;#anyURI">
           &travel;#place
     </process:parameterType>
    <rdfs:label>miesto, z ktoreho vychadzame</rdfs:label>
</process:Input>
<process:Input rdf:ID="ToPlace">
    <process:parameterType rdf:datatype="&xsd;#anyURI">
           &travel;#place
    </process:parameterType>
    <rdfs:label>miesto, na ktore chceme premiestnit</rdfs:label>
</process:Input>
```

PDDL jednoduchá akcia:

```
(:action Driving
       :parameters
              ( ?Thing - thing  ?FromPlace ?ToPlace - place)
       :precondition
              (and
                     (at ?FromPlace ?Thing)
                     )
       :effect
              (and
                     (not
                            (at ?FromPlace ?Thing)
                            )
                     )
)
```

Pr. 9 Tvorba PDDL akcie z OWL-S atomického procesu

6.1.5.2 Jednoduchý proces

Jednoduché procesy nie sú priamo vykonateľné a nie sú ani prepojené s groudingom služby. Podobne ako atomické procesy sú však chápané ako jednokrokové procesy. Jednoduché procesy slúžia na abstraktné popísanie atomických a zložených procesov. Na rozdiel od atomických procesov môžeme pri jednoduchých procesoch vidieť ich vnútornú štruktúru. Samotný proces je z tohto dôvodu zložitejší a nie je možné predpokladať, že ho pôjde priamo transformovať na jednoduchú PDDL akciu ako to bolo možné pri atomických procesoch. Daný jednoduchý proces môže abstraktne popisovať viacero atomických alebo zložených procesov. Preto je potrebné do tela PDDL akcie pridať ďalšie elementy, pomocou ktorých budeme schopný vytvoriť PDDL akciu z jednoduchého OWL-S procesu (aj zo zloženého, kap. 6.1.5.3):

<action-def body>::=[:precondition <GD>]
[:effect <effect>]
[:expansion <action spec>]`:action-expansions`
[:expansion :methods] `:action-expansions`
[:maintain <GD>]`:action-expansions`
[:only in-expansion <boolean] `:action-expansions`

Okrem predpokladov a efektov pribudli syntaktické elementy súvisiace s požiadavkou domény **`:action-expansions`**. Expanzia akcie (angl. action expansion) určuje spôsob, akým môže byť akcia rozložená do jednoduchších blokov (obyčajne predstavujúcich priamo atomické akcie, popr. jednoduchšie zložené akcie). Treba poznamenať, že použitie efektov akcie a expanzie akcie naraz nie je povolené. Je to dané tým, že v prípade použitia expanzie akcie sa predpokladá, že akcia nie je

primitívna a jej vykonanie spočíva vo vykonaní primitívnych akcií, ktoré obsahuje. Čiže definícia efektov pre takúto akciu nie je možná.

V prípade atomických akcií sa uvažovala akcia, ktorá sa skladala z troch hlavných častí: parametre, predpoklady a efekty. V prípade jednoduchých a zložených akcií pribudne ďalšia časť, konkrétne časť expanzie. Expanzia informuje o tom, akým spôsobom je daný proces vytvorený (zložený) z menších podprocesov.

V prípade transformovania jednoduchého OWL-S procesu do PDDL akcie je možné postupovať podľa funkcie uvedenej nižšie (Alg. 11).

```
FUNCTION owls2pddlAction_SimpleProcess(P :
                                OWL-S Jednoduchý Proces) : PDDL Akcia
  VAR
    A : PDDL akcia;
    Pr : Predikáty z PDDL domény;
    Pred : Predpoklady;
    Z : Zoznam Procesov;
  BEGIN
    názov akcie A := názov procesu P;
    FOR i TO všetky vstupy procesu P DO BEGIN
      pridaj i so svojim typom medzi parametre A;
    END;
    usporiadaj parametre akcie A podľa typov;

    FOR p TO všetky predpoklady procesu P DO BEGIN
      namapuj daný predpoklad p na odpovedajúci predikát z Pr;
      vlož upravený predpoklad p do množiny Pred;
    END;
    vlož predpoklady z Pred ako konjunkciu do predpokladov akcie A;
    efekty akcie A := prázdna množina;

    Z := [p1, p2, ..., pn] - množina atomických a zložených procesov,
      ktoré abstraktne popisuje proces P;

    expanzia akcie A je typu choise (PDDL);
    FOR p TO Z DO BEGIN
      p' := (in-context p : precondition (vykonateľnosť(p)));
      vlož do časti expanzie akcie A p';
    END;
    owls2pddlAction_AtomicProcessOutputs : A;
  END;
```

Alg. 11 Transformácia jednoduchého procesu na PDDL akciu

V programe použitý „in-context“ syntaktický element má nasledujúcu formu:

```
(in-context <action spec>
            :precondition P
            :maintain M)
```

Súvisí s definíciou predpokladov a obsluhujúcich podmienok akcií, pričom tieto definície platia pre akciu len v kontexte expanzie. Nesúvisia so samotnou definíciou akcie (napr. primitívnej akcie). Funkcia `vykonateľnosť(p)` vracia hodnotu pravda, v prípade ak proces p je vykonateľný, inak vracia nepravda. Daná funkcia sa pridáva medzi predpoklady.

6.1.5.3 Zložený proces

Zložený proces je proces skladajúci sa z podprocesov. Dekompozícia zložených procesov do menších celkov sa uskutočňuje pomocou konštruktov. Vykonanie zložených procesov spočíva vo vykonaní ich častí, čiže v konečnom dôsledku v exekúcii atomických procesov.

OWL-S umožňuje použitie nasledovných konštruktov (kapitola 2.6):

1) postupnosť (**sequence**) – zoznam procesov, ktoré sa môžu vykonávať v danom poradí.
2) delenie (**split**) – skladá sa z viacerých procesov, ktoré sa môžu vykonávať paralelne.
3) delenie+spájanie (**split+join**) – spočíva v rozdelení procesov do skupín a v paralelnom vykonávaní týchto skupín.
4) ľubovoľné poradie (**any-order**) – umožňuje vykonávať procesy v nešpecifikovanom poradí, nie však paralelne.
5) výber (**choise**) – vyberá sa určitý proces s množiny procesov, ktorý môže byť vykonaný.
6) ak-potom-inak (**if-then-else**) – testuje sa podmienka, ktorá keď je splnená nastáva vetva potom (**then**), keď nie je splnená nastáva vetva inak (**else**).
7) iterácie (**iterate**) – je abstraktná trieda, čo znamená že sama o sebe nemôže byť inštanciovaná. Slúži ako supertrieda pre definovanie cyklov z bodu 8.
8) opakuj – až do (**repeat-until**) a opakuj pokiaľ (**repeat-while**) – reprezentujú cykly, ktoré sa opakujú pokiaľ je splnená daná podmienka.

