Otvorená implementácia architektúry Agent-Space

Andrej Lúčny

Katedra aplikovanej informatiky, Fakulta matematiky, fyziky a informatiky, Univerzita Komenského
Mlynská Dolina, 842 48 Bratislava
lucny@fmph.uniba.sk

Abstrakt

Architektúra Agent-Space v sebe spája myšlienky Minského societného modelu mysle a Brooksovej subsumpčnej architektúry vyjadrené jazykom agentovo orientovaného programovania. Vysvetlíme základnú myšlienku tejto architektúry a zopár technických detailov spojených s vydaním jej otvorenej implementácie a jej použitia v kognitívnej robotike.

1 Úvod

Architektúra Agent-Space myšlienkovo vychádza z Marvina Minského [8] a Rodneyho Brooksa [1]. Je v podstate vyjadrením myšlienok subsumpčnej architektúry [1], pomocou jazyka multiagentových systémov, t.j. vyjadrujeme modulárny systém ako systém distribuovaný. O relatívne starých myšlienkach takto hovoríme novým jazykom, ktorý nám dáva viac možností. Zmenou tohto náhladu dostávame zo subsumpčnej architekúry vhodný implementačný prostriedok pre Minského societný model mysle [9].

Prvú verziu architektúry Agent-Space navrhol autor tohto príspevku v roku 1997 a implemental ju v C pre operačný systém reálneho času QNX4, ktorý bol v tom čase jedinou dostupnou platformou na ktorej by dokázala fungovať v reálnom čase. Jej nároky na výpočtový výkon sú totiž značné, hoci dobre škálovateľné. Architektúra vyžaduje multi-procesové alebo aspoň multi-vláknové prostredie s čo najnižšou latenciou (čo je čas potrebný na prepnutie dvoch procesov alebo vlákien v procesore pri ich preemptívnom preplánovaní). Ako sa zvyšoval výkon počítačov, zvyšovali sa aj možnosti použitia architektúry. Implementácia pod QNX4 bola použitá v rade komerčných produktov a to monitorovacích a riadiacich systémov, ktoré za svoj úspech vďačia mimo iné i jej vlastnostiam [7]. V roku 2001 dostala pod vplyvom koordinačného programovania [2] svoje dnešné meno (predtým zvaná Agent-Environment) a bola prvý krát použitá na akademické účely a to na simuláciu zakladania potomstva kutavkou [4], čo je jedno z najzložitejších vrodených správaní v ríši hmyzu. V roku 2004 bola prvý krát celistvo publikovaná [4], prenesená na prístupnejšiu platformu Java a použitá ako riadiaci systém mobilného robota (Pingpong). Táto implementácia potom bola použitá na viacero pokusov s mobilnými robotmi v nasledujúcich rokoch. V roku 2005 bola prenesená do VRML na použitá na reimplementáciu robota ALLEN [1] vo virtuálnej realite. V roku 2006 bola implementácia v Jave použitá v projekte na 3D rekonštrukciu pohybu chodcov po prechode. V roku 2011 bola vytvorená implementácia v C++ za účelom previazania s OpenCV (podobná myšlienka ako má veľkolepejší projekt ROS).

Pomocou diplomantov boli vytvorené ďalšie implementácie a aplikácie:

- Ronald Weiss vytvoril prostredie na vývoj riadiaceho systému robota v robotickej scéne so vzdialeným prístupom vývojára
- Milan Leitman riadil pomocou agent-space hráčov vo virtuálnom robotickom futbale
- Andrej Riška riadil kráčajúci model mravca pod MRDS
- Branislav Malinovský ovládal model lietajúceho vtáka

Okrem toho sme použili túto architektúru na tzv. robotických prázdninách, kde sa každoročne schádza malý tím nadšencov.

