Reaktívny model inteligentného systému

Andrej Lúčny

Matematicko-fyzikálna fakulta, Univerzita Komenského v Bratislave
lucny@fmph.uniba.sk
http://www.fmph.uniba.sk/~lucny

Všetko treba vysvetliť najjednoduchšie
ako je možné. Len nie jednoduchšie.

Albert Einstein

1. Prístupy k tvorbe inteligentných systémov

Cieľom umelej inteligencie je vytvárať umelé systémy, ktoré sa vyznačujú vlastnosťou inteligencie. Z takejto definície síce nie je úplne jasné aké systémy sa to vlastne pokúšajú „umelí inteligenti“ vytvárať (nakoľko pojem „inteligencia“ sa definuje rôzne a vágne), zato však z nej striktne vyplýva požiadavka na reálnosť ich vytvorenia a ich funkčnosti. V rámci umelej inteligencie teda nehľadáme matematické konštrukty spĺňajúce určité vlastnosti a nesnažíme sa o nich ukazovať, že potom už musia mať nevyhnutne ďalšie zaujímavé vlastnosti. My veríme, že v hromade všetkých strojových kódov (včítane dát) aké môžu byť interpretované našimi počítačmi existujú také, ktoré sa vyznačujú vlastnosťami, ktoré nám pri bežných počítačových systémoch chýbajú, avšak pozorujeme ich na sebe i na primitívnejších živých systémoch – a snažíme sa nájsť spôsob ako sa k týmto kódom reálne dopracovať. Na umelú inteligenciu teda možno nazerať ako na snahu hľadať postupy vedúce k nakódovaniu a dátovému naplneniu systémov s určitými špeciálnymi (v súčasnosti neobvyklými) vlastnosťami, ktorým sa až dodatočne pridáva všeobecný atribút inteligencie: súbor týchto vlastností je vlastne dodatočnou odpoveďou na otázku v čom spočíva jej podstata. O takýchto umelých systémoch možno potom s istou dávkou trúfalosti vyhlásiť, že z určitej stránky aproximujú systémy živé.

Umelá inteligencia by teda mala byť pomerne praktická záležitosť, i keď veľmi ľahko môže skĺznuť do úvah a sporov, ktoré zaujímajú viac filozofov než programátorov. Je totiž veľmi otázne, čo všetko je v živých systémoch nevyhnutnou podmienkou pre ich inteligentné správanie. V mene ich programátorov môžeme povedať, že pýtať sa či „je možné urobiť z počítača systém, ktorý by pociťoval svoju existenciu tak, ako ju my sami pociťujeme a prežívame“ považujeme za menej plodné (i keď rozhodne zaujímavejšie) než pýtať sa či „je možné urobiť z počítača systém, ktorý sa vás neopýta či chcete pri odchode z editora uložiť na serveri rozeditovaný súbor, keď vám tesne predtým oznámil, že tento server je mimo prevádzky“. Aby bola druhá otázka plodnejšia, je samozrejme potrebné predpokladať, že vlastnosť spomínaná v prvej otázke nie je nevyhnutná pre realizáciu vlastnosti požadovanej v otázke druhej, t.j. je od nej oddeliteľná. Neostáva nám nič iné než túto oddeliteľnosť predpokladať, aby sme mohli k vytváraniu inteligentných systémov pristúpiť. To, či budú výtvory skonštruované v rámci týchto pozícií uznané za inteligentné a v akej miere, ako i to, či ich vytvorenie vnieslo určité svetlo na podstatu živých inteligentných systémov, sa, pravda, ukáže vždy až po ich vytvorení.

Existujú v podstate tri spôsoby ako k tvorbe inteligentných systémov pristúpiť. Každý z nich sa zakladá na získaní predstavy o určitej vlastnosti živých systémov, ktorá sa považuje za najdôležitejšiu. V prvom prípade ide o princíp evolúcie. Tu sa tvorca snaží urobiť si predstavu o tom, ako a s použitím akého fyzického základu vývoj živých systémov prebieha a snaží sa ho zopakovať: inteligencia sa tu teda chápe ako podmnožina súboru schopností prežiť a presadiť sa v nasledujúcej generácii. V druhom prípade ide o získanie predstavy o fyzickom základe učenia sa, ktoré sprevádza ontogenetický vývoj živého inteligentného jedinca. Tvorca sa v tomto prípade snaží vytvoriť prázdny systém schopný učenia sa a podrobiť ho správnej sérii vstupov za účelom adekvátneho dobudovania jeho vnútornej štruktúry. Inteligencia sa v tomto prípade chápe ako podmnožina súboru vlastností získaných učením a samotná schopnosť učiť sa. Na záver sme si nechali historicky najstarší a stále najrozšírenejší spôsob, čo je neobvyklé, ale máme na to dobré dôvody. Je totiž na založený na predstave, že podstatu inteligencie možno oddeliť nielen od problémov typu „duša - telo“, ale taktiež od evolúcie a učenia sa. Na predstave, že je možné pochopiť v čom spočíva funkčnosť už vyvinutého a už naučeného systému, a

následne vytvoriť systém, ktorý sa už nepotrebuje vyvíjať ani učiť. Tvorca v tomto prípade kopíruje svoju predstavu o funkčnosti živého systému do systému umelého: triafa sa do výsledku, ktorý príroda dosiahla evolúciou, jedinec učením sa a podobne, inteligencia je preňho vlastnosť zvolenej architektúry, ktorá vyjadruje vzťah medzi vnútornou štruktúrou systému a kvalitou jeho vonkajších prejavov. Už na prvý pohľad sa tento spôsob javí ako menej biologicky relevantný. Programátorom je však rozhodne najbližší. Zodpovedá jednak základnému štýlu programátorskej práce, jednak sa pri ňom darí urobiť si dostatočne dobrú predstavu o fyzickom základe vytváraného systému. Pokiaľ sme totiž spomínali pri alternatívnych spôsoboch vytvárania inteligentných systémov ich fyzický základ, sledovali sme tým skutočnosť, že s implementáciou netriviálnych systémov sú (zatiaľ) pri ich použití problémy. (V prípade evolúcie problém spočíva v tom, že nepoznáme netriviálnu algoritmickú štruktúru ktorá pri mutácii iba málo zmení svoju interpretáciu: keď v bubblesorte zmeníme a[j-1]:=:a[j] na a[j-2]:=:a[j], nedostaneme quicksort, ale range-check error. Mutácie sa preto spravidla odohrávajú iba nad vektorom konštánt nemenného algoritmu a to je skôr naladenie než vytvorenie systému. V prípade učenia zase vznikajú problémy, keď sa neučíme jedinú konkrétnu vec, ale široké spektrum vecí: systém má tendenciu obsahovať (minimálne dočasne) koncepty ako „lietajúca mucha so stoličkou bez kvetináča“, čo vedie k exponenciálnym nárokom pamäť a čas od počtu naučených želaných konceptov. Taktiež vyčlenenie učiaceho algoritmu ako samostatnej jednotky v systéme má za následok jeho monolytickosť, nevýhody čoho uvedieme nižšie). Preto väčšina systémov umelej inteligencie vznikla práve v duchu tretieho uvedeného spôsobu.

