Agenti ali samo še en objektno usmerjen pristop?

Dejan Lavbič, Matjaž Gams
Dejan.Lavbic@fri.uni-lj.si, Matjaz.Gams@ijs.si

Povzetek

Agentno usmerjene tehnologije veljajo za najmodernejše in najbolj perspektivne pristope k razvoju programske opreme. Cilj agentnih tehnologij je predvsem izgradnja sistemov v dinamičnih in odprtih okoljih, v katerih se lahko prilagajajo spremembam z uporabo avtonomnih komponent. V primerjavi z objekti lahko opazimo številne podobnosti, vendar tudi številne razlike, zaradi katerih je agente smiselno obravnavati ločeno od objektov. V ta namen so bile razvite številne metodologije. Zavedati se moramo, da gre za novo tehnologijo, ki jo v praksi pestijo številne uvajalne težave, a kljub temu veliko obeta.

Ključne besede: agent, objekt, objektno usmerjen pristop, agentno usmerjen pristop, metodologija

Abstract

Agents or no More than Another Object-Oriented Approach?

Agent-oriented technologies are one of the most perspective and modern approaches to software development. The main goal of agent technologies is creating systems situated in a dynamic and open environment, being able to adapt to changes with the use of autonomous components. The comparison with objects identified several similarities, but there are still many distinct features that make different handling of objects and agent reasonable. Several agent-oriented methodologies have been developed that help users cope with complexity of dynamic environments. Agents are a new technology with several introductory issues, but with a great potential.

Keywords: agent, object, object-oriented approach, agent-oriented approach, methodology

1 Uvod

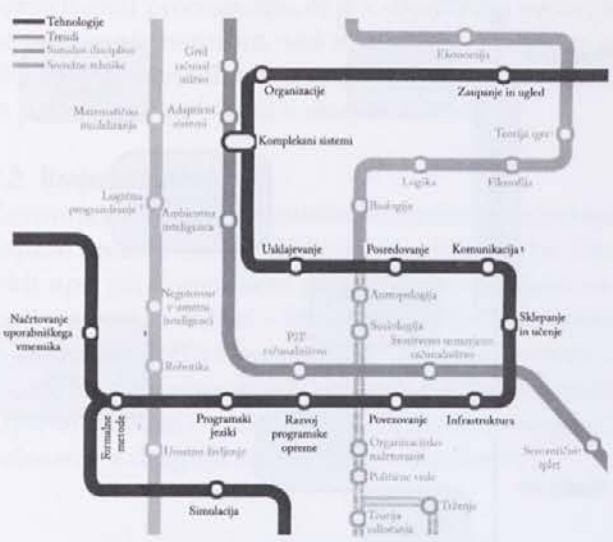
Agentno usmerjeni informacijski sistemi so trenutno eno bolj odmevnih in pomembnih raziskovalnih področij na področju informacijske tehnologije. Agenta lahko preprosto opredelimo kot računalniški sistem, ki je sposoben prilagodljivega in avtonomnega delovanja v dinamičnem in nepredvidljivem okolju [28]. Odprta in dinamična okolja, v katerih morajo računalniški sistemi delovati, zahtevajo izboljšave tradicionalnih računalniških modelov in paradigem. Tako se je pojavila zahteva po določeni stopnji avtonomije, ki bi omogočala dinamično samostojno odzivanje na spremenljive okoliščine. Veliko raziskovalcev je prepričanih, da so po objektno usmerjenem pristopu agenti ena pomembnejših paradigem pri razvoju programske opreme.

Koncept agenta se je izoblikoval iz različnih disciplin na področju informacijske tehnologije (slika 1) – računalniška omrežja, razvoj programske opreme, umetna inteligenca, porazdeljeni sistemi, pridobivanje in obvladovanje informacij, kontrolni sistemi, podpora odločanju, elektronsko poslovanje itd. V današnjem času, predvsem pri implementaciji, spletne storitve ponujajo nov način poslovanja s pomočjo standardiziranih orodij in podpirajo storitveno usmerjen pogled na neodvisne programske komponente, ki s pomočjo medsebojnega sodelovanja prinašajo funkcionalno dodano vrednost. V podobnem kontekstu se uporabljajo agentne tehnologije, ki počasi prihajajo v ospredje.

V nadaljevanju bo predstavljena opredelitev agentov, objektov in večagentnih sistemov ter opredelitev glavnih lastnosti. Predstavljene bodo najbolj pogosto omenjane lastnosti agentov in primerjava z objekti na ravni teh lastnosti in iz načrtovalskega stališča. Sledila bo predstavitev evolucije programskih jezikov in kako so se skozi čas spreminjali različni pristopi, vse od monolitnega pristopa do agentov in večagentnih sistemov. Ker nove tehnologije za razvoj informacijskih sistemov potrebujejo orodja in metodologije, bo primerjalno predstavljeno tudi to področje. Poudarek bo na opredelitvi glavnih pojmov, ki se uporabljajo ter na genealogiji agentno usmerjenih metodologij. Pred koncem sledi še razdelek, v katerem so predstavljena mnenja analitikov o zrelosti agentno usmerjenega pristopa.

2 Agenti in objekti

Opredelitev agentov ni trivialna, saj na to področje obstajajo številni pogledi [15] z različnih področij.