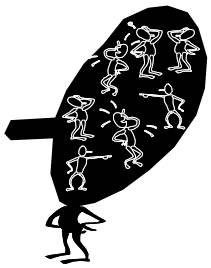
Všetky tieto aktivity volajú po zastrešení nejakým open-source balíkom s dostatočnou dokumentáciou, čo práve činíme. Tento príspevok neprináša nové poznatky, ale sumarizuje rôzne zdroje informácií o nej na jedno miesto.

2 Opis architektúry

Architektúra Agent-Space sa snaží štruktúrou umelého systému napodobniť živé systémy a samotnú ľudskú myseľ. Predpokladá, že najpodstatnejšími vlastnosti inteligentných systémov sú:

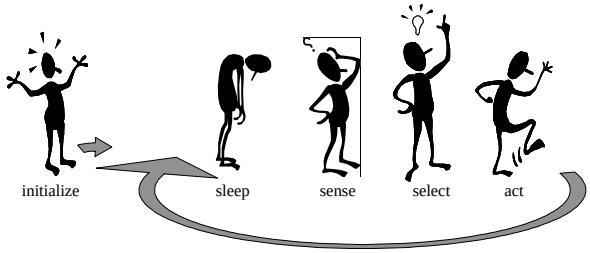
- modularita založená na decentralizácii a masívnom paralelizme
- práca v reálnom čase

Z hľadiska organizácie modulov systému, agent-space nasleduje Minského koncepciu mysle ako množiny tzv. agentov (v neskorších prácach používa Minsky miesto pojmu agent pojem zdroj), kde je želaná globálna aktivita systému dosiahnutá správnou aktiváciou jednotlivých agentov v správnych časových okamihoch (Obr. 1). Správanie jednotlivého agenta je pritom relatívne jednoduché a môže byť ľahko nakódované vývojárom, zatiaľ čo globálne správanie systému je komplexné a povstáva zo vzájomnej interakcie agentov.



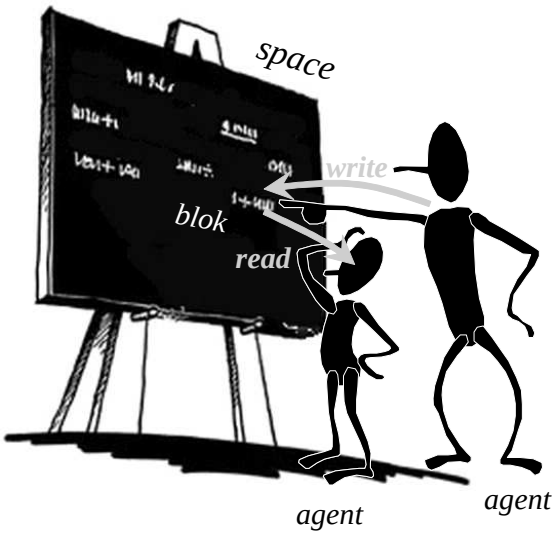
Obr. 1. Systém ako množina agentov

Každý agent má vlastné riadenie nezávislé na ostatných agentoch a vykonáva vlastný kód, pričom tento beží v nekonečnom cykle (Obr. 2). Pri každom prechode týmto cyklom sa na základe vnímania aktuálnej situácie a ďalších informácii uložených v tzv. vnútornom stave agenta vypočíta a vykoná náležitá akcia. Pritom je možné tomuto výpočtu jasne priradiť jeho poslanie v systéme – inými slovami: každému agentovi možno priradiť určitý cieľ.

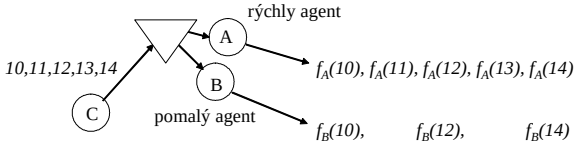


Obr. 2. Životný cyklus agenta.