2. Inteligentné systémy v reálnom čase a prostredí

Predpokladajme v ďalšom, že inteligentný systém, ktorý vytvárame, musí pracovať v reálnom čase a prostredí. (Koniec koncov iba takýto systém je nám na niečo priamo dobrý. Systém, ktorý po 2 hodinách výpočtu presunie kocku z políčka [1,2] na políčko [2,2], nás môže akurát poučiť o tom, ako by sa dali vytvoriť systémy ekonomicky prínosné.) Uvidíme, že prijatie tohto predpokladu obráti v prach väčšinu tradičných systémov umelej inteligencie (žiaľ), zároveň však aj ukáže niektoré nevyhnutné vlastnosti, ktorými sa musí inteligentný systém vyznačovať. Napodiv pôjde o vlastnosti programátorom dobre známe, ktoré čo to povedia nielen o inteligentných, ale o všetkých počítačových systémoch vôbec.

Pozrime sa teraz očami predpokladu práce v reálnom čase a prostredí na tradičné systémy umelej inteligencie. Jadrom týchto je kognitívny podsystém, založený na jedinom algoritme (napr. STRIPS) operujúcom nad reprezentáciou - t.j. nad množinou dát určitého typu (napr. klauzuly). Na jednu časť týchto dát sú premieňané vstupné signály z receptorov. Ďalšia časť reprezentácie je interpretovaná ako plány pre riadenie efektorov. Reprezentáciu sú taktiež vyjadrené ciele (napr. „odstráň z podlahy všetky kocky“), ktoré sa systém svojimi akciami snaží v prostredí dosiahnuť alebo udržať. Prax ukázala, že tieto systémy nedokážu pracovať v reálnom čase a reálnom prostredí. Teoreticky sa toto zlyhanie vysvetľuje väčšinou neadekvátnym spôsobom budovania systému a jeho členenia na časti. Poukazuje sa nato, že najprv bol spravidla vyvinutý kognitívny podsystém. O tomto sa potom ukázalo, že správne operuje nad reprezentáciou, t.j. nad dátovými typmi jemu vlastnými. Nakoniec sa umelo napasoval do väčšieho celku (napr. do robotického systému SHAKEY) pričom sa ad-hoc špekulovalo ako previesť vstupné signály z receptorov na reprezentáciu a reprezentáciu na výstupné signály riadiace efektory. Pritom na vstupe je dosť problematické pri tomto prevode nestratiť podstatnú zložku vstupnej informácie a zároveň udržať veľkosť reprezentácie v rozumných medziach (odvodenie plánu v kognitívnom podsystéme má spravidla exponenciálnu zložitosť od veľkosti reprezentácie). Na výstupe je zase nesmierne ťažké premeniť reprezentovaný plán („vytlač kocku K na pódium P“) na atomické akcie („cúvni o 90 cm“). Ťažko proti tomu niečo namietať, ale podľa nášho názoru je rovnako významná - ak nie významnejšia - skutočnosť, že kognitívny podsystém je z hľadiska svojej štruktúry monolytický – t.j. tvorí ho jediný program. To ho zvádza k tomu, že počas operácií nad reprezentáciou opakovane vzťahuje k sebe reprezentácie vecí sématicky absolútne nesúvisiacich (napr. v priebehu hľadania plánu pre vyššie uvedený cieľ sa STRIPS snaží opakovane použiť v rezolvencii reprezentáciu skutočnosti, že „kváder je na podlahe“). Sú to práve tieto jalové operácie, ktoré zdvihnú jeho výpočtovú zložitosť tak, že sa stáva reálne nepoužiteľným. O týchto systémoch sa síce často hovorí, že „vedia čo robia“, nakoľko v ich pamäti možno objaviť náprotivky jednotlivých vecí z prostredia a možno v nej pozorovať zmeny zodpovedajúce procesu logického uvažovania o týchto veciach, avšak vzhľadom na vykonávanie spomínaných jalových operácií by bolo správnejšie hovoriť, že „nemajú najmenšej potuchy o tom s čím robia“ (nerobia s kockami ale s *„kocka“*-mi).

Odklon časti umelej inteligencie od takýchto systémov má svoje paralely aj v iných oblastiach. V biológii sa napríklad na cytoplazmu bunky nazeralo ako na roztok, ktorého činnosť sa dá modelovať na báze matematickej štatistiky a termodynamiky – dnes sa o nej vie, že má zložitú štruktúru, ktorá pravdepodobne dokáže úplne zabrániť priebehu určitých chemických reakcií, ktoré by boli v rámci roztoku možné. Podobne pri modelovaní ekosystémov typu dravec - korisť sa uvažuje iba o nikách typu „savana, kde každý pobehuje na vlastnú päsť

a môže sa každý s každým stretnúť“. Na systémy sa všade začína nazerať nielen ako na hromadu elementov, ale ako na štruktúrované stavby, ktoré nepripúšťajú interakciu medzi ktorýmikoľvek elementami.

3. Multiagentové systémy

Na základe uvedomenia si týchto skutočností vznikli na okraji umelej inteligencie prúdy orientujúce sa na tvorbu modulárnych systémov, ktoré by boli schopné zachytiť štruktúrovanosť procesov v nich prebiehajúcich. Či sa na to išlo s heslami proti reprezentácií, budovania jednoduchších ale reálne fungujúcich systémov, alternatívnych dekompozícií systému, či používania metódy zdola – nahor pri ich tvorbe, pravdepodobne nebolo rozhodujúce. Našimi očami to chápeme ako snahu o hľadanie takej modulárnej štruktúry, ktorá by zabezpečila:

- **aby boli k sebe vzťahované iba sémanticky príbuzné dáta (či už signály alebo ich reprezentácie)**
- **aby jednotlivé sémantické vzťahy zabezpečovali špecializované algoritmy**

