MATEMATICKO-FYZIKÁLNA
FAKULTA

UNIVERZITY KOMENSKÉHO

Andrej Lúčny

Architektúra inteligentných programových systémov

Projekt dizertačnej práce

BRATISLAVA 1997

Kapitola 1

Úvod

I najbežnejší užívatelia programových systémov (t.j. programových produktov pozostávajúcich z viacerých navzájom komunikujúcich procesov bežiacich pod operačným systémom s preemptívnym multitaskingom) čoskoro spoznajú, že i keď im tieto systémy v ich práci výrazne napomáhajú, v spolupráci s nimi dokážu urobiť také chyby, ktoré by sami nikdy neurobili. O hlbokej pravde predchádzajúceho tvrdenia som mal možnosť sa presvedčiť počas môjho trojročného pôsobenia v praxi. Pritom spomínané chyby vznikajú v situáciách, kedy by triviálna reakcia systému typu „stalo sa toto, urobím tamto“ dokázala hroziacu chybu eliminovať. Pokiaľ sa pri používaní určitého nasadeného systému takýto prípad vyskytne, obyčajne sa premietne do snahy túto reakciu do systému zabudovať. Naráža to spravidla na tri problémy:

- z užívateľského hľadiska triviálna reakcia môže byť ťažkým orieškom pokiaľ v systéme nie sú prítomné dáta, ktoré indikujú, kedy treba reakciu uplatniť a služby, ktoré ju zrealizujú
- zabudovanie takýchto reakcií je konštrukčne náročné a pri postupnom zabudovávaní ďalších a ďalších reakcií dochádza k zvyšovaniu štrukturálnej zložitosti kódov zúčastnených procesov, čo vedie stále k väčšej a väčšej pravdepodobnosti vzniku chýb, až tento proces dosiahne stav, kedy si tvorca netrúfa viac do svojho diela zasiahnuť
- po zabudovaní určitej reakcie jej význam v ďalšej interakcii užívateľa so systémom klesá a po čase sa ukáže potreba zaviesť ďalšiu reakciu, rafinovanejšiu ako prvú a potom ďalšiu a ďalšiu, až sa ukáže, že nasledujúcu reakciu si už netrúfame do systému implementovať, lebo to predstavuje zhruba takú prácu ako vybudovanie nového systému.

Predstavme si napríklad, že tvorcami File Manager-a, v ktorom sa maže tlačidlom F8. Jedného dňa nám zavolá zúfalý zákazník a oznámi, že omylom stačil F8 a zmazal si súbor, ktorý písal dva dni a povedzme, že ho už nejde obnoviť. Dorobíme teda do svojho systému vlastnosť, že pri stlačení F8 sa nemaže súbor hneď, ale najprv sa užívateľovi položí otázka, či ho chce zmazať. Už tu môže nastať prvý problém, lebo pokiaľ nemáme v systéme službu na otvorenie dotazovacieho okna, musíme si ho kvôli tejto úprave naprogramovať. Novú verziu dodáme užívateľovi a ten bude spokojný. Ale iba do času. Čoskoro si užívateľ na otázku zvykne a potvrdí ju automaticky prv ako si stihne uvedomiť, čo sa ho to systém vlastne pýta. Prejaví sa teda tretí problém a opäť bedáka, že si zmazal súbor, ktorý písal týždeň a že by sme s tým mali niečo urobiť. Takto môžeme postupne zabudovať do nášho systému vlastnosti ako: „pýta sa dvakrát“, „ak je súbor nulový nepýta sa ani raz“, „súbory s určitými menami a príponami podľa konfigurácie maže bez opýtania a na niektoré sa pýta aj tri krát“, „určité súbory vôbec nedovolí zmazať“, až zistíme, že skutočné vyriešenie tohto problému spočíva v tom, že náš File Manager nerozumie obsahu súborov, ktoré maže. Tu ešte chvíľu môžeme pokračovať reakciami typu „ak sa súbore vyskytuje takýto reťazec, pýtaj sa štyrikrát“, pričom sú tieto modifikácie sprevádzané prácnym zabudovávaním možností konfigurácie týchto reakcií, až sa dostaneme k potrebe, aby systém skutočne rozumel obsahu súboru a to nás už prejde akákoľvek chuť takúto črtu do systému zapracovávať, vzhľadom na jej obtiažnosť. Skončíme tým, že keď nám zákazník nabudúce ohlási, že si vymazal súbor, ktorý písal dva mesiace, poriadne ho vyhrešíme a poučíme ho, že si má dávať pri práci pozor. A pretože sa z nášho kódu stala medzičasom obluda, ktorej nepomôže už ani reverse engeneering, urýchlene hľadáme inú zákazku.

Dajme si teda za cieľ vyvíjať také programové systémy, ktoré by sa vyvarovali akcií, ktoré by rozmýšľajúci užívateľ nikdy nechcel vykonať. Vzhľadom na nádej, že pri naplnení tohto cieľa sa priblížime i k vyšším métam, budeme tieto systémy nazývať inteligentnými programovými systémami. Zároveň si uvedomme, že naplnenie tohto cieľa nie je jednoduché: musíme navrhnúť konštrukciu takéhoto systému a postup jeho tvorby, pričom oboje musí jednak umožňiť náš cieľ a jednak - z pochopiteľných dôvodov - vychádzať z vývojovej línie tradičných programových systémov. Z uvedeného príkladu jasne vidieť, že takáto architektúra systému[1] musí v prvom rade

- zabrániť znehodnocovaniu jeho kódu počas opakujúcich modifikácií systému
- zabrániť narastaniu prácnosti vykonávania dorábok a prerábok
- umožniť rozsiahlu mieru konfigurovateľnosti systému

Pouvažujme teraz, v akom vzťahu je nami zvolený cieľ k vednej disciplíne, ktorá sa už desaťročia zaoberá oveľa odvážnejšími cieľmi - k umelej inteligencii. Základným cieľom umelej inteligencie je budovanie inteligentných systémov, systémov, ktoré majú

[1] Pod architektúrou systému budeme vždy rozumieť jeho konštrukciu a metódu jeho tvorby

vlastnosti, ktorými sa tak veľmi vyznačujú i najprimitívnejšie živé systémy a tak málo súčasné systémy umelé. Systémov, ktoré sú schopné priaznivo až rozumne reagovať na zmeny vo svojom prostredí v rámci širokej **triedy povolených stavov prostredia**[2]. Systémov, ktoré sa hľadiska subjektívneho názoru ľudského pozorovateľa prejavujú rovnaké alebo podobné správanie sa ako živé systémy. Rôzne pokusy v oblasti umelej inteligencie ukázali, že k tomuto cieľu možno smerovať, problematickým sa však spravidla javí širokosť spomínanej triedy možných stavov prostredia, pri ktorých je správanie sa systému také, aké má byť. Táto trieda je práve spravidla veľmi úzka a z rôznych dôvodov, často záhadných, nie je možné ani jej ďalšie rozširovanie.

Nuž, nami vytýčený cieľ je zrejme zúžením cieľa umelej inteligencie. Možno však ide o zúženie, ktoré je tým správnym stupienkom, z ktorého je cieľ umelej inteligencie vidieť oveľa ostrejšie. Pokiaľ sa tento momentálne javí ako príliš nenáročný na výstup, je to len zdanie. Dopredu môžeme prezradiť, že z tohto stupienka sa nám naskytne jednotný pohľad na jednu celú časť umelej inteligencie, pričom bude vidieť širokoďaleko i do tvorby programových i operačných systémov a do robotiky.

Svojho času som sa u jedných zákazníkov dal do reči s tamojším vrátnikom. Rozprával o svojich názoroch na používanie počítačov a i keď u neho absentovalo akékoľvek vzdelanie v tomto odbore, rozprával zaujímavo. Odvtedy mám na pamäti, že urobiť programový systém, s ktorým by bol spokojný, povedzme vrátnik, je problém umelej inteligencie a je to ťažký, nesmierne ťažký oriešok. Nuž, poďme sa s týmto orieškom pohrávať.

[2] Prosíme čitateľa o zapamätanie si tohto zvratu, budeme ho často používať. Na objasnenie uvedieme, že inteligentný systém nemusí reagovať inteligentne na každú zmenu. Ani človeku nepomôže inteligencia, keď sa náhle teplota jeho prostredia zmení z 20 $^{\circ}C$ na -200 $^{\circ}C$, keď sa však zmení na 5 $^{\circ}C$, mal by sa lepšie oblieť. Inteligentné reakcie umelých systémov sa vzťahujú na ešte menšiu triedu povolených stavov prostredia, dokážu reagovať napríklad iba na zmeny výkonu tridsiatich dvoch motorov raketoplánu a jeho požadovanej dráhy. Ich prínos však spočíva v tom, že triedy ich povolených stavov prostredia spravidla nie sú podtriedami tried povolených stavov prostredia (bežných) ľudí, hoci sú tieto oveľa širšie.

Kapitola 2

Predpoklady tvorby inteligentných programových systémov

Pokiaľ chceme posunúť vpred kvalitu svojich programových systémov, t.j. niečo získať, musíme za to niečím zaplatiť, musíme niečo obetovať. Čo konkrétne to bude, závisí čiastočne od nášho výberu, avšak v rozhodujúcej miere je to determinované všeobecnými vlastnosťami inteligentných systémov. Z hľadiska doterajšej tvorby programových systémov ide o vlastnosti nie veľmi prirodzené. Predstava, že programový systém sa s určitou možnosťou správa nekorektne, že jeho tvorcovia nie celkom rozumejú ako funguje, že sa za tých istých okolností zachová ináč, že realizácia jednorázovej akcie si vyžiada mnoho jej opakovaní a podobne, dokáže vystrašiť i názorovo otvoreného programátora. Pokiaľ sa však majú programové systémy priblížiť inteligentným systémom, musí sa na ne rozšíriť platnosť toho, čo z rôznych dôvodov platí pre inteligentné systémy ako také.

2.1 Inteligentné systémy

Základným cieľom umelej inteligencie je budovanie inteligentných systémov. Čo všetko možno za takéto systémy považovať, je predmetom pragmatických úvah, pri ktorých sa hľadajú také predpoklady, ktoré vedú k pokroku v problematike, ktorou sa zaoberáme. Najvýstižnejšie by bolo povedať, že ide o systémy, ktoré majú vlastnosti, ktoré nám na bežných umelých systémoch chýbajú a pozorujeme ich pritom na systémoch živých. Naopak najvšeobecnejšie môžeme definovať inteligentný systém ako systém majúci vlastnosť inteligencie a odsunúť tak ťažisko podstaty definície na rozhranie medzi umelou inteligenciou a psychológiou, ktorá pod inteligenciou rozumie schopnosť adaptovať sa na zmeny prostredí.

Z hľadiska tvorby umelých systémov musí teda na základe akcií v prostredí dochádzať v umelom systéme k adekvátnym reakciám, ktoré možno nazvať rozumnými. Tu samozrejme musíme povedať, čo sú to rozumné reakcie. Aby sme toto mohli posúdiť, nezostane nám nič iné, ako predpokladať u umelého systému existenciu nejakého cieľa, ktorý má plniť a za účelom ktorého bol vytvorený. Rozumnými reakciami teda budeme chápať reakcie vedúce k dosiahnutiu či dosahovaniu určitého cieľa.

Pravda, tu je dosť veľká diferencia medzi práve definovanou rozumnosťou a rozumnosťou ako ju chápeme v bežnom živote. Alternatívou našej definície, ktorá by bola v bližšom vzťahu k živým systémom, je definovať rozumné reakcie ako tie, ktoré by sme sami v danej situácii podnikli[3]. Pri ľuďoch samozrejme nemožno všeobecne hovoriť o cieľoch pre ktoré prišli na svet. Naopak, pri umelých systémoch to vieme veľmi presne.

2.2 Turingov test

V predchádzajúcej podkapitole sme si rámcovo zadefinovali čo chápeme pod inteligentnými systémami. Otázkou však ostáva ako preveríme či nejaký nami vytvorený systém je inteligentný a hlavne ako o tom presvedčíme ostatných.

Principiálne sa nám ponúkajú dve možné metodiky vychádzajúce z uvedených dvoch definícií rozumnosti. Prvá je založená na tom, že sa na základe merateľných kvantitatívnych údajov vyhodnotí nakoľko sa systému darí splniť stanovený cieľ, ide teda o objektívne kritérium. Žiaľ, jeho aplikovateľnosť silne závisí od povahy cieľa: čím je tento komplexnejší, tým je jeho aplikovateľnosť menšia. Druhá je založená na subjektívnom viacnásobnom porovnaní správania sa umelého systému s analogickým živým systémom, pričom pozorovateľ nevie, ktorý zo systémov je živý a ktorý umelý, nemá možnosť zistiť to na základe ich konštrukcie a ani usudzovaním z reakcií na akcie týkajúce sa vlastnej konštrukcie systémov[4].

Toto subjektívne kritérium zvané Turingov test[5] je podľa môjho názoru jediným kritériom na posúdenie prítomnosti inteligentného správania u systémov s komplexným cieľom. Skutočne komplexný cieľ sa totiž nedá exaktne vyjadriť v nejakej

[3] Tu je dôležité všimnúť si gramatickú stavbu tejto vety a odlíšiť jej zmysel od „pre ktoré by sme sa v danej situácii rozhodli (ak by sme mali dostatočne veľa času na uvažovanie)“ alebo „ktoré by sme považovali za dobré v danej situácii podniknúť (dodatočne)“.

[4] Spravidla sa však tieto dosť náročné podmienky nahradzujú dobrou vôľou pozorovateľa. Čo sa týka poslednej podmienky, ide o to aby sa pozorovateľ nemohol napr. priamo opýtať jedného zo systémov či je umelý a na základe jeho pravdomluvnosti usúdiť, že naozaj je. Takúto možnosť spravidla nemá, vzhľadom na nedokonalú podobu interakcie umelých systémov s pozorovateľom.

[5] Turing tento test prezentoval vo forme imitačnej hry, ktorá je navrhnutá tak, aby už z principiálnych dôvodov spĺňala uvedené podmienky [Kelemen 1989].

skrátenej forme, ale iba poukázaním na analogický systém. Keby som napríklad chcel urobiť robota čistiaceho kuchyňu, som schopný nadefinovať, že má udržiavať v čistote okná, dlážku a riady a výsledok si môžem skontrolovať na oknách, dlážke a riadoch. Keď však budem chcieť urobiť robota, ktorý sa správa ako môj sused, napriek tomu že suseda dobre poznám, nebudem vedieť naraz našpecifikovať jeho správanie. Budem sa musieť uspokojiť s tým, že našpecifikujem určité hlavné črty a výsledok si budem istý čas všímať. Postupne si uvedomím, čo je v správaní výsledku odlišné od správania môjho suseda a čo mi tam naopak chýba. V takomto momente je dôležité, aby konštrukcia systému umožnila tieto moje nové požiadavky do systému ľahko premietnuť. Pokiaľ taká je, môžem zahájiť druhé kolo všímania si systému a ďalej systém inkrementálne modifikovať, až to bude skoro ono. Nedostaneme síce kópiu môjho suseda, ale zato budeme mať jeho dobrú aproximáciu[6].