Najvýznamnejšou zložkou vnímania je pre každého agenta komunikácia s inými agentami. Práve tento komunikačný mechanizmus zaraďuje agenta do systému. V architektúre agent-space je tento mechanizmus založený výlučne na nepriamej komunikácii a je realizovaný tzv. čiernou tabuľou (nástenkou), ktorú nazývame prostredím (v angličtine doslova „priestor“, t.j. *space*). (Obr. 3). Prostredie dokáže uskladňovať pomenované dáta, tzv. bloky, ktoré agenti dokážu čítať, zapisovať a mazať. Agenti pritom musia poznať ich mená a formát dát v nich zapísaných. V osobitých prípadoch je možné pre potreby čítania bloky referencovať aj maskou (regex alebo hviezdičková konvencia).



Obr. 3. Komunikácia medzi agentami cez prostredie.



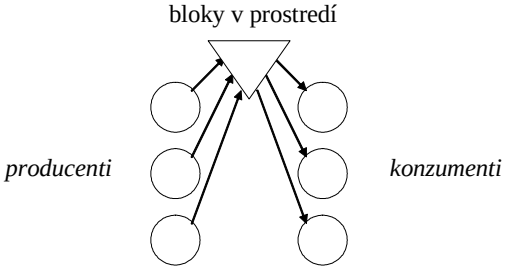
Obr. 4. Implicitné vzorkovanie

Niekoľko detailov manipulácie s blokmi je podstatných, aby to celé fungovalo naozaj dobre. Najdôležitejší z nich spočíva v tom, že na vytvorenie bloku v prostredí nie je potrebné volať špecifickú funkciu. Bloky sa fyzicky vytvárajú prvým zápisom. Čítať ich však možno už predtým a to bez toho, že to vyvolalo nejakú výnimku: pre tento prípad je agent pri čítaní povinný zadefinovať preddefinovanú hodnotu, ktorá sa pri čítaní neexistujúceho alebo prázdneho bloku vráti, ako keby v ňom bola zapísaná. Blok môže obsahovať iba jedinú hodnotu, takže hodnota zapísaná jednou operáciou zápisu je prepísaná nasledovnou. Hodnota zapísaná v bloku je prepísaná bez ohľadu na to, či ju nejaký agent stihol prečítať alebo nie. Takže ak producent zapíše nasledujúcu hodnotu do bloku prv, ako konzument stihne blok prečítať, tak konzument o túto hodnotu proste príde – z jeho pohľadu nikdy v bloku ani nebola. Takže ak konzument nie je dostatočne rýchly, aby stihol z bloku prevziať všetky zapisované hodnoty, automaticky tu príde k ich vzorkovaniu (Obr. 4). Vďaka tomuto javu nie je možné systém preťažiť a vždy beží (ako dobre – to už je iná otázka) v reálnom čase. Je taktiež ľahké kombinovať

pomalé a rýchle moduly, čo je typický problém pre tradičné architektúry, kde má celková rýchlosť spracovania tendenciu prispôsobovať sa najpomalšiemu modulu.

Okrem zápisu zo strany agentov na hodnotu blokov vplýva výrazne i prostredie a to dvomi mechanizmani: časovou platnosťou hodnôt a prioritou hodnôt.