Tieto nevyslovené ale nesporne sledované vlastnosti priviedli časť umelej inteligencie do úzkeho kontaktu s komunikáciou medzi procesmi a konkurenčným objektovo orientovaným programovaním či inými formami modularity decentralizovaných programových systémov. (Termínom „programový systém“ označujeme systém zložený z viacerých vzájomne komunikujúcich programov vykonávaných na jednom alebo viacerých procesoroch, systém skôr konkurenčný než paralelný.) Došlo k búrlivému procesu vzájomného ovplyvňovania sa, ktorý trvá podnes a ktorý dal vzniknúť veľkému množstvu frakcií s rôznym stupňom relevantnosti k umelej inteligencii. Hranice medzi nimi sú také nejasné a terminológia nimi používaná tak variabilná, že ťažké je ich čo i len rozumne kategorizovať. Problém je ich aj spoločne pomenovať: Nová UI, Distribuovaná UI, sociorobotika, softwaroví agenti, či multiagentové systémy sú skôr názvy častí než celku, obsahujúce navyše k nášmu ponímaniu problematiky irelevantný balast. Pokiaľ si máme predsa len vybrať určitý spoločný názov, prikláňame sa k posledne uvedenému. Väčšina týchto frakcií sa zamerala na pre ňu vlastnú časť prostriedkov, ktorými sa snaží obohatiť tradičné softwarové či robotické produkty a povýšiť ich na nejakú vyššiu kvalitu. Niektoré kladú dôraz na normalizáciu komunikácie medzi procesmi (CORBA, KQML, FIPA), iné na mobilitu algoritmov (Telescript), ďalšie na softwarové (DESIRE, BDI) či robotické architektúry (subsumpčná architektúra). Každá z týchto frakcií ponúka určitú **architektúru**, t.j. určitý typ základných stavebných kameňov a metódu ako z nich tvoriť systém, vyznačujúci sa určitými zaujímavými vlastnosťami.

Agenti

Nazeranie na základný stavebný kameň systémov je v rámci všetkých frakcií zhruba rovnaké. Svorne ho nazývajú **agent** a bez ohľadu na to, čo si na ňom najviac cenia (autonómnosť, vnútornú stavbu, mobilitu, komunikačné rozhranie, …) rozumejú pod ním **niečo, čo neustále (resp. opakovane) vníma svoje prostredie, na základe toho volí akcie, ktoré má vo tomto prostredí vykonať, aby sa dosiahol stanovený cieľ a následne tieto akcie vykonáva.** Pritom pod prostredím sa rozumie nielen napr. fyzické okolie mobilného robota, ale čokoľvek, k čomu môže agent pristupovať bez následného zablokovania jeho riadenia. Pri interakcii (t.j. vnímaním a vykonávaním akcií) agenta s prostredím môže ísť spravidla o

- prijímanie signálov zo senzora (vstupného zariadenia) alebo vysielanie signálov na aktuátor (motor) (výstupné zariadenie)
- (**priamu**) **komunikáciu** s inými agentami, t.j. príjem alebo vyslanie správy
- čítanie a zápis zdieľaných dát (**nepriamu komunikáciu**), realizovanú obyčajne ako vyslanie správy a príjem odpovede na špeciálny proces stelesňujúci prostredie

Pritom vo všetkých prípadoch pristupujeme ku každej vstupnej informácii ako keby ju agent vnímal určitým receptorom a ku každej výstupnej ako keby agent vykonal nejakú akciu určitým efektorom. Z hľadiska vnútra agenta, ktoré zabezpečuje voľbu akcie, má teda vstup i výstup rovnakú povahu bez ohľadu na konkrétny spôsob interakcie.

Spôsob, akým je toto vnútro agenta realizované, t.j. povaha algoritmu, ktorý volí akcie, určuje druh agenta. Na prvý pohľad sa môže táto povaha zdať nepodstatná, nakoľko agent je formalizovateľný napr. funkciou, ktorá histórii vstupov priraďuje určitý výstup. Nám však nejde o skúmanie výpočtovej sily alebo inej podobnej vlastnosti uvažovaného systému, ale o hľadanie spôsobov, ako ho tvoriť (programovať, vyvíjať) tak, aby sme pritom splnili vyššie stanovené podmienky a dosiahli určité kvality o ktorých bude ešte reč. Spôsob voľby akcie je základným stavebným kameňom multiagentových systémov a preto niet divu, že sa v rámci jednotlivých frakcií zásadne odlišuje. Niektoré frakcie vkladajú do agenta celý tradičný systém a „agentovosť“ sa tým pádom stáva iba nástrojom normalizovanej komunikácie rôznych tradičných systémov. Iné obmedzujú tvar vstupov a výstupov vstupujúcich do procesu voľby, t.j. definujú určitý jazyk, či protokol. Ďalšie zase definujú tvar

voliaceho algoritmu, prípadne používajú jediný pevný algoritmus (v rôznych konfiguráciách). (Posledne menované môže sa zdať v rozpore s druhou požadovanou vlastnosťou, t.j. aby jednotlivé sémantické vzťahy zabezpečovali špecializované algoritmy. Tieto špecializované algoritmy však nemusíme ztotožňovať s algoritmami pre voľbu akcie. Niekoľko agentov používajúcich ten istý algoritmus na voľbu akcie môže spoločnými silami realizovať špecializovaný algoritmus. Požadovaná špecializovanosť sa teda nemusí objaviť v základných stavebných kameňoch systému, ale až pri stavbách z nich postavených.)

Ďalším dôležitým parametrom agenta je spôsob akým je do jeho vnútra zahrnutý cieľ. Pokiaľ si agent buduje na základe histórie vstupov určitú reprezentáciu a používa na to jednotný vyjadrovací prostriedok, je možné týmto prostriedkom **explicitne** vyjadriť aj cieľ. Pre takéhoto agenta sa ujalo pomenovanie cieľavedomý, t.j. **deliberatívny agent** a je veľmi obľúbenou a rozšírenou platformou (najmä v rámci distribuovanej UI). Explicitné vyjadrenie cieľa mu umožňuje voliť akciu napríklad výberom zo všetkých možných akcií na základe predvídania ich následkov. To je na jednej strane v zhode s tým ako prebieha (resp. ako si predstavujeme, že prebieha) riešenie problémov v našej mysli, avšak na strane druhej od tradičných systémov umelej inteligencie sa to líši iba v tom, že v rámci jednotlivých agentov je možné používať rôznu reprezentačnú platformu a špecializovať algoritmy predvídania následkov akcií. Práca systému zloženého z takýchto jednotiek v reálnom čase je tiež dosť otázna. Ide tu o to, nakoľko agent následky možných akcií domyslí.

Ak začne kvalitu voľby akcie posudzovať podľa ďalšieho možného vývoja a bude zisťovať, či skutočne povedie k cieľu, nemôže byť o práci v reálnom čase ani reči. Navyše to bude v rozpore s tým, že navrhujeme za základný stavebný kameň inteligentných systémov niečo, čo vie riešiť problémy (t.j. postaviť adekvátny plán na základe vnútorných virtuálnych pochodov) a pritom poznáme veľa živých systémov, ktoré sa neštítime nazývať inteligentnými a riešiť problémy (v nami používanom užšom význame slova) vôbec nevedia. V istých prípadoch by teda celok mal mať menšie schopnosti než jeho časti. Zaujímavé by bolo dosiahnuť skôr opak.