Slika 1: **Agentne tehnologije in povezava z drugimi področji [15]**

Ključni lastnosti agentov, ki ju vsi priznavajo, sta *avtonomnost* in *proaktivnost*. Druge lastnosti, kot sta npr. socialnost in mobilnost, so dodatne in opredeljujejo poseben podtip agenta, medtem ko nekaterih drugih lastnosti ne moremo označiti kot deterministične, saj si jih agenti delijo z objekti. V tem kontekstu je *avtonomen agent* neodvisna entiteta, ki se lahko povsem samostojno odloča o svojem načinu delovanja, še posebno kako se odziva na zahteve drugih agentov. Za proaktivnega agenta velja, da lahko samostojno deluje brez vpliva navodil uporabnika. Zavedati se moramo, da se pri tej opredelitvi pojavijo določene težave, saj številni prispevki s področja reaktivnih agentov (tistih, ki reagirajo na razmere brez modela sveta) po tej kategorizaciji ne spadajo med agente. Čeprav je vloga reaktivnih agentov na nekaterih področjih dominantna, je v realnosti večina agentov s proaktivnimi in reaktivnimi lastnostmi. Tako je uravnovešenje obeh delov ključni izziv za načrtovalce agentno usmerjenih informacijskih sistemov.

Nekateri viri agente primerjajo z objekti, zlasti pri snovanju sistemov. Nekateri vidijo agente kot »pametne objekte« ali »objekte, ki lahko rečejo ne«. Intuitivno se zdi, da imajo agenti in objekti marsikaj skupnega. S takšno opredelitvijo je izvedljivost hibridnih agentno objektnih sistemov na dlani. Spet drugi vidijo agente na veliko višji ravni abstrakcije, podobno kot strokovnjaki OO gledajo na komponente na več granularnih ravneh.

Ena od posledic te visokonivojske opredelitve je, da agenti sodelujejo v ciklusih sprejemanja odločitev. Za dosego tega je treba upoštevati tudi druge lastnosti na nižjih ravneh, kot so npr. vloge, ki jih opravljajo agenti, podoba agentov z mentalnim stanjem, kar vključuje spretnosti in odgovornosti ter tudi zmožnosti in sposobnosti. Ko govorimo o interakciji s pomočjo zaznav okolja in akcijami, ki jih izvedejo nad drugimi agenti ter okoljem, se ukvarjamo s pojmi, kot so zaznava, akcija, cilji in jeziki za komuniciranje med agenti. Sposobnost pogajanja vključuje Contract Net, strategije dražb in idejo tekmovanja ter sodelovanja. Nekateri avtorji [10] celo navajajo, da imajo socialni interaktivni večagentni sistemi v odprtem okolju večjo računsko moč kot univerzalni Turingovi stroji.

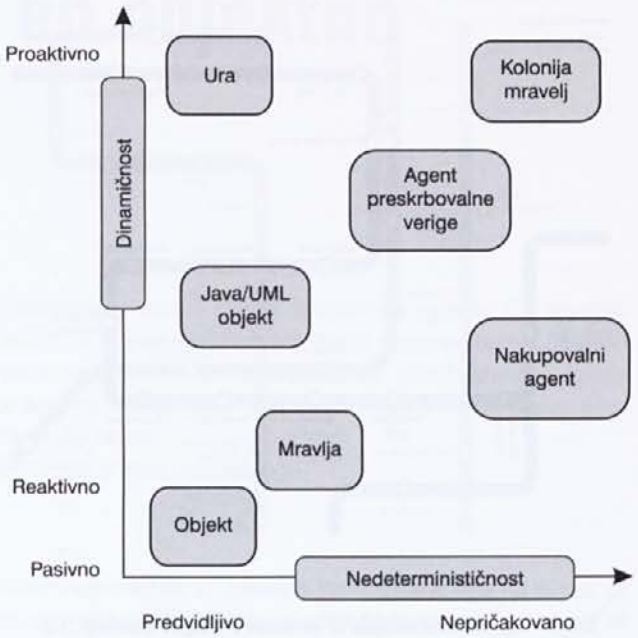
Na podlagi omenjenega lahko objekt opredelimo kot entiteto, ki ograjuje stanje in je sposobna izvajanja akcij ali metod v okviru tega stanja ter medsebojno komunicira z drugimi entitetami s pomočjo izmenjave sporočil. Agent je po drugi strani računalniški sistem, ki izvaja avtonomne akcije z namenom doseganja načrtovanih ciljev.

Opredelitev večagentnega sistema (MAS) prav tako ni preprosta. Kljub temu skoraj vse opredelitve, ki jih lahko najdemo v literaturi, MAS opredeljujejo kot sistem, ki je sestavljen iz sodelujočih agentov z namenom dosega individualnega ali skupnega cilja. S stališča razvoja programske opreme je ena najpomembnejših lastnosti MAS-a ta, da končna množica agentov v času načrtovanja ni znana (opredeljena je zgolj začetna množica), marveč se dokončno oblikuje šele pri izvajanju. To v praksi dejansko pomeni, da MAS temelji na odprtih arhitekturah, ki omogočajo novim agentom dinamično priključevanje v sistem in zapuščanje le-tega [4]. Največja razlika med pristopoma AO in OO je v tem, da lahko pri pristopu OO novi objekti algoritemsko prihajajo in zapuščajo sistem med izvajanjem, medtem ko lahko agenti to počnejo avtonomno s svojim proaktivnim načinom delovanja, ki ni popolnoma napovedljiv oz. algoritemsko vnaprej določljiv. MAS je sestavljen iz agentov (in ne objektov), tako da ima posledično tudi lastnosti agentov. Tako lahko kot ključne lastnosti MAS-a izpostavimo avtonomnost, situiranost, proaktivnost in socialnost. Izmed omenjenih je morda proaktivnost najbolj sporna, saj smo že prej omenili, da agenti delujejo tako proaktivno kot tudi reaktivno. Poleg teh dveh lastnosti so še

druge višjenivojske lastnosti, kot so prilagodljivost in vzdržljivost.