```
-----------------------------------------------------------------------------------------------------

FUNCTION owls2pddlAction_CompositeProcess(P :
                  OWL-S Zložený Proces) : PDDL Akcia
  VAR
    A : pddl akcia;
    Pr : predikáty z PDDL domény;
    Pred : predpoklady;
  BEGIN
    názov akcie A := názov procesu P;
    FOR i TO všetky vstupy P DO BEGIN
      pridaj i so svojim typom medzi parametre A;
    END;
    usporiadaj parametre A podľa typov;

    FOR p TO všetky predpoklady P DO BEGIN
      namapuj daný predpoklad p na odpovedajúci predikát z Pr;
      vlož upravený predpoklad do množiny Pred;
    END;
    vlož predpoklady z Pred ako konjunkciu do predpokladov  akcie A;

    efekty akcie A := prázdna množina;

    IF (proces má konštrukt postupnosť) DO BEGIN
      expanzia A := constructSequence(P);
    END;
    IF (proces má konštrukt delenie) DO BEGIN
      expanzia A := constructSplit(P);
    END;
    IF (proces má konštrukt výber) DO BEGIN
      expanzia A := constructChoice(P);
    END;
    IF (proces má konštrukt ak-potom-inak) DO BEGIN
      expanzia A := constructIfThenElse(P);
    END;
    owls2pddlAction_AtomicProcessOutputs : A;
  END;
-----------------------------------------------------------------------------------------------------
```

Alg. 12 Transformácia zloženého procesu na PDDL akciu

Zložené procesy sa transformujú do zložených akcií v PDDL. Zložené akcie popri parametroch, predpokladoch a efektoch obsahujú aj definície expanzie danej akcie (viď. kapitola 6.1.5.2). Na základe konštruktov použitých pre definovanie zloženého procesu v OWL je vytváraný výraz pre expanziu zloženej PDDL akcie.

V prípade PDDL akcií nie je možné definovať iterácie. Z toho dôvodu cykly definované v OWL-S nie je možné transformovať ako expanzie akcií v PDDL (konštrukty 7 a 8). Podobne PDDL priamo nepodporuje konštrukty pre ľubovoľné poradie (any-order) a delenie+spájanie (split+join). Tieto konštrukty však je možné vyjadriť pomocou dostupných konštruktov v PDDL. Ďalej sú bližšie popísané konštrukty, ktoré môžu byť priamo transformované do PDDL – postupnosť, delenie, výber a ak-potom-inak.

Transformácie parametrov a predpokladov akcie bude pre každý zložený proces rovnaká. Preto je možné algoritmus pre transformáciu zloženej akcie zapísať tak, ako je to uvedené vyššie (Alg. 12).

6.1.5.3.1 Postupnosť – sequence

```
-----------------------------------------------------------------------------------------------------
FUNCTION constructSequence(P : OWL-S Zložený Proces)
                                        : PDDL action-expansion
  VAR
    exp : výraz expanzie PDDL akcie;
    Z : zoznam procesov;
  BEGIN
    konštruktor expanzie PDDL akcie := series;
    Z := [p1, p2, ..., pn] - množina atomických a zložených procesov,
      ktoré abstraktne popisuje proces P;

    FOR p TO Z DO BEGIN
      p‘:=(in-context p: precondition (vykonateľnosť(p)));
      vlož do exp p’;
    END;
    constructSequence : exp;
  END;
-----------------------------------------------------------------------------------------------------
```

Alg. 13 Transformácia zloženého procesu - postupnosť

6.1.5.3.2 Delenie – split

```
-----------------------------------------------------------------------------------------------------
FUNCTION constructSplit(P : OWL-S Zložený Proces)
                                        : PDDL action-expansion
  VAR
    exp : výraz expanzie pddl akcie;
    Z : zoznam procesov;
  BEGIN
    konštruktor expanzie := parallel;
```

```
    Z := [p1, p2, ..., pn] - množina atomických a zložených procesov,
      ktoré abstraktne popisuje proces P, ktoré sú vykonávané
      paralelne;
    FOR p TO Z DO BEGIN
      vlož do exp p;
    END;
    constructSequence := exp;
  END;
```

Alg. 14 Transformácia zloženého procesu - delenie

V prípade paralelného vykonávania procesov jednotlivé procesy od seba nezávisia. Preto nie je potreba osobitne upravovať tieto procesy ako tomu bolo pri sekvenčnom spracovaní, kde bolo nutné kontrolovať, či daný proces môže byť vykonaný.

6.1.5.3.3 Výber – choice

```
FUNCTION constructChoice(P : OWL-S Zložený Proces)
                                    : PDDL action-expansion
  VAR
    exp : výraz expanzie pddl akcie;
    Z : zoznam procesov;
  BEGIN
    konštruktor expanzie := choice;

    Z := [p1, p2, ..., pn] - množina atomických a zložených procesov,
      ktoré abstraktne popisuje proces P, ktoré reprezentujú množinu;

    FOR p TO Z DO BEGIN
      vlož do exp p;
    END;
    constructSequence := exp;
  END;
```

Alg. 15 Transformácia zloženého procesu - voľba

6.1.5.3.4 Ak-potom-inak

```
FUNCTION constructIfThenElse(P : OWL-S Zložený Proces)
                                    : PDDL action-expansion
  VAR
    exp : výraz expanzie pddl akcie;
    Z : zoznam procesov;
  BEGIN
```

```
    konštruktor expanzie := series;
    Z := [p1, p2] - množina atomických alebo zložených procesov, ktoré
      abstraktne popisuje proces P;
    exp := (series (in-context p1 : precondition podmienkaif)
                   (in-context p2 : precondition (not podmienkaif)))

            /* (in-context action precondition pre) je v PDDL
             * definovanie  akcie, ktorá môže byť vykonateľná, ak je
             * pre splnené
             */
    constructSequence := exp;
  END;
```

Alg. 16 Transformácia zloženého procesu – ak-potom-inak

6.2 Tvorba PDDL plánovacieho problému

Kompletná štruktúra PDDL plánovacieho problému je definovaná v kapitole 4. V prípade plánovania pre kompozíciu webových služieb je možné štruktúru zjednodušiť. Zjednodušená EBNF forma štruktúry PDDL plánovacieho je uvedená nižšie:

```
<problem> ::= (define (problem <name>)
                      (:domain <name>)
                      [<require-def>]
                      [<object declaration>]
                      [<init>]
                      <goal>+)
```

Štruktúra PDDL problému v tomto prípade pozostáva z nasledovných definícií:

- meno problému,
- meno domény, na ktoré sa daný problém viaže,
- zoznam požiadaviek problému,
- definícia objektov problému,
- počiatočný stav,
- koncový stav.

Tvorba každej jednej časti je popísaná v osobitnej podkapitole.

6.2.1 PDDL Problém – meno problému a meno domény

Na začiatku tvorby PDDL plánovacieho problému je potrebné definovať meno problému a meno domény. Tieto je možné vygenerovať z počiatočného stavu domény. Počiatočný stav je reprezentovaný ontológiou, ktorá vznikla z pôvodnej (hlavnej)

ontológie vytvorením inštancií tried a ich popísaním pomocou objektových a dátových vlastností. Predpokladajme, že názov tejto ontológie ostal zachovaný z pôvodnej. Získaním tohto názvu dostávame názov domény (viď. kapitola 6.1.1 - napr. **`travel`**).

Pre získanie mena problému je možné jednoducho pridať ku názvu domény reťazec „Problem“ (napr. **`travelProblem`**). Je možné si však predstaviť aj veľa iných spôsobov, napr. definícia oboch mien priamo v hlavičke pomocou importovania určitej ontológie a použitie vhodných konceptov na tieto popisy a podobne.

6.2.2 PDDL Problém – požiadavky problému

Definovanie požiadaviek problému sa robí analogicky ako definovanie požiadaviek v doméne (viď kapitolu 6.1.2).

6.2.3 PDDL Problém – objekty

Objekty v PDDL doméne sú konkrétne objekty, ktoré je možné dosadiť za parametre predikátov a vytvárať tak fakty v plánovacej doméne. Napr. majme objekty **`Zvolen`** a **`Zilina`** typu **`mesto`** a v doméne definovaný predikát **`(cesta ?X ?Y - mesto)`**, ktorý vyjadruje informáciu, že z mesta X existuje cesta do mesta Y. Vytvorenie konkrétneho faktu pomocou existujúcich dvoch objektov je nasledovné: **`(cesta Zvolen Zilina)`**.

EBNF forma pre definovanie objektov v plánovacom probléme má tvar:

`<object declaration> ::= (:objects <typed list (name)>)`

Ide o zoznam, kde jednotlivé prvky zoznamu sú reťazce **`name`** (musia spĺňať určité podmienky, viď. kapitola 6.1.1). V prípade ak je daná aj požiadavka **`:typing`**, je potrebné aby každý prvok zoznamu mal jednoznačne pridelený typ. V prípade práce s ontológiami je táto požiadavka potrebná. Napr.

`(:objects Zilina Kosice Zvolen – mesto Peter Pavol - osoba)`

Pri tvorbe objektov v plánovacom probléme sa využívajú z OWL ontológií inštancie tried a vlastnosti s nimi súvisiace. Ako už bolo spomínané, objekty slúžia na vytváranie konkrétnych faktov v plánovacej domény, pričom ich dosádzame za parametre do predikátov. Preto v prípade ontológií nás budú zaujímať len tie inštancie, ktoré spĺňajú špecifickú podmienku, že sú inštanciami takých tried, ktoré tvoria **rozsah** určitej vlastnosti. Z definície predikátov je vidieť (viď. kapitolu 6.1.4), že objekty vytvorené z takýchto inštancií môžeme dosadiť za parametre predikátov. Na tvorbu týchto objektov bolo potrebné využiť objektové vlastnosti z ontológie. Okrem

týchto vlastností ontológie obsahujú aj dátové vlastnosti, ktoré sa tiež využívali na definíciu PDDL predikátov. Rozsah dátových vlastností je tvorený určitým dátovým typov z XML schémy. Preto aj každá hodnota určitej dátovej vlastnosti musí byť považovaná za objekt v PDDL plánovacej doméne.

Algoritmus pre získavanie objektov z OWL ontológií je popísaná nižšie (Alg. 17).

```
FUNCTION owl2pddlObjects(O : OWL Ontológia): PDDL Objekty
  VAR
    OV : Množina konkrétnych objektových vlastností z O;
    DV : Množina konkrétnych dátových vlastností z O;
    MO : Množina PDDL objektov;
  BEGIN
    FOR v TO OV DO BEGIN
      VAR
        i : Inštancia OWL triedy (OWL Individual);
      i := hodnota rozsahu vlastnosti v;
      IF (MO neobsahuje i) THEN BEGIN
        vlož i to MO aj s typom (OWL trieda);
      END;
    END;
    FOR v TO DV DO BEGIN
      VAR
        i : dátová premenná;
      i := hodnota rozsahu vlastnosti v;
      IF (MO neobsahuje i) THEN BEGIN
        vlož i do MO aj s typom (dátový typ);
      END;
    END;
    usporiadaj MO podľa typov;
    owl2pddlObjetcs : MO;
  END;
```

Alg. 17 Tvorba PDDL objektov z OWL ontológie

Na Pr. 10 je zobrazená definícia PDDL objektov, ktoré vznikli z OWL ontológie.

Fragment OWL ontológie:

```
<!-- Individuals -->
<owl:NamedIndividual rdf:about="&travel;#a">
    <rdf:type rdf:resource="&travel;#place"/>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="&travel;#a1">
    <rdf:type rdf:resource="&travel;#at"/>
    <has_place rdf:resource="&travel;#a"/>
    <has_thing rdf:resource="&travel;#jack"/>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="&travel;#a2">
```

```
        <rdf:type rdf:resource="&travel;#at"/>
        <has_thing rdf:resource="&travel;#bulldozer"/>
        <has_place rdf:resource="&travel;#e"/>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="&travel;#b">
        <rdf:type rdf:resource="&travel;#place"/>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="&travel;#bulldozer">
        <rdf:type rdf:resource="&travel;#vehicle"/>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="&travel;#c">
        <rdf:type rdf:resource="&travel;#place"/>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="&travel;#e">
        <rdf:type rdf:resource="&travel;#place"/>
    </owl:NamedIndividual>
    <owl:NamedIndividual rdf:about="&travel;#jack">
        <rdf:type rdf:resource="&travel;#person"/>
    </owl:NamedIndividual>
```

PDDL objekty:

```
(:objects
         a e b c - place
         bulldozer - vehicle
         jack - person
             )
```

Pr. 10 Tvorba PDDL objektov z OWL inštancií

6.2.4 PDDL Problém – počiatočný stav

EBNF forma počiatočného stavu PDDL plánovacej úlohy má tvar:

<init> ::= (:init <literal (name)>+)

Z EBNF formy je vidieť, že počiatočný stav v PDDL probléme je reprezentovaný množinou literálov, čiže buď pozitívnymi alebo negatívnymi atomickými formulami. V kapitole 6.1.4 pri popise predikátov bolo uvedené, že predikáty sú v PDDL reprezentované kostrami atomických formul. Pri tvorbe atomickej formule preto ide o namapovanie konkrétnych objektov za premenné kostry atomickej formule. Preto je v prípade tvorby počiatočného potrebné poznať predikáty z PDDL domény (kapitola 6.1.4) a objekty z PDDL problému (kapitola 6.2.3).

Pri tvorbe PDDL počiatočného stavu sa bude vychádzať z OWL individualít (inštancií) nachádzajúcich sa v OWL ontológii. Budú sa prechádzať všetky konkrétne objektové a dátové vlastnosti, pričom sa budú grupovať tie, ktoré majú rovnakú inštanciu v doméne vlastnosti. Takto získané množiny sa potom mapujú na konkrétne predikáty. Na záver ešte treba spracovať tie inštancie, ktoré sa nenachádzajú v doméne žiadnej vlastnosti. Pre tieto inštancie sú vytvorené špeciálne predikáty (viď kapitolu 6.1.4, napr. Pr. 6 a Pr. 10). Algoritmus tvorby PDDL počiatočného stavu z OWL ontológie je zobrazený na Alg. 18.