Ak deliberatívny agent vyhodnotí iba určité kvantitatívne ukazovatele priamych následov, zrejme ho to príliš nezdrží, avšak na druhej strane sa cieľ preňho stáva menej dôležitý než súbor trikov na predvídanie následkov akcií. Pokiaľ bude tento súbor dostatočne špecializovaný, bude dobre fungovať iba pre tento cieľ a pre nijaký inakší. Spomínané triky sa v hraničnom prípade stanú implicitným vyjadrením explicitne vyjadreného cieľa. Pokiaľ si prestanú všímať cieľ úplne, agent prestáva byť deliberatívnym a stáva sa agentom, pre ktorého sa ujalo pomenovanie reaktívny.

Reaktívni agenti sú založení na predstave o adekvátnych podvedomých reakciách. Reaktívny agent pracuje spôsobom „urobím toto, lebo som v situácii, kedy sa to robieva“ na rozdiel od deliberatívneho, pre ktorého platí „urobím toto, lebo chcem dosiahnuť tamto a takto to (snáď) dosiahnem“. Hovorí sa, že kým deliberatívny agent sa „rozhoduje“, reaktívny „reaguje“ – i keď tu v podstate ide spojité spektrum, ktoré sa polarizuje iba v rámci snahy o kategorizáciu.

Reaktívny agent je iba súborom trikov definujúcich, čo sa za akých okolností robieva na dosiahnutie jediného cieľa. O tomto cieli nemá ani potuchy, lebo nemá potuchy vôbec o ničom. Na receptoroch (vstupoch) neustále prijíma signály, na ktoré sa vrhá algoritmus pevne nadrátovaný v jeho vnútri, následkom čoho sa produkujú signály na jeho efektory (výstupy).

Na tento algoritmus je možné klásť ďalšie ohraničenia. S jednoduchosťou voľby akcie sa pritom dostávame na hranice možností, kde zdanlivo hrozí, že z tak jednoduchých elementárnych jednotiek bude problém vybudovať zložitý systém, t.j. že ak to vôbec bude možné, bude to ťažšie než pri použití zložitejších elementárnych jednotiek. Jedno z týchto ohraničení je používanie vnútorného stavu agenta. Algoritmus voľby akcie normálne môže používať globálnu pamäť, ktorá mu umožňuje vziať do úvahy nepriamo aj hodnoty vstupov, výstupov a vnútorných stavov z minulosti. Pokiaľ to zakážeme, agent si nemôže pamätať nič z predchádzajúcich krokov. Pamäť môže využiť len v jednom kroku ako jednorázovú pomôcku. Takýto oklieštený agent sa zvykne nazývať čisto reaktívnym. Aby predsa len **čisto reaktívni agenti** dokázali meniť svoju vôľbu akcií aj ináč než na základe zmeny na vstupoch, vybavujú sa niekedy prídavným vstupom, ktorý poskytuje hodnoty z náhodného generátora. Algoritmus voľby akcií ostane i za takýchto podmienok deterministický, ale jeho správanie sa voči skutočným vstupom bude stochastické. Týmto spôsobom sa dá čiastočne vykompenzovať stratená pamäť (napríklad miesto „otoč sa doľava v každom 20-tom kroku“ sa dá použiť „s pravdepodobnosťou 1/20 sa otoč doľava) a navyše sa získa väčšia variabilita v správaní sa systému.

Metódy tvorby

Význam každého stavebného kameňa nespočíva ani tak v tom, čo sa dá z neho postaviť, ale hlavne ako sa dá z neho stavať. Teoreticky to totiž poväčšine vychádza tak, že z každého typu kameňa sa dá postaviť akýkoľvek dom. Prakticky však každá stavba naráža na určité ohraničenia vyplývajúce zo spôsobu ako sa dom stavia. Stavba 100 poschodového domu kladie celkom iné nároky na spôsob stavania ako stavba 10 poschodového domu. Pritom zdanlivo ide len o kvantitatívny rozdiel. Pritom keby sa niekto pokúšal stavať 100 poschodový dom metódami, ktoré sa používajú pri stavbe 10 poschodového, dopadlo by to asi tak, že by po 25 poschodiach zo zúfalstva urobil strechu a „išiel od toho“.

Analogická situácia sa týka vytvárania multiagentových systémov (a programových systémov vôbec). Na každej architektúre je vlastne podstatnejšia metóda tvorby než typ agenta, na druhej strane musí typ agenta danú metódu podporovať (jednotlivé syntaktické štruktúry reprezentujúce agentov sú síce väčšinou vzájomne konvertovateľné, avšak nie každá syntax je rovnako vhodná na používanie tvorcom systému). Metóda tvorby multiagentových systémov pritom spočíva v zvolení postupu ako budovať vnútro systému, aby na povrchu produkoval konkrétne požadované správanie. Od tohto postupu očakávame, že nám napovie ako systém dekomponovať (t.j. ako ho pri návrhu rozdeliť na jednotlivé časti) a ako tieto časti postupne implementovať. Pritom pod časťou rozumieme určitú skupinu kódov a dát organizačne izolovanú od ostatných. V prípade multiagentových systémoch ide o teda o kódy a dáta prislúchajúce určitej skupine agentov.

Pri dekompozícii systému je podstatné akú zvolíme ekvivalenciu. Táto definuje ktoré atribúty budú dáta a kódy systému spájať a ktoré oddeľovať. V tradičných systémoch jednotlivé časti tvorili spravidla algoritmicky príbuzné dáta a kódy – išlo o dekompozíciu funkciou. Napríklad jednu časť robota SHAKEY tvorilo spracovanie vstupu zo senzorov, druhú výber spracovaného signálu a jeho premena na reprezentáciu, tretiu kognitívny podsystém (plánovač), štvrtú vykonávanie plánov a piatu riadenie efektorov. Inou možnosťou je zvoliť za ekvivalenciu sémantickú príbuznosť kódov a dát. Dostaneme tak systém rozdelený na jednotlivé aktivity – **dekompozíciu aktivitou**. Napríklad robot ALLEN tvoria tri časti: dopredný pohyb s vyhýbaním sa prekážkam, túlanie a skúmanie. Dekompozícia funkciou má výhodu v tom, že môže byť rovnaká pre dva systémy vykonávajúce rôznu činnosť a teda je možné poznatky získané pri jednom systéme znovu použiť pri systéme druhom. Naproti tomu dekompozícia aktivitou je podobná iba ak systémy vykonávajú podobné činnosti. Vedieť správne dekomponovať systém na aktivity je spravidla umenie. Sila tejto dekompozície však spočíva v tom, že podporuje inkrementálnu implementáciu navrhnutých častí. Umožňuje zamerať sa najprv na základné aktivity a potom postupne ďalšie aktivity pridávať. Pritom systém je vždy po pridaní ďalšej aktivity spustenia schopný a vieme aké správanie by mal mať pri daných implementovaných aktivitách – môžeme ho teda ladiť ako celok. Tým pádom nám nehrozí, že keď jednotlivé (bárs aj osobitne odskúšané) časti systému spojíme, vzniknutý celok nebude funkčný. Toto nešťastie je charakteristické práve pre systémy, pri tvorbe ktorých bola použitá dekompozícia funkciou.