V praksi agenti precej zaostajajo za definicijami, vendar ne glede na to, s katerimi lastnostmi opišemo agenta, se razlikuje od tradicionalnega objekta. Pri načrtovanju agentov pogosto uporabljamo pojme, kot je znanje, prepričanje, namen, obveznost, medtem ko objekte načrtujemo preprosto kot enkapsulacijo njihove notranje strukture v obliki metod in atributov [21]. Definiciji agenta in objekta se zelo razlikujeta. Stopnja avtonomnosti agentov in objektov je zelo različna. Objekti nimajo nadzora nad svojim delovanjem, ker jih kliče zunanja entiteta. Agenti se prav tako lahko odločijo, ali bodo določeno akcijo izvedli ali ne. Za notranjo predstavitev znanja se pogosto uporabljajo ontologije, kar olajšuje izmenjavo znanja med agenti in tudi človeškim uporabnikom.

V nadaljevanju bodo predstavljene razlike med objekti in agenti z vidika nekaterih lastnosti agentov, ki se omenjajo najbolj pogosto.



Slika 2: **Primerjava ravni avtonomnosti pri objektih in agentih**

2.1 Avtonomnost

Kot ključno lastnost agentov lahko izpostavimo avtonomnost kot sposobnost izvajanja akcij neodvisno od zunanjih entitet. Avtonomnost laže obravnavamo v obliki stopnje prisotnosti, kot pa če preprosto določimo, da je navzoča ali pa je ni. Za lažje razumevanje si predstavljajmo avtonomnost z dveh vidikov: **dinamičnost** in **nedeterminističnost** (slika 2).

Dinamična avtonomnost se odraža v izvajanju aktivnosti na pasiven, reaktiven ali proaktiven način. Proaktivni agenti se glede na stanje okolja, dogodkov in sporočil, odločajo, katero akcijo bodo izvedli. Torej se agent lahko sam odloči, kdaj bo izvedel akcijo. Objekti so po drugi strani pasivni, saj se klic metode zgodi v izvajalni niti entitete, ki je zahtevala to metodo. Koncept avtonomnosti zato težko povežemo z objekti, saj je njihovo izvajanje in klic dejansko odvisen od drugih komponent sistema. Kljub temu lahko najdemo v jezikih za modeliranje (npr. UML) in programskih jezikih (Java, .NET itd.) dogodkovno usmerjena ogrodja, ki omogočajo objektom večjo mero avtonomnosti.

Pri nekaterih agentih se srečamo tudi z nedeterminističnim avtonomnim delovanjem. Z vidika okolja se lahko delovanje agenta giblje od popolnoma predvidljivega do nenapovedljivega. Objekti prav tako niso vedno popolnoma predvidljivi, čeprav objektno usmerjeni jeziki težijo k predvidljivem pristopu kot temeljni filozofiji sistema. Na primer, ko se objektu pošlje sporočilo, se metoda, ki sporočilo obdela, kliče popolnoma predvidljivo. Res je, da je lahko v metodi opredeljeno, ali se zahteva obdela ali ne, in če se obdela, kako. Kljub temu se v splošni praksi v primeru, da objekt zavrne sporočilo, to obravnava kot napaka, medtem ko pri agentih ni tako. Delovanje agenta je nedeterministično tudi zato, ker je enkapsulacija pri agentno usmerjenem pristopu obravnavana bolj nejasno. Lahko se zgodi, da zahtevani način delovanja, ki ga opravi agent, sploh še ni znan. To je zelo jasna razlika med objekti in agenti, saj objektni jeziki omogočajo delo le z objekti, ki podpirajo določen vmesnik. Pri objektnem pristopu zato kodiranje dinamičnega delovanja postane težje, saj mora programer natančno poznati vmesnik objekta, s katerim bo komuniciral. V objektno usmerjenem svetu prav tako ne obstajajo mehanizmi za »objavo« vmesnikov, medtem ko se pri agentih uporabljajo drugi mehanizmi, kot je storitev objave in prijave, registracijski protokol, storitev rumenih, zelenih in belih strani itd. Pri agentih medsebojna komunikacija poteka asinhrono, kar pomeni, da ne obstaja točno določen kontrolni tok od enega agenta do drugega. Agent lahko avtonomno spremeni način delovanja in ne samo, ko prejme sporočilo. Zelo pomembna je tudi podpora

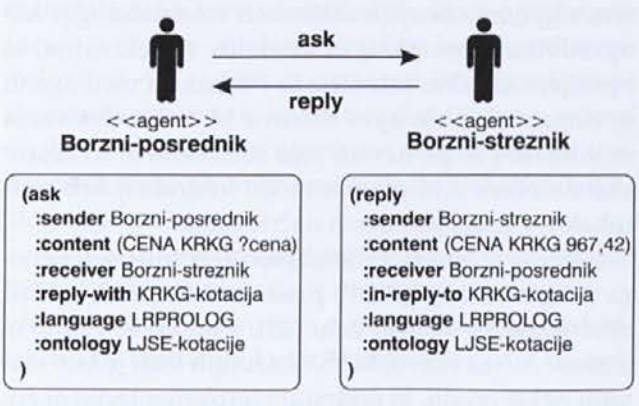
paralelnemu procesiranju, ki je v objektnem svetu po navadi implementiran nad objektnim modelom in objektno usmerjenim okoljem. Pri agentih so objekti in paralelizem povezani v skupen model.

2.2 Vzajemno delovanje

Ravno tako kot avtonomnost vzajemno delovanje najlaže predstavimo v obliki stopnje interakcije. Objekti npr. pri vzajemnem delovanju uporabljajo najbolj preprosto obliko – klic metode. Bolj zapletena interakcija je odziv na zaznane dogodke v okolju, medtem ko v večagentnih sistemih najdemo najbolj zapleteno obliko – večkratno in vzporedno vzajemno delovanje z drugimi agenti (družba agentov).

