SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Andrea Kljaić

JEZICI ZA MODELIRANJE VIŠEAGENTNIH SUSTAVA

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Andrea Kljaić

Matični broj: 45334/16-R

Studij: Baze podataka i baze znanja

JEZICI ZA MODELIRANJE VIŠEAGENTNIH SUSTAVA

DIPLOMSKI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Markus Schatten

Varaždin, lipanj 2021.

Andrea Kljaić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

---

Sažetak

U radu se opisuju postojeći jezici za modeliranje višeagentnih sustava, uključujući grafičke jezike. Jezici za modeliranje višeagentnih sustava su temelji za razvoj višeagentnih aplikacija i provođenje simulacija. Uz opis svakog jezika za modeliranje, opisat će se načini modeliranja višeagentnih sustava uključujući prikaz pojedinih jezika na jednostavnim primjerima, kritički osvrt na njih te zaključak i literatura.

Ključne riječi: višeagentni sustav; modeliranje; grafički jezici; zahtjevi modeliranja; Agent UML; PLKTF; GPLKTF

Sadržaj

1. Uvod

Tehnologije koje se temelje na agentima se sada primjenjuju u razvoju velikih komercijalnih i industrijskih sustava. Takva specifična vrsta sustava koja se sastoji od više inteligentnih agenata koji međusobno djeluju radi postizanja određenih ciljeva nazivamo višeagentnim sustavima. Ti sustavi su složeni, uključuju stotine, možda tisuće agenata čime postoji potreba za tehnikama modeliranja sustava koje omogućuju učinkovito upravljanje njihovom složenošću i metodologijama. Bez odgovarajućih tehnika koje podržavaju proces projektiranja, takvi sustavi neće biti dovoljno pouzdani, održivi ili proširivi, teško su razumljivi i njihovi elementi neće biti moguće ponovno koristiti.

Metodologije za razvoj višeagentnih sustava su aktivno područje istraživanja. Pokušava se odgovoriti na pitanje kako oblikovati, analizirati i optimizirati zajednicu agenata, za razliku od implementacije pojedinih agenata. Postoje mnogi pristupi, ali za sada ne postoji univerzalni i općeprihvaćeni pristup za razvoj i modeliranje višeagentnih sustava. [1]

Postoji nekoliko jezika za modeliranje (MA-UML, KAOS, GBRAM, Tropos,...), gdje svaki jezik pokriva pojedinu karakteristiku VAS-a koje inače nisu uobičajene u konvencionalnim softverskim sustavima.

Na početku ćemo definirati agenta i njegovo značenje u višeagentnim sustavima i opisati proces modeliranja višeagentnih sustava kroz metodu zahtjeva. Pojasnit ćemo kako modelirati u Agent UML-u. Agent UML koristimo kako bismo povezali modeliranje VAS-a s tehnologijom koja je najbliža tome - objektno orijentirani razvoj softvera i uveli odgovarajući artefakte koji podržavaju razvojno okruženje tijekom cijelog životnog ciklusa sustava. Proći ćemo kroz UML prikaz agenta o unutarnjem ponašanju agenta, povezivanje unutarnjeg opisa s vanjskim ponašanjem korištenjem i proširenjem dijagrama klasa te opisivanjem protokola interakcije agenta na nov način.