2.3 Vzťah programových a inteligentných systémov

Zaujímavé je uvedomiť si podobnosť vyššie spomínaného procesu s tvorbou bežných programových systémov, ktorým inteligenciu nikto nepripisuje a ani ju od nich (pôvodne) neočakáva. Tu možno napísať dobré zadanie, dôkladne popísať bázu dát, dátové toky, služby, grafické rozhranie, potom vybudovať systém, ktorý toto všetko zahŕňa a otestovať ho podľa uvedeného popisu a povedať: „je to ono!“. Ťažkosti tohto prístupu spočívajú v tom, že každý reálne nasadený systém vyžaduje dodatočné modifikácie (je to spôsobené jednak tým, že ani najstarostlivejšie vypracované zadanie nie je dokonalé, jednak tým, že trvajúcou interakciou užívateľa s programovým systémom sa menia jeho predstavy a nároky), ktoré ak užívateľ nepožaduje, nie je to preto, že by si ich neželal, ale preto, že napríklad na ne nemá dostatok financií, alebo mu to neumožňujú zmluvné vzťahy, alebo by po zmene bolo treba systém podrobiť novej certifikácií a podobne. V súčasnosti sa veľmi veľa energie venuje práve zdokonaleniu tohoto prístupu (okrem iného zavádzaním noriem ISO), lebo je pre tvorcov programových systémov ekonomicky nevyhnutný. Tvorca systému sa totiž týmto spôsobom vyvaruje komplikácií, nakoľko užívateľa dostane do situácie, keď nemôže proti finálnym vlastnostiam systému nič namietať, keďže v projekte svieti pod týmito vlastnosťami jeho podpis. Uznávam, že vzhľadom na ekonomickú stránku tvorby programových systémov v súčasnosti pre ich tvorcov niet inej cesty, napriek tomu však považujem túto cestu za slepú uličku. Dôvod spočíva v mojom vyššie uvedenom názore na jedinečnosť Turingovho testu ako kritéria pre posúdenie rozumnosti či inteligencie systémov s komplexným správaním.

[6] Turingov test má štatistickú povahu, preto ním táto aproximácia prejde napriek tomu, že medzi ňou a aproximovaným vzorom ostanú nejaké rozdiely. Dokonca môže ísť o závažné rozdiely, ktoré sa však prejavujú veľmi zriedkavo a majú nepatrný vplyv na správanie vzoru. Napríklad správanie kupujúceho v supermarkete možno veľmi dobre aproximovať jeho zameraním sa na minimálne výdaje a maximálny úžitok napriek tomu, že skutočný kupujúci občas dá drobné žobrákovi pri vchode.

Na prvý pohľad ide pri tvorbe bežných programových systémov o iný druh systémov, než sme uvažovali pri úvahách o Turingovom teste. Avšak keď si uvedomíme, že pri programových systémoch neustále vzrastá miera ich autonómnosti, komplexnosti, kvality interakcie s užívateľom a využívania prvkov umelej inteligencie, presvedčíme sa, že tieto systémy smerujú k systémom inteligentným. Užívateľ už nebude požadovať zoznam služieb, ale inteligenciu. Nebude od systému očakávať vykonávanie postupnosti nejakých príkazov, ktorá je ukončená odovzdaním výsledku, ale komplexné, nepretržité a autonómne správanie. Pri ich konštrukcii nepôjde o napodobnenie existujúcich živých systémov, ale o implementáciu fiktívnych živých systémov, ktoré svojimi kvalitami viažucimi sa k ich použitiu prekonávajú tie existujúce. Zámer umelej inteligencie napodobniť živé systémy sa v tomto prípade rozširuje na tvorbu bežných programových systémov určitým špecifickým spôsobom, ktorým je možné spomínané živé systémy napodobniť[7]. Práve toto je podľa mňa cesta, ktorá na rozdiel od vyššie spomínanej, nie je slepá. Na systémy vytvorené týmto špecifickým spôsobom sa samozrejme naše úvahy ohľadne používania Turingovho testu vzťahujú, takže pokiaľ budú chcieť byť aj komplexné[8], neostane iná možnosť ako tvoriť ich inkrementálne. T.j. nie spôsob tvorby sa má prispôsobovať ekonomickej stránke veci, ale ekonomická stránka veci sa musí prispôsobiť novému spôsobu zadávania a tvorby.

Aby to však bolo vôbec možné, musíme pri tejto tvorbe použiť inú architektúru než doteraz, lebo ako sme spomínali v úvode, inkrementálny postup tvorby pri použití doterajšej architektúry vedie k znehodnocovaniu konštrukcie systému. Existujú rôzne metódy ktoré toto znehodnocovanie dokážu spomaliť (napr. CASE), ale nie zastaviť. Znehodnocovanie spočíva v tom, že

- tvorcovia sa v systéme čoraz menej vyznajú
- že dorobiť ďalšie zmeny je čoraz ekonomicky náročnejšie
- že pri dorobení ďalších zmien začínajú vznikať šíriace sa chyby (pri odstránení jednej chyby, vzniknú dve nové)

Je naivné si myslieť, že sa nám podarí nájsť takú architektúru ktorá nebude mať tieto zlé vlastnosti a ostanú jej pritom všetky dobré. Tomuto cieľu budeme musieť niečo obetovať. Tu prezradím, že môj návrh obetúva porozumenie systému ako celku[9] a jeho efektívnosť[10]. Týmto je zároveň stratená možnosť odvodiť správanie systému z kódov jeho procesov. Jediné testovanie tohto systému spočíva teda v jeho vonkajšom pozorovaní, ktoré zároveň slúži i na ďalší vývoj systému. Turingov test tu teda degraduje na ľudové kritérium „oko pozrie, oko vidí“, nejde teda o nič pozoruhodné, pozoruhodným je jedine fakt, že toto kritérium je jediné možné.

[7] Podobný posun môžeme vidieť v objektovo-orientovanom programovaní. Pôvodný zámer pracovať v počítači s objektami reálneho sveta, ako sú dom, pracovník, výrobok a pod. sa posunul do práce s objektami fiktívnych svetov obsahujúcich napríklad okno, tlačidlo alebo menu.

[8] čo bude pravidlom, lebo inteligencia a komplexnosť idú ruka v ruke

[9] napríklad tvorca systému nevie, aké sú v ňom dátové toky

[10] výrazne vzrastie počet operácií pri propagácii dát

2.4 Metóda tvorby inteligentného programového systému

Treba však povedať, že Turingov test nie je ani v rámci oblasti umelej inteligencie všeobecne prijímaným kritériom. Podarilo sa totiž skonštruovať umelé systémy, ktoré Turingovým testom prešli napriek tomu, že konštrukčne boli veľmi jednoduché. Takýmto systémom je napríklad Weizenbaumov program ELIZA, ktorého cieľom je viesť v úlohe psychiatra liečebný dialóg s pacientom. Pacienti komunikujúci s týmto systémom odmietali uveriť, že nekomunikujú so skutočným psychiatrom, pritom program robil iba jednoduché reťazcové operácie nad vetami pacienta a slovami a vzorkami viet vo svojom slovníku [Ferko - Kalaš - Kelemen 1990]. Toto priviedlo mnohých na odmietnutie Turingovho testu ako dobrého kritéria.

Ja si naopak myslím, že tieto systémy sa vyznačovali architektúrou, ktorú hľadáme, presnejšie povedané boli jej (rozsiahlym) zúžením. Za to, že sa i pri takomto zúžení dostavil ich úspech, vďačia hlavne prirodzenej úzkosti svojej triedy povolených stavov prostredia[11]. Medzi veľkosťou povolených stavov prostredia a zložitosťou štruktúry systému je ľahké domyslieť si priamu úmeru. Teda namiesto toho, že tieto systémy vyvracajú prípustnosť Turingovho testu, môžeme vyhlásiť tieto systémy za skutočne inteligentné a stanoviť si za cieľ naučiť sa budovať podobné so širšou triedou povolených stavov prostredia (a teda i zložitejšou vnútornou štruktúrou).

Už som naznačil, že na dosiahnutie tohto cieľa obetujeme niečo z nášho porozumenia systému, ktorý tvoríme. Znie to absurdne, že nebudeme celkom rozumieť tomu čo robíme. Na tomto mieste to môžeme upresniť ešte tak, že síce budeme rozumieť každému lokálnemu toku dát v systéme, pri určitej zložitosti však stratíme prehľad o globálnych dátových tokoch. To znamená, že budeme rozumieť tomu, čo sa odohráva v systéme na ktoromkoľvek mieste, ale nebudeme celkom rozumieť tomu, ako systém globálne pracuje. Je to zvláštne, ale povedzme, že poznáme takú konštrukciu, ktorá sa týmto vyznačuje. Potom je vhodné položiť si otázku, či sme za týchto podmienok vôbec schopní urobiť taký systém, ktorý má také použitie, aké sme pôvodne zamýšľali. T.j. či sme ho schopní udržať pri jeho vývoji v určitej línií, aby z neho bolo to, čo chceme aby bolo. Tejto požiadavke samozrejme musí zodpovedať metóda jeho tvorby a už sme naznačili, že jej podstata bude spočívať v jej inkrementálnosti.

Dúfame, že na tomto mieste sa nám podarilo čitateľa presvedčiť o tom, že by mohla existovať taká architektúra programového systému, aplikovanie ktorej nám umožní budovať inteligentné programové systémy s rozsiahlejšími triedami povolených stavov prostredia. Vydajme sa teda hľadať túto architektúru. Treba povedať,

[11] V prípade systému Eliza ide o to, že spomínaný dialóg sa viaže k špecifickej téme, pri ktorej nás neprekvapí aj nie bežná akcia či reakcia psychiatra, lebo v nich vidíme nejaký jeho zámer či úskok. Keď si uvedomíme, že Turingovým testom prejde každá dobrá aproximácia, nemalo by nás prekvapiť, že ním prešiel systém, ktorý iba primitívne napodobňuje určitý vzor. Znamená len toľko, že ho napodobňuje veľmi dobre a samozrejme, že tento vzor je ľahko napodobiteľný.

že nepôjde o objavenie niečoho, čo tu doteraz nebolo. Budeme len aplikovať existujúce myšlienky na oblasti do ktorých ešte neprenikli. Konkrétne pôjde o aplikáciu myšlienok z oblasti novej umelej inteligencie na oblasť programových systémov. Obe spomínané oblasti si preto v nasledujúcich dvoch kapitolách priblížime.

Na záver tejto filozofickejšie orientovanej kapitoly ešte jedno malé upozornenie. Jedným z predpokladov umelej inteligencie je, že inteligentné systémy sú realizovateľné na bežných počítačoch, t.j. ide v konečnom dôsledku o nejaký sled bežných strojových inštrukcií. Nie každý takýto sled, ani nie každý zmysluplný, sa po spustení prejavuje ako inteligentný systém, umelá inteligencia však predpokladá, že existujú aj také. Cieľom umelej inteligencie teda nie je vymýšľať nový fyzikálny základ pre tvorbu inteligentných systémov, ale vymýšľať postup, ako sa k týmto žiadaným sledom strojových inštrukcií dostať. Samozrejme teoreticky na to stačí znalosť týchto inštrukcií a ich ukladanie vedľa seba. Prakticky však takýmto spôsobom nie možné vytvoriť ani väčší zmysluplný sled inštrukcií, nieto ešte sled inštrukcií inteligentného systému. Každý vie, že čo je možné teoreticky ešte nemusí byť možné prakticky. Pritom teoreticky možný postup naráža často na prekážky, ktoré nemožno presne špecifikovať[12]. Keď teda nájdeme nejakú architektúru, pomocou ktorej dostaneme sled strojových inštrukcií, ktorý sa inteligentne správa, má to svoju hodnotu[13].

[12] Nikto napríklad presne nevie prečo sa Babbageovi nepodarilo skonštruovať analytický stroj, bolo to proste preto, že si vzal do úst príliš veľké sústo [Kelemen 1989].

[13] i keď si odborníci z teórie jazykov a automatov myslia niečo iné, keďže na ich výpočtové modely nie je možné aplikovať Turingov test, v dôsledku čoho je pre nich každý sled inštrukcií rovnako hodnotný ako ktorýkoľvek iný

Kapitola 3

Možnosti aplikácie ideí novej umelej inteligencie

V polovici 80-tych rokov sa v rámci umelej inteligencie vyčlenila podoblasť, ktorá do umelej inteligencie vniesla nové myšlienky a ktorá si dala za cieľ budovať inteligentné systémy nového charakteru. Vzišla z oblasti autonómnej robotiky a ujal sa pre ňu názov **nová umelá inteligencia** [Brooks 1991].

Ako uvidíme ďalej, cieľ tejto disciplíny sa mierne líši od cieľa tradičných oblastí umelej inteligencie, čo sa premieta i do posunu v jej filozofických východiskách. Preto považujeme za oprávnené rozlišovať tradičnú a novú umelú inteligenciu.

Na konci 70-tych rokov vzišla z oblasti prieniku umelej inteligencie a distribuovaných výpočtových systémov **distribuovaná umelá inteligencia**, ktorá sa zamerala na aplikovanie myšlienok oboch týchto oborov v oblasti sociológie a ekonómie [Doran 1992].

Ťažko povedať, či v prípade novej a distribuovanej umelej inteligencie ide o dobývanie toho istého vrcholu z dvoch rôznych strán. Nepochybné je že tieto dva prúdy sa navzájom výrazne ovplyvňujú. Pritom väčší vplyv má distribuovaná umelá inteligencia na novú, než naopak. Dalo by sa povedať, že nová umelá inteligencia preberá nástroje distribuovanej, zatiaľ čo distribuovaná preberá myšlienky novej. Najväčší rozdiel medzi nimi však spočíva v oblasti ich aplikácií, čo zase nie je taký zásadný rozdiel, aby sa nedali tieto dve oblasti postaviť na jednotnú platformu.

Čo sa týka vzťahu týchto dvoch disciplín k tejto práci, táto práca spočíva v prvom rade v aplikovaní myšlienok novej umelej inteligencie. Pritom však používa i nástroje a pojmový aparát vlastné distribuovanej umelej inteligencii.

3.1 Tradičná umelá inteligencia

Ako sme už spomínali, cieľom umelej inteligencie je budovať inteligentné systémy. Avšak spomínali sme i to, že dôležitou charakteristikou takéhoto systému je veľkosť jeho triedy povolených stavov prostredia, t.j. rozsah pôsobnosti jeho inteligentného správania. Treba si uvedomiť, že táto trieda je pri rozumovo náročných, ale dobre sformulovaných problémoch (napr. šachový automat) úzka, zatiaľ čo pri rozumovo nenáročných a ťažko sformulovateľných problémoch (napr. trhanie kvetov na lúke) je táto trieda veľmi široká. Preto sa ťažký problém môže pri bližšom skúmaní ukázať ako ľahký[14], zatiaľ čo ľahký problém sa môže ukázať ako ťažký[15].

Tradičná umelá inteligencia zvolila pre budovanie inteligentných systémov také metódy, pri ktorých sa jej podarilo skonštruovať cenné systémy, ktoré boli prínosom (napr. expertný systém pre riadenie motorov raketoplánu). Keď sa však nad nimi zamyslíme, zistíme, že ich prínos spočíval v tom, že ich trieda povolených stavov prostredia (v spomínanom prípade fyzikálne veličiny charakterizujúce jednotlivé motory raketoplánu) nebola podtriedou triedy povolených stavov prostredia (bežných) ľudí. T.j. robila to, čo ľudia nedokázali. Keď sa však bližšie pozrieme na veľkosť tejto triedy zistíme, že je v porovnaní s analogickou triedou ktoréhokoľvek živého systému rádovo menšia.