Slika 3: **Primerjava ravni vzajemnega delovanja pri objektih in agentih**

Sporočilo pri objektu za zahtevo izvedbe operacije lahko zahteva izvajanje samo ene operacije. Pogoj je tudi, da je poslano sporočilo točno določene oblike. Pri medsebojni komunikaciji med agenti se uporablja Agent Communication Language (ACL), ki je vsebinsko bolj bogat kot objektno usmerjeni. Medtem ko lahko ACL vsebuje niz poljubne oblike, ki je še vseeno skladen s formalno sintakso, mora objektno usmerjena metoda vsebovati parametre, katerih število in zaporedje je fiksno. Jezik ACL je tako nujno potreben za komunikacijo med agenti in včasih med agenti in



Slika 4: **Primer komunikacije FIPA-ACL iz domene borznega poslovanja**

objekti. V praksi se najbolj pogosto uporabljata standardizirana formata KQML in FIPA ACL. S pomočjo omenjenih jezikov lahko agenti medsebojno komunicirajo, pri tem uporabljajo semantiko v obliki ontologij in sodelujejo v vzorcih sodelovanja (protokoli pogajanja, kot je npr. Contract Net).

Agenti se poleg enostavnih klicev metod uporabljajo pri dolgotrajnem vzajemnem delovanju. Prav tako lahko vzporedno sodelujejo pri večkratnih transakcijah s pomočjo uporabe večnitnega pristopa. Vsaka vzpostavljena seja pridobi enolični identifikator. Z običajni objektnimi jeziki težko zadostimo takšnim zahtevam, vendar se moramo zavedati, da se objekti kljub temu lahko uporabljajo kot elementi za izmenjavo pri komunikaciji med agenti.

Pri komunikaciji med agenti nam jezik ACL omogoča uporabo številnih načinov medsebojnega delovanja, ki niso standardni. Lahko npr. opredelimo osrednjega posrednika, ki odpošilja zahteve določenim dobaviteljem storitev na podlagi algoritma. Ta algoritem pa je lahko neodvisen od tipa vmesnika in je npr. odvisen od cene ali dostopnosti. Prav tako je pri agentih dokaj preprosto opredeliti entiteto za zagotavljanje anonimnosti pri zahtevah storitev. Z močno tipizirani modeli, kot so CORBA, RMI in Jini, lahko podpremo takšne vzorce, vendar je to veliko bolj zahtevno kot z uporabo agentnih tehnologij.

2.3 Druge lastnosti

V praksi sta avtonomnost in vzajemno delovanje ključni lastnosti pri razlikovanju objektov in agentov. Vendar so še številne druge lastnosti, po katerih se agenti razlikujejo od objektov.

Lastnosti na ravni instanc. Vsak objekt vsebuje lastnosti, ki so opredeljene v pripadajočem razredu. Podobno analogijo najdemo tudi v agentnem svetu, vendar agenti lahko imajo (ali jih spremenijo) poleg vnaprej določenih še druge lastnosti, ki niso opredeljene na razredni ravni, marveč na ravni posameznega agenta (instance). To je recimo zelo priljubljeno pri genetskem programiranju, pri katerem potomcu vsak od staršev prispeva del genov, medtem ko se potomec kasneje uči naprej. Takšen pristop najdemo pri večagentnih sistemih na področju umetnega življenja (slika 1).

Večkratna in dinamična klasifikacija. Pri objektnih jezikih je objekt instanca razreda. Ko objekt enkrat kreiramo, ne more spremeniti svojega razreda oz. postati instanca kakšnega drugega razreda (razen pri

dedovanju). Pri agentih imamo na voljo bolj prilagodljiv pristop, saj ima lahko neki agent v določenem trenutku več vlog – oseba, zaposleni, zakonski partner, stranka ipd. Če agenta odstranimo z delovnega mesta, vsi podatki, povezani s to vlogo, agentu niso več na voljo. Tako agent ostaja ves čas ista entiteta z različnim naborom lastnosti. Tudi v objektnem svetu obstaja pojem opredelitve vlog, vendar večina jezikov OO tega mehanizma ne podpira neposredno, kljub temu da ga podpira UML. Prav tako imajo lahko agenti različne vloge na različnih področjih delovanja (vloga zaposlenega v službi in vloga zakonskega partnerja doma).

Analogija z naravo. Avtonomni in interaktivni značaj agentov je veliko bliže naravnim sistemom kot pa objekti. Agenti lahko delujejo v skladu z lastnostmi zajedavstva, simbioze ali posnemanja. Prav tako so lahko del reprodukcije, v kateri sledijo Darwinovim načelom. Družba agentov odraža politične in organizacijske posebnosti, npr. način organizacije, ki je lahko anarhičen ali demokratičen. Narava torej ponuja številne ideje, ki jih lahko uporabimo pri načrtovanju večagentnih sistemov.

3 Razvojni pristopi in programski jeziki

Slika 5 prikazuje evolucijo programskih jezikov od monolitnega, modularnega, objektno in agentno usmerjenega pristopa. Najprej je bila osnovna enota celoten program, nad katerim je imel programer popoln nadzor. Programer je bil odgovoren za stanje programa, medtem ko je za klice skrbel sistemski operater. Takrat ni bilo mogoče govoriti o modulih, saj še ni bil znan koncept ponovne uporabljivosti.

Zaradi naraščajoče kompleksnosti programov so programerji uvedli določeno mero organiziranosti. To je pripeljalo do modularnega pristopa, pri katerem je bila koda organizirana v manjše enote, te pa je bilo mogoče uporabiti v različnih rešitvah. V tem obdobju procedur je bilo stanje enote določeno od zunaj s pomočjo argumentov, klici pa so bili prav tako zunanji.

Objektno usmerjen pristop je modularnemu dodal tako obvladovanje segmentov kode kot tudi dostop do lokalnega nadzora nad spremenljivkami, do katerih dostopajo metode. Kljub temu v tradicionalnem objektno usmerjenem pristopu objekte obravnavamo kot pasivne, saj pride do klica njihovih metod le takrat, ko jim neka zunanja entiteta pošlje sporočilo.

Agenti, kot zadnja stopnja v tem pregledu, pa imajo svoj nadzor nad izvajanjem, pri katerem je poleg načina delovanja in stanja lokaliziran tudi njihov klic.