```
FUNCTION owl2pddlInitState(O : OWL Ontológia): PDDL Počiatočný Stav
  VAR
    OV : Množina konkrétnych objektových vlastností;
    DV : Množina konkrétnych dátových vlastností;
    IS : Množina literálov – počiatočný stav;
    I : Množina inštancií;
    I1: Množina inštancií;
    PR : Množina predikátov;
  BEGIN
    PR := owl2pddlPredicates(O);
    FOR v TO OV DO BEGIN
      VAR
        i : Inštancia OWL triedy;
      i := hodnota domény vlastnosti v;
      IF (I neobsahuje i) THEN BEGIN
        vlož i to I;
      END;
    END;
    FOR v TO DV DO BEGIN
      VAR
        i : inštancia triedy;
      i := hodnota domény vlastnosti v;
      IF (I neobsahuje i) THEN BEGIN
        vlož i to I;
      END;
    END;
    FOR i TO I DO BEGIN
      VAR
        a : PDDL Atomicka formula;
        l : Literál;
      a.predicate := typ inštancie i;
      a.zoznam premenných := pre každé i získaj všetky vlastnosti,
      v ktorých sa i nachádza v doméne;
      FOR p TO PR DO BEGIN
        a namapujeme na príslušný predikát;
      END;
      l := a;
      l vložíme do IS;
    END;
    I1 je množina inštancii, ktoré sa nenachádzajú v doméne žiadnej
      vlastnosti;
    FOR i TO I1 DO BEGIN
      FOR p TO PR DO BEGIN
        i namapujeme na príslušný predikát;
```

```
      END;
      l := a;
      l vložíme do IS;
    END;
    owl2pddlInitState : IS;
  END;
```

Alg. 18 Definovanie algoritmu pre tvorbu počiatočného stavu z OWL ontológie

Na Pr. 11 je zobrazený príklad tvorby počiatočného stavu plánovacej úlohy. Fragment OWL ontológie, ktorého využitie je pri tejto transformácii rozhodujúcej je podobný aký bol zobrazený na príklad Pr. 10. Výsledkom danej transformácie je časť PDDL problému, ktorá reprezentuje počiatočný stav (množine literálov).

Fragment OWL ontológie – podobný ako v Pr. 10.

PDDL počiatočný stav:

```
(:init
        (at e bulldozer)
        (place a)
        (at a jack)
        (vehicle bulldozer)
        (place e)
        (place b)
        (place c)
        (person jack)
       )
```

Pr. 11 Tvorba PDDL počiatočného stavu z OWL ontológie

6.2.5 PDDL Problém – koncový stav

EBNF forma cieľového stavu PDDL plánovacej úlohy má tvar:

<goal> ::= (:goal <GD>)

Jeden zo spôsobov tvorby koncového stavu je jeho tvorba pomocou dvoch ontológií: ontológia počiatočného stavu (OPS) a ontológia koncového stavu (OKS) (Alg. 19). OPS bola používaná v predchádzajúcej kapitole. OKS bude vychádzať z ontológie OPS. Je však potrebné zmeniť v nej definované fakty, ktoré majú byť cieľom plánu. Tvorba cieľového PDDL stavu potom bude záležať od tých faktov, ktoré sú v OKS nové.

```
FUNCTION owl2pddlGoalState(OInit, OGoal : OWL Ontológie)
                                    : PDDL Cieľový Stav
  VAR
```

```
    IS : Množina literálov – počiatočný stav;
    GS : Množina literálov – koncový stav;
    G : PDDL koncový stav;
    X : Pomocná množina literálov;
  BEGIN
    IS := owl2pddlInitState(OInit);
    GS := owl2pddlGoalState(OGoal);

    X := literály z GS, ktoré sa nenachádzajú v IS;

    G := konjunkcia literálov z množiny X;
    owl2pddlGoalState : G;
  END;
```

Alg. 19 Definovanie algoritmu pre tvorbu koncového stavu z OWL ontológie

Oproti príkladu Pr. 11 boli v ontológii zmenené nasledovné fakty:

- bullozer sa nachádza na mieste a
- jack sa nachádza na mieste e

Tieto fakty boli porovnaním počiatočnej a koncovej ontológie použité na vytvorenie koncového stavu PDDL plánovacieho problému (Pr. 12).

Fragment OWL ontológie:

```
<!-- Individuals -->
 <owl:NamedIndividual rdf:about="&travel;#a">
     <rdf:type rdf:resource="&travel;#place"/>
 </owl:NamedIndividual>
 <owl:NamedIndividual rdf:about="&travel;#a1">
     <rdf:type rdf:resource="&travel;#at"/>
     <has_place rdf:resource="&travel;#e"/>
     <has_thing rdf:resource="&travel;#jack"/>
 </owl:NamedIndividual>
 <owl:NamedIndividual rdf:about="&travel;#a2">
     <rdf:type rdf:resource="&travel;#at"/>
     <has_thing rdf:resource="&travel;#bulldozer"/>
     <has_place rdf:resource="&travel;#a"/>
 </owl:NamedIndividual>
 <owl:NamedIndividual rdf:about="&travel;#b">
     <rdf:type rdf:resource="&travel;#place"/>
 </owl:NamedIndividual>
 <owl:NamedIndividual rdf:about="&travel;#bulldozer">
     <rdf:type rdf:resource="&travel;#vehicle"/>
 </owl:NamedIndividual>
 <owl:NamedIndividual rdf:about="&travel;#c">
     <rdf:type rdf:resource="&travel;#place"/>
 </owl:NamedIndividual>
 <owl:NamedIndividual rdf:about="&travel;#e">
     <rdf:type rdf:resource="&travel;#place"/>
 </owl:NamedIndividual>
 <owl:NamedIndividual rdf:about="&travel;#jack">
     <rdf:type rdf:resource="&travel;#person"/>
 </owl:NamedIndividual>
```

PDDL koncový stav:

```
(:goal
```

```
(and
        (at a bulldozer)
        (at e jack)
        )
)
```

Pr. 12 Tvorba PDDL koncového stavu z OWL ontológie

Iný spôsob predstavuje tvorba cieľovej ontológie z hlavnej ontológie, pričom sa priamo definuje koncový stav. Tvorba PDDL koncového stavu z tejto ontológie by potom mohla vychádzať z tvorby počiatočného stavu definovanému v kapitole 6.2.4, s tým rozdielom, že zatiaľ čo v prípade počiatočného stavu je stav definovaný množinou literálov, koncových stav je definovaný popisom cieľa (PDDL element GD). V kapitole 6.1.5 bola prezentovaná EBNF forma GD pre potreby kompozície webových služieb v zjednodušenej orezanej forme. GD tu môže byť reprezentovaný literálmi termov, alebo to môže byť konjunkcia GD. Čiže v konečnom dôsledku je koncový stav reprezentovaný ako konjunkcia literálov.

6.3 Definovanie plánovacej úlohy

Plánovacia úloha v PDDL spočíva v plánovacej doméne a odpovedajúcom plávacom probléme. V predchádzajúcich častiach bolo uvedené všetky potrebné algoritmy pre definovanie plánovacej domény a problému, takže teraz je možné načrtnúť celkový problém pre tvorbu plánovacej úlohy z dostupných ontologických popisov stavov a webových služieb.

Predpokladajme, že je daná OWL ontológia popisujúca určitý problém. Ontológia v sebe zahŕňa triedy, objektové a dátové vlastnosti. Táto ontológia sa bude nazývať základná (doménová) ontológia (ontológia **O** – napr. (príloha A príklad 3.1)). Počiatočný stav bude reprezentovať ontológia počiatočného stavu (**initO -** napr. (príloha A príklad 3.2)). Túto je možné získať vytvorením konkrétnych inštancií tried v doménovej ontológii, spolu s prislúchajúcimi vlastnosťami pre tieto inštancie.

Cieľový stav bude reprezentovať ontológia cieľového stavu (**goalO** - napr. (príloha A príklad 3.1)). Túto je možné získať podobne ako ontológiu počiatočného stavu. Ďalej budú k dispozícii OWL-S ontológie pre popis webových služieb (**$Owls_1$**, **$Owls_2$**, ... , **$Owls_n$**). Algoritmus (Program)(Alg. 20) pre tvorbu PDDL plánovacej úlohy ku svojej činnosti používa funkcie popísané v predchádzajúcich kapitolách.

```
PROGRAM tvorbaPDDLPlanovacejUlohy
```

```
VAR
  O : Hlavná doménová OWL ontológia;
  initO : OWL ontológia počiatočného stavu;
  goalO : OWL ontológia koncového stavu;
  owlS [Owls1, Owls2, ... , Owlsn] : množina OWL-S popisov služieb;
  domena : PDDL domena;
  prob : PDDL problem;
BEGIN
  domena.MenoDomeny := owl2pddlDomainName(O);
  domena.Poziadavky := (:strips, :typing, :action-expansions);
  domena.Typy := owl2pddlTypes(O);
  domena.Predikaty := owl2pddlPredicates(O);

  FOR p TO owlS DO BEGIN
    VAR
      A : PDDL akcia;
    A := owls2pddlActions(p);
    vlož A medzi akcie domena.Akcie;
  END;