Táto skutočnosť vyplávala na povrch v okamihu, keď v umelej inteligencii zaznamenali prvé pokusy o vytvorenie systémov interagujúcich v prirodzenom prostredí. V robotike tomu zodpovedala snaha o skonštruovanie kognitívnych robotov, ktoré by v prirodzenom prostredí dokázali splniť užívateľom zadaný cieľ. Od týchto systémov sa napríklad očakávalo, že po zadaní príkazu na postavenie červenej kocky na zelenú dokážu tento príkaz zrealizovať i keď na to budú musieť najprv zložiť z červenej kocky modrú. Ukázalo sa, že s takýmito problémami by spomínané systémy nemali najmenšie problémy, keby dokázali napr. pohybovať sa tak ako treba, vnímať to čo treba, a podobné, podľa ľudského chápania triviálne problémy. Tieto problémy sa však ukázali tak závažné, že vývoj takýchto robotických systémov ustrnul na mŕtvom bode. Dostal sa z neho zhruba po pätnástich rokoch a to práve aplikovaním myšlienok novej umelej inteligencie. Pritom za skutočne kognitívneho robota môžeme považovať až METATOTO ([Stein 1991]), vyvinuté zhruba dvadsať rokov po skrachovaní spomínaných ambicióznych projektov tradičnej umelej inteligencie [Mlichová 1993].

Z dnešného pohľadu kľúčovým momentom, ktorý tu zohral svoju negatívnu úlohu bola predstava, že inteligentný systém interagujúci v prirodzenom prostredí vytvára

[14] V čase písania tejto práce práve počítačový program prvý krát porazil majstra sveta v turnaji, pričom v jednom dueli ho dokázal poraziť už dávnejšie

[15] Do polovice 80-tych rokov sa nepodarilo skonštruovať autonómneho mobilného robota, ktorý by dokázal plniť v prirodzenom prostredí čo i len najjednoduchší cieľ, pritom pokusy v tomto smere sa datujú od šesťdesiatych rokov

vo svojom vnútri náprotivky všetkých objektov, ktoré sa nachádzajú v jeho prostredí a akcie, ktoré v tomto prostredí vykonáva, volí na základe manipulácie s týmito náprotivkami. Na jednej strane je nepochybné, že človek si takéto náprotivky vo svojom vnútri buduje. Na strane druhej je však otázne, pri akých inteligentných činnostiach túto metódu používa a k akým objektom náprotivky buduje a k akým nie. Tradičná umelá inteligencia vychádza z predstavy, že túto metódu používa pri všetkých inteligentných činnostiach a že táto metóda je jediná. Ďalej predpokladá, že medzi objektami prostredia systém dokáže robiť určitý sofistikovaný výber tých, ktoré sú preňho relevantné. Keďže zložitosť manipulácie má spravidla exponenciálnu závislosť od počtu vybraných objektov, vystupuje tu do popredia neriešiteľný problém: buď vyberieme málo objektov, a potom sa nám môže stať, že si nevšimneme niečo dôležité, čo môže viesť až k zhavarovaniu systému, alebo ich vyberiem veľa a potom sa nám môže stať, že príslušné akcie dopočítame, keď už je neskoro. Tento problém, nazývaný problémom rámca ([Kelemen a kol. 1992]) je v podstate neriešiteľným problémom v dynamickom prostredí. Preto všetky autonómne robotické systémy vytvorené na báze myšlienok tradičnej umelej inteligencie fungovali dobre len v statickom prostredí, v ktorom boli sami jedinými hýbateľmi.

Ďaleko ťažšie problémy sú spojené s vykonávaním zvolených akcií. Povaha akcií, ktoré sa v prostredí vykonávajú, totiž úzko súvisí s povahou tohto prostredia. Pri umelom či fiktívnom prostredí ide o jednoduché akcie typu "ťahaj koňom z e5 na f7". Avšak v prirodzenom prostredí ide o akcie typu "zlož červenú kocku zo zelenej", čo predstavuje nesmierne zložitú vec. Ťahu koňa na šachovnici (ak ide na voľné políčko) totiž nemá čo zabrániť, je jasné na akých súradniciach sa nachádza a i na ktoré sa má premiestniť. V druhom prípade však robot musí najprv nájsť červenú kocku, musí obísť všetky prekážky a by sa k nej dostal, musí ju uchopiť, nájsť miesto kam ju môže položiť a položiť ju. Pritom v dynamickom prostredí sa mu môže stať, že mu niekto druhý červenú kocku medzičasom odnesie preč, a podobne [Kelemen 1994].

Programové systémy, ktorých budovanie nevyžadovalo v takej miere potrebu tvorby autonómnych systémov, t.j. systémov, ktoré na zabezpečenie svojho správania nevyžadujú interakciu s užívateľom[16], podobná mizéria dosiaľ nepostihla. Dôvody spočívajú jednak v tom, že tieto systémy spravidla nie sú mobilné a jednak v tom, že tieto systémy mali slabé možnosti interakcie so svojím prostredím. Preto bola možnosť vytvorenia užitočných aplikácií uvažovaného druhu oveľa menšia než v robotike. K programovému systému potom bola vzhľadom na účel aplikácie spravidla možnosť posadiť operátora, ktorý svojím adekvátnym správaním zapĺňal medzery v správaní sa systému.

Vplyv multimédií, www a rozvoja embeded systémov, ako i potreba budovať čoraz zložitejšie systémy, pri ktorých operátorove možnosti nemusia stačiť vykryť diery v

[16] i keď môžu s ním interagovať za účelom odovzdania výsledku či prijatia cieľa

správaní sa programového systému, však vývoj programových systémov tlačí smerom na tenký ľad, kde ho čakajú podobné problémy aké postihli svojho času robotiku. Keďže táto práca sa snaží aplikovať v oblasti programových systémov myšlienky, ktoré vyviedli zo spomínaných problémov robotiku, nesmelo si dáva za cieľ do istej miery predchádzať analogickej situácii v programových systémoch.

Do programových systémov síce potrebujeme dostať prvky umelej inteligencie a povýšiť ich na inteligentné programové systémy, ale tradičná umelá inteligencia nám v tomto prípade nepomôže. Na zdôvodnenie tejto skutočnosti zaznieva často rôzne orientovaná kritika tradičnej umelej inteligencie. Za najvážnejšiu výhradu ja osobne považujem kritiku tvorby inteligentného systému metódou zhora nadol [Brooks 1986]. Pri takomto postupe sa spravidla systém rozdelí na

- spracovanie vstupu z receptorov
- vybudovanie či úpravy vnútornej reprezentácie prostredia (náprotivky)
- plánovanie t.j. manipuláciu s touto reprezentáciou za účelom získania plánu akcií, ktoré treba vykonať v prostredí, aby sa dosiahol určitý cieľ
- vykonávanie týchto plánov
- spracovanie výstupu pre efektory

Keď si teda predstavíme systém ako koláč, metóda zhora-nadol nás vedie pri prvom krájaní k rozkrájaniu na malý počet zložitých častí, predstavujúcich základné funkčné moduly systému (dekompozícia funkciou). Tie potom môžeme ďalej krájať podľa potreby. Takéto krájanie nám veľmi vyhovuje, lebo odkrojí ako jeden celok modul plánovania, v ktorom sa sústreďuje inteligencia systému, a ktorý si vieme veľmi dobre predstaviť. Na tomto kúsku je nám umožnené pochutnať si, ale chuť nás rýchlo prejde, keď sa pustíme do ostatných kúskov. V samej radosti sme totiž nepostrehli, že nám pri krájaní ostali kusy, ktoré nemožno ani ďalej krájať ani prehltnúť. Mohli by sme na druhý krát skúsiť iné krájanie, ale skúsenosť naznačuje, že chyba spočíva v samotnom krájaní. Jediné východisko je v tejto chvíli opustiť myšlienku krájania koláča a nás si celkom inú metódu.

Zaujímavé je, že pri vývoji programových systémov sa metóda zhora nadol nielen používa, ale považuje sa za jedinú správnu. Všeobecne sa verí, že systémy, ktoré by nevznikli na základe plánu ich výstavby, ktorý sa vypracúva od hrubých rysov k stále jemnejším podrobnostiam, by boli zle navrhnuté[17]. Toto je samozrejme veľká prekážka pre aplikáciu iných myšlienok. V tejto práci sa práve o túto aplikáciu pokúsime. Osudy podobných snažení nás však presviedčajú, že pokiaľ sa nám to má podariť, s tradičnou umelou inteligenciou sa musíme aspoň dočasne rozísť. Bolo by však nespravodlivé rozísť sa s ňou v zlom. Spravodlivejšie by jej bolo vymedziť nový rámec pôsobnosti, menší síce ako ten pôvodný, ale zato nesporne užitočný.

[17] V každom programátorskom desatore nájdete prikázanie: „Používaj metódu zhora-nadol !“.

V súlade s týmto predsavzatím považujeme za cieľ tradičnej umelej inteligencie budovať inteligentné systémy s malými ale zaujímavými triedami povolených stavov prostredia[18]. Takéto inteligentné systémy by sme mohli nazvať špecializovanými inteligentnými systémami. Na budovanie takýchto systémov vypracovala tradičná umelá inteligencia dobré aplikovateľné metódy a patrí jej za to vďaka. Žiaľ, ukázalo sa, že tieto metódy nemožno rozšíriť na budovanie inteligentných systémov so širokou triedou povolených stavov prostredia. Neostáva nám teda nič iné, ako sa týmito komplexnými inteligentnými systémami zaoberať v rámci iného prúdu umelej inteligencie. Nie je vylúčené, že po preskúmaní možností jeho metód, ho stihne podobný osud a ambiciózne ciele prevezme ďalší metodologický prúd v umelej inteligencii.

3.2 Nová umelá inteligencia

Ako sme naznačili v závere predchádzajúcej podkapitoly, cieľom novej umelej inteligencie je budovanie komplexných inteligentných systémov. Od ich triedy povolených stavov prostredia teda očakávame, že bude v prvom rade dostatočne široká, pričom jej zaujímavosť vo vzťahu k analogickej triede u (bežných) ľudí pre nás nie je podstatná. Nejde nám o to, aby sa tieto systémy vyznačovali schopnosťou vykonávať vysoké intelektuálne činnosti, ktoré by dokonca presahovali rámec bežných činností ľudí. Ide nám o postupné pokrytie povolených stavov prostredia ľudí od menej náročných činností k náročnejším, od jednoliatych k zloženým a od jednoduchých k zložitým.

Samozrejme nič nezaručuje, že sa nám metódami novej umelej inteligencie podarí rozšíriť spomínanú triedu až na oblasti, ktoré nie sú podtriedou triedy povolených stavov prostredia ľudí. Z tohto hľadiska by sa systémy novej umelej inteligencie môžu ukázať menej užitočné ako systémy tradičnej umelej inteligencie[19]. My však veríme, že sa ukážu aspoň tak užitočné, obzvlášť ak sa podarí myšlienky novej umelej inteligencie aplikovať i na oblasti, do ktorých dosiaľ umelá inteligencia ešte výrazne neprenikla, napríklad na oblasť tvorby programových systémov (čo je cieľom tejto práce).

V čom teda spočívajú myšlienky novej umelej inteligencie? Často sa uvádza, že rozdiel medzi tradičnou a novou umelou inteligenciou spočíva vo filozofických východiskách. Videl som dokonca jeden článok, v ktorom bolo uvedených niekoľko filozofických otázok, na ktoré umelý inteligent odpovedal áno-nie a na základe takéhoto tiketu mu autor povedal jeho príslušnosť k tomu či onomu názorovému prúdu. Ja som naopak presvedčený, že rozdiel spočíva jedine v cieli týchto disciplín, pričom sa

[18] Ich zaujímavosť môže spočívať napríklad v tom, že nie sú podtriedami povolených stavov prostredia ľudí a teda dokážu niečo čo ľudia nedokážu

[19] Nejaká banka určite skôr kúpi expertný systém, zaoberajúci sa úverovou problematikou, ako autonómneho mobilného robota, ktorý zametá priestory banky

snaha dosiahnuť svoj cieľ premieta na základe pragmatického prístupu i do filozofických úvah. Najzávažnejšou úvahou v tomto smere je úvaha o deliberatívnosti, t.j. úmyselnosti (cieľavedomosti).

Predstavte si, že idete do odpadového koša vysypať z tanierika zvyšky jedla. Otvoríte kôš, nožom zhŕňate zvyšky a zrazu vám do koša padne celý tanierik. Nastala tým určitá zmena v prostredí na ktorú by ste mali ako inteligentný systém adekvátne zareagovať. Zohnete sa do koša a tanierik z neho vytiahnete. Čo sa pritom odohralo vo vašej mysli?

Podľa jednej predstavy, vznikol pádom tanierika do koša problém, ktorý musíte vyriešiť. Tento problém je daný skutočným stavom prostredia (tanierik v koši) a želaným (cieľovým) stavom prostredia (tanierik v ruke) a rozhodnete sa ho riešiť vytiahnutím tanierika. Tanierik teda vytiahnete na základe toho, že s ním máte nejaký ďalší úmysel, ktorý by sa nedal realizovať, ak by ostal v koši. Pritom riešenie problému prebieha tak, že agent zvažuje rôzne následky vykonania rôznych postupností akcií[20] a na základe toho sa rozhodne ktorou cestou sa vydá.

Podľa druhej predstavy vznikla pádom tanierika do koša zmena v prostredí na ktorú treba správne zareagovať. Vy zo skúsenosti viete, že keď niečo čo nemá skončiť v koši padne do koša treba to vytianuť. Takže tanierik vytiahnete bez akéhokoľvek rozmýšľania na základe vedomosti, že v danej situácii sa tanierik vyťahuje.

Systém pracujúci podľa prvej predstavy nazývame deliberatívnym, t.j. úmyselným, lebo keby sa ho opýtali, prečo vytiahol z koša tanierik, odpovedal by „Chcel ho zaniesť do dresu na umytie a na to som ho musel držať v ruke a ja som ho nedržal a tak som ho chytil a vytiahol“. Keby sme tú istú otázku položili systému pracujúcemu podľa druhej predstavy, odpovedal by „Lebo dobre viem, že keď je tanierik v koši, treba ho vytiahnuť“. Takýto systém nazývame reaktívnym[21].

Nuž, ktorý spôsob bol použitý vo vašej mysli ? Zdanlivo prvý, t.j. deliberatívny. Lenže je tu jeden háčik. Plán vytvorený na základe úmyslu nemožno tvoriť priamočiaro, vyžaduje si to prehľadávanie do hĺbky či do šírky. V takomto prípade nemožno urobiť prvú akciu plánu prv, než nemáme potvrdené, že cesta, ktorú ňou začneme, vedie k cieľu. Musíme mať teda vytvorený celý plán, aby sme mohli urobiť jeho prvú akciu. Vykonanie prvej akcie by sa dalo urýchliť pri použití prehľadávania do šírky, pri ktorom sa môže po istom čase ukázať, že šancu dosiahnuť cieľ majú už len tie cesty, ktoré začínajú touto akciou. Každopádne sa však pritom musí istý čas rozmýšľať. Je však nepochybné, že každý ten tanierik vytiahne okamžite, ako mu

[20] Momentálne abstrahujeme od spôsobu ako si systém vyhodnocuje, ktorá ciest je vhodná. Väčšinou sa to robí porovnaním diferencií stavu, ktorý by vznikol po aplikovaní akcie, s cieľovým stavom. Iný spôsob uvádza napríklad [McFarland - Bosser 1993].

[21] požíva sa i termín nedeliberatívny

do koša padne[22]. Domnievam sa, že je to preto, lebo každý vie, že tanieriky do koša jednoducho nepatria, ide teda skôr o druhý, t.j. reaktívny spôsob.