	Monolitni pristop	Modularni pristop	Objektivno usmerjeni pristop	Agentno usmerjeni pristop
Način delovanja enote	Ne-modularen	Modularen	Modularen	Modularen
Stanje enote	Zunanje	Zunanje	Notranje	Notranje
Klic enote	Zunanji	Zunanji (klic)	Zunanji (sporočilo)	Notranji (pravila, cilji)

Slika 5: **Evolucija programerskih pristopov [25]**

Takšni agenti imajo določena pravila in cilje, zato jih nekateri imenujejo tudi aktivni objekti. Kdaj in kako bo agent sprožil neko akcijo, je tako določeno s strani agenta [16].

4 Metodologije

Številne agentno usmerjene (AO) metodologije (npr. Gaia, Tropos itd.) uporabljajo idejo človeške organizacije (razdeljeno v podorganizacije), v kateri agenti igrajo eno ali več vlog in vzajemno sodelujejo med seboj. Strukture in modeli človeške organizacije se zato uporabljajo za načrtovanje MAS (podrobneje je to predstavljeno z arhitekturnimi vzorci v metodologiji Tropos in organizacijskimi modeli v MAS-CommonKADS). Koncepti, kot so vloge, socialna odvisnost in organizacijske vloge, se ne uporabljajo samo za modeliranje okolja, v katerem bo sistem deloval, marveč tudi za sam sistem. Z organizacijsko naravo MAS je ena najpomembnejših aktivnosti v metodologiji AO opredelitev interakcij in modelov sodelovanja, ki opisujejo socialna razmerja in odvisnosti med agenti in vlogami, ki jih imajo v sistemu. Modeli sodelovanja in interakcij so po navadi zelo abstraktni in so dejansko določeni z implementacijo interakcijskih protokolov v kasnejših fazah načrtovanja.

Čeprav je Shoham paradigmo agentno usmerjenega programiranja (AOP) predstavil že pred več kot desetimi leti v svojem delu [23], v praksi še vedno ni jezikov AO za razvoj MAS. V zadnjih letih je bilo razvitih nekaj orodij, ki podpirajo implementacijo agentov in večagentnih sistemov, vendar nobeno od njih

ne temelji na agentno usmerjenem jeziku. Obstajajo sicer agentni jeziki za medsebojno komunikacijo med agenti, kot sta KQML in FIPA-ACL. Zelo zanimiv in dolg seznam orodij za razvoj agentov se nahaja na spletni strani AgentLink.[1] Trenutna razvojna orodja temeljijo na Javi in za implementacijo uporabljajo objektno usmerjeno paradigmo. Metodologije AO po navadi niso usmerjene v fazo implementacije, čeprav jih večina poda smernice za to. Najbolj uporabljani razvojni orodji sta JACK [1] in JADE [2].

4.1 Kaj je agentno usmerjena metodologija

Metodologija je sestavljena iz dveh pomembnih komponent: prva opisuje **procesne elemente** pristopa, druga pa je osredinjena na **produkte** in **dokumentacijo**. Predvsem druga je bolj vidna pri sami uporabi metodologije, kar je razlog, da se jezik za modeliranje OO – UML [18], tako pogosto (in napačno) enoti z vsemi »koncepti OO« ali se ga celo opisuje kot metodologija. Jezik za modeliranje, kot je ta ali na drugi strani AUML [17], ki je usmerjen v agentno usmerjeni razvoj, vsekakor prinaša dodano vrednost, vendar ima omejeno področje delovanja v kontekstu metodologije. Poudarki na strani produktov metodologije se razlikujejo med avtorji. Nekateri uporabljajo UML/AUML, medtem ko se drugi tega izogibajo zaradi nezadostne podpore konceptom, vezanim na agente, in zato uporabljajo svojo notacijo in koncepte. Pri uporabi diagramov iz družine UML se od bralca pričakuje, da je seznanjen s takšno grafično notacijo, v nasprotnem primeru avtorji opredelijo notacijsko množico ikon, ki se uporablja v tem posebnem metodološkem pristopu.

Vsaka metodologija mora podpirati zadostno stopnjo abstrakcije, da lahko v celoti modeliramo agente in MAS. Izkazalo se je, da so preproste nadgradnje metodologij OO preveč omejujoče, saj so osredinjene samo na objekte. Po drugi strani pa imamo pri metodologijah AO (agentno usmerjenih) organizirane skupnosti agentov, ki igrajo določene vloge v svojem okolju. V takšnem večagentnem sistemu agenti vzajemno delujejo glede na protokole, ki jih opredeljujejo vloge agentov.

Prav tako se moramo vprašati, kaj za metodologijo pravzaprav pomeni, da je »agentno usmerjena«, v istem smislu kot govorimo o metodologiji OO v kontekstu objektne tehnologije. V metodologiji OO za opis metodologije uporabljamo koncepte OO, ki se v nadaljevanju uporabijo za izgradnjo objektno usmerjenih sistemov. V nasprotju pri metodologiji AO v splošnem nimamo v mislih metodologije, ki je zgrajena iz agentno usmerjenih konceptov, marveč je le usmerjena k izgradnji agentno usmerjene programske opreme.