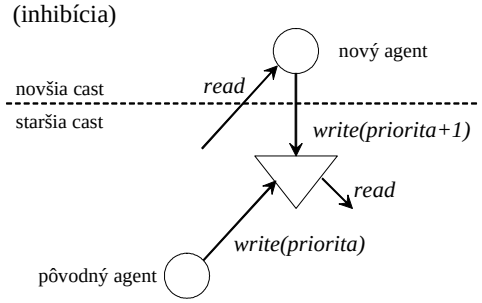
Časová platnosť hodnoty môže byť definovaná agentom pri jej zápise. Agent môže vtedy definovať dobu, po uplynutí ktorej hodnota z bloku automaticky zmizne (ako keby bola vymazaná agentom). Pokiaľ časová platnosť pri zápise nie je definovaná, hodnota ostáva v bloku natrvalo, respektíve do najbližšieho prepísania. Časová platnosť je dobrý nástroj na vyjadrenie dočasného charakteru informácie. Navyše umožňuje nastoliť dátový tok od mnohých producentov k mnohým konzumentom (Obr. 5). Keď máme napríklad viac metód na výpočet určitej hodnoty, pričom každá vyprodukuje hodnotu len za určitých podmienok, môžeme tieto metódy premeniť na agentov zapisujúcich rovnaký blok. Takto sa konzumenti nemusia starať z akého zdroja hodnota pochádza. Musíme však zabrániť tomu, aby nám agent, ktorý nič nespočítal, neprepísal použiteľnú hodnotu od iného agenta. Preto agenti môžu blok prepísať iba keď niečo dobré spočítajú, inak sú ticho. Potom ale zákonite hrozí, že nik nespočíta nič a v bloku ostatne stará posledná hodnota. Tomu však zabráni práve časová platnosť, ktorú v tomto prípade musíme použiť.



Obr. 5. Dátový tok od mnohých producentov ku mnohým konzumentom

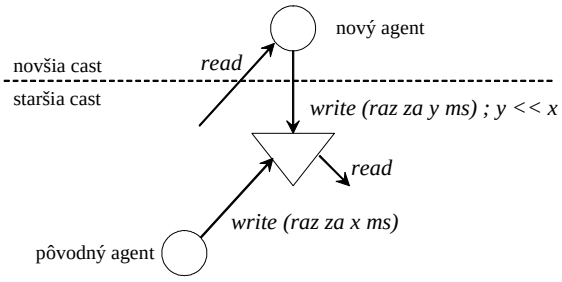
Priorita hodnôt nám zase umožňuje preferovať hodnotu od určitého producenta, čo je užitočné aj vo vyššie uvedenom príklade, keď majú rôzne metódy rôznu presnosť. Priorita je taktiež definovaná agentom pri zápise hodnoty a do života ju uvádza prostredie: keď sa nejaký agent snaží túto hodnotu prepísať, prostredie hodnotu zmení iba ak má zapisovaná hodnota aspoň rovnakú prioritu (zapisujúcemu agentovi sa pritom zdá, že sa jeho hodnota zapíše vždy). Tento mechanizmus je dôležitý, keď sa pri inkrementálnom vývoji snažíme z novo pridaných častí systému ovplyvniť pôvodné (Obr. 6), čo je podstata Brooksovej subsumpčnej architektúry [1]. Pri agent-space sa novo pridaní agenti miešajú do komunikácie medzi pôvodnými:

– noví čítajú bloky, cez ktoré si pôvodní vymieňajú informácie (odpočúvanie)
– noví prepisujú hodnotu v týchto blokoch (supresia)
– noví vymazávajú hodnotu v týchto blokoch (inhibícia)

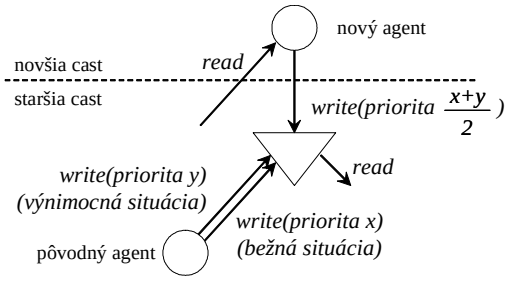


Obr. 6. Supresia starších častí z novších

Agent-space však vie nielen vyjadriť všetky mechanizmy sumbsumpcie (odpočúvanie, supresiu a inhibíciu), ale i komplikovanejšiu koordináciu nových a pôvodných agentov. Nový agent môže použiť rovnakú prioritu ako starý a súperiť tak s ním na základe rôznej frekvencie zápisu: môže ho takto prehlušiť, ale nie úplne umlčať (takže napríklad priemer hodnôt za určité časové obdobie je zhodný s hodnotou nového agenta, ale minimum či maximum sa výrazne líšia) (Obr. 7)