Čo sa týka implementácie navrhnutých častí, t.j. ich nakódovania, panuje v programátorskom svete jednotný názor: správny postup je zhora – nadol. Implementácia určitých typov multiagentových systémov je však náchylná otriasť i touto pravdou. Každý komplikovanejší systém sa totiž tvorí **inkrementálne**. Pokiaľ v ňom nejaká časť využíva prítomnosť časti druhej, pri metóde zhora nadol bude najprv implemntovaná časť, ktorá využíva a až potom časť, ktorá je využívaná. Pokiaľ teda chceme systém ladiť, musíme využívanú časť dočasne nahradiť nejakým triviálnym modulom. Takýto celok však nebude vykazovať žiadne relevantné správanie a preto takéto ladenie nič nepovie o funkčnosti implementovanej časti. Ak však použijeme metódu **zdola – nahor**, ladenie je plne umožnené. Samozrejme tu však hrozí nebezpečenstvo, že pri návrhu neohalíme všetky potrebné detaily na to, aby neskôr implementovaná časť dokázala využiť skôr implementovanú. Toto nebezpečenstvo môžeme čiastočne znížiť tým, že z princípu nebudeme implementovať určitú časť kvôli tomu, aby ju nejaká iná v budúcnosti využívala, ale len kvôli samotnej aktivite ktorú zabezpečuje. Pokiaľ potom budeme chcieť pri implementácii ďalšej časti využiť inú existujúcu časť, musíme toto využitie podporiť špeciálnymi prostriedkami. Napríklad tým, že nová časť nebude existujúcu časť „volať“, ale sa „votrie“ do jej činnosti. T.j. existujúca časť nebude poskytovať zapuzdrené rozhranie na to, aby ju niekto zavolal. Novej časti bude akurát dovolené infiltrovať externé dáta používané existujúcou časťou. Keď to prevedieme do reči multiagentových systémov, nová skupina agentov bude infiltrovať prostredia stávajúcich agentov. Pritom k tejto infiltrácií nesmie dochádzať neustále, nakoľko by to v podstate vyradilo existujúce časti z ich pôvodnej činnosti. Avšak počas situácie na ktorú pôvodné časti nevedia adektvátne zareagovať, je infiltrácia želaná, ak sa jej pôsobením spomínaná adekvátna reakcia zabezpečí. Takéto pripojenie novej časti do systému v zásade neovplyvní podiel stávajúcich častí na správaní systému, iba do neho zavedie určité výnimky. To nám umožňuje pri inkrementálnej tvorbe po každom pridaní novej časti systém testovať a odladiť. Tým získavame istotu, že pri tvorbe sledujeme výsledné želané správanie systému – že postupujeme správnym smerom a nestane sa nám, že po ukončení tvorby bude systém robiť niečo iné než sme pôvodne zamýšľali. Tento spôsob, ktorý používa robotická subsumpčná architektúra je len jedným z mnohých spôsobov (a podľa nás nielen jedným ale aj veľmi dobrým) ako dosiahnuť, aby sa na povrchových prejavoch systému podielala vždy správna zostava jeho častí. Umenie vytvoriť multiagentový systém totiž spočíva práve v tom, ako zariadiť, aby sa **v každom okamihu aktivovali tí agenti, ktorých zlúčený prejav dáva na povrchu systému želané správanie.**

Z hľadiska biologickej relevantnosti je pritom podstatné ako sú prejavy aktivovaných agentov (t.j. tých, ktorí práve zvolili a vykonávajú určité akcie spôsobujúce zmenu v ich prostredí) zlúčené do celkového prejavu systému. Spravidla to vyzerá tak, že pokiaľ sa v prostredí systému nič nedeje, v systéme prebiehajú len najzákladnejšie aktivity. Tie sú potlačené vyššími v okamihu, keď príde určitý podnet. Za istý čas po jeho odznení, keď už bol tento podnet „vybavený“, vyššie aktivity opäť ustúpia do ústrania. Aktivity teda pracujú synchrónne, je medzi nimi zavedená prísna hierarchia. Zaujímavejšie by podľa nás bolo, keby hierarchicky nižšia aktivita mala šancu odolať potlačeniu zo strany hierarchicky vyššej aktivity, akurát by to bolo málo

pravdepodobné. Napríklad, keď sme hladní, ale veľmi túžime ísť na seminár z umelej inteligencie, dokážeme tento hlad potlačiť tak, že celé minúty o ňom nevieme. On sa však bude v pravidelných intervaloch hlásiť o slovo ako aj predtým – nám sa iba podarí predĺžiť tieto intervaly zo sekúnd na minúty.

Zaujímavé vlastnosti

Ako sme naznačili, multiagentové systémy majú istý vzťah k umelej inteligencii, ale často majú oveľa bližšie k iným oblastiam. Ich „verejným" cieľom nie je vytvoriť inteligentné systémy, ale systémy, ktoré sa vyznačujú určitými zaujímavými vlastnosťami. Pritom ide väčšinou o kvalitatívne vlastnosti týkajúce sa spôsoby tvorby systémov (čo si samozrejme vyžaduje aj adekvátnu vnútornú štruktúru), ktoré umožňujú rozšíriť limity pre ich kvantitatívne parametre. (T.j. hľadajú nové technológie ako stavať dom, aby sa dal postaviť nie maximálne 25 poschodový, ale 100 poschodový. Pritom žiadnej technológii sa pravdepodobne nemôže podariť úplne zrušiť tento limit, môže ho iba oddialiť.) Pokúsime sa vymenovať aspoň najčastejšie uvažované vlastnosti:

- **decentralizovanosť systému**: V tradičných systémoch je každý proces naviazaný na procesy nižšej úrovne, tie na procesy ešte nižššej úrovne, … až v konečnom dôsledku na zariadenia. Toto naviazanie má taký charakter, že samotný proces bez nižšších vrstiev nedokáže vyvíjať vlastnú aktivitu. Proces spravidla požiada nižšie vrstvy o určitú službu (napríklad „chcem čítať znaky zo sériovej linky") a potom je už odkázaný na to, že mu tieto vrstvy dajú určitý podnet k jeho činnosti („prišiel ti znak, prečítaj si ho"). Zdá sa, že to tak musí byť, lebo proces by musel inak čítať znaky neustále – cyklil by sa a vyčerpával by všetku výpočtovú kapacitu procesora. „Neustále" však stačí interpretovať ako dostatočne často. (Ak je buffer na príjem znakov 1kB, a rýchlosť linky 9600 bps, stačí procesu raz za 0.5 s prečítať všetky znaky z buffra – a nepotrebuje aby ho niekto upozorňoval, že mu nejaké znaky prišli.) Pritom už procesor nemusí zaťažiť vôbec a nižšie vrstvy volá „z vlastnej vôle" – nie je centrálne riadený. Aplikovaním takejto zmeny na všetky časti systému sa dá dosiahnuť, že medzi procesmi v systéme sa nachádzajú iba „služobníci" a žiadni „riaditelia". Aj samotné časovanie procesu je iba služba – nikto procesu nehovorí „teraz sa zobuď", ale proces vraví „teraz chcem pol sekundy spať".

- **práca v reálnom čase**: Je ľahké si všimnúť, že v predchádzajúcom príklade sme zamlčali skutočnosť, že je nevyhnutné, aby bol v systéme niekto, kto vie vykonať žiadosť „chcem pol sekundy spať" tak, že to bude naozaj 0.5 s a nie 2 s. Na to je potrebné, aby žiadny proces neokupoval permanentne procesor. Musí teda ísť o systém reálneho času. Ďalším zamlčaným faktom je, že decentralizované procesy vykonávajú množstvo jalovej práce (v našom príklade ide o čítanie z linky aj vtedy, keď z nej žiadne znaky neprichádzajú). Tu si treba uvedomiť, že táto jalová práca neznižuje možnosti vykonávania užitočnej práce (systém musí byť totiž dimenzovaný tak, že tie znaky môžu prichádzať kedykoľvek, t.j. aj stále).

- **normalizácia komunikačných rozhraní**: Komunikácia medzi procesmi v programových systémoch je spravidla založená na známom vzťahu klient-server. V týchto aplikáciách poskytuje obyčajne každý server sebe vlastné komunikačné rozhranie pre svojich klientov. Klient sa potom na každý server musí obrátiť spôsobom, ktorý je vlastný danému serveru, .t.j. na každý server iným spôsobom. Pritom kódy v serveri zabezpečujúce komunikáciu s klientami sú veľmi podobné. Vzniklo preto niekoľko pokusov normalizovať komunikáciu medzi klientom a serverom. Z nich sú pre nás zaujímavé tie, ktoré dekomponujú serverov a klientov na agentov a ich prostredia.

- **zníženie nárokov na výpočtovú a prenosovú kapacitu**: V rámci architektúry multiagentového systému je potrebné sledovať, aby od počtu pridávaných agentov rástla výpočtová zložitosť zlúčenia ich prejavov do výsledného prejavu systému v najhoršom prípade lineárne. T.j. metóda jeho tvorby musí dôsledne podporovať implementáciu sémantických vzťahov, aby bola interakcia medzi agentami iba lokálna. Pokiaľ napríklad agenti používajú priamu adresnú komunikáciu, nesmie autor zaviesť v systéme komunikáciu každý s každým, pričom každý zahodí z prijatých správ tie, ktoré nie sú preň podstatné. Významným faktorom zlúčenia prejavov agentov je taktiež prenosová kapacita, obzvlášť vtedy, keď je programový systém vykonávaný na vzdialených procesoroch. Ak z hľadiska sémantickej povahy veci určitý agent nepotrebuje spojenie so vzdialeným agentom permanentne, môže byť výhodnejšie dočasne presťahovať na vzdialený počítač tohto agenta, než komunikovať na ďiaľku (pripomíname, že toto by nebolo možné urobiť s centrálne riadeným procesom, respektíve bolo by to možné iba za strašnú cenu).

- **procesná modularita**: Dômyselné použitie určitých typov agentov môže mať následok, že v systéme vznikne oveľa viac (a menších) procesov než by tomu bolo v analogickom programovom systéme (pokiaľ by taký bolo možné reálne vytvoriť).To sa môže kladne prejaviť napr. na znovuvyužiteľnosti kódov. Táto síce funguje iba na princípe zhody, avšak zhody tak malých a jednoduchých častí, že je jej použitie dosť

pravdepodobné. Čím jednoduchšie ciele priradíme k agentom, tým je potenciálne všetkých možných cieľov (a následne kódov agentov) menej a častejšie sa vyskytujú. Máme tu dočinenia s novým druhom modularity systémov.

- **zotaviteľnosť**: Pri tradičných systémoch často reálne hrozí, že narazia na prekážku, ktorú nedokážu zdolať. Napr. nejaký proces, ktorý plní určitú dôležitú službu má v sebe chybu (reálne sa musí počítať s tým, že čím viac kódov sa vytvorí a čím sú menej kvalitne ladené, tým viac chýb obsahujú) a za istých okolností spadne. Tento je možné znovu spustiť a ak už pominul podnet, ktorý vyvolal pád, proces bude ďalej bežať. Ale ak procesy, ktoré jeho službu používali, sa naň odkazovali jeho adresou, táto už bude neplatná, a zreštartovaný proces bude pre ne nepoužiteľný. Použitie nepriamej komunikácie (t.j. komunikácie odkazom cez prostredie) v multiagentových systémoch umožňuje, aby sa procesy využívajúce službu chybného procesu o jeho páde a reštartovaní vôbec nedozvedeli. Je taktiež možné vytvoriť zálohovanie služieb, keď jednu a tú istú službu vykonávajú dvaja rôzni agenti rôznym spôsobom, ktorý dáva zhruba tie isté výsledky. Pritom užívatelia služby nevedia koľko procesov ju v skutočnosti realizuje.

- **robustnosť**: Môže sa taktiež stať, že pri implementácií nejakého procesu niečo prehliadneme, následkom čoho nebude určitá služba v systéme pre danú situáciu adekvátna. V podstate tu ide tiež o chybu, len nespôsobuje pád procesu. Architektúra systému by pre tento prípad mala zabezpečiť, že sa táto chyba natrvalo nevypropaguje do vonkajších prejavov systému. Malo by to spôsobiť buď žiadne, alebo len chvíľkové zaváhanie v správaní systému.

- **Emergencia**: Už sme naznačili, že medzi jednotlivými typmi agentov uprednostňujeme tie jednoduchšie. Je takýto postoj oprávnený? Napríklad už na prvý pohľad je jasné, že reaktívny a tobôž čisto reaktívny agent nedokáže sledovať zložitejšie ciele. To je aj pravda, ale reaktívni a dokonca aj čisto reaktívni agenti to dokážu. Ich sila spočíva v schopnosti vytvárať celky, ktoré prejavujú vlastnosti, ktoré oni sami prejaviť nevedia. Väčšinou na tom nie je nič svetoborné a ľahko si je predstaviť syntaktickú transformáciu spôsobujúcu prenos podstaty týchto vlastností z nižších úrovní systému do vyšších.

  Niektoré typy deliberatívnych agentov napríklad vedia riešiť problémy. Reaktívni agenti to nevedia. Ale je ľahké predstaviť si, že z deliberatívnych agentov vytiahneme všetky reprezentačné štruktúry do ich prostredia (toto sa dá urobiť len vtedy, keď agenti používajú nepriamu komunikáciu – každá priama komunikácia sa však dá transformovať na nepriamu) a proces logického odvodzovania zrealizujeme ako reakcie teraz už reaktívnych agentov na hodnoty týchto štruktúr.

  Podobne reaktívni agenti majú vnútorný stav. Čisto reaktívni ho nemajú. Ale opäť nie je nič jednoduchšie, než vyhodiť globálnu pamäť z reaktívneho agenta do jeho prostredia. Bude si ju musieť akurát zakaždým z tohto prostredia prečítať a opäť ju tam zapísať.

  Medzi jednotlivými architektúrami sa dá teda prechádzať. Otázne je, čo sa tým získa. Jednou z nádejí je, že pri určitých architektúrach sa po implementácií n-vlastností v správaní systému, objaví želaná (n+1)-vá, a my budeme prekvapení a vzrušení kde sa tam sama od seba vzala. Tento jav sa nazýva emergencia – samoobjavenie sa. Je otázne, či emergenciou môže vzniknúť niečo naozaj zaujímavé. Isté však je, že naše vzrušenie sa veľmi rýchlo stratí, lebo keď si položíme otázku „kde sa tá vlastnosť vzala?“, zrejme si na ňu budeme vedieť odpovedať. Pri emergencií máme teda dočinenia skôr s neschopnosťou tvorcu vidieť vopred určité sémantické vzťahy a schopnosťou architektúry umožniť dobrú implementáciu sémantických vzťahov tvorcom videných.

Kto nechce zaťahovať multiagentové systémy do umelej inteligencie, môže tieto vlastnosti pokojne ďalej nazývať zaujímavými. Pri trocha odvahy a nadhľadu nad detailmi však môžeme pokladať tieto vlastnosti za prostriedky k implementácií toho, čo nazývame inteligenciou. Na otázku „prečo vývoj v rôznych oblastiach speje k multiagentovým systémom ?“ si trúfame odpovedať: „preto, lebo v rámci týchto oblastí existuje vedomá alebo nevedomá snaha o vytvorenie inteligentných systémov a multiagentové systémy sú jednou z použiteľných platforiem pre ich vývoj“.

4. Multiagentové systémy zložené z čisto reaktívnych agentov

Možno nie pre všetky odrody multiagentových systémov je toto relevantné, ale pre niektoré určite áno. Odroda, preferovaná autorom tohto príspevku, sú **multiagentové systémy zložené z čisto reaktívnych agentov používajúcich nepriamu komunikáciu cez prostredie, dekomponované aktivitou a implementované inkrementálne metódou zdola-nahor**. Tieto systémy sú totiž prirodzeným vylepšením tradičných programových systémov. Vylepšením, ktoré prináša pokrok pri všetkých vlastnostiach, ktoré sa v oblasti

programových systémov cenia. Ide tu o modulárnosť, otvorenosť, modifikovateľnosť, konfigurovateľnosť a komplexnosť týchto systémov.

Tradičné vzťahy klient - server sú v týchto systémoch jednoduchou syntaktickou transformáciou prebudované na princípe decentralizovaného systému. Ide o to, že server sa dekomponuje na časť, ktorá slúži ako komunikačné rozhranie pre klientov (túto nazveme v zhode s „multiagentovou" terminológiou prostredie) a na jednotlivé obsluhy služieb, ktoré sa stanú samostatnými procesmi – agentami. Klienti sa tým pádom stanú rovnakými agentami ako služby. Prostredie, ktoré vzišlo zo servera, musíme ďalej podriadiť pevne definovanému komunkačnému rozhraniu, ktoré bude rovnaké pre agentov bez ohľadu na to, či vzišli zo servera alebo z klientov. Z každého servera teda vzíde rovnaké prostredie a klienti, ktorí dovtedy používali rôzne rozhrania pre rôznych serverov sa zaobídu s jediným rozhraním pre prostredie.

Agenti v systéme nebudú tým pádom komunikovať priamo, ale iba odkazmi, ktoré budú zanechávať v prostrediach. Týmto odkazom budú rozumieť iba tí agenti, ktorí ich do prostredia zapisujú, alebo ich odtiaľ čítajú. Pre prostredie to budú nič neznamenajúce buffre určitej dĺžky.

Agenti o sebe navzájom nič nevedia, čo ich núti žiť agentím životom: každý agent sa musí opakovane pozerať do prostredia, či tam nenastala nejaká zmena v odkazoch. Prostredie by mu síce dokázalo povedať, že určité odkazy boli prepísané, ale kazilo by to decentralizačný charakter architektúry. Zabudnime teda na zmeny. Pokiaľ by agenta zaujímali v prostredí zmeny, musel by mat globálnu pamäť (aby si pamätal predchádzajúcu hodnotu). Taktiež jeho tvorcu by to mohlo zviesť k písaniu čo najefektívnejšieho kódu, ktorý by sa snažil neprepočítavať to, čo nebolo zmenené a kód by sa tak stával zložitejším. Náš systém však beztak musí počítať s tým, že sa všetky odkazy týkajúce sa agenta môžu zmeniť naraz. Preto si môžu naši agenti dovoliť zakaždým prečítať všetky svoje odkazy z prostredia, spracovať ich jednotným kódom a zapísať do prostredia všetky vypočítané odkazy pre iných agentov (a prípadne aj pre seba). Výhoda čisto reaktívnych agentov spočíva v tom, že žiadny iný prístup neumožňujú a tým nedovoľujú tvorcovi pustiť sa na bočné chodníčky.

Keďže prostredia sú všetky (okrem tých, ktoré stelesňujú prepojenie na fyzické prostredie) rovnaké (a to nielen v rámci jedného systému) a krajne jednoduché, je dosť pravdepodobné, že ich dokážeme napísať bez chyby. Agenti naopak môžu chyby obsahovať a môžu napríklad padnúť. Je však možné ich zreštartovať bez toho, že by sa museli reštartovať akékoľvek ďalšie procesy. Z toho vyplýva veľká schopnosť zotaviteľnosti systému.

Skutočnosť, že všetka pamäť agentov sa nachádza v prostrediach sa veľmi pozitívne prejaví na konfigurovateľnosti systému. Z každej konštanty v systéme sa totiž musí urobiť odkaz. Tento sa zdanlivo celkom zbytočne pri každom kroku agenta číta. Zbytočné je to však len do chvíle, kedy sa zistí že túto konštantu treba meniť počas behu systému. Táto zmena, v obyčajných systémoch bez globálneho reštartu veľmi komplikovaná, je tu zadarmo.

Keďže čisto reaktívny agent má všetku svoju pamäť vyhodenú do prostredia, môže ju ktorýkoľvek iný agent infiltrovať a niekedy tak celkom pozmeniť funkciu agenta. (Ideálne by tu bolo, keby bolo možné v agentovi zmeniť takýmto spôsobom nielen dáta ale aj kód, napr. sčítanie na násobenie – tak ďaleko však ešte nie sme.) To umožňuje širokú mieru otvorenosti a modifikovateľnosti systémov.

Modifikovateľnosť ide pritom ruka v ruke s inkrementálnou implementáciou. Práve na základe infiltrácie odkazov v prostrediach sa môže vyššia a neskôr implementovaná vrstva votrieť do činnosti nižších vrstiev. Pritom je nádej, že také postupy rozširovania a modifikácie systémov budú len veľmi pomaly neznehodnocovať kódy. Dúfame, že sa teda takýmto spôsobom do istej miery odolávať javu, ktorý programátori dobre poznajú pod názvom softwarová kríza. Tento jav je charakteristický tým, že každá zmena a každé rozšírenie programového systému je ďalším a ďalším klincom do jeho rakvy. Po čase sa všetky jeho kódy znehodnotia natoľko, že nik už nie je schopný urobiť v ňom ďalšie rozšírenia a modifikácie a pokiaľ sa o to pokúsi, prejaví sa to stratou funkčnosti novej verzie systému.

Ešte mnoho by sa dalo napísať o vlastnostiach uvažovanej odrody multiagentových systémov, ale z hľadiska čitateľa bude asi dôležitejšie zodpovedať otázku, prečo sa tu týmto praobyčajnými vlastnosťami zaoberáme. Veď tie zaujímajú skôr programátorov, softwarových inžinierov a nie umelých inteligentov. Pokúsime sa teraz presvedčiť čitateľa o opaku. Pokúsime sa ho presvedčiť, že práve v súbore takýchto vlastností spočíva prirodzená i umelá inteligencia.

5. Nové pohľady na umelú inteligenciu

Dá sa vybrať z množstva definícií inteligencie jedna, ktorá by obhajcovi vyššie uvedenej myšlienky poslúžila viac než iné? Myslíme, že áno. V našom prípade je výhodné inteligenciu chápať ako **schopnosť systému adekvátne reagovať na zmeny odohrávajúce sa v jeho prostredí**. Tu si však hneď treba uvedomiť, že nemožno požadovať od inteligentného systému adekvátnu reakciu na akúkoľvek zmenu jeho prostredia. (Ani my sa netvárime najmúdrejšie keď nás ponoria do kotla s vriacou vodou.) Musíme množinu spomínaných zmien prostredia obmedziť len na **určitú triedu** a snažiť sa, aby táto bola netriviálna a čo najširšia. Bude závisieť od

šírky a „tvaru“ tejto triedy a od charakteru správania, ktorého produkciu od nášho systému požadujeme, či bude uznaný za inteligentný alebo nie. Môžeme sa na celú vec pozerať tak., že pre každé požadované správanie existuje určitá latka, ktorú musí šírka a „tvar“ spomínanej triedy preskočiť, aby bol systém, ktorý sa ňou vyznačuje považovaný (ľuďmi) za inteligentný. Existuje teda nejaká spojitá veličina, ktorá je vlastnosťou systému, ktorá ak prekročí určitý limit, systém je inteligentný, inak nie je. Spomeňme napríklad program ELIZA, ktorý dokázal hrať úlohu psychiatra pri rozhovore s pacientom tak dobre, že ho pacienti jednoznačne označovali za skutočného psychiatra. Pritom tento program napíše každý študent umelej inteligencie za jeden večer. To je vysvetliteľné tým, že spomínaná latka pre správanie psychiatra je tak nízko, že ju aj taký jednoduchý program ako ELIZA dokáže preskočiť.

Takéto nazeranie na vec je zaujímavé preto, lebo systémy umelej inteligencie sú pri ňom v limitnom prípade, keď sa spomínaná spojitá veličina blíži k nule (t.j. v prípade triviálnej triedy zmien, nad ktorou systémy operujú adekvátne), totožné s obyčajnými programovými systémami (respektíve s akýmikoľvek počítačovými systémami od ktorých požadujeme nie výsledok – ale správanie sa). To, čo týmto „neinteligentným“ systémom chýba, je teda schopnosť pojať do seba netriviálne širokú triedu zmien a adekvátne nad ňou reagovať. Spomínanou spojitou veličinou nie je teda nič iné, než praobyčajná **komplexnosť**, t.j. schopnosť systému pojať do seba veľa vecí bez toho, aby v nich vznikol zmätok. Na základe toho možno povedať, že **inteligencia je obyčajnou dvojhodnotovou kategorizáciou komplexnosti.**

Umelú inteligenciu potom možno v užšom význame slova chápať ako disciplínu, ktorá sa zaoberá hľadaním kódových a dátových štruktúr a postupmi ich nakódovania a naplnenia, ktoré systému na nich založenom umožňujú dosiahnuť čo najvyššiu komplexnosť.

6. Záver

Na záver by autor príspevku rád konštatoval, že všetky informácie, ktoré v ňom uviedol berie vážne. Nemôže sa však zaručiť, že je to naozaj všetko tak. Možno na celom svete niet človeka, ktorý by dokázal uvedené myšlienky kvalifikovanie prijať alebo odmietnuť. Niet však inej cesty ako k ich odmietnutiu alebo prijatiu dospieť, než pokúšať sa aplikovať uvedené myšlienky v praxi a hľadať pre ne teoretický podklad. Na to treba v prvom rade tieto myšlienky šíriť a teda napríklad písať takéto príspevky.

Literatúra

[1] Brooks R. A.: Achieving artificial intelligence through building robots, A. I. Meno 899, MIT, Cambridge, Mass., (1986)

[2] Brooks R. A.: A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, RA2, (1986), 14--23.

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

[4] Brooks R. A.: Intelligence without representation. Artificial Intelligence 47, (1991), 135-159.

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

[6] Jozef Kelemen: Strojovia a agenty, Archa, Bratislava, (1994)

[7] Andrej Lúčny: Architektúra inteligentných programových systémov, Projekt dizertačnej práce, Matematicko-fyzikálna fakulta, Univerzita Komenského v Bratislave (1997)

[8] Renáta Mlichová: Niektoré experimenty so subsumpčnou architektúrou v nedeliberatívnej robotike, diplomová práca, Matematicko-fyzikálna fakulta, Univerzita Komenského v Bratislave (1993)