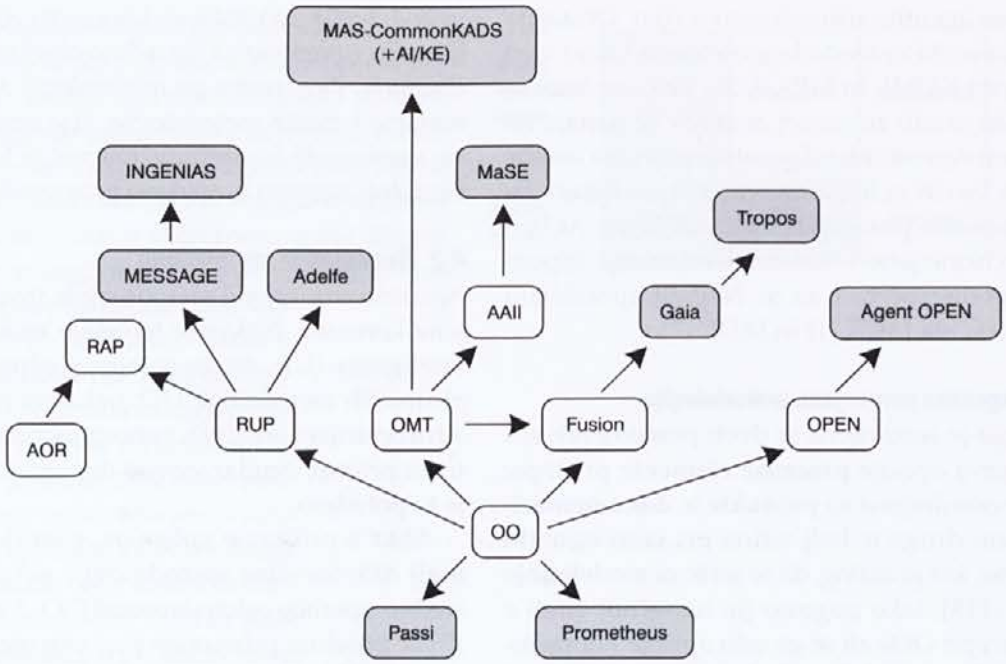
4.2 Genealogija metodologij

Agentno usmerjene metodologije imajo zelo razvejene korenine. Nekatere temeljijo na idejah umetne inteligence (UI), druge so neposredne izpeljanke iz obstoječih metodologij OO, nekatere pa poskušajo z združevanjem teh dveh konceptov predstaviti inovativen pristop, vendar vseeno dopuščajo ideje OO, ko je to potrebno.

Slika 6 prikazuje vplive oz. genealogijo metodologij AO. Številne metodologije priznavajo neposredno uporabo celotnih metod OO. Avtorji **MaSE** [9, 27] še posebno priznavajo prej omenjene vplive [12] in AAII [13], katero je prav tako močno zaznamovala metodologija OO OMT [22]. Prav tako so se pri načrtovanju metodologije **Gaia** [28] avtorji zgledovali po metodologiji OO Fusion [7]. Kot osnova pri izgradnji metodologij AO sta bila uporabljena tudi druga dva pristopa OO. RUP [14] je nastopil kot osnova za **ADELFE** [3] in **MESSAGE** [6], pri katerih je z vplivi drugega nastal tudi **INGENIAS** [20]. Pri novejši metodologiji RAP [24] se je kot vhodni element pri izgradnji poleg RUP-a pojavil tudi AOR [26]. Zelo zanimiv je pristop OPEN, ki je v veliki meri razširil pristop OO razvoja s podporo agentom ter dobil ime **Agent OPEN** [8]. Na koncu pa moramo omeniti še dve metodologiji, ki sta nastali pod neposrednim vplivom metodološkega pristopa OO. **Prometheus** [19] sicer ni čisto neposredni naslednik OO, vendar predlaga uporabo diagramov in konceptov OO takrat, ko ti obstajajo in so združljivi z agentno usmerjeno paradigmo. Podobno je z metodologijo **PASSI**, ki združuje ideje OO in MAS in uporablja notacijo UML. Od drugih se nekako razlikuje metodologija **MAS-CommonKADS** [11], saj je edina, ki temelji pretežno na konceptih AI, čeprav avtorji trdijo, da so na razvoj močno vplivale tudi metodologije OO.

Poleg omenjenih metodologij so tudi metodologije, ki ne priznavajo neposrednih povezav s pristopi, kot sta OO ali AO. Ena od teh metodologij je **Tropos** [5], ki poudarja predvsem zgodnje modeliranje

[1] http://www.agentlink.org



Slika 6: **Neposredni in posredni vplivi objektno usmerjenenih metodologij na agentno usmerjene metodologije**

zahtev. Osredinjena je na opisovanje ciljev, ki poleg obravnave vprašanj tipa »kaj« in »kako«, odgovarjajo tudi na vprašanja tipa »zakaj«. Tropos med drugim uporablja za modeliranje jezik i* (še posebno v fazi analize in načrtovanja), kar ga razlikuje od drugih, ki za notacijo večinoma uporabljajo Agent UML oz. AUML. To tudi pomeni, da razmišljanje, drugačno od OO, omogoča uporabnikom Troposa v metodološkem smislu inovativen pristop k modeliranju agentov.