Deliberatívnym systémom (systémom tradičnej umelej inteligencie) teda nazývame systém, ktorý svoje akcie získava na základe manipulácie so stavom prostredia, svojím vnútorným stavom a cieľom, pričom uvažuje viaceré alternatívy zvolených akcií a ich následky, t.j. rozhodovaním. Naopak, **reaktívnym systémom** (systémom novej umelej inteligencie) nazývame systém, ktorý svoje akcie získava na základe manipulácie iba so stavom prostredia a svojím vnútorným stavom, pričom neuvažuje alternatívy ani následky svojich akcií, t.j. reagovaním. (Reaktívne systémy samozrejme tiež sledujú určitý cieľ, ale tento cieľ v nich nikde nie je explicitne prítomný.) Mali by sme teda tendenciu povedať, že deliberatívny systém vie čo robí, zatiaľ čo reaktívny nie. Nie je to pravda. Ak si vôbec dovolíme použiť výraz „vedieť“ na umelý systém, pravdou je, že deliberatívny systém vie čo robí explicitne (preveruje si na predpokladaných následkoch správnosť vybraných akcií), zatiaľ čo reaktívny to vie implicitne[23] (nedokáže vybrať iné ako správne akcie).

Tento rozdiel sa v implementačných podmienkach výrazne premieta do architektúry uvažovaných systémov aj do nimi produkovaného správania. Dôležitú rolu tu hrajú také faktory ako

1. vnútorná reprezentácia
2. porozumenie tvorcu činnosti vytvoreného systému
3. schopnosť systému zotaviť sa z kritických stavov
4. determinizmus
5. predikovateľnosť
6. stelesnenosť
7. emergencia

1. Keďže deliberatívny systém uvažuje celkový stav prostredia a dokonca pri uvažovaní následkov vykonaných akcií s týmto stavom uskutočňuje zložité manipulácie, je pre neho nevyhnutné stav prostredia kompletne vnútorne reprezentovať, t.j. mať ho celý vyjadrený v deklaratívnej podobe. Naproti tomu reaktívny systém takúto reprezentáciu nevyhnutne nepotrebuje.

2. Na prvý pohľad sa zdá absurdné, že by tvorca systému nerozumel tomu, ako jeho systém funguje. Lenže jedna vec je rozumieť konštrukcií systému a druhá je rozumieť akým procesom bolo zabezpečené určité správanie systému. Teoreticky je vždy možné v každom modulárnom systéme vysledovať, aké moduly sa podieľali na výslednom správaní a ako. Prakticky je to tým zložitejšie, čím

[22] keby sme namiesto koša uvažovali dynamickejšie a mohutnejšie zariadenie, isto by sme si spomenuli na časté úrazy, ktoré sa v tejto situácii vyskytujú

[23] Cieľ v implicitnej podobe je naoko nevýhodou, skutočne komplexnému cieľu však ťažko môžeme dať explicitnú podobu.

väčší je počet týchto modulov a čím zložitejší je spôsob interakcie medzi nimi. Úmerne týmto kvantitám stúpa i omylnosť tohto usudzovania. Reaktívne systémy sa vyznačujú **modularitou** s veľkým množstvom modulov, ktorý je tým väčší, čím komplexnejší je ich cieľ, takže toto usudzovanie nie je jednoduchou záležitosťou. Koniec koncov ani u objektových programových systémov používajúcich dynamickú väzbu nie je podobné usudzovanie ľahké.

3. Pokiaľ sa prostredie deliberatívneho systému dostane do stavu mimo triedy jeho povolených stavov prostredia, obyčajne na takúto zmenu nedokáže zareagovať žiadnou akciou. To môže viesť až do stavu jeho „zaseknutia". Naproti tomu od systémov novej umelej inteligencie očakávame v danej chvíli to, že budú konať, iba ich konanie nemusí byť adekvátne (inteligentné). Týmto konaním majú šancu vrátiť prostredie do povoleného stavu a správať sa opäť inteligentne[24] Vedia sa teda zotaviť z krízového stavu.

4. V súvislosti so zotavením sa z krízových stavov nedeterminizmus môže výrazne napomôcť vygenerovať náhodnú akciu, ktorá dostane prostredie do povoleného stavu. Preto by sa systémy novej umelej inteligencie mali vyznačovať nedeterminizmom. Dôležitosť nedeterminizmu stúpa s faktom, že systém má spravidla obmedzené možnosti vnímania svojho prostredia, determinované sadou jeho receptorov. V takomto prípade mu môže z prostredia chýbať veličina, podľa ktorej by sa dalo v kritickom prípade inteligentne zareagovať. Nedeterminizmus môže do istej miery túto chýbajúcu veličinu nahradiť. Treba však povedať, že nejde o hocijaký nedeterminizmus. Správanie systému nemôže úplne náhodne kolísať, ale musí sa pevne držať určitej línie. Nedeterminizmus sa teda prejaví len určitými fluktuáciami na tejto línii.

5. Správanie systémov novej umelej inteligencie možno spravidla predpovedať iba rámcovo, presnejšie povedané, dá sa predpokladať vykonanie určitého okruhu akcií vedúcich k dosiahnutiu cieľa. Spravidla však nie je možné predpovedať akciu, ktorú systém za daných podmienok zvolí a to z dôvodu zložitosti systému a dynamiky prostredia, v ktorom sa nachádza. Teoreticky takejto predpovedi nič nebráni, ale prakticky toho nie sme schopní[25]

6. Správanie nášho systému nás teda neraz prekvapí, i keď je presne determinované konštrukciou systému. Preto pri vývoji systému je nevyhnutné pracovať so ste-

[24] Sú chrobáky, ktoré keď prevrátite na rovnej a hladkej doske na chrbát nedokážu sa prevrátiť do normálnej polohy. Napriek tomu mykajú nekoordinovane nohami. Výnimočne sa im takto podarí aj rozhojdať sa a prevrátiť.

[25] Plne to potvrdzuje moja vlastná skúsenosť s predvádzaním môjho veľmi jednoduchého mobilného robota, podobného robotu TUTBOT ([Jones - Flynn 1993]). Mal jediný cieľ: sústavne sa pohybovať, t.j. nezaseknúť sa na prekážke. I keď celkovo prezentácia nedopadla najlepšie, v rohu miestnosti urobil akciu, ktorá prekvapila nielen publikum ale i mňa samého. To som sa naozaj nečakal.

lesneným prototypom, systém sa nedá najprv vyvinúť niekde na papieri a potom skonštruovať.

7. Systém novej umelej inteligencie dokáže svojím správaním nielen prekvapiť svojho tvorcu, ale dokonca prekvapiť ho priaznivo. To nastane v momente, kedy začne adekvátne reagovať v situácii, ktorú sme pri návrhu systému neuvažovali. Túto vlastnosť nazývame **emergenciou**, t.j. vynorením sa, samoobjavením sa. Boli sme to pochopiteľne my, kto do systému túto správnu reakciu vložil, ale vložili sme ju tam implicitne a nevedomky ako dôsledok vedomého a explicitného vkladania aktivít pre riešenie iných situácií. Niekedy potom dokážeme spätne usúdiť, prečo tak systém reaguje, ale vedieť to vopred nie sme schopní. Ide teda o našu neschopnosť urobiť niečo, čo je teoreticky možné.

Keď si spomenieme na podobenstvo s krájaním koláča pri tvorbe systému tradičnej umelej inteligencie, spomenieme si na predsavzatie dekomponovať systém novej umelej inteligencie nejakým celkom iným spôsobom. Vzhľadom na reaktívnu povahu systému ponúka sa systém rozdeliť na moduly zabezpečujúce tieto reakcie. Tieto moduly sa môžu spájať do väčších celkov, ktoré zabezpečujú reakcie týkajúce jednotlivých aktivít systému (dekompozícia aktivitou [Brooks 1986] [Kelemen 1994]). Napr. autonómny mobilný systém sa spravidla dekomponuje na aktivity:

- vyhýbanie sa prekážkam
- túlanie sa
- prehľadávanie priestoru
- budovanie mapy prostredia
- ...

Výhoda tejto dekompozície spočíva v tom, že akcie volí takto dekomponovaný systém majú spravidla atomickú povahu, napríklad „zapni motor“, „vypni motor“ a podobne. Problém rámca tu takisto nehrozí, lebo každá aktivita si z receptorov berie práve tie informácie, ktoré sú pre ňu relevantné.

Teraz už máme akú takú predstavu o tom, ako má byť systém vo svojej finálnej podobe dekomponovaný, avšak ešte nevieme, ako túto dekompozíciu dosiahnuť. Na začiatku tvorby systému podľa zamýšľaného cieľa[26] získame pomerne dobrú predstavu o jednotlivých aktivitách systému[27], avšak my musíme moduly zabezpečujúce tieto aktivity poprepájať tak, aby bol medzi nimi súlad, aby sa vždy dostala na základe určitej aktivity ku slovu tá reakcia, ktorá je adekvátna zmenám v prostredí. Nič iné nám teda neostáva ako použiť metódu zdola-nahor a tieto aktivity spájať do väčších celkov až dostaneme celý systém [Brooks 1986].

[26] upozorňujeme, že tento cieľ je iba rámcovo špecifikovaný

[27] zabudovanie týchto aktivít si však môže vyžiadať pridanie ďalších, t.j. na začiatku nepoznáme úplne všetky aktivity, ktoré systém bude obsahovať po dokončení

Na toto spájanie bolo vyvinutých niekoľko postupov, pričom za najvhodnejší z nich považujeme **inkrementálnu metódu**. Táto metóda tvorby systému bola navrhnutá R. Brooksom pre jeho subsumpčnú architektúru autonómnych robotov [Brooks 1989]. Jej podstatu krátko priblížime na príklade a podrobne sa jej budeme venovať v samostatnej kapitole. Vráťme sa teda k situácii s tanierikom, ktorý sme použili pri demonštrácii rozdielov medzi systémom tradičnej a novej umelej inteligencie. Spomínali sme, že systém novej umelej inteligencie vytiahne tanierik z koša na základe aktivity „tanieriky do koša nepatria". Keď budeme takýto systém testovať, možno sa nám podarí niektorý tanierik pri páde do koša rozbiť. Ten už, pravda, nemá zmysel z koša vyťahovať, ale náš systém sa o to pokúsi[28]. Musíme ho teda opraviť. To urobíme tak, že do systému dodáme aktivitu „rozbité tanieriky si nevšímaj", ktorá zdetekuje, či je tanierik rozbitý, a ak áno, potlačí aktivitu „tanieriky do koša nepatria". Takto opravený systém sa už správa korektnejšie. Takýmto spôsobom zamýšľa nová umelá inteligencia budovať čoraz komplexnejší a komplexnejší systém, až dosiahne systém, ktorý dokáže plniť stanovaný cieľ. Tento cieľ sa pritom premieta do systému v podstate opakovaným prechádzaním systému cez Turingov test, z ktorého vyplynie potreba pridania nových aktivít. Umeleckou časťou inkrementálnej metódy pritom zostáva určité utriedenie aktivít, ktoré determinuje, ktoré aktivity do systému zabudujeme skôr a ktoré neskôr.

3.3 Distribuovaná umelá inteligencia

Distribuovaná umelá inteligencia sa zaoberá v prvom rade pochopením a modelovaním vzťahov medzi živými systémami a až druhotne budovaním umelých inteligentných systémov. Tak ako sa umelá inteligencia čerpá podnety zo psychológie, distribuovaná umelá inteligencia čerpá podnety zo sociológie a ekonómie. Kým umelá inteligencia sa zaoberá jednotlivými systémami, distribuovaná sa zaoberá vždy skupinou takýchto systémov. Skúma teda distribuované systémy, v ktorých základné jednotky sú inteligentné systémy [Singh 1994].

Distribuovaná umelá inteligencia vyvinula pre svoje potreby veľmi silné nástroje a pojmy, aplikovateľné nielen v jej rámci. Kľúčový význam malo nahradenie pasívnej základnej jednotky distribuovaného systému aktívnou, zvanou agent. Táto jednoduchá myšlienka má také široké uplatnenie, že považujeme za oprávnené v súvislosti s ňou hovoriť o **agentovej paradigme**.

Uplatnenie agentovej paradigmy na distribuované systémy následne umožnilo nahradiť centrálne riadený distribuovaný systém decentralizovaným. Vzhľadom na to, že takýto decentralizovaný systém sa spravidla skladá z veľkého počtu agentov, ujalo sa preň pomenovanie **multiagentový systém**.

[28] t.j. vytiahne jeden kúsok

Oba tieto pojmy sú pre túto prácu kľúčové. Treba však poznamenať, že ich budeme chápať v širšom význame, ako je to bežné v distribuovanej umelej inteligencii. Toto rozšírenie spočíva v uplatnení rekurzie.

Pri štandardnom pohľade na hierarchiu multiagentového systému sa zvyknú uvažovať tri hierarchické úrovne:
- moduly
- z modulov sa skladajúce agenty
- z agentov sa skladajúci multiagentový systém

Túžbou tvorcu takéhoto systému je emergencia, t.j. aktivita multiagentového systému by mala byť viac než suma aktivít jednotlivých agentov. Pokiaľ navyše moduly týchto agentov stelesňujú jednotlivé aktivity agenta (t.j. agent je systém novej umelej inteligencie), to isté sa očakáva i na tejto nižšej úrovni.

My túto hierarchiu rozšírime na ľubovoľný počet úrovní, na čo nám stačí urobiť rekurzívny krok spočívajúci v tom, že musíme agenta vyjadriť ako multiagentový systém nižšej úrovne.

Na prvý pohľad sa zdá, že ide o jednoduchú záležitosť. Práce zaoberajúce sa formálnou stránkou hierarchických systémov, spravidla nemajú problém pracovať s ľubovoľným počtom úrovní, kde na každej úrovni sú určité komponenty skladajúce sa z komponentov nižšej úrovne medzi ktorými prebieha lokálna interakcia [Forest 1991]. Pokiaľ však tvoríme nejaký skutočný systém, rekurzívny krok spočíva nielen vo formálnom vyjadrení. My musíme vyjadriť agenta tak, aby to malo význam pri tvorbe systému, t.j. nielen ho rozložiť na časti, ale rozložiť ho na také časti, ktoré sú blízke chápaniu tvorcu.

Ak sa nám toto podarí, priblížime multiagentové systémy nielen hierarchickým formálnym modelom, ale i filozofickým predstavám o konštrukcii mysle od M. Minského ([Minsky 1986]) a D. Dennetta (spoločenstvo kompetitívnych homunkulov) [Kelemen 1994].

Takéto zjednocovanie rôznych predstáv na platforme niečoho, čo je prakticky použiteľné, je podľa nášho názoru viac než sľubné. Pustime sa teda do podrobností.

3.4 Agentová paradigma

Agent je niečo, čo **neustále vníma** svoje prostredie, na základe toho **volí** akcie, ktoré má v tomto prostredí vykonať aby dosiahol určitý **cieľ** a následne tieto akcie **vykonáva** [Doran 1992].

Na vnímanie prostredia slúžia agentovi vhodné receptory (senzory), na vykonávanie akcií vhodné efektory (aktuátory) a na voľbu akcií radič, ktorý prijíma dáta

z receptorov a vysiela dáta na efektory. Pritom pod vhodnými receptormi a efektormi rozumieme tie, ktoré agentovi poskytnú dostatok informácií a dostatok motorických schopností k splneniu určitého cieľa. Dôležité je uvedomiť si, že agent vníma prostredie, volí akcie a vykonáva akcie **neustále**, t.j. v diskrétnom prípade **cyklicky**[29].

Povaha agenta závisí od povahy radiča, t.j. od spôsobu voľby akcií.

Keďže radič môže mať ľubovoľnú štruktúru, klasifikácia agentov ([Goodwin 1993]) nie je jednoduchou záležitosťou. Pre naše úvahy však bude stačiť uvažovať tri druhy agentov: **deliberatívne**, **reaktívne** a **čisto reaktívne**.

Deliberatívny agent má radič, ktorý je deliberatívnym systémom, t.j. akcie volí na základe rozhodovania sa medzi ich predpokladanými následkami.

Reaktívny agent má radič, ktorý je reaktívnym systémom, t.j. akcie volí na základe reakcie na stav prostredia a svoj vnútorný stav.