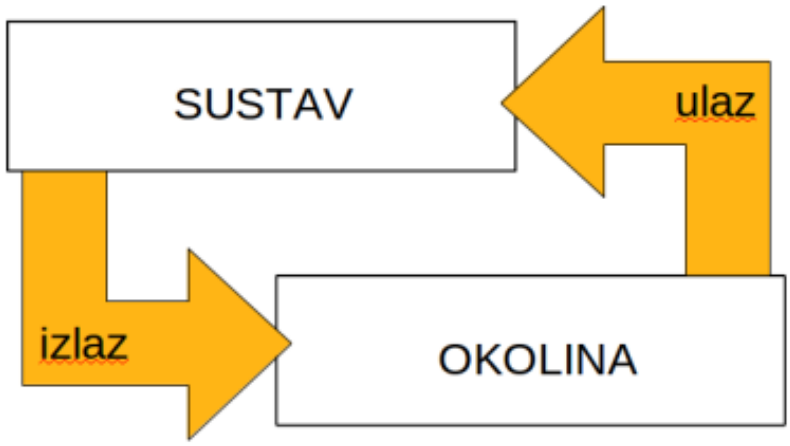
Proći ćemo i kroz grafički jezik GPLKTF koji se temelji na jeziku PLKTF, jeziku propozicijske logike proširen znanjem, temporalnim operatorom i operatorom zaboravljanja. Koristimo ga za modeliranje složenih VAS i njihovih odnosa u kojemu agenti stječu znanje od agenata s kojima surađuju u grupi kroz interakciju s njima. Primjenom znanja, grupnog znanja, distribuiranog znanja i operatora zajedničkog znanja, zajedno s uobičajenim vremenskim operatorima i operatorom zaboravljanja, možemo formalizirati učenje i složene procese koji se odvijaju u njima.

Na kraju ćemo se osvrnuti na ranije spomenute jezike i pokušati objediniti njihove prednosti i nedostatke u kritičkom osvrtu.

2. Što je agent?

Računala nisu veoma dobra u određivanju što im je činiti: programer mora svaku radnju koju računalo izvrši izričito predvidjeti, planirati i kodirati. Ali u slučaju da računalo naiđe na situaciju koju njegov dizajner nije predvidio, rezultat može biti veoma ružan - pad sustava u najboljem, gubitak života u najgorem slučaju. Ta činjenica je srž našeg odnosa s računalima. Za osobu koja je računalno pismena takve se stvari podrazumijevaju, dok za one koji se prvi put susreću s time je iznenađujuće. Većim dijelom rado prihvaćamo računala kao poslušne i nemaštovite sluge za aplikacije gdje je takva primjena sasvim prihvatljiva. Međutim, za sve veći broj aplikacija potrebni su nam sustavi koji mogu sami odlučiti što trebaju učiniti kako bismo postigli ciljeve koje im delegiramo. [2] Računalni sustavi danas nisu više samostojeće jedinke već su umreženi u velike distribuirane sustave (npr. internet). Takvu kompleksnost zadataka potrebno je automatizirati i delegirati računalima te time predajemo upravljanje računalima, čak i u situacijama kritičnim za sigurnost (npr. autopilot u zrakoplovu). Delegiranje i inteligencija impliciraju potrebu gradnje računalnih sustava koji mogu efektivno djelovati umjesto nas što znači da računalni sustavi moraju djelovati nezavisno i zastupajući naše interese prilikom interakcije s drugim ljudima ili sustavima, implicirajući da sustavi moraju biti u stanju surađivati i ostvarivati dogovore s drugima koji mogu zastupati sasvim drukčije interese. [3] Takvi su računalni sustavi poznati kao agenti.

Agenti koji moraju brzo djelovati u mijenjajućim, nepredvidivim ili otvorenim okruženjima, gdje postoji značajna mogućnost da radnje ne uspiju nazivamo inteligentnim, ponekad autonomnim, agentima. Njihova najvažnija karakteristika je autonomnost, da su u stanju samostalno djelovati i upravljati svojim unutrašnjim stanjem u nekom okruženju kako bi postigao ciljeve svog oblikovanja (Slika 1).



Slika 1: Autonomni agent (Izvor: Schatten, 2018)

Fleksibilno ponašanje u nekoj okolini, što uključuje reaktivnost, proaktivnost i društvenost, je karakteristično ponašanje za inteligentnog agenta. Pomoću reaktivnošću mora biti u stanju postaviti si pitanje isplati li mu se pokrenuti te ostvariti kontinuiranu interakciju sa svojom okolinom i pravovremeno reagirati na promjene koje se u njoj događaju. Međutim, kako bi agent pravovremeno reagirao, proaktivnošću agent generira i ostvaruje ciljeve, prepoznaje

prilike i preuzima inicijativu. Zbog toga je potrebno uskladiti njegovu reaktivnost i proaktivnost kako bi pravovremeno reagirali na promjenu situacije na odgovarajući način i rade sistematično na ostvarivanju dugoročnih ciljeva te time dolazimo do treće karakteristike fleksibilnosti - društvenost. Ono je agentova sposobnost interakcije s drugim agentima (i/ili ljudima) putem nekog jezika za međuagentnu komunikaciju kao i suradnja s drugim agentima. [4]

U kontekstu agenata se često spominju i sljedeće osobine [4]:

- mobilnost - sposobnost agenta da se kreće unutar mreže
- iskrenost - agent neće (namjerno) komunicirati lažne informacije
- benevolentnost - agent neće imati međusobno konfliktne ciljeve, te će stoga svaki agent uvijek pokušati ostvariti ono što se od njega traži
- racionalnost - agent će uvijek djelovati na način da postigne svoje ciljeve, tj. neće vlastitim djelovanjem onemogućiti njihovo postizanje (ukoliko mu to dopuste njegova uvjerenja)
- učenje/adaptibilnost - agenti poboljšavaju svoje performanse tijekom vremena

2.1. Formalizacija agenata

Pretpostavimo da je okružje konačan skupa O diskretnih, instantnih stanja:

$$O = \{o_1, o_2, ...\}$$

Pretpostavimo nadalje da svaki agent ima konačan repertoar akcija koje transformiraju stanje okružja:

$$A_k = \{a_1, a_2, ...\}$$

Prolaz p agenta kroz okružje je naizmjenični niz stanja okružja i akcija agenta:

$$p : o_0 \xrightarrow{a_0} o_1 \xrightarrow{a_1} o_2 \xrightarrow{a_2} ... \xrightarrow{a_{n-1}} o_n$$

Neka je P skup svih konačnih prolaza (nad O i A_k).
Neka je P^{A_k} podskup svih onih prolaza koji završavaju akcijom.
Neka je P^O podskup svih onih prolaza koji završavaju stanjem okružja.

Funkcija promjene stanja je funkcija koja će za zadano stanje okružja i zadanu akciju odrediti buduće stanje okružja, tj.

$$\tau : P^{A_k} \to \varsigma(E)$$

Formalno kažemo da je okružje Okr uređena trojka $Okr = (O, o_0, \tau)$ u kojoj je O skup mogućih stanja okružja, o_0 početno stanje i τ funkcija transformacije stanja.

Prema tome dolazimo do definicije agenta kao funkcije koja prolazima pridružuje akcije:

$$Ag : P^O \rightarrow A_k$$

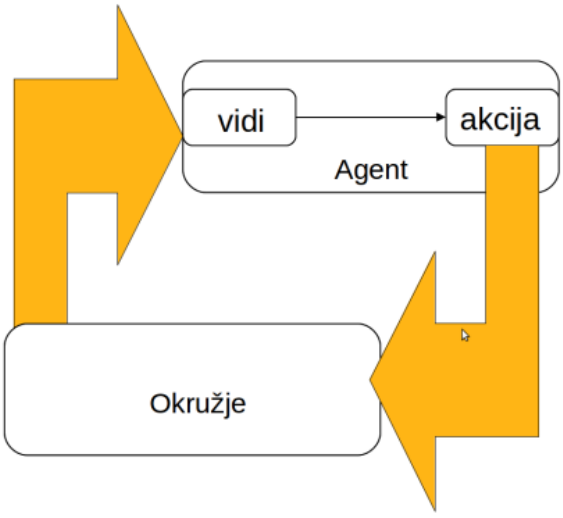
Sustav je par koji se sastoji od jednog agenta i jednog okružja. Svaki sustav imat će definirani niz prolaza. Skup prolaza agenta Ag u okružju Okr označavamo s $P(Ag, Okr)$. Pretpostavljamo da $P(Ag, Okr)$ sadrži samo završene prolaze.

Formalno gledano, niz $(o_0, a_0, o_1, a_1, ...)$ bit će prolaz agenta Ag kroz okružje $Okr = (O, o_0, \tau)$ ako vrijedi:

- o_0 je početno stanje Okr
- $a_0 = Ag(o_0)$
- za $n > 0$ vrijedi:

$$o_n \in \tau((o_0, a_0, ...a_{n-1})) \text{ gdje je } a_n = Ag((o_0, a_0, ...o_n))$$

Neki agenti odlučuju o svojim akcijama bez razmatranja prošlosti te donose odluke isključivo na temelju trenutnog stanja okružja (npr. termostat). Takve agente nazivamo čisto reaktivnim agentima te zbog njih uvodimo percepcijski sustav (Slika 2).



Slika 2: Agent i percepcija (Izvor: Schatten, 2018)

Funkcija $vidi$ je agentova sposobnost percipiranja okoline, dok funkcija $akcija$ predstavlja agentov proces donošenja odluka. Izlaz iz funkcije $vidi$ je doživljaj:

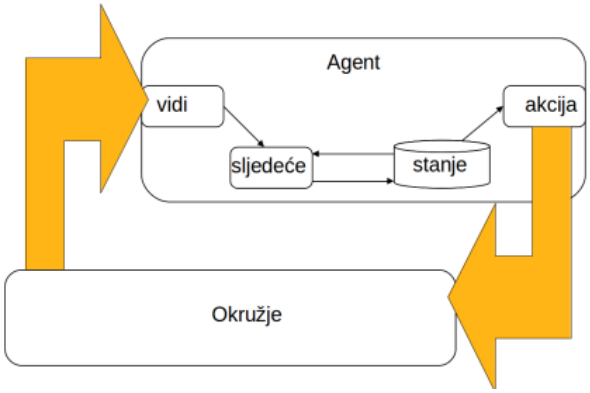
$$vidi : O \rightarrow Do$$

Dakle funkcija stanjima okoline pridružuje doživljaj. Stoga funkcija $akcija$:

$$akcija : Do* \rightarrow A_k$$

doživljajima pridružuje akcije.

Razmotrimo sad agente koji održavaju unutarnje stanje (Slika 3):



Slika 3: Agent i unutarnja stanja (Izvor: Schatten, 2018)

Ovakvi agenti sadrže neku vrstu interne podatkovne strukture, koja se tipično koristi za bilježenje informacija o stanju okružja i povijesti.
Neka je U skup svih unutarnjih stanja agenta.
Funkcija percepcije za agenta sa stanjem ostaje ista:

$$vidi : O \rightarrow Do$$

Funkcija odlučivanja o akciji sada se definira kao preslikavanje:

$$akcija : U \rightarrow A_k$$

dakle iz unutarnjih stanja u akcije. Definira se dodatna funkcija $sljedece$ koja parovima unutarnjih stanja i doživljaja pridružuje unutarnja stanja:

$$sljedece : UxDo \rightarrow U$$

Iz toga svega, kontrolna petlja agenta bi izgledala ovako [4]:

1. Agent započinje s nekim početnim stanjem u_0
2. Promatra svoje okružje o te generira doživljaj $vidi(o)$
3. Unutarnje stanje agenta sada se ažurira putem funkcije $sljedece$ i postaje $sljedece(u_0, vidi(0))$
4. Akcija koja se odabire je $akcija(sljedece(u_0, vidi(o)))$
5. Idi na korak 2.

2.2. Višeagentni sustavi

Sustav koji se sastoji od niza agenata koji su u međusobnoj interakciji, ponašaju se u skladu s različitim ciljevima i motivima korisnika te za uspješnu interakciju im je potrebna sposobnost suradnje, koordinacije i pregovaranja, nazivaju se višeagentnim sustavima. [3] Mogu se koristiti za rješavanje problema koje je teško ili nemoguće za riješiti pomoću monolitnih ili pojedinačnih agenata. Posljednjih godina predložene su različite metodologije za razvoj sustava s više agenata, poput Troposa (Giorgini i sur., 2005), Ingenije (Gómez-Sanz & Pavón, 2003.), Gaia (Zambonelli i sur., 2003.) itd. Međutim, unatoč važnosti faze zahtjeva u razvoju softverskih sustava, mnogi od predloženih metodologija za razvoj VAS-a ne pokrivaju na odgovarajući način fazu inženjeringa zahtjeva, usredotočujući se uglavnom na projektiranje i fazu provedbe. Štoviše, nedavna studija o primjeni zahtjeva inženjerske tehnike u razvoju višeagentnih sustava je otkrilo da 79% postojećih metodologija za razvoj VAS-a koristi inženjerske tehnike prilagođene drugim paradigmama (object orientation, knowledge engineering, itd.). Ove tehnike nisu dovoljne za pokrivanje prirode VAS-a, budući da ti sustavi, zajedno sa svojim organizacijskim, strukturnim ili funkcionalnim svojstvima, imaju karakteristike koje nisu uobičajene u konvencionalnim softverskim sustavima kao što su proaktivnost, prilagodljivost, suradnja, istinost ili dispozicija. Stoga postoji potreba za novim metodama i tehnike koje omogućuju odgovarajuće tretiranje VAS-a. U nastavku ćemo proći kroz evoluciju ranijih prijedloga modeliranja bez obzira na korištenu metodologiju analize i projektiranja te pokriti četiri bitne perspektive VAS-a, organizacijsku, strukturnu, funkcionalnu i društvenu. Također je vrijedno spomenuti da tehnologije koje koriste agente korisna u mnogim složenim domenama, od e-commerce, zdravstva, burze, do proizvodnje i igrama. [5] Razvoj igara je unazad nekoliko godina doživio najveći rast na poslovnom tržištu te zbog toga agenti su od sve veće važnosti jer njegove karakteristike poput suradnje, pregovaranja, povjerenja, ugleda i sl. dolaze do izražaja.

3. Metoda zahtjeva modeliranja

Važnost faze zahtjeva u razvoju softvera nadaleko je poznata. Međutim, definiranje i modeliranje zahtjeva sustava nije beznačajan zadatak, posebice višeagentni sustavi koji zahtjevaju apstrakcije, tehnike i oznake koji su specifično određeni i prilagođeni za njih. Pozabavit ćemo se s četiri osnovne perspektive za modeliranje zahtjeva VAS-a: organizacijsko, strukturno, funkcionalno i društveno ponašanje. Organizacijska perspektiva podržana je prijedlozima poput GBRAM (Anton, 2006).

GBRAM je relevantan tradicionalni inženjerski prijedlog usmjeren na ciljeve i daje proceduralni vodič za identifikaciju i razvoj ciljeva poznavajući tehnike koje pomažu u njihovoj metodičkoj i sustavnoj analizi. GBRAM ima veliki nedostatak u smislu formalnosti, što uključuje nedostatak modela, formalnih oznaka i alata koji podržavaju modeliranje za ovu metodu. Ipak, smjernice i razina jasnoće koje nudi vrlo su dobre. Štoviše, GBRAM naglašava provjeru zahtjeva kroz svoju fazu usavršavanja koja specificira određene smjernice kojih se treba pridržavati čineći ovaj proces pouzdanijim te je moguće pratiti zahtjeve odražavajući sljedivost koju nudi metoda.

Drugi prijedlog za modeliranje zahtjeva koji podržava organizacijsku perspektivu je i* framework (Yu, 1997), uspostavljen kao osnova za Tropos metodologiju. Tropos je na odgovarajući način prilagođen za stjecanje i modeliranje aktera u sustavu i njegovom okruženju, npr. aktera, ciljeva, zadataka, interakcija, ovisnosti, potrebnih resursa, itd. Međutim, ne dopušta potpuni prikaz ograničenja niti predlaže okruženje za modeliranje. Budući da nam je orijentacija na cilj od posebnog interesa za zahtjeve VAS-a, smatramo da je potrebno analizirati druge metode koje su komplementarne ovom pristupu.

Strukturna perspektiva podržana je prijedlozima poput AUML-a (Agent UML) (Odell et al., 2000). AUML se nastoji afirmirati kao notacijski standard u različitim metodologijama te je jedan od najčešćih prijedloga za fazu zahtjeva upravo use case dijagram. Ovakav tip formalizma je pokazao dobre rezultate u predstavljanju funkcionalnih zahtjeva, a također je i dobar alat za komunikaciju sa sudionicima. Ipak, use case dijagram ima ograničenja u bilježenju kvalitativnih aspekata okoliša i interakcija s njim. Osim toga, zanimljiv doprinos AUML-a je protokol za interakciju agenata (AIP) koji predstavlja središnji aspekt za VAS, specificiran pomoću dijagrama protokola.

Drugi prijedlog koji pokriva strukturnu perspektivu je KAOS (Van Lamsweerde i sur., 1998), prijedlog modeliranja zahtjeva kroz ciljeve. KAOS se sastoji od specifikacije jezika, metoda razrade i znanje na meta razini koje se koristi kao vodič. KAOS model sadrži ciljeve, zahtjeve informacijskog sustava, očekivanja okruženja sustava, sukobe između ciljeva, prepreka, entiteta, agenata, itd. Jedna od prednosti prijedloga je njegova upotreba formalnosti za postizanje ispravka. Štoviše, ideja ograničenja korisna je u identificiranju nekih vanjskih problema integriteta, a to pridonosi robusnosti sustava. Međutim, uspješna implementacija metode uvelike ovisi o iskustvu programera u domeni i koliko je dobro definiran problem koji treba riješiti.

Ostali prijedlozi ne podržavaju organizacijsku i strukturnu perspektivu. To vidimo u slučaju CREWS (Maiden, 1998) koji se usredotočuje na perspektive funkcionalnog i društvenog ponašanja. CREWS se temelji na objektnim modelima sustava koji su apstrakcije ključnih značajki različitih kvaliteta domene problema. CREWS koristi ove modele za generiranje normalnih sce-

narija, a zatim koristi teorijska i empirijska istraživanja kognitivne znanosti, interakcije čovjeka i računala, sustava suradnje i softverskog inženjeringa kao osnovu za generiranje alternativnih tečajeva za te scenarije. Potencijalna slabost pristupa CREWS-a je u tome što je generiranje scenarija usmjereno na domenu, za razliku od analize scenarija usmjerene na ciljeve i modeliranje use case orijentiranog na zadatke. Ako se namjerava scenarijima potvrditi zahtjeve, ti zahtjevi trebaju biti usmjereni na stvaranje scenarija.
Ukratko, organizacijska perspektiva obuhvaćena je prijedlozima kao što su GBRAM i i*, a strukturna perspektiva obuhvaćena je prijedlozima kao što su KAOS i AUML. Većina prijedloga na neki način pokriva, potpuno ili djelomično, funkcionalnu i društvenu perspektivu ponašanja, kao u slučaju CREWS. Međutim, nema metode koja u potpunosti pokriva sve četiri perspektive potrebne za razvoj VAS-a. [5]

3.1. Različite perspektive za višeagentne sustave

Nadalje ćemo proći kroz prethodno navedene četiri perspektive VAS-a: organizacijska, strukturna, funkcionalna i društveno ponašanje. Kako bi se te perspektive kontekstualizirale, predstavljen je njihov pregled koji naglašava i društveno ponašanje i organizacijske aspekte, budući da su to ključni aspekti za razvoj VAS-a.

3.1.1. Organizacijska perspektiva

U organizacijskoj perspektivi, organizacija je predstavljena kao sustav koji ima određene ciljeve i postiže ih dosljednim djelovanjem, koje koristi resurse sustava i mijenja željeno stanje sustava. Neki autori poput Zambonelli i sur., 2003. smatraju da je ljudska organizacijska metafora veoma primjerena za sustave koji se nalaze u otvorenim i promjenjivim okruženjima. Oni definiraju VAS kao softverski sustav koji je zamišljen kao računska instanca grupe interaktivnih i autonomnih pojedinaca (agenata). Na svakog se agenta može gledati da ima jednu ili više određenih uloga, ima niz odgovornosti ili ciljeva u kontekstu cjelokupnog sustava i odgovoran je za njihovo samostalno provođenje.
Interakcije su jasno identificirane i lokalizirane u definiciji same uloge te pomažu u karakterizaciji cjelokupne strukture organizacije i položaja agenta u njoj. Evolucija aktivnosti u organizaciji, koje proizlaze iz autonomnog izvršenja agenata i iz njihovih interakcija, određuju postizanje cilja aplikacije.

3.1.2. Strukturna perspektiva

Strukturna perspektiva prikazuje arhitekturu sustava u smislu entiteta i statičkog odnosa među njima. Modeliranje ovih entiteta i odnosa pruža apstraktnu strukturnu perspektivu sustava. Vjerujemo da je ova perspektiva neophodna za identifikaciju entiteta koji će biti potrebni za izgradnju budućeg VAS-a. Da bi se statički i strukturni odnosi točno zabilježili, razvojna metoda mora uključivati formalizme i tehnike za specificiranje odnosa hijerarhije (nasljeđivanje), semantičke ovisnosti (asocijacija) i dijela odnosa (agregacija).

3.1.3. Funkcionalna perspektiva

Funkcionalna perspektiva prikazuje semantiku povezanu s uslugama organizacijskih uloga koje su motivirane pojavom događaja. U tom kontekstu, organizacijsku ulogu shvaćamo kao predstavljanje apstraktnog entiteta koji pruža (više) metoda ili usluga sustava. Događaj je nešto što se događa u okruženju i na to što organizacijska uloga reagira pokretanjem metode ili usluge. Ova se perspektiva fokusira na modeliranje funkcionalnih zahtjeva koje moraju ispunjavati uloge u budućem VAS-u.

3.1.4. Perspektiva društvenog ponašanja

Perspektiva društvenog ponašanja pokazuje moguće sekvence događaja ili usluga na koje agent može odgovoriti ili koje se dogode tijekom njegova života, zajedno sa aspektima interakcije, poput komunikacije između agenata, a to se često predstavlja kao dijagram stanja ili aktivnosti. Kao što je gore objašnjeno, osim organizacijskih, strukturnih i funkcionalnih svojstava, VAS također zahtjeva karakteristike koje se obično ne zahtijevaju u konvencionalnim softverskim sustavima, kao što su proaktivnost, prilagodljivost, suradnja, istinitost ili dispozicija. Ove karakteristike su označene kao društveno ponašanje. Stoga vjerujemo da je pokrivanje ove perspektive u prijedlogu modeliranja zahtjeva za VAS važan doprinos razvoju takvih sustava, budući da je bit tih sustava izvođenje složenih zadataka koje druge vrste sustava nisu sposobne riješiti.

3.2. Zahtjevi modeliranja višeagentnih sustava

Kako bismo podržali organizacijske, strukturne, funkcionalne i društvene perspektive, predlažemo proces modeliranja zahtjeva koji je raščlanjen na dvije glavne aktivnosti, definicija zahtjeva i specifikacija zahtjeva. Posebne potrebe korisnika identificirane su u aktivnosti Definicija zahtjeva.

To podrazumijeva:

- organizacijska struktura sustava
- uloge koje su potrebne u svako podorganizaciji
- ciljevi uloga
- društveno ponašanje potrebno da uloge ostvare svoje ciljeve
- relevantni subjekti okruženja.

Detaljni zahtjevi za programere navedeni su u aktivnosti Specifikacija zahtjeva. Specifikacije izvučene iz aktivnosti Definicije zahtjeva poboljšavaju se i povećava se razina detalja kako bi se identificirali artefakti koji su bliži analizi i razvoju sustava: aktivnosti i interakcije, resursi sustava, dopuštenja koja uloge imaju u tim resursima i organizacijskim pravilima.

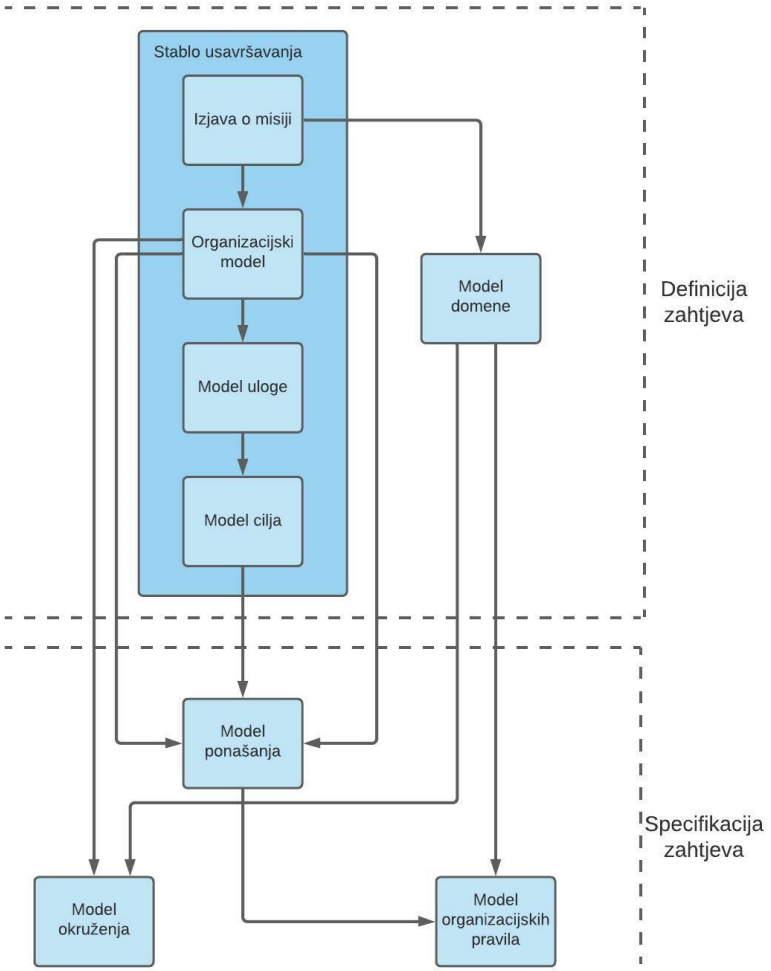
Štoviše, proces se temelji na definiciji modela potrebnih za opisivanje problema u konkretnijim aspektima koji tvore različite perspektive sustava.

Konkretno, u aktivnosti Definicije zahtjeva (eng. Requirements Definitions) moguće je identificirati (Slika 4):

1. stablo usavršavanja (eng. Refinement Tree) - identificira i predstavlja ciljeve sustava i njihovu hijerarhiju, uloge koje će te ciljeve provoditi u organizacijskom kontekstu i organizacijska struktura sustava
   (a) izjava o misiji (eng. Mission Statement) - glavni cilj koji sustav u razvoju pruža okruženju kako bi se identificirao opći cilj unutar organizacije u cjelini
   (b) organizacijski model (eng. Organizational Model) - predstavlja podorganizacije od koji se sastoji globalna organizacija
   