5 Mnenja analitikov

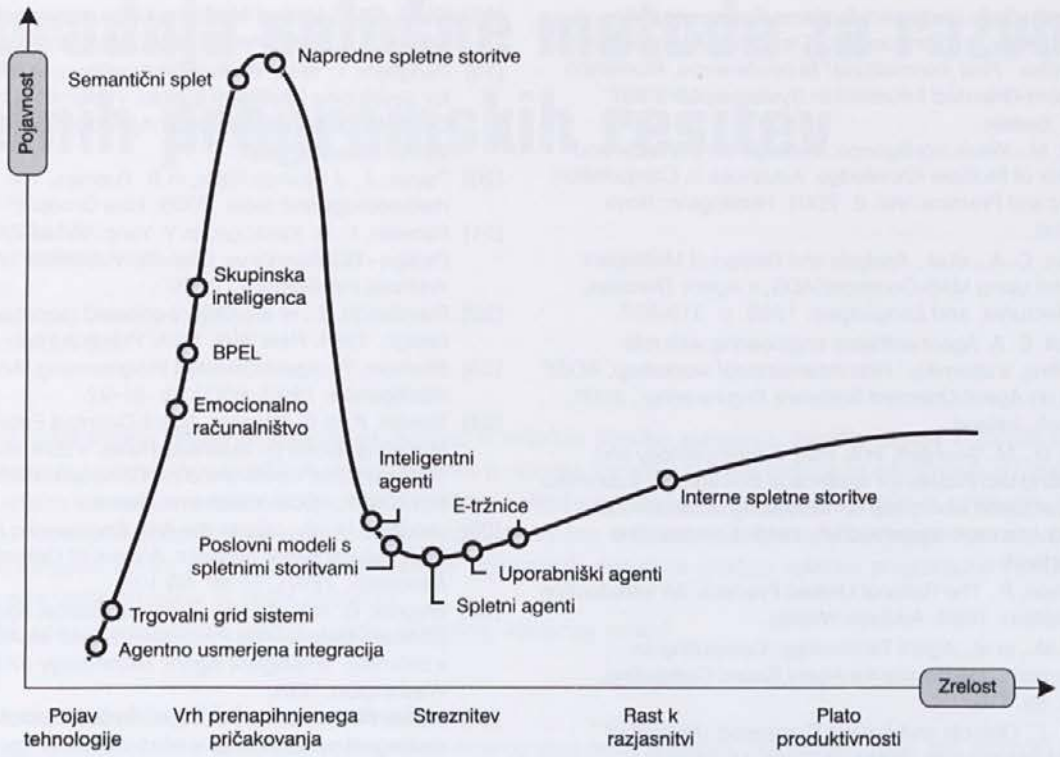
Slika 7 prikazuje Gartnerjevo analizo različnih tehnologij in aplikacij s področja agentov. Na področju infrastrukture se pojavlja jezik za izvajanje poslovnih proces, BPEL, s približno 1- do 5-odstotno prodornostjo trga. Spletne storitve v osnovni obliki za opredelitev storitev in integracijo med aplikacijami s pomočjo SOAP in WSDL rastejo k fazi razjasnitve, saj jih podpirajo že vsi večji ponudniki ter dosegajo 20 do 50 odstotkov prodornosti. Napredne spletne storitve za doseganje večje kakovosti, ki naj bi omogočale napredne kritične poslovne funkcije s pomočjo standardov SOAP, WSDL, UDDI, WS-Security in WS-R, so odvisne predvsem od standardov in še niso tako podprte s strani večjih ponudnikov informacijske tehnologije.

Semantični splet se nahaja prav v vrhu pričakovanj, predvsem s transformacijsko vlogo, vendar je njegova prodornost na trgu zgolj 1 odstotek. Podobno vlogo imajo tudi trgovalni grid sistemi za podporo virtualnim organizacijam in se tudi nahajajo na začetku cikla.

Inteligentni agenti so bili v preteklosti precenjeni kot rezultat prevelikih pričakovanj in premalo aplikacij, vendar vedno bolj postajajo zanimivi spletni in uporabniški agenti, ki dosegajo tja do 5 odstotkov tržnega deleža. V vseh primerih agentov je vedno govor o t. i. »lahkih« agentih, pri čemer se še vedno čaka na glavnino uporabe agentno usmerjenih tehnologij. Primer je npr. agentno usmerjena integracija, ki se ukvarja s porazdeljenimi aplikacijami, ki zahtevajo določeno mero avtonomnosti in prilagodljivosti. V tem primeru je komercialna tehnologija relativno nova, na tem področju pa so številni manjši projekti in manjše število sodelujočih. Gartner ocenjuje prodornost na trgu na 1 odstotek načrtovane. Glede na položaj semantičnega spleta to niti ni presenetljivo.

6 SKLEP

Agenti so avtonomne entitete, ki lahko vzajemno delujejo z okoljem, v katerem se nahajajo. Ali so agent-



Slika 7: **Gartnerjev zrelostni model za agentne tehnologije**

ne tehnologije kljub temu zgolj objektne tehnologije z dodatnimi lastnostmi ali predstavljajo popolnoma drugačen pristop? Na idejni ravni so razlike med objektnimi in agentnimi tehnologijami zelo signifikantne, v praksi nekaj manjše, a še vedno pomembne.

Agentne tehnologije bodo prišle v širšo uporabo šele takrat, ko bodo na voljo strategije za migracijo iz drugih obstoječih okolij. Razvijalci programske opreme pri razvoju agentov svetujejo, da pri implementaciji izhajamo in temeljimo na sistemih OO. Številne strukture in dele agentov lahko dokaj preprosto izrazimo z objekti, npr. imena agentov, komponente jezika za medsebojno komunikacijo med agenti (kodiranje, ontologije, elementi podatkovnega slovarja itd.), vzorce medsebojnega delovanja idr.

Pri načrtovanju in razvoju informacijskih rešitev lahko izbiramo med številnimi možnostmi obeh pristopov. Lotimo se lahko celo implementacije sistemov, pri kateri so agenti sestavljeni iz objektov in agentov ter obratno.

7 Viri in literatura

[1] AOS. JACK Intelligent Agents User Guide. dostopno na: http://www.jackagents.com/docs/jack/html/index.html.

[2] Bellifemine, F., et al. JADE Administrator's guide. Posodobljeno 10. 11. 2006; dostopno na: http://jade.tilab.com/doc/administratorsguide.pdf.

[3] Bernon, C., et al. The ADELFE Methodology for an Intranet system design. v zborniku *'Agent-Oriented Information Systems (AOIS 2002)'*. 2002. Bologna, Italy.

[4] Bežek, A., Avtomatsko modeliranje večagentnih sistemov, v *Faculty of Computer and Information Systems*. 2006, University of Ljubljana: Ljubljana. Str. 144.

[5] Bresciani, P., et al., Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 2004. 8(3): str. 203-236.

[6] Caire, G., et al., Agent Oriented Analysis using Message/ UML. *Lecture Notes in Computer Science*, 2001. 2222: str. 119-135.