Čisto reaktívny agent je špeciálnym prípadom reaktívneho agenta. Na rozdiel od neho nemá vnútorný stav, t.j. nepoužíva žiadnu pamäť. Na prvý pohľad sa zdá, že takéhoto agenta môžeme použiť iba na zabezpečenie veľmi jednoduchých aktivít. Neskôr však ukážeme, že koncept takéhoto agenta je rovnako silný ako koncept reaktívneho agenta a že tu ide iba o drobný konštrukčný rozdiel.

Význam agentov široko prekračuje rámec distribuovanej umelej inteligencie. Významnú zmenu prinesie použitie agentov v ktoromkoľvek modulárnom konkurenčnom systéme, t.j. v systéme, ktorý je zložený z modulov či procesov, ktoré sa striedajú pri vykonávaní svojho kódu (striedajú sa v procesore) na základe určitého plánovacieho algoritmu.

Obyčajný modul či proces vykonávajúci tomto systéme určitú funkciu koná práve vtedy, keď je to potrebné, inak spí. Výhoda tohto prístupu spočíva v tom, že je efektívny (z výpočtového hľadiska). Avšak má to i svoju nevýhodu: v systéme musí existovať niekto múdry, ktorý ho v správny okamih zobudí, modul je teda centrálne riadený. Pri centrálnom riadení sa v systéme neudeje nič, čo by jeho tvorcom nebolo vopred presne našpecifikované. Na základe toho, čo sme doteraz v práci uviedli, by malo byť zrejmé, že takéto systémy majú pramalú nádej produkovať komplexné inteligentné správanie. Preto by sme radšej privítali niečo, čo nepotrebuje centrálne riadenie, aj za cenu toho, že to nebude tak efektívne. Presne toto dosiahneme agentom, ktorý sa sám každú chvíľu zobudí, pozrie sa, či netreba vzhľadom na jeho funkciu niečo urobiť a ak treba, tak to urobí.

Ako príklad si uvedieme spôsob čítania dát z nejakého reálneho zariadenia, napríklad sériovej linky. Takéto zariadenie buď neposkytuje údaje, alebo ich poskytuje

[29] t.j. v každom časovom kroku respektíve raz za určitý čas

s určitou rýchlosťou, povedzme 9600 bps[30]. Údaje sa zhromažďujú vo vyrovnávacej pamäti, ktorá má povedzme 1024 bytov. Klasickým spôsobom by určitý modul čítal tieto dáta tak, že by si podal žiadosť na operačný systém, aby ho zobudil, keď nejaké dáta z linky prídu. Pokiaľ dáta z linky nechodia, modul čítania spí. Pokiaľ prídu, operačný systém na základe generovaných prerušení modul zobudí a on si prečíta obsah vyrovnávacej pamäti. Naproti tomu agentovi stačí raz sekundu sa zobudiť a prečítať obsah vyrovnávacej pamäti, prerušenia ho vôbec nezaujímajú. Keďže za sekundu nemôže prísť z linky viac než 960B, čo menej ako je obsah vyrovnávacej pamäti (1024B), nemôže agent žiadne dáta stratiť, pracuje teda rovnako dobre ako klasický modul.

Tento prípad je poučný i z hľadiska efektívnosti systému. Na prvý pohľad sa zdá, že použitie agentov je menej efektívne. Avšak ak je systém dobrý, musí byť predsa navrhnutý tak, aby bolo stále dosť procesorového času pre potenciálne možné čítanie z linky. Teda pokiaľ modul na čítanie linky spí, vo využívaní procesoru musí beztak ostať určitá časová rezerva, ktorá je nevyužitá. Preto povedať o procese, ktorý sa budí i keď to nie je potrebné, že je neefektívny je príliš tvrdé, spravodlivejšie by bolo povedať, že viac využíva rezervovaný procesorový čas.

Ďalší prínos použitia agentov je v uvoľnení väzieb medzi modulmi, tu môžeme smelo hovoriť o „voľnej modularite“. Bežný modul nemôžeme zo systému svojvoľne odstrániť alebo do systému pridať, prípadne ho preniesť do iného systému, lebo pravdepodobne má za cieľ niekoho budiť, alebo je niekým budený. Takáto operácia si vyžiada zmeny aj v ostatných moduloch. S agentom môžeme nakladať oveľa voľnejšie.

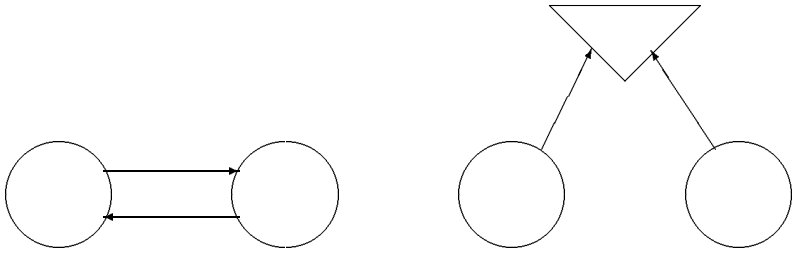
Modularita a konkurenčnosť sú vlastnosti systémov širokého spektra najrôznejších vedných disciplín. Vzhľadom na výhody použitia agentov sa posledné obdobie vyznačuje snahou nahrádzať v týchto systémoch klasické moduly agentmi[31]. Túto snahu a očakávania plynúce z jej dovŕšenia si dovoľujeme v súlade s T. S. Kuhnom nazvať agentovou paradigmou. K tejto paradigme sa hlásia nielen odnože umelej inteligencie, ale i objektovo orientovaného programovania, CIM-u[32], simulačných modelov a hlási sa k nej samozrejme i táto práca.

3.5 Multiagentové systémy

Pod multiagentovým systémom rozumieme systém skladajúci sa z veľkého množstva komunikujúcich agentov.

[30] t.j. 9600 bitov za sekundu, respektíve 960 bytov dát za sekundu (pri jednom štartbite a stopbite)
[31] nie je to však také jednoduché ako by sa mohlo zdať z týchto riadkov
[32] Computer integrated manufacturing

Podstata tejto komunikácie spočíva v konštrukčnom vzťahu agentov a ich prostredia a realizovať ju možno dvoma základnými spôsobmi. Prvý spôsob je založený na predstave, že prostredie agenta tvorí skupina agentov s ktorými vie komunikovať, ide teda o **priamu komunikáciu** medzi agentmi (keďže pri implementácii tejto komunikácie sa agentom zvyčajne priraďujú jednoznačné adresy, nazýva sa i adresnou komunikáciou). Druhý spôsob je založený na predstave, že agenti sú naozaj umiestnení v určitom prostredí a teda nemôžu spolu priamo komunikovať, ide teda o **komunikáciu cez prostredie** (obrázok 1[33]). Prvý spôsob možno prirovnať k používaniu vysielačky, druhý k ponechávaniu si lístkov na dohodnutých miestach. Štandardným spôsobom, ktorý používa distribuovaná umelá inteligencia a poväčšine i sociorobotika je ten prvý. Prvý spôsob je však konštrukčne náročnejší, problematickou sa stáva hlavne synchronizácia agentov a voľnosť modularity je pri ňom v podstate totožná s voľnosťou modularity bežného systému bez agentov. Sme preto horlivými zástancami druhého spôsobu. (Neskôr ukážeme, že tu opäť ide skôr o konštrukčný než filozofický rozdiel.)



priama komunikácia komunikácia cez prostredie

obrázok 1

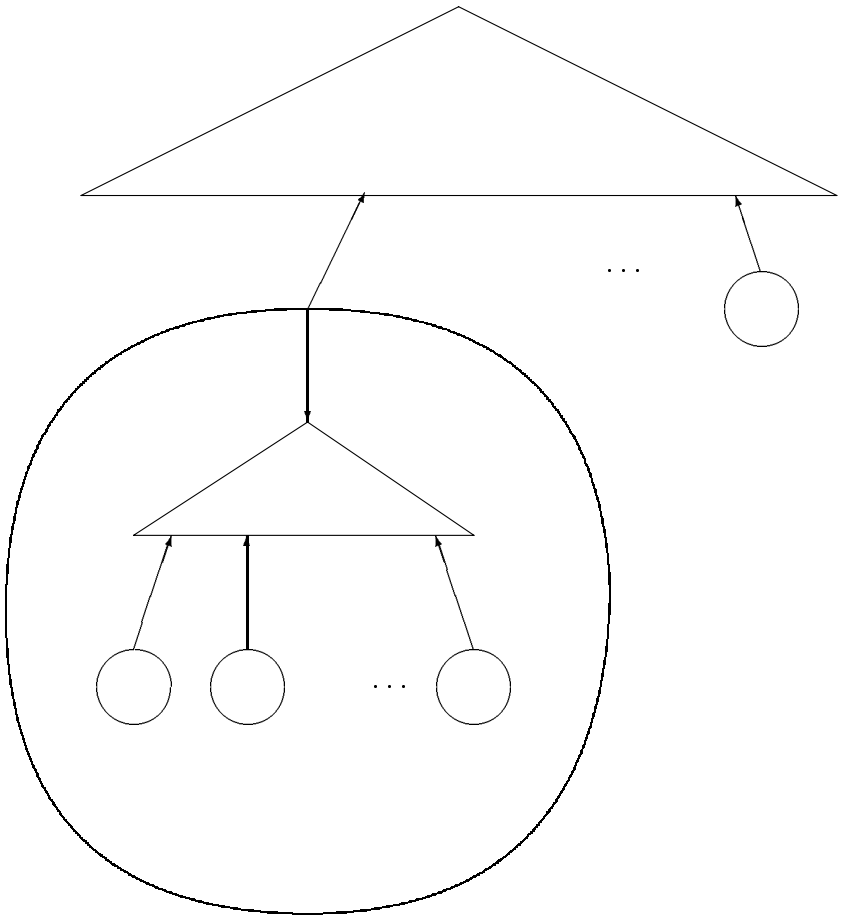
Náš pohľad na komunikáciu medzi agentmi teda nie je celkom štandardný. Zato o našom pohľade na agenta možno vyhlásiť, že je celkom neštandardný. V distribuovanej umelej inteligencii je agent spravidla zložitý deliberatívny systém, v novej umelej inteligencii zase jednoduchý reaktívny. My budeme používať agenta, ktorý je buď atomický, alebo sa viaže k určitej hierarchickej úrovni. Atomický agent je jednoduchý a reaktívny, t.j. taký, aký sa bežne používa v systémoch novej umelej inteligencie. Na druhej strane agent viažúci sa k určitej hierarchickej úrovni je multiagentový systém zložený z agentov nižšej hierarchickej úrovne.

Takto chápaný multiagentový systém zodpovedá Minského predstave o mysli. M. Minsky predpokladá, že myseľ je organizovaným spoločenstvom komunikujúcich

[33] v celej práci agentov kreslíme ako kružnice a prostredia ako trojuholníky, šípka naznačuje vnímanie a vykonávanie akcií

agentov plniacich jednoduché špecifické funkcie. Inteligencia tohto systému nie je výsledkom postupov aké používajú títo agenti, ale spôsobu, ako sú títo agenti prepojení do väčších celkov, do „agentúr" a tieto do ešte väčších, zložitejších a náročnejšie funkcie zabezpečujúcich agentúr. Pritom v inteligentnom správaní vidí M. Minsky v prvom rade správanie komplexné a podstatu systému v spôsobe aktivácie správnych agentúr v správnu chvíľu [Minsky 1986] [Kelemen 1994][34].

Vráťme sa teda k základnému problému a vyjadrime agenta ako multiagentový systém. Už sme spomínali, že urobiť to nejako nie je problém, ale zásadný problém je urobiť to tak, aby to malo prínos pri tvorbe systému.



obrázok 2

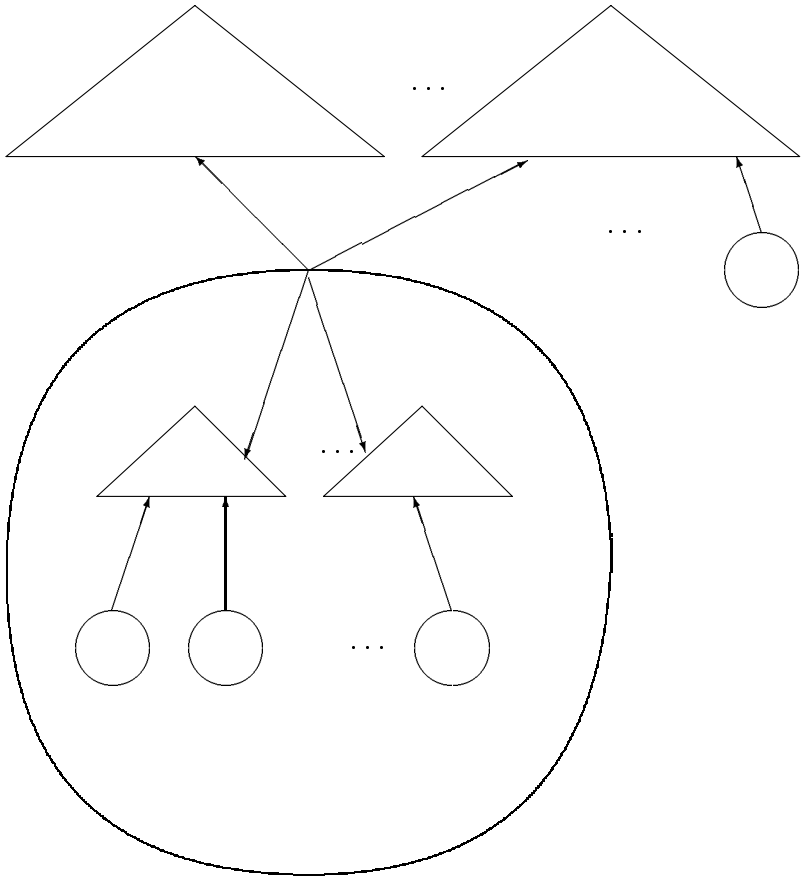
Základná idea[35] spočíva v tom, že takýto agent pracuje (v našom prípade reaguje) nad dvomi prostrediami: nad vonkajším a vnútorným. Nad vnútorným prostredím

[34] V tejto práci budeme tieto myšlienky aplikovať v oblasti programových systémov, pričom náš podiel na zásluhách bude spočívať v použití konkrétneho spôsobu prepojenia a aktivácie agentúr.

[35] Táto idea, napriek tomu, že som sa s týmto problémom dlho trápil, nepochádza z mojej hlavy. Dozvedel som sa ju od J. Kelemena

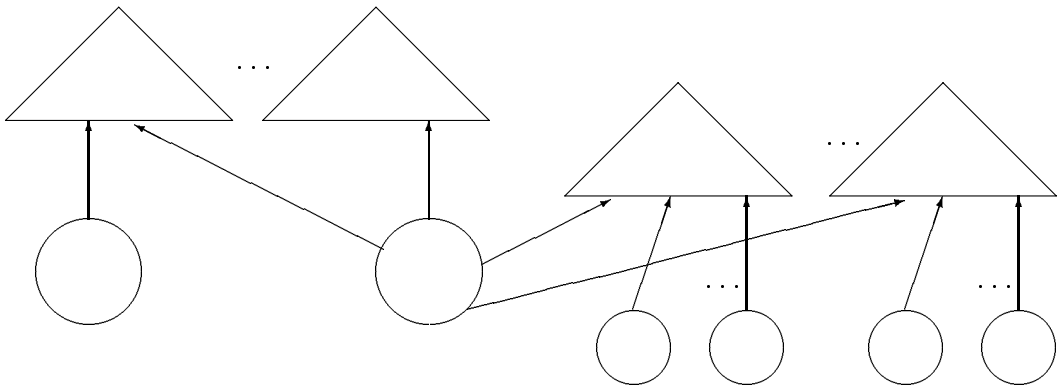
pracujú okrem neho jeho vnútorní agenti, ktorí zabezpečujú určité aktivity nižšej hierarchickej úrovne. Cez vnútorné prostredie agent jednak monitoruje činnosť na tejto úrovni (pričom určitý stav prostredia ho môže priviesť k aktivite vo vonkajšom prostredí), jednak môže ovplyvniť aktivity svojich vnútorných agentov (na základe stavu vonkajšieho prostredia) (obrázok 2). Takýto agent môže byť jednoduchý (napríklad reaktívny) a zároveň môže vykonávať zložitú aktivitu, lebo jednoduchým spôsobom zastrešuje zložitú činnosť multiagentového systému, ktorý sa nachádza akoby v jeho vnútri.