  problem.MenoProblemu := owl2pddlProblemName(initO);
  problem.MenoDomeny := owl2pddlDomainName(O);
  problem.Objekty := owl2pddlObjects(initO);
  problem.PociatocnyStav := owl2pddlInitState(initO);
  problem.KoncovyStav := owl2pddlGoalState(initO, goalO);
END;
```

Alg. 20 Definovanie algoritmu pre tvorbu PDDL plánovacej úlohy s využitím sémantických znalostných technológií OWL a OWL-S

Kompletný príklad výslednej PDDL domény a PDDL problému (spolu reprezentujú PDDL plánovaciu úlohu), ktoré vznikli transformáciou z OWL a OWL-S ontológií, je možné vidieť v Prílohe A príklady 5.1 a 5.2.

7 Záver

7.1 Zhrnutie

Hlavným cieľom práce bolo využiť znalostné prístupy pre účely automatickej kompozícii webových služieb. V teoretickej časti boli analyzované technológie a štandardy súvisiace s webovými službami. Medzi tieto štandardy patria aj sémantické štandardy ako ontológie OWL a ontologické popisy webových služieb OWL-S. Ďalej sa teoretická časť práce zaoberala problematikou kompozície webových služieb a je tu popísaná vhodnosť využitia plánovacích metód umelej inteligencie pre potreby automatickej kompozície webových služieb. Na záver teoretickej časti práce je prezentovaných niekoľko existujúcich prístupov a systémov pre kompozíciu webových služieb. V praktickej časti je vzhľadom na poznatky z teoretickej časti navrhnutý systém pre kompozíciu webových služieb, ktorý využíva znalostné prístupy pre definovanie problému kompozície. Nakoniec bolo v praktickej časti navrhnutých viacero algoritmov, ktoré daný prístup využíva.

7.2 Výsledky práce

Predkladaná práca sa zaoberá problematikou kompozície webových služieb. Výsledky práce sú prezentované vzhľadom na ciele, ktoré boli uvedené na začiatku práce (str. 5).

1. V kapitolách 2 a 3 je analyzovaná problematika webových služieb a kompozície webových služieb. V práci sa pracuje so SOAP WS popísanými pomocou WSDL. Okrem spomenutých dvoch technológií sú tu ďalej popísané OWL ontológie, ktoré sú neskôr využívané pri tvorbe problému kompozície WS. V kapitole 3 je načrtnutý veľmi všeobecný model kompozície webových služieb. Model sa skladá z niekoľkých častí, kde za jednu z hlavných častí možno považovať modul tvorby procesov. Daný modul má na starosti generovanie procesov, ktoré sú reprezentované pracovným a dátovým tokom, a ktorých vykonaním je vyriešený problém kompozície.
2. V kapitole 4 je popísaná problematika plánovania metódami umelej inteligencie a vhodnosť využitia metód plánovania na riešenie problému kompozície WS. Je tu zobrazená súvislosť problému kompozície WS

s definovaním plánovacej úlohy. V kapitole 4.2 je predstavených niekoľko hlavných plánovacích metód vhodných na kompozíciu WS. Zo všeobecnej definície problému kompozície je dané, že systém realizujúci kompozíciu WS obyčajne popisuje problém kompozície pomocou dvoch špecifikácií. Externá špecifikácia používa na definovanie problému štandardy webových služieb (ako napr. WSDL, OWL, OWL-S, popr. iné). Interná špecifikácia problému je špecifikácia, s ktorou pracuje generátor procesov (v práci je ním plánovač umelej inteligencie). V práci sa na internú špecifikáciu problému kompozície WS používa jazyk PDDL, ktorý je popísaný v kapitole 4.3. Kapitola 4.5 popisuje niekoľko existujúcich návrhov a systémov pre (semi-)automatickú kompozíciu webových služieb.

3. V praktickej časti (kapitola 5) je predstavený vlastný návrh systému, ktorý pri kompozícii WS vychádza z použitia znalostných prístupov, ako je použitie ontológií OWL a OWL-S. Navrhnutý systém vychádza so štúdia problematiky kompozície webových služieb a z existujúcich systémov pre kompozíciu WS. Pre popis problému kompozície (externá špecifikácia problému) sú zvolené ontológie OWL. Pomocou daných ontológii sa popisujú stavy domény, v ktorej kompozícia prebieha. Webové služby sú popísané pomocou ontológií OWL-S. Pre popis problému kompozície pomocou plánovacej úlohy (interná špecifikácia) je zvolený jazyk PDDL.

4. Kapitola 6 sa zaoberá tvorbou PDDL plánovacej úlohy z ontológií (OWL a OWL-S). Z externej špecifikácie problému kompozície WS určenej pomocou ontológií OWL a OWL-S je potrebné vygenerovať PDDL plánovaciu úlohu, ktorú je možné vyriešiť pomocou zvoleného plánovača umelej inteligencie. Z tohto dôvodu je v práci prezentovaných viacero pôvodných algoritmov, ktoré zabezpečujú tvorbu PDDL (kap. 6.1) plánovacej domény a PDDL plánovacieho problému (kap. 6.2).

7.3 Prínos práce

Z pohľadu celkovej problematiky kompozície webových služieb sa práca zaoberá hlavne špecifikáciou problému kompozície WS. Hlavná časť práce je preto venovaná transformácii sémanticky popísaného problému kompozície webových služieb (pomocou OWL a OWL-S ontológií) do plánovacej úlohy popísanej pomocou PDDL

jazyka. Práca navrhuje využitie ontologického popisu domény, v ktorej sa následne uskutočňuje kompozícia. Ďalej je spracovaná problematika transformácie týchto ontologických popisov spolu s OWL-S popismi služieb do PDDL plánovacej úlohy. Pri OWL-S popisoch služieb sa využívajú SWRL podmienky pre definovanie predpokladov a efektov služieb. Tieto podmienky sa využívajú na tvorbu efektov a predpokladov PDDL akcií. PDDL plánovacia úloha sa skladá z domény a problému. Každá časť obsahuje niekoľko podčastí. Napr. definícia plánovacej domény spočíva v definícii typov, predikátov, akcií a podobne. Plánovací problém obsahuje objekty, definíciu počiatočného a koncového stavu. V práci pre tvorbu každej časti zo sémantického popisu vyčlenená osobitná kapitola.

Medzi hlavné prínosy predkladanej práce patrí komplexný návrh problematiky spracovania externej špecifikácie reprezentovanej pomocou OWL a OWL-S ontológií a internej špecifikácie reprezentovanej pomocou PDDL plánovacej úlohy. Pri návrhu spôsobu transformácie sa autor inšpiroval existujúcimi systémami, predovšetkým prácou [37], v ktorej autori predstavujú algoritmy na tvorbu operácií a metód v SHOP2 doméne pomocou OWL-S popisov, a prácou [52], v ktorej autori nadviazali hlavne na predchádzajúcu prácu a teoreticky spracovali problematiku transformácie OWL-S ontológií na PDDL plánovaciu úlohu. V predkladanej práci sa detailnejšie spracoval spôsob spracovania OWL-S ontológie, ktoré sú používané výlučne na definovanie PDDL akcií. Podobne bol v práci vymedzený jazyk SWRL na definovanie podmienok, ktoré sa v OWL-S popisoch používajú na definovanie predpokladov a efektov. Na definovanie ostatných častí PDDL plánovacej úlohy, ako predikáty, typy, objekty, počiatočný a koncový stav a podobne sa používajú doménové OWL ontológie a ontológie počiatočného a koncového stavu, ktoré vychádzajú z daných doménových ontológií. Pre spracovanie spomenutých ontológií bolo v práci navrhnutých niekoľko pôvodných algoritmov, ktoré spolu s upravenými existujúcimi algoritmami pre spracovanie OWL-S umožňujú tvorbu PDDL plánovacej úlohy.

7.4 Otázky problematiky

V závere práce je vhodné aspoň v stručnosti načrtnúť nevyriešené, resp. nedostatočne spracované problémy, ktorými nebol priestor sa zaoberať v práci a všeobecné problémy súvisiace s problematikou automatickej kompozície WS, ktoré by bolo vhodné spracovať.

Proces transformácie OWL a OWL-S ontológii, ktoré reprezentujú primárnu špecifikáciu problému kompozície, na PDDL plánovaciu úlohu je v práci relatívne detailne spracovaný. Pri PDDL plánovacej doméne a probléme sa však vychádzalo zo zjednodušených štruktúr (viď 6.1 a 6.2), ktoré boli pre danú transformáciu postačujúce. Vynechali sa niektoré časti, ako napr. možnosť definovania konštánt v PDDL doméne, možnosť prepojenia domény na už existujúce domény a iné, ktoré by bolo vhodné v ďalšom vývoji problematiky spracovať. Podobne by bolo veľmi vhodné spracovať problematiku PDDL GD (Goal Description), ktorá sa používa napr. na definovanie PDDL cieľov v probléme, ale aj na definovanie predpokladov pri PDDL akciách. EBND forma pre element GD je nasledovná:

```
<GD> ::= <literal(term)>
<GD> ::= (and <GD>*)
<GD> ::=:disjunctive-preconditions (or <GD>*)
<GD> ::=:disjunctive-preconditions (not <GD>)
<GD> ::=:disjunctive-preconditions (imply <GD> <GD>)
<GD> ::=:existential-preconditions (exists (<typed list(variable)>*) <GD> )
<GD> ::=:universal-preconditions (forall (<typed list(variable)>*) <GD> )
```

Z danej definície PDDL GD je vidieť, že expanzia daného elementu môže prebiehať pomocou viacerých pravidiel. Definície cieľov (GD) v PDDL obsahujú literály, čiže pozitívne alebo negatívne atomické formule. Pri definícii cieľov môžu byť použité kvantifikátory (logické a existenčné). Pre potreby práce sa spracovalo len použitie logického kvantifikátora **`and`** a negácie pomocou **`not`**. Preto pri ďalšom vývoji by bolo vhodné načrtnúť možnosti spracovania ostatných kvantifikátorov pri definícii cieľov.

V prípade OWL-S predpokladov a efektov sa predpokladalo, že sú definované pomocou SWRL podmienok. Okrem SWRL podmienok je v OWL-S popisoch možné použiť aj KIF podmienky. KIF (Knowledge Interchange Format) umožňuje definovať podmienky v jazyku syntakticky veľmi blízkom jazyku PDDL, a preto by bolo veľmi vhodné navrhnúť spôsob spracovania daných podmienok v prípade tvorby PDDL akcie.

V práci bolo spomenuté (kapitola 5.1.1), že pre prípad laického používateľa systému AKWS je vhodné mať v systéme nástroj, ktorý bude používateľovi uľahčovať spôsob tvorby externej špecifikácie problému kompozície. V dôsledku zamerania práce, nebol daný návrh ďalej spracovaný. V prípade systému ZKWS (kapitola 5) by preto bolo potrebné spracovať, ako najjednoduchšie by laický používateľ mohol tvoriť počiatočné a cieľové stavy pomocou dostupných doménových ontológií bez potreby toho, aby musel poznať problematiku OWL ontológií.

Za najzaujímavejšiu oblasť možného výskumu v oblasti AKWS autor považuje možnosť dynamického zasahovania do systému počas plánovania. Dynamické zasahovanie do systému súvisí napr. s tvorbou plánu, kedy používateľ môže dynamicky meniť plán (napr. odstrániť nežiadanú akciu), pričom systém musí na používateľov zásah reagovať. Veľmi vhodné by bolo načrtnúť možnosť ukladania takýchto používateľových zásahov pre ďalšie plánovanie. Daný systém by potom mohol byť považovaný za inteligentný a prispôsoboval by sa potrebám konkrétneho používateľa, resp. viacerým používateľom.

Zoznam použitej literatúry

[1] AUSTIN, D. - BARBIR, A. – FERRIS, Ch. - S. GARG: Web Services Architecture Requirements, W3C Working Group Note, http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/, február 2004

[2] HAAS, H. – BROWN, A.: Web Services Glossary, W3C Working Group Note, http://www.w3.org/TR/ws-gloss/, február 2004

[3] SRIVASTAVA, Biplav - KOEHLER, Jana: Web Service Composition - Current Solutions and Open Problems. In: ICAPS 2003 Workshop on Planning for Web Services, p. 28-35, 2003

[4] GRAY, N.A.B.: Comparison of Web Services, Java-RMI, and COBRA Service Implementations. Fifth Australasian Workshop on Software and System Architectures (ASWEC 2004), Melbourne, Australia, apríl 2004

[5] CHRISTENSEN, E. – CURBERA, F. – MEREDITH, G. – WEERAWARANA, S: Web Services Description Language (WSDL) 1.1, W3C Note, http://www.w3.org/TR/wsdl, 2001

[6] BRAY, T. - HOLLANDER, D. - LAYMAN, A. - TOBIN, R. - THOMPSON, H.: Namespaces in XML 1.0 (Third Edition), W3C Recommendation, http://www.w3.org/TR/REC-xml-names/, 2009

[7] GUDGIN, M. – HADLEY, M. – MENDELSOHN, N. – MOREAU, J. – NIELSEN, H. – KARMARKAR, A. – LAFON, Y.: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation, http://www.w3.org/TR/soap12-part1/, apríl 2007

[8] UDDI: Universal Description, Discovery, and Integration, Online community for the Universal Description, Discovery, and Integration OASIS Standard, http://uddi.xml.org/

[9] SMITH, M. – WELTY, Ch. – McGUINESS, D.: OWL Web Ontology Language Guide, W3C Recommendation, http://www.w3.org/TR/2004/REC-owl-guide-20040210/, 2004

[10] BECHHOFER, S. – DEAN, M. – VAN HARMELEN, F. – HENDLER, J. – HORROCKS, I. – McGUINESS, D. – PATEL-SCHNEIDER, P. – SCHREIBER, G. – STEIN, L.: Owl Web Ontology Language Reference, W3C Proposed Recommendation. http://www.w3.org/TR/owl-ref/, 2004.

[11] MARTIN, D. – BURSTEIN, M. – HOOBS, J. – LASSILA, O. – McDERMOTT, D. - McILRAITH S. – NARAYANAN, S. – PAOLUCCI, M. – PARSIA, B. – PAYNE, T. – SIRIN, E. – SRINIVASAN, N. – SYCARA, K.: OWL-S: Semantic Markup for Web Services, W3C Member Submission, http://www.w3.org/Submission/OWL-S/, 2004

[12] HORROCKS, I. – PATEL-SCHNEIDER, P. – BOLEY, H. – TABET, S. – GROSOF, B. – DEAN, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, http://www.w3.org/Submission/SWRL, 2004

[13] Knowledge Interchange Format. Draft proposed American National Standard (dpANS) NCITS.T2/98-004, http://logic.stanford.edu/kif/dpans.html, 1998

[14] MARTIN, D. – BURSTEIN, M. – LASSILA, O. – PAOLUCCI, M. – PAYNE, T. – McILRAITH, S.: Describing Web Services using OWL-S and WSDL, http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html

[15] ČULEN, Martin: XML, databázy a webové služby - meteorologické webové služby, Diplomová práca, Univerzita Komenského, 2005

[16] ABELA, Charlie: Semantic Web Service Composition, Department of Computer Science and AI, University of Malta, 2003

[17] RAO, J. – SU, X.: A Survey of Automated Web Service Composition Methods. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA, 2004.

[18] NILSSON, Nils J. – FIKES, Richard E.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, Artificial Intelligence, 2(3):189-208, 1971.

[19] BONET, Blai – GEFFER, Hector: HSP: Heuristic Search Planner, Entry at AIPS-98 Planning Competition, AI Magazine Vol 21(2), 2000

[20] BONET, Blai – GEFFER, Hector: Heuristic Search Planner Ver. 2.0., AI Magazine. Fall 2001. Pages 77-80, 2001

[21] HOFFMAN, Jorg: FF: The Fast Forward Planning System, AI Magazine, Vol. 22 #3, pp 57-62, 2001

[22] MARTÍNEZ, E. – LESPÉRANCE, Y.: Web Service Composition as a Planning Task: Experiments using Knowledge-Based Planning. Proceedings of the ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services, pp. 62-69, Whistler, BC, June 3-7, 2004.

[23] NIEMELA, I. – SIMONS, P.: Smodels - An Implementation of the Stable Model and Well-Founded Semantics for Normal LP. In Proceedings of the 4th international Conference on Logic Programming and Nonmonotonic Reasoning (July 28 - 31, 1997). J. Dix, U. Furbach, and A. Nerode, Eds. Lecture Notes In Computer Science, vol. 1265. Springer-Verlag, London, 421-430, 1997

[24] DIMOPOULOS, Yannis – NEBEL, Bernhard – KOEHLER, Jana: Encoding Planning Problems in Nonmonotonic Logic Programs. In Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning (September