[7] Coleman, D., et al., Object-oriented development: The Fusion method. 1994, New York, USA: Prentice Hall.

[8] Debenham, J. K. in B. Henderson-Sellers, Designing agent-based process systems - Extending the OPEN Process Framework. *Intelligent Agent Software Engineering*, 2003: str. 160-190.

[9] DeLoach, S. A. Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems. v zborniku *'First International Bi-conference, Workshop on Agent-Oriented Information Systems (AOIS'99)'*. 1999. Seattle.

[10] Gams, M., Weak Intelligence: Through the Principle and Paradox of Multiple Knowledge. Advances in Computation: Theory and Practice. Vol. 6. 2001, Huntington: Nova Science.

[11] Iglesias, C. A., et al., Analysis and Design of Multiagent Systems using MAS-CommonKADS, v *Agent Theories, Architectures, and Languages*. 1998. p. 313-327.

[12] Kendall, E. A. Agent software engineering with role modelling. v zborniku *'First international workshop, AOSE 2000 on Agent-Oriented Software Engineering'*. 2001. Limerick, Ireland.

[13] Kinny, D., M. Georgeff, in A. Rao. A methodology and modelling techniques for systems of BDI agents. v zborniku *'7th European workshop on Modelling autonomous agents in a multi-agent world'*. 1996. Einhove, The Netherlands.

[14] Krutchen, P., The Rational Unified Process: An Introduction - 2nd edition. 1994: Addison Wesley.

[15] Luck, M., et al., Agent Technology: Computing as Interaction - A Roadmap for Agent Based Computing. 2005. str. 110.

[16] Odell, J., Objects and Agents Compared. *Journal of Object Technology*, 2002. 1(1): str. 41-53.

[17] Odell, J., H. V. Parunak, in B. Bauer. Extending UML for Agents. v zborniku *'Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence'*. 2000. Austin, USA.

[18] OMG. OMG Unified Modeling Language specification, version 1.4. dostopno na: http://www.omg.org.

[19] Padgham, L. in M. Winikoff. Prometheus: A Methodology for developing intelligent agents. v zborniku *'Internation Conference on Autonomous Agents (AAMAS'2002)'*. 2002. Bologna, Italy.

[20] Pavon, J., J. Gomez-Sanz, in R. Fuentes, The INGENIAS methodology and tools. 2005: Idea Group.

[21] Rahwan, I., R. Kowlczyk, in Y. Yang. Virtual Enterprise Design - BDI Agents vs. Objects. v zborniku *'Advances in Artificial Intelligence'*. 2000.

[22] Rumbaugh, J., et al., Object-oriented modeling and design. 1991, New York, USA: Prentice Hall.

[23] Shoham, Y., Agent-Oriented Programming. *Artificial Intelligence*, 1993. 60(1): str. 51-92.

[24] Taveter, K. in G. Wagner. Agent-Oriented Enterprise Modeling Based on Business Rules. v zborniku *'International conference on Conceptual Modeling (ER2001)'*. 2001. Yokohama, Japan.

[25] van Dyke H., P., 'Go to the Ant': Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 1997(75): str. 69-101.

[26] Wagner, G. in K. Taveter. Towards Radical Agent-Oriented Software Engineering Processes Based on AOR Modeling. v zborniku *'Intelligent Agent Technology (IAT.04)'*. 2004. Washington, USA.

[27] Wood, M. F. in S. A. DeLoach. An overview of the multiagent systems engineering methodology. v zborniku *'First International workshop, AOSE 2000 on Agent-Oriented Software Engineering'*. 2001. Limerik, Ireland.

[28] Wooldridge, M., An Introduction to MultiAgent Systems. 2002, Chichester, England: John Wiley & Sons.

▪

Dejan Lavbič je leta 2004 diplomiral na področju računalništva in informatike na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer je trenutno doktorski kandidat in zaposlen kot mladi raziskovalec. Na raziskovalnem področju se ukvarja z inteligentnimi agenti, večagentnimi sistemi, ontologijami, poslovnimi pravili in semantičnim spletom. Predvsem ga zanima uporaba tehnologij semantičnega spleta v poslovnih aplikacijah. Sodeloval je pri številnih gospodarskih in raziskovalnih projektih s področja strateškega planiranja, metodologij razvoja informacijskih sistemov, uporabe inteligentnih agentov in prenove ter avtomatizacije poslovnih procesov.

▪

Matjaž Gams je raziskovalni svetnik in vodja odseka za inteligentne sisteme na Inštitutu Jožef Stefan. Predava inteligentne sisteme na nekaj fakultetah, med drugim na Ekonomski fakulteti in Fakulteti za računalništvo in informatiko v Ljubljani. Učil je tudi na PEF, FDV, FM, FPP, VŠUP, en semester leta 2002 tudi na University of Applied Sciences v Nemčiji. Od leta 1990 je urednik revije Informatica, član uredniških odborov več mednarodnih revij (AICOM, IDA, KAIS idr.), član programskih odborov številnih priznanih mednarodnih konferenc. Je ustanovitelj in dva mandata predsednik Društva za kognitivne znanosti, večletni predsednik Društva za umetno inteligenco, soustanovitelj Inženirske akademije Slovenije in njen prvi tajnik, sedaj tajnik in predstavnik slovenske AI v IFIP, predsednik ACM Slovenija, član več nacionalnih svetov (za informatizacijo šolstva, za informacijsko družbo, sedaj za znanost in tehnologijo). Bil je podpredsednik sindikata SVIZ in predsednik oz. podpredsednik konference visokega šolstva in znanosti pri SVIZ. Ukvarja se z inteligentnimi sistemi, informacijsko družbo, umetno inteligenco, inteligentnimi agenti, kognitivnimi znanostmi. Objavil je več kot 350 del, pretežno znanstvenih člankov in referatov. Sodeloval je pri nekaj najodmevnejših aplikacijah inteligentnih sistemov v Sloveniji.