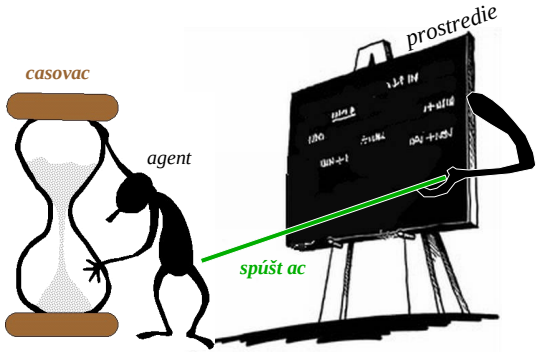
Obr. 7. Čiastočná supresia na základe rôznej frekvencie



Obr. 8. Selektívna supresia na základe priority

Taktiež je možné zariadiť, aby starý agent pri zápise používal rôzne priority a nový na svoju aktivitu využíval prioritu medzi mini. Takto je možné zariadiť, že za bežných podmienok potlačí nový agent starého, ale za výnimočných okolností je to naopak (Obr. 8).

Po vysvetlení mechanizmu manipulácie s blokmi ostáva teraz vysvetliť, čo vedie agentov k tomu aby ju konali práve v určitom čase. Pri každom prechode svojím cyklom sa agent najprv zablokuje a zaspí. V tomto stave strávi väčšinu času, čím dopraje výpočtovú kapacitu ostatným agentom. K prebudeniu môže prísť na základe dvoch podnetov: časovača (angl. Timer, Obr. 9 vľavo) alebo spúšťača (angl. Trigger, Obr. 9 vpravo). Časovač budí agenta v pravidelných časových intervaloch. Dôsledkom toho je isté oneskorenie pri propagácii udalosti reťazou takto budených agentov. Toto oneskorenie môže dokonca pri komplikovanejších tokoch dát medzi agentmi viesť k dočasným nekonzistentným stavom, ale obvykle sa s ním dá žiť. Za určitých okolností sa dokonca môžu tieto stavy prejaviť na globálnom správaní systému ako tvorivý prvok.



Obr. 9. Budenie agenta časovačom a spúšťačom

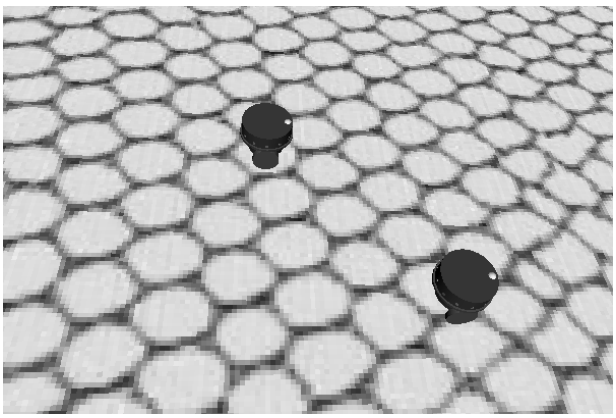
Vyhnúť sa im dá použitím spúšťača, ktorý zobudí agenta v okamihu keď dôjde ku zmene hodnoty niektorého bloku, ktorého meno či mennú masku agent zaregistroval u prostredia. Má zmysel uvažovať viac druhov spúšťačov podľa toho akú informáciu vedia pri zobudení agentovi poskytnúť: nič, mená zmenených blokov, alebo aj hodnoty, ktoré blok nadobúdal, kým agent spal.

3 Implementácie

Na www.agentspace.org sme v ostatnom čase zverejnili dve otvorené implementácie.