Ďalej sa odpútajme od predstavy, že agent pracuje v jednom prostredí, či v dvoch prostrediach. I v prípade, že uvažujeme fyzické prostredie systému, tento systém vníma toto prostredie rôznymi receptormi, ktoré nie sú nijak prepojené. Z jeho pohľadu je to teda niekoľko rôznych prostredí, dokonca by sa mohli nachádzať i na rôznych hierarchických úrovniach. Agent teda v našom ponímaní môže operovať nad viacerými prostrediami svojej alebo nižšej hierarchickej úrovne. (obrázok 3)



obrázok 3

Určitý problém tu predstavuje enkapsulácia, t.j. či ostatní agenti nachádzajúci



obrázok 4

sa na tej istej hierarchickej úrovni ako neatomický agent môžu „vidieť do jeho vnútra“, prípadne „robiť zásahy v jeho vnútri“. Porušenie enkapsulácie (pokiaľ sa ukáže výhodné[36]) však nepovažujeme za zrútenie našej predstavy o hierarchii. Hierarchia tým iba stráca „poriadok“, tvorca systému získava väčšiu voľnosť.

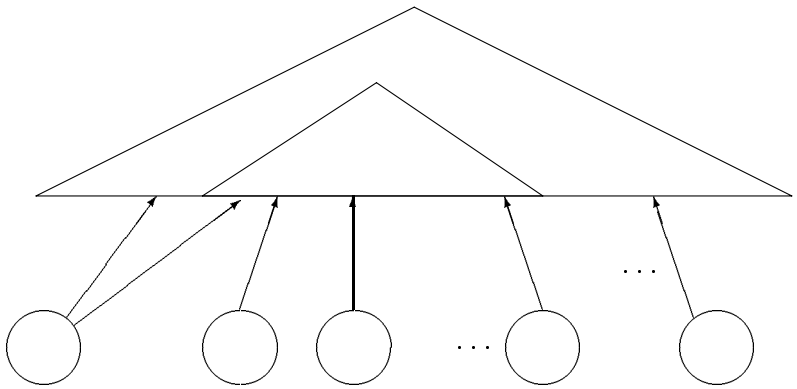
To, že takto chápaná hierarchia je aj prakticky užitočná ukážeme, keď sa budeme zaoberať inteligentnými programovými systémami. Objavíme i to, že už dávno je prítomná v robotických systémoch novej umelej inteligencie. Teraz si ukážeme len toľko, že táto hierarchia je blízka uvažovaniu tvorcu systému. Keďže na každej hierarchickej úrovni ide o tie isté komponenty líšiace sa akurát zložitosťou aktivít, ktoré zabezpečujú, môžeme túto hierarchiu rozvinúť do tvaru, v ktorom sú agenty, ktoré predtým boli „pod sebou“, „vedľa seba“ (obrázok 4).

Táto konštrukcia je voľnejšia a na porušenie enkapsulácie pri nej nepotrebujeme žiadny prídavný mechanizmus. Pravda, možno namietať, že takáto konštrukcia sa už dosť podobá pôvodnej predstave bez hierarchie, ktorú dostane tak, že zjednotíme všetky prostredia do jediného (obrázok 5).

Výhoda nášho riešenia spočíva hlavne v prínose pre tvorbu systému inkrementálnou metódou. Hierarchia nás teda nebude až tak trápiť z hľadiska konštrukcie, zato bude pre nás podstatná z hľadiska postupnej tvorby systému.

3.6 Inkrementálna metóda

[36] náš predpoklad je v tejto chvíli taký, že to výhodné bude, nakoľko nám to umožní v systéme používať „agentov-parazitov“



obrázok 5

V predchádzajúcej kapitole sme sa zaoberali konštrukciou inteligentného systému, poďme sa teraz zaoberať metódou jeho tvorby. Spomínali sme už, že vzhľadom na implicitnú povahu cieľa nie možné použiť ľubovoľnú metódu, ktorá dodrží uvažovanú konštrukciu. Metóda musí zabezpečiť, že vývoj systému bude k danému cieľu smerovať. Jedným z možných spôsobov ako to zabezpečiť, je použitie **inkrementálnej metódy**.

Inkrementálna metóda spočíva v postupnom budovaní a testovaní systému. Oproti bežnému postupu sa táto metóda líši v tom, že sa systém testuje z hľadiska plnenia cieľa (funkčnosti) prv ako je celý vybudovaný. Použitie tejto metódy predpokladá na začiatku určitú predstavu o hierarchii aktivít v systéme, a o adekvátnych prejavoch komponentov zabezpečujúcich tieto aktivity, t.j. o parciálnych, hierarchicky nižšie položených cieľoch.

Systém začneme budovať od najjednoduchších aktivít. Akonáhle máme niektorú novú aktivitu zabudovanú, otestujeme, či na danej hierarchickej úrovni pozorujeme adekvátne prejavy.

Tu sa vtiera otázka „A čo by sme tam asi tak pozorovali ? Veď tam musí byť to, čo sme tam zabudovali !“ To je síce pravda, že je tam to, čo sme tam zabudovali, ale to ešte nemusí byť to, čo tam chceli zabudovať. Toto je podstatný fakt, keď nami tvorený systém má pracovať v prostredí, ktoré sme my nevytvorili a nemôžeme sa spoľahnúť na to, že jeho dynamika je nám úplne známa. Na prvý pohľad sa táto úvaha týka iba robotických systémov, ale ak si odmyslíme teóriu, týka sa i programových systémov (kde rôzne časti obsahujú rôzne neznáme chyby). Nevyhnutnou podmienkou tvorby systému inkrementálnou metódou je teda **stelesnenosť**, návrh musí ísť ruka v ruke s implementáciou.

Pokiaľ teda systém javí adekvátne prejavy, môžeme pristúpiť k zabudovaniu ďalších hierarchicky vyšších a vyšších aktivít. Pokiaľ nie, musíme špecifikovať príčinu (t.j stav prostredia, ktorý je pre náš výtvor nepovolený, ale my chceme aby bol povo-

lený) a poopraviť realizáciu pridanej aktivity alebo pridať novú aktivitu, ktorá ošetrí kritický stav.

Pritom činnosť hierarchicky vyššej aktivity je založená na **„vtieraní sa“** do činnosti hierarchicky nižších aktivít. Toto vtieranie sa je zabezpečené tým, že moduly zabezpečujúce hierarchicky vyššiu aktivitu, môžu určitým spôsobom manipulovať s tokmi dát medzi modulmi zabezpečujúcimi hierarchicky nižšie aktivity.

Konkrétny spôsob vtierania sa závisí na konkrétnej architektúre, presnejšie povedané konštrukcii systému (lebo teraz uvažujeme len tie architektúry, ktoré ako metódu tvorby systému používajú inkrementálnu metódu). Demonštrujeme ho na dvoch prípadoch. Prvým bude známa subsumpčná architektúra, druhým architektúra multiagentového systému s konštrukciou, ktorú sme navrhli v predchádzajúcej podkapitole.

3.6.1 Subsumpčná architektúra

[Brooks 1989]

Subsumpčná architektúra R. Brooksa bola prvou robotickou architektúrou[37], ktorá umožnila skonštruovať autonómne roboty[38] operujúce v dynamickom a prirodzenom prostredí.

Konštrukcia systému je podľa tejto architektúry postavená na modularite, pričom tieto moduly sú jednoduché: ide o konečno-stavové automaty rozšírené o vnútorné časovače, vnútorné výpočtové procedúry, vstupno-výstupné registre a interné registre (pamäť). Komunikácia medzi modulmi prebieha posielaním signálov (správ).

Tvorba systému je, ako sme už uviedli pri subsumpčnej architektúre, inkrementálna. Predpokladá rozdelenie systému na postupne implementované vrstvy, pričom neskoršie implementované vrstvy sa vtierajú do činnosti tých skôr implementovaných. Toto vtieranie sa je zabezpečené dvomi mechanizmami (resp. dvomi obdobami jedného mechanizmu): inhibíciou a supresiou.

Inhibícia spočíva v potlačení výstupu z modulu v skôr vybudovanej vrstve na základe signálu z neskôr vybudovanej vrstvy, ide teda o odstavenie určitého modulu z činnosti.

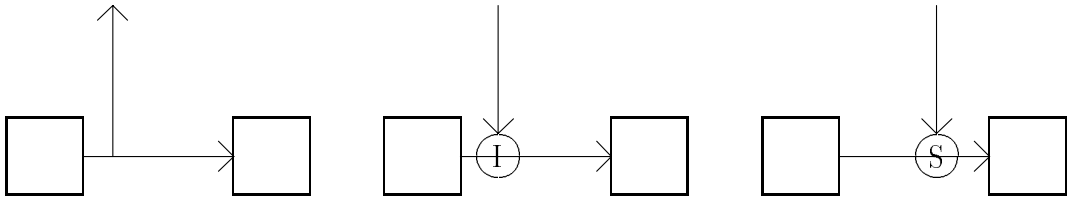
Supresia spočíva v nahradení vstupu do modulu v skôr vybudovanej vrstve signálom z neskôr vybudovanej vrstvy, ide teda o dočasné prepojenie vstupu do určitého modulu s výstupom z iného modulu.

Okrem toho majú moduly z neskoršie vybudovanej vrstvy možnosť priviesť si na vstup výstup z ktoréhokoľvek modulu skôr vybudovanej vrstvy.

[37] nie však jedinou [Mlichová 1993]

[38] ALLEN, TOM a JERRY, SQUIRT, HERBERT, TOTO, a ďalšie

Pri fyzickej realizácii sú moduly v rámci jednej vrstvy prepojené obyčajným dátovým vedením (drôtom), zatiaľ čo prepojenia medzi vrstvami sú realizované buď zdvojením dátového vedenia alebo špeciálnymi hardwarovými zariadeniami (inhibítormi a supresormi), ktoré sa podľa potreby vložia do už existujúcich obvodov vybudovaných úrovní (obrázok 6).



obrázok 6

Vstup a výstup niektorých modulov je prepojený s receptormi a efektormi bez ohľadu na vrstvu, v ktorej sa modul nachádza. Tým je umožnené i dodatočné rozšírenie receptorov a efektorov a teda i potenciálnych schopností robotického systému[39].

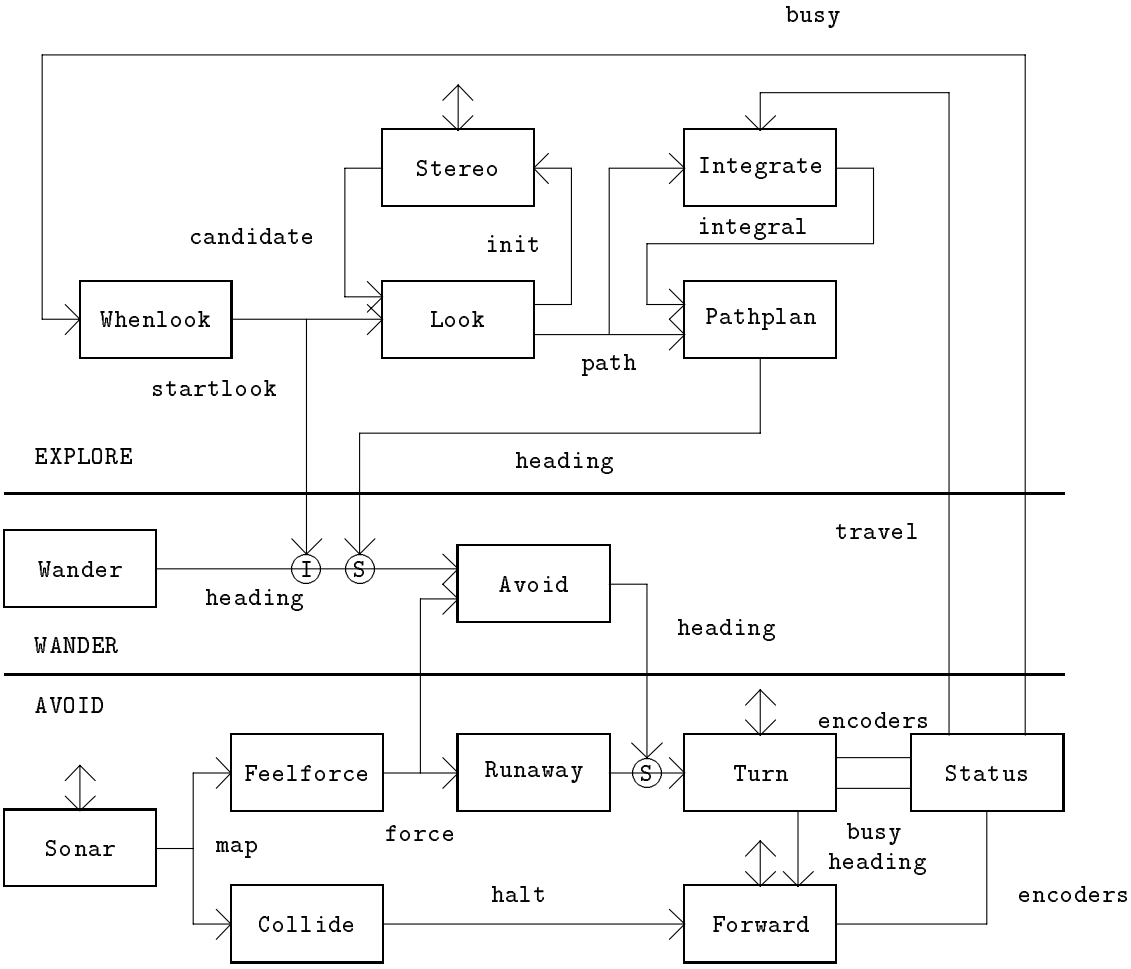
Ako ukážku[40] systému so subsumpčnou architektúrou si predvedieme pomerne jednoduchý robotický systém ALLEN.

ALLEN je zložený z troch vrstiev: `AVOID`, `WANDER` a `EXPLORE` (obrázok 7). Jeho cieľom je prečesávať prostredie a dostávať sa ňom na nové miesta.