24 - 26, 1997). S. Steel and R. Alami, Eds. Lecture Notes In Computer Science, vol. 1348. Springer-Verlag, London, 169-181, 1997

[25] REITER, Ray: On knowledge-based programming with sensing in the situation calculus. ACM Trans. Comput. Logic 2, 4 (Oct. 2001), 433-457, 2001

[26] PONNEKANTI, Shankar R. – FOX, Armando: Sword: A developer toolkit for web service composition. In Proceedings of the 11th International WWW Conference (WWW2002), Honolulu, HI, USA, 2002.

[27] DOSTÁL, Adam: Demonstrace metod plánovaní, Vysoké učení technické v Brně, Fakulta infromačných technológií – Ústav inteligentných systému. Brno, 2007

[28] EROL, Kutluhan – HENDLER, James – NAU, Dana S.: Complexity Results for HTN Planning, Annals of Mathematics and Artificial Intelligence, 69 – 93, 1996

[29] EROL, Kutluhan – HENDLER, James – NAU, Dana S.: Semantics for Hierarchical Task-Network Planning, Technical report CS-TR-3239, UMIACS-TR-94-31, University of Maryland, March 1994

[30] WU, Dan – SIRIN, Evren –HENDLER, James – NAU, Dana – PARSIA, Bijan: Automatic Web services composition using SHOP2, In Workshop on Planning for Web Services, 2003

[31] BLUM, Avrim L. – FURST, Merrick L.: Fast planning through planning graph analysis, In Artificial Intelligence journal volume 90, 1636 -1642, 1997

[32] PEER, Joachim: Web Service Composition as AI Planning - A Survey, Dissertation, University of St. Gallen, Switzerland, 2005.

[33] PEER, Joachim: A PDDL Based Tool for Automatic Web Service Composition. Lecture Notes in Computer Science, Springer Berlin / Heidelberg. ISBN 978-3-540-22961-2, 149-163, september 2004

[34] Nau, Dana – CAO, Y – LOTEM, A – MUNOZ-AVILA, H.: SHOP: Simple Hierarchical Ordered Planner. In IJCAI-99, pp. 968-973, 1999

[35] McILRAITH, S. – SON, T.: Adapting Golog for Composition of Semantic Web Services. In Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), 482-493, Toulouse, France, April 22-25, 2002.

[36] SIRIN, Evren – HENDLER, James – BIJAN, Parsia: Semi-automatic composition of web services using semantic descriptions in Web services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003, Angers, France, April 2003

[37] SIRIN, Evren – BIJAN, Parsia – WU, Dan - HENDLER, James: HTN planning for web service composition using SHOP2, Journal of Web Semantics 1 (4), pp.377-396, 2004

[38] KLUSCH, M. - GERBER, A. - SCHMIDT, M.: Semantic Web Service Composition Planning with OWLS-Xplan.1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA, 2005

[39] GERBER, Andreas – BUTT, Suleman – KLUSCH, Matthias: OWLS-XPlan: OWL-S Semantic Web Service Composition Planner, User Manual Version 1.0, DFKI Saarbrücken, Germany: February 20, 2006

[40] OH, S. - LEE, D. - KUMARA, S. R.: A comparative illustration of AI planning-based web services composition. ACM SIGecom Exch. 5, 5 Jan., 2006

[41] SUBRAHMANIAN, V. S. – ZANIOLO, C.: Relating stable models and AI planning domains. In International Conference on Logic Programming, pages 233 – 247, 1995

[42] GHALLAB, Malik – NAU, Dana – TRAVERSO, Paolo: Automated Planning Theory and Practice. Morgan Kaufmann Publisher Inc., San Francisco, CA, USA, ISBN 1-55860-856-7, 2004

[43] GHALLAB, M. – HOWE, A. – KNOBLOCK, C. – McDERMOTT, D. – RAM, A. – VELOSO, M. – WELD, D. – WILKINS, D.: PDDL - The Planning Domain Definition Language, Version 1.2. Yale Center for Computational Vision and Control, Tech Report CVC TR-98-003/DCS TR-1165, October, 1998

[44] SCOWEN, Roger S.: Extended BNF — A generic base standard. Software Engineering Standards Symposium 1993.

[45] FOX, M. - LONG D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research 20 (2003) 61-124, Submitted 09/02; December 2003

[46] McDERMOTT, D.: The formal semantics of processes in PDDL. In Proceedings of the ICAPS-03 Workshop on PDDL, 2003

[47] HOFFMANN, J. – EDELKAMP, S.: The Deterministic Part of IPC-4: An Overview. Journal of Artificial Intelligence Research 24 (2005) 519-579. October 2005

[48] GEREVINI, A. – LONG, D.: BNF Description of PDDL3.0. Technical report, http://ipc5.ing.unibs.it, 2005

[49] GEREVINY, A – LONG, D.: Plan constraints and preferences in PDDL3, Technical Report RT-2005-08-47, Dipartimento di Elettronica per l'Automazione, Universitá di Brescia, 2005

[50] SIRIN, Evren: Combining description logic reasoning with AI planning for composition of Web services. Dissertation Thesis, University of Maryland, 239 p., 2006

[51] ŽÁKOVÁ, Monika – KŘEMEN, Petr – ŽELEZNÝ, Filip – LAVRAČ, Nada: Using Ontological Reasoning and Planning for Data Mining Workflow Composition. SoKD: ECML/PKDD 2008 workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery, 2008.

[52] YANG, Bo - QIN, Zheng: Composing Semantic Web Services with PDDL. Information Technology Journal 9 (1), ISSN 1812-5638, 48-54, 2010

[53] EROL, K. – HENDLER, J. – NAU, D.: UMCP: A sound and complete procedure for hierarchical task-network planning. In Proc. Internat. Conf. on AI Planning Systems (AIPS), pp. 249–254, June 1994.

[54] CHAN, K. S. M. – BISHOP, J. – BARESI, L.: Survey and comparison of planning techniques for web services composition. Technical report, Univ. of Pretoria, 2007

[55] SRINIVASAN, Naveen - PAOLUCCI, Masiimo - SYCARA, Katia: Adding OWL-S to UDDI, implementation and throughput. In Proc. 1st Intl. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), p. 6-19, 2004

[56] KUMAR, Sandeep - MISHRA, R. B.: Semantic Web Service Composition, IETE Technical Review, Vol. 25, No. 3, May-June'08, 2008

[57] PISTORE, Marco - BARBON, Fabio - BERTOLI, Piergiorgio - SHAPARAU, Dmitry - TRAVERSO, Paolo: Planning and Monitoring Web Service Composition. AIMSA, 106-115, 2004

[58] HABALA, Ondrej - PARALIČ, Marek - ROZINAJOVÁ, Viera - BARTALOS, Peter: Semantically-Aided Data-Aware Service Workflow Composition. SOFSEM '09: Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science, ISBN 978-3-540-95890-1, p. 317-328, 2009

[59] Resource Description Framework (RDF), http://www.w3.org/RDF/, 2004

[60] BABIK, Marian: Optimizing description logic reasoning for the matchmaking and composition of semantic web services. Dissertation Thesis. Institute of Informatics, Slovak Academy of Sciences, Bratislava, p. 137, 2007

[61] PEDNAULT, E.: ADL and the state-transitionmodel of action. Journal of Logic and Computation 4, p. 467–512, 1994

[62] FIELDING, Roy Thomas: Architectural Styles and the Design of Network-based Software Architectures. Dissertation Thesis. University of California, Irvine, p. 180, 2000

Prílohy

Príloha A: Príklady – vybrané príklady demonštrujúce prezentované štandardy súvisiace s AKWS

Príloha B: CD médium – dizertačná práca v elektronickej podobe, prílohy v elektronickej podobe, program ku dizertačnej práci s používateľskou, systémovou príručkou a Javadoc dokumentáciou.