Implementácia v Java pre VRML

V multivláknovom prostredí Javy sme implementovali agenta ako objekt s vlastným vláknom, prostredie (space) ako objekt vzoru singleton. Celá implementácia je kompatibilná s Cortonou 4.2, ktorá je dostupným VRML prehliadačom s dobrou podporou merania vzdialeností a detekciou kolízii (na rozdiel od jej neskorších verzií). To si vyžiadalo obmedziť sa Javu nižšej verzie, preto táto platforma nie je veľmi progresívna, nič menej sa v nej dajú poľahky robiť experimenty. Platforma je dostupná spolu s príkladom použitia pre modelovanie mechanizmu Freudovej „pumpy slasti“ (Obr. 10)
www.agentspace.org/download/PleasurePump-ver3.zip
Video, ktoré sa z takto vykonanej simulácie dá generovať v Cortona Movie Makeri je na:
http://youtu.be/Kmk2LI2rRpQ



Obr. 10. Príklad použitia v kognitívnej robotike

Implementácia v C++

Implementáciu pre GCC pod MinGW sme postavili na knižnici pthread a agenta implementovali ako objekt s vlastným vláknom. Hojne sme museli využívať parametrický polyformizmus v C++. Zdrojové kódy sú dostupné na: www.agentspace.org/uiakv3

4 Záver

Na rozdiel od tradičných prístupov, architektúra Agent-Space upúšťa od rozdelenia systému na softwarové moduly, ktorých rozloženie sa ničím neodlišuje od rozloženia hardware, t.j. podobá sa plošné spoje prepojené vodičmi. Hoci i pri takomto rozložení sa dajú vytvoriť užitočné systémy (predovšetkým tu máme na mysli riadiace systémy priemyselných robotov), už dnes čelíme úlohám (hlavne v mobilnej robotike), ktorých riešenie si vyžaduje zložitejšie dátové toky medzi modulmi. Ich rozloženie sa potom skôr ponáša na organizáciu živých organizmov. Jedna časť systému musí povstávať z druhej a druhá zase z prvej. Napríklad rozpoznávanie objektu na videu závisí od kvality

segmentácie obrazu. Akonáhle je však objekt rozpoznaný, je možné podnietiť väčšiu kvalitu segmentácie v danom mieste obrazu. Vzhľadom na charakter komunikácie cez prostredie, architektúra Agent-Space umožňuje takáto toky v systéme zavádzať. A to bez nebezpečenstva komunikačného uviaznutia či ohrozenia práce v reálnom čase.

Poďakovanie

Tento príspevok vznikol za podpory grantovej agentúry ASFEU v rámci grantovej úlohy KC-INTELSYS, ITMS ITMS 26240220072.

Literatúra

[1] R. A. Brooks: *Cambrian Intelligence*, The MIT Press, Cambridge, Mass., 1999.

[2] Gelernter, D.: *Generative Communication in LINDA.* ACM on Transactions on Programming Languages and Systems, Volume 7(1), 80-112, 1985

[3] Kelemen, J.: *The Agent Paradigm.* Computing and Informatics, Vol.22. (2003), pp. 513-519

[4] Lúčny, A.: *Hľadanie kvalitatívneho rozdielu.* In: Kognice a umělý život (Kelemen, J. – Kvasnička, V. – Pospíchal, J., eds.), FPF SU, Smolenice, 2001, pp. 167-176

[5] Lúčny, A.: *Building Complex Systems with Agent-Space Architecture*. Computing and Informatics, Vol. 23 (2004), pp. 1001-1036

[6] Lúčny, A.: *Od medzimodulových spojení k nepriamej komunikácii medzi agentami.* Znalosti, VŠE Ostrava, 2007.

[7] Lúčny, A.: *Advantages of Multi-agent Approach to Building of Monitoring Systems.* In: Informatics 2007 (Ivan Plander ed.), SSAKI, Bratislava, 2007

[8] Lúčny, A.: *Multiagentový prístup k modelovaniu mysle - alebo ako sledovať pingpongovú loptičku*. In: Umelá inteligencia a kognitívna veda III. (Kvasnička V. ed.), STU, Bratisva, 2011

[9] Minsky, M.: *Society of Mind.* Simon & Schuster, New York, 1986