Jeho prvá implementovaná vrstva je vrstva `AVOID`. Začína modulom `Sonar`, ktorý prijíma dáta z dvanástich sonarov snímajúcich na dvanásť častí rozdelených 360°. Každý z týchto sonarov dokáže vo svojom zornom uhle určiť vzdialenosť najbližšieho predmetu, na základe čoho modul `Sonar` vybuduje mapu relatívnych smerov a vzdialeností blízkych predmetov. Časť týchto informácií (predný výrez) vstupuje potom do modulu `Collide`, ktorý vyhodnotí, či už nastal, respektíve či už už nastáva náraz a v tom prípade spôsobí v module `Forward`, ktorý ovláda zadné kolesá robota, spätný chod na určitý čas, ak to povolí stav natočenia predných kolies z modulu `Turn`. Mapa taktiež vstupuje do modulu `Feelforce`, ktorý podľa nej zráta príťažlivosť smerov a vyberie najpríťažlivejší smer (t.j. smer v ktorom najviac hrozí zrážka). Modul `Runaway` sa na základe tohto smeru snaží vplývať na modul `Turn` riadiaci predné kolesá vychyľujúce smer jazdy tak, aby sa išlo z tohoto smeru preč. Smer chodu zadných

[39] Táto vlastnosť je veľmi dôležitá, lebo pôvodné receptory a efektory sa môžu ukázať ako nedostačujúce

[40] táto ukážka nie je samoúčelná, použijeme ju na zviditeľnenie analógií medzi nami navrhovanou architektúrou multiagentového systému a subsumpčnou architektúrou



obrázok 7

kolies robota, natočenia predných kolies a indikátor, ktorý indikuje, že modul `Turn` pracuje na uniknutí prekážke, sú monitorované (pre využitie neskoršie zabudovanými vrstvami) v module `Status`[41]. Vrstva `AVOID` je samostatne schopná zabezpečiť chod robota bez nárazov, s vyhýbaním sa prekážkam a dokonca i únik pred dynamickou prekážkou. Pokiaľ sa však v okolí robota nevyskytne prekážka, pôjde robot stálym smerom.

Vrstva `WANDER` má za úlohu zabezpečiť zaujímavejší pohyb robota, t.j. túlanie. Modul `Wander` vysiela v pravidelných intervaloch návrhy na otočenie o náhodne generovaný uhol do modulu `Avoid`, ktorý zároveň prijíma signály z modulu `Feelforce` a ktorý, ak netreba unikať pred priveľmi príťažlivým smerom, supresuje signál prichádzajúci do modulu `Turn` z modulu `Runaway` smerom, ktorý navrhol modul `Wander`.

Vrstva `EXPOLORE` robí pohyb robota ešte zložitejší. Zabezpečuje, aby robot pri

[41] Toto monitorovanie by však mohlo urobiť aj neskôr, t.j. tvorca by nemusel naň v čase tvorby tejto vrstvy bezpodmienečne myslieť.

pohybe nezotrvával v malej oblasti, ale aby orientoval svoj pohyb do oblastí, ktoré ešte „nepreskúmal“. Na to potrebuje dlhšiu dobu držať určitý absolútny smer smer a to s i ohľadom na vyhýbanie sa prekážkam. Modul `Whenlook` detekuje podľa signálu z modulu `Status`, že práve neprebieha ani vyhýbanie sa prekážkam, ani náhodná zmena smeru pri túlaní a raz za čas sa rozhodne vydať sa do inej oblasti. Vtedy prostredníctvom modulu `Look` aktivuje modul `Stereo`, ktorý na základe informácií zo sonarov hľadá voľný smer, v ktorom sa dá dlhšie pohybovať. Aby počas tohto hľadania nedošlo k zbytočnému otáčaniu sa robota, je už počas hľadania inhibovaný výstup z modulu `Wander`. Po čase sa modulu `Stereo` podarí navrhnúť kandidáta. Pokiaľ ho modul `Look` prijme, nevšíma si určitý čas signály prichádzajúce z modulu `WhenLook` a aktivuje modul `Pathplan`. Ten zabezpečuje natočenie sa robota do správneho smeru supresiou vstupu do modulu `Avoid`, t.j. zabezpečuje akési „riadené túlanie“. V prvom momente mu stačí navoliť smer, ktorý mu určil modul `Look`. Ako sa však robot začne otáčať, potrebuje vedieť nakoľko sa už robot k danému smeru priblížil. Takisto pri obchádzaní prekážok musí vedieť vrátiť robota do správneho smeru. Na to mu slúži modul `Integrate`, ktorý od momentu zvolenia smeru počíta na základe informácií o otáčaní z modulu `Status` rozdiel aktuálneho smeru robota od zvoleného smeru.

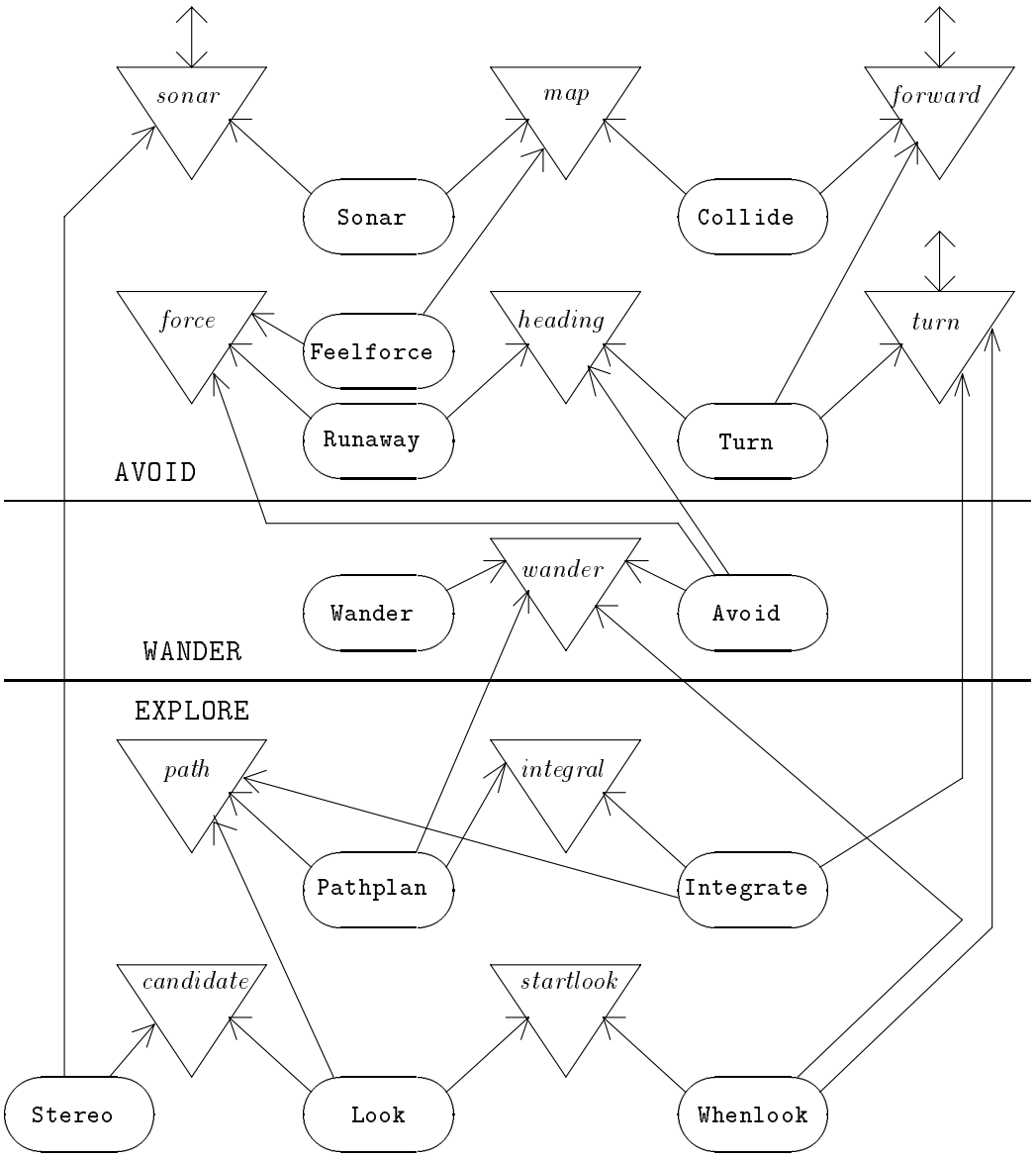
3.6.2 Architektúra multiagentového systému

Uvažujme teraz náš multiagentový systém. Z konštrukčného hľadiska je vybudovaný z agentov a prostredí, pričom priamo komunikujú iba agenti s prostrediami. Prostredia slúžia na uchovávanie odkazov, ktoré zanecháva jeden agent druhému, avšak nemôže zabrániť tomu, aby odkaz prečítal iný agent alebo ho nahradil iným odkazom. Okrem toho niektoré prostredia slúžia na poskytnutie údajov z fyzických receptorov a na príjem údajov pre riadenie fyzických efektorov. Ku všetkým prostrediam agenti pristupujú jednotne: môžu z nich údaje prečítať a môžu do ich údaje zapísať.

Pri tvorbe systému postupujeme podľa všeobecných pravidiel inkrementálnej metódy. Vtieranie sa neskôr vybudovaných (hierarchicky vyšších) aktivít do činnosti skôr vybudovaných (hierarchicky nižších) aktivít je realizované na základe čítania a zápisu nových agentov do už vybudovaných prostredí. Čítaním odkazov použitých pri komunikácii v nižšej vrstve môžu agenti monitorovať činnosť nižšej vrstvy a na základe toho sa do jej činnosti votrieť prepísaním alebo vymazaním (t.j. prepísaním na nejakú štandardnú hodnotu) týchto odkazov.

Robotický systém ALLEN by sa v tejto architektúre zrealizoval tak, ako to vidíme na obrázku 8.

Opäť by sme mali tri hierarchické vrstvy `AVOID`, `WANDER` a `EXPLORE`. Vo vrstve `AVOID` by agent `Sonar` čítal údaje poskytované „fyzickým“ prostredím *sonar*, na zá-



obrázok 8

klade čoho by do prostredia *map* zapisoval mapu relatívnych smerov a vzdialeností blízkych predmetov. Túto mapu by čítal agent `Collide`, ktorý by v prípade, že hrozí náraz, zapísal do prostredia *forward* povel na spätný chod zadných kolies. Ten by mu mohol prípadne prepísať agent `Turn`, pokiaľ by bol spätný chod neprípustný, vzhľadom na natočenie predných kolies. Mapu v prostredí *map* ďalej číta agent `Feelforce`, na základe čoho zapisuje do prostredia *force* najpríťažlivejší smer. Ten odtiaľ číta agent `Runaway`, ktorý zase do prostredia *heading* zapisuje smer, ktorým treba unikať pred prekážkou. Ten číta agent `Turn`, ktorý na jeho základe zapisuje do „fyzického“ prostredia *turn* smer, ktorým sa majú natočiť predné kolesá.

Vrstva `WANDER` obsahuje jediné prostredie *wander*, do ktorého zapisuje agent `Wan-`

`der` náhodne vygenerovaný smer. Ten číta agent `Avoid`, ktorý ak podľa obsahu prostredia *force* neusúdi, že sa treba vyhýbať prekážke, prepisuje týmto smerom smer uložený v prostredí *heading*.

Vrstvu `EXPLORE` aktivuje agent `Whenlook` sledujúci povely prichádzajúce do prostredia *turn*, zapísaním povelu na vybratie smeru do prostredia *startlook*. Zároveň v prostredí *wander* vymazáva smer zvolený agentom `Wander`, t.j. prepisuje ho hodnotou, ktorou si agent `Avoid` v prostredí značí, že už prečítal odkaz agenta `Wander`. Prostredie *startlook* číta agent `Look`, ktorý pokiaľ si v ňom nájde povel na vydanie sa určitým smerom, prečíta si kandidáta z prostredia *candidate* a zvolí smer, ktorý zapíše do prostredia *path*. Pritom hľadanie kandidáta zabezpečuje agent `Stereo` na základe vnímania fyzického prostredia *sonar*, na rozdiel od subsumpčnej architektúry však hľadá tohto kandidáta stále, t.j. agent `Look` namiesto inicializácie hľadania prevezme momentálny návrh[42]. Zvolenie smeru z prostredia *path* zdetekuje agent `Integrate`, ktorý neustále počíta na základe vnímania fyzického prostredia *turn* rozdiel aktuálneho a zvoleného smeru, na základe čoho toto počítanie resetuje. Výsledok počítania pritom neustále zapisuje do prostredia *integral*. Na základe zvoleného smeru z prostredia *path* a rozdielu aktuálneho smeru od tohto smeru v prostredí *integral* agent `Pathplan` určuje správny momentálny smer a prepisuje ním smer v prostredí *wander*. Aby túto hodnotu nezmazal agent `Whenlook` musí agent `Look` pri zvolení smeru poznačiť do prostredia *startlook*, že zvolenie už prebehlo a agent `Whenlook` musí na to adekvátne zareagovať.

Toto samozrejme je len rámcový popis systému, lebo zatiaľ nám chýba znalosť presného spôsobu komunikácie agentov a prostredí, časovania agentov a štruktúry prostredí. K týmto parametrom sa viažu rôzne problémy. V prvom rade ide o synchronizáciu. Bez synchronizácie napríklad agent `Wander` dokáže v prostredí *wander* prepísať nielen štandardnú hodnotu od agenta `Whenlook` ale i zvolený smer od agenta `Pathplan`. Tento problém sa dá vyriešiť:

- synchronizáciou procesov, t.j. `Avoid` bude čítať až keď postupne zapíšu `Wander`, `Whenlook` a `Pathplan`. Takýto prístup si žiada jednotné časovanie agentov pričom v jednom časovom okamihu sa agenti dostávajú k slovu v poradí, v akom boli do systému postupne pridávaní. Väčšie časové periódy si potom agenti musia odvodiť z tohto základného časovania, napríklad agent `Wander` bude dekrementovať počítadlo a keď klesne na nulu, zapíše do prostredia *wander* svoj návrh a nastaví ho. Môžu taktiež použiť generátor náhody, v prípade agenta `Wander` by tu išlo o zapísanie návrhu s určitou malou pravdepodobnosťou (tento prístup je z hľadiska budovania zaujímavo sa správajúcich systémov účinnejší ako používanie počítadla [Lúčny 1994]).

[42] tento spôsob je „agentovejší“

- prídavnými informáciami v štruktúre prostredí. Napríklad by sme ku každému odkazu v prostredí držali informáciu o priorite agenta, ktorý ho tam zapísal. Prepísať túto hodnotu by mohli iba agenti s väčšou prioritou. V tomto prípade by si nemohli agenti čítajúci takýto odkaz v prostredí poznačiť, že ho prevzali, tým, že by ho prepísali štandardnou hodnotou, takže toto by si museli poznačiť ako ďalšiu prídavnú informáciu. Počet takýchto prídavných informácii by nebol triviálny.

- rôznym časovaním agentov a ich naprogramovaním tak, aby vždy keď sa dostanú ku slovu prehodnotili celú svoju činnosť. Keď budeme časovať agenta `Pathplan` oveľa rýchlejšie ako agenta `Wander`, nepotlačíme síce vplyv agenta `Wander` na smer jazdy, ale zato jeho vplyv znížime na minimum. Takéto uvažovanie má ďalekosiahle následky, vedúce k rámcovému zachovaniu správania systému a pritom k zjednodušeniu jeho konštrukcie i návrhu. Napríklad nemá zmysel agent `Whenlook` potláčal činnosť agenta `Wander` počas hľadania vhodného kandidáta na smer pohybu do novej oblasti. Jednak ani zvolenie kandidáta nezaručí potlačenie činnosti agenta `Wander`, jednak sa tento agent beztak nedostáva k slovu veľmi často.

V tejto práci preferujeme posledné riešenie a máme na to tri dôvody:

- Prvé dve riešenia sú iba čiastočné, neplatia všeobecne. Kontrapríkladom sú dve prostredia nad ktorými operujú tí istý agenti a povaha údajov v prostrediach vyžaduje v každom z nich iné poradie prístupu agentov, respektíve inú prioritu agentov. Tento kontrapríklad je, pravda, z praktického hľadiska príliš vyšpekulovaný, ale predsa to len znižuje našu náchylnosť tieto riešenia akceptovať.

- Uplatnením posledného riešenia získame „vtieranie sa“ novej kvality. Tu už nejde o zablokovanie nižšej vrstvy a jej nahradenie vyššou vrstvou[43], t.j. vyriešenie špeciálneho prípadu na inom mieste. Tu ide o aktiváciu vyššej vrstvy a na vyriešenie špeciálneho prípadu v spolupráci s nižšou vrstvou. Schopnosti nižšej vrstvy teda ostávajú k dispozícii.

- Posledné riešenie najlepšie vyhovuje možnostiam konkurenčných operačných systémov (s časovaním)[44]. Keďže v tejto práci nám vlastne nejde všeobecne o multiagentové systémy ale o ich inštanciu, ktorá by bola novou generáciou programových systémov, na tomto rázcestí si vyberáme cestu, o ktorej predpokladáme, že na nej spomínanú inštanciu nájdeme.

3.6.3 Aplikácia agentovej paradigmy na subsumpčnú architektúru

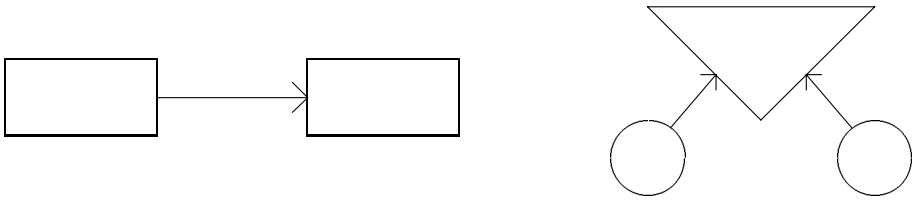
Na prvý pohľad je zrejmé, že medzi obomi uvádzanými architektúrami je silná korelácia. Dá sa povedať, že obe poskytujú iný pohľad na tú istú vec. To, že sme

[43] ako je to pri subsumpčnej architektúre

[44] na ktorých prevádzkujeme programové systémy

použili druhý pohľad, má svoj význam. V porovnaní so subsumpčnou architektúrou sme získali jednak jednotný komunikačný mechanizmus (založený na komunikácii cez prostredie), jednak trochu iné „vtieranie sa“. V prvom rade sme však dostali pohľad, ktorý môžeme aplikovať na programové systémy. Zatiaľ sme síce nič nepovedali o komunikačných mechanizmoch, ktoré sa pri programových systémoch uplatňujú, ale predstava vedenia (drôtu), aspoň pre moderné programové systémy[45] nie je vyhovujúca.

O vzájomnom vzťahu týchto dvoch architektúr možno povedať, že našu architektúru môžeme dostať zo subsumpčnej aplikovaním agentovej paradigmy[46]. Teda moduly (presnejšie povedané väčšinu modulov) v subsumpčnej architektúre môžeme nahradiť agentmi. Zaujímavé si je všimnúť ako sa pritom zjednocuje povaha prepojení medzi modulmi. Pôvodné jednoduché vedenie (drôt) sa transformuje na prostredie obsahujúce určitý odkaz (obrázok 9). Zdvojené vedenie je to isté, iba k prostrediu pribudne ďalší čitateľ. Vedenie s inhibítorom je opäť to isté, iba pribudne jeden zapisovateľ-vymazávateľ. Takže už nás neprekvapí ani to, že vedenie so supresorom je zase prostredie, iba pribudne jeden jeho zapisovateľ-prepisovateľ.



obrázok 9

Pravda, sú tu i problémové špeciality. V prípade robota ALLEN je to napríklad modul `Status`, ktorý monitoruje stavy efektorov Turn a Forward a poskytuje ich iným modulom. V našej architektúre je toto typické prostredie, nakoľko však prepojenie s efektormi zabezpečujú „fyzické“ prostredia, môžu dotyční agenti získať tieto informácie priamo v týchto prostrediach. Teda nie každý modul sa v transformácii zachová.

Náročnejší problém predstavujú vedenia obsahujúce väčší počet inhibítorov a supresorov. Poradie týchto zariadení na vedení totiž zabezpečuje synchronizáciu, resp. prioritu modulov. My sme sa však v našej architektúre vydali inou cestou. Myslíme si že to bolo správne. Veď prečo by sme sa snažili silou-mocou zaviesť poriadok niekde, kde každý element robí len to, čo považuje za najprospešnejšie pre celý systém ?

[45] t.j. také ktoré používajú posielanie správ

[46] V skutočnosti som však najprv navrhol túto architektúru a až potom som objavil jej vzťah k subsumpčnej architektúre

Kapitola 4

Tézy dizertačnej práce

V predchádzajúcom sme načrtli východiská s ktorými vstupujeme do fázy aplikácie vybraných myšlienok novej umelej inteligencie na tvorbu programových systémov, ktoré by sme mohli v zmysle uvedených kritérií považovať za inteligentné programové systémy. Uvedieme teraz etapy, ktorými budeme v ďalšej práci prechádzať.

4.1 Komunikačný model

Prv, než vstúpime do spomínanej fázy aplikácie, musíme analyzovať existujúce programové systémy a to v prvom rade z hľadiska komunikácie medzi procesmi. Budeme si musieť zvoliť presný komunikačný model. V úmysle máme použiť komunikáciu cez posielanie správ a triggery. Z hľadiska syntaxe budeme používať komunikačný model operačného systému QNX4, ktorý považujeme z tohto hľadiska za najmodernejší. Posielanie správ budeme teda rozlišovať blokujúce (vzťah procesov Sender-Receiver, primitívy Send, Receive a Reply) a neblokujúce (vzťah procesov ProxyUser-ProxyOwner, primitívy Trigger a Receive).

4.2 Model časovania

Zvolenie operačného systému QNX4 ako vzoru je výhodné i pre model časovania procesov, nakoľko ide o operačný systém umožňujúci kvalitnú prácu v reálnom čase. Časovanie procesu bude teda zabezpečené cez virtuálny proces timer.

4.3 Implementačné prostredie

Otázkou ostáva nakoľko bude možné použiť operačný systém QNX4 pre naše experimenty. Problematickým je tu počet procesov, lebo tento je v systémoch aké sa chystáme navrhovať ďaleko vyšší ako obyčajne. Štandardne sa jadro QNX4 konfiguruje na 150 procesov, sú nám známe i dobre fungujúce aplikácie s 500 procesmi, avšak ťažko povedať čo sa stane pri väčšom počte procesov. Navyše z principiálnych dôvodov nemožno počítať s viac ako 65535 procesmi, vrátane procesov operačného systému a virtuálnych procesov. Samozrejme bežného programátora nikdy nenapadne používať také počty procesov, lebo pri zložitejších komunikačných vzťahoch sa už len ťažko dajú udržať v poriadku dátové toky, začínajú sa objavovať dátové nekonzistentnosti. My sme sa už však rôznych predsudkov zbavili a javy normálne neželané plánujeme využiť na náš prospech.

4.4 Komunikačné architektúry

Nad zvoleným komunikačným modelom popíšeme bežne používanú architektúru, založenú na vzťahu Client-Server a hierarchických úrovniach s obmedzením smeru tohto vzťahu (pri neviazanej komunikácii vznikajú komunikačné problémy ako napríklad deadlock). Pritom akékoľvek vzťahy medzi procesmi budeme definovať na základe konštrukcie procesov, pre ktorú budeme používať bežnú syntax jazyka C. Naznačíme ďalšie komunikačné problémy architektúry Client-Server a spôsoby ich riešenia, pričom sa zameriame na tie, pri ktorých sa objavujú techniky podobné tým technikám, na ktorých sa budeme snažiť vybudovať inteligentné programové systémy.

Na základe štúdia týchto techník a našej predstavy multiagentového systému zavedieme architektúru, ktorej sme dali pracovný názov **Agent-Environment** na základe komunikačného vzťahu agenta s prostredím, ktorý je špeciálnym prípadom vzťahu Client-Server. Táto architektúra sa na jednej strane vyznačuje komunikačnou bezproblémovosťou, na strane druhej jej konštrukcia spĺňa naše predstavy o inteligentných programových systémoch.

V danej chvíli nastane problém akého agenta v tejto architektúre používať, t.j. ako sa v agentoch volia akcie (prostredia sú z hľadiska konštrukcie definovateľné v podstate jediným spôsobom). Na základe prínosu pre bežné vlastnosti programových systémov ako sú modifikovateľnosť, konfigurovateľnosť a rekonfigurovateľnosť počas behu, prekvapujúco vyjde v tomto výbere víťazne čisto reaktívny agent, ktorého použitie dáva možnosť kedykoľvek bezo zmeny jeho kódu zmeniť nielen jeho vplyv v systéme (t.j. inhibovať alebo supresovať jeho činnosť), ale i jeho samotné správanie.

Súčasťou architektúry Agent-Environment je samozrejme i metóda tvorby systému založená na inkrementálnej metóde.

4.5 Experimenty

Naším zbožným želaním je samozrejme doviesť celú vec do stavu experimentov. Tu je zásadný problém vo výbere cieľa experimentálne implementovaných systémov. Ponúkajú sa nasledovné možnosti:

1. Riešiť tradičné problémy. Už teraz vieme pomerne dobre skonštruovať napríklad systém na upravovanie algebraického výrazu na základný tvar, systém analogický systému ELIZA a podobne. Týchto problémov sa nebudeme zriekať, lebo umožňujú získavať skúsenosti a ich implementácia nevyžaduje veľkú námahu. Plne si však uvedomujeme, že takéto experimenty nedokážu potvrdiť či je používaná metodika vhodná pre budovanie systémov s komplexným správaním.

2. Riešiť problém komunikácie v prirodzenom jazyku. Výhoda tohto problému spočíva v tom, že nevyžaduje technicky náročné riešenie. (Prepojiť systém s modulmi syntézy a rozpoznávania reči sa dá dodatočne).

3. Riešiť nejaký reálny praktický technologický problém, napríklad riadenie nejakej výrobnej linky. Tu je možno dobre odskúšať silu používanej architektúry pre jej praktické uplatnenie a komplexnosť problému tiež nie je najhoršia.

4. Riešiť nejaký problém správy dát z oblasti multimédií alebo virtuálnej reality. Tu je veľkým problémom technická a finančná náročnosť.

5. Vybudovať mobilný robotický systém s našou architektúrou. To je opäť technicky a finančne náročná záležitosť. Príťažlivejšou by sa tento smer mohol stať v okamihu, keď bude daný do predaja operačný systém QNX Neutrino, ktorý má umožniť priamočiare prevedenie programového systému vyvinutého pod obyčajným PC na embeded systém. I tak je však otázne, či súčasné technické možnosti veľkosti pamätí embeded systémov umožňujú vytvoriť embeded systém zložený z tak veľkého počtu procesov.

Ktorý smerom sa nám podarí vydať v tejto chvíli ešte nevieme. Dúfame však, že sa nám i na experimentálnom poli niečo podarí.

Literatúra

[Brooks 1986]

Brooks R. A.: *A Robust Layered Control System for a Mobile Robot.* IEEE Journal of Robotics and Automation, RA-2, (1986), 14–23.

[Brooks 1989]

Brooks R. A.: *A Robots that Walks: Emergent Behaviors from a Carefully Evolved Network.* Neural Computation 1:2, Summer, (1989).

[Brooks 1991]

Brooks R. A.: *Intelligence without representation.* Artificial Intelligence 47, (1991), 139–159.

[Brooks 1993]

Brooks R.: *Building Brains for Bodies.* (A. I. Memo No. 1439), MIT AI Lab, Cambridge, Mass., 1993.

[Doran 1992]

Doran J.: *Distributed AI and its Applications.* In: Advanced Topics in Artificial Intelligence (V. Mařík, O. Štěpánková, R. Trappl, eds.) Springer-Verlag, Berlin, 1992.

[Ferko - Kalaš - Kelemen 1990]

Ferko A., Kalaš I., Kelemen J.: *Počítač Hamlet.* Mladé letá, Bratislava, 1990.

[Jones - Flynn 1993]

Jones L. J., Flynn A. M.: *Mobile robots: Inspiration to Implementation* AK Peters. Ltd., Wellesley, Mass., 1993.

[Forest 1991]

Forest S.: *Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks.* Introduction to the Proceedings of the Ninth Annual CNLS Conference. In: Emergent Computation (Forest S., ed.), The MIT Press, Cambridge, Mass., 1991, 1–11.

[Goodwin 1993]

Goodwin R.: *Formalizing Properties of Agents.* (Report CMU - CS - 93 - 159), Computer Science Department, Carnegie Mellon University, Pittsburgh, Pensylvania, 1993.

[Kelemen 1989]

Kelemen J.: *Myslenie, počítač ...* Spektrum, Bratislava, 1989.

[Kelemen a kol. 1992]

Kelemen J., Ftáčnik M., Kalaš I., Mikulecký P.: *Základy umelej inteligencie.* Alfa, Bratislava, 1992.

[Kelemen 1993]

Kelemen J.: *Multiagent symbol systems and behavior-based robots.* Applied Artificial Intelligence 7, (1993), 419–432.

[Kelemen 1994]

Kelemen J.: *Strojovia a agenty.* Archa, Bratislava, 1994.

[Kelemen - Kelemenová 1992]

Kelemen J., Kelemenová A.: *A grammar-theoretic treatment of multiagent systems.* Cybernetics and Systems 26, (1992), 621–633.

[Knight 1993]

Knight K.: *Are Many Reactive Agents Better Than a Few Deliberative Ones?* In: Proc. IJCAI '93, Chambery, 1993, 132–137.

[Lúčny 1994]

Lúčny A.: *Emergentné správanie v kolóniách agentov.* Diplomová práca, Katedra umelej inteligencie, Matematicko-fyzikálna fakulta, Univerzita Komenského, Bratislava 1994.

[Maes 1989]

Maes P.: *How To Do the Right Thing.* (A. I. Memo No. 1180), MIT AI Lab, Cambridge, Mass., 1989.

[Mataric 1994]

Mataric M. J.: *Interaction and Intelligent Behavior.* Submitted to the Department of Electrical Engineering in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy, MIT AI Lab, Cambridge, Mass., 1994.

[McFarland, Bosser 1993]

McFarland D., Bosser T.: *Intelligent Behavior in Animals and Robots.* The MIT Press, Cambridge, Mass., 1993.

[Minsky 1986]

Minsky M.: *The Society of Mind.* Simon&Schuster, New York, 1986.

[Minsky 1996]

Minsky M.: *Konštrukcia mysle.* (Kelemen J., ed.), Archa, Bratislava, 1996.

[Mlichová 1993]

Mlichová R.: *Niektoré experimenty so subsumpčnou architektúrou v nedeliberatívnej robotike.* Diplomová práca, Katedra umelej inteligencie, Matematicko-fyzikálna fakulta, Univerzita Komenského, Bratislava 1993.

[Parker 1992a]

Parker L. E.: *Adaptive Action Selection for Cooperative Agent Teams.* In: From Animals to Animates, Proc. 2nd International Conference on

Simulation of Adaptive Behavior (Meyer J. A., Roirblar H., Wilson S., eds.), MIT Press, Cambridge, Mass., 1992, 442–450.

[Parker 1992b]

Parker L. E.: *Local versus Global Control Laws for Cooperative Agent Teams.* (A. I. Memo No. 1357), MIT AI Lab, Cambridge, Mass., 1992.

[Parker 1994]

Parker L. E.: *Heterogeneous Multi-Robot Cooperation.* Submitted to the Department of Electrical Engineering in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy, MIT AI Lab, Cambridge, Mass., 1994.

[Resnick 1994]

Resnick M.: *Turtles, Termites, and Traffic jams.* The MIT Press, Cambridge, Mass., 1994.

[Singh 1994]

Singh M. P.: *Multiagent Systems.* Springer-Verlag, Berlin, 1994.

[Stein 1991]

Stein L. A.: *Imagination and Situated Cognition* (A. I. Memo No. 1277), MIT AI Lab, Cambridge, Mass., 1991.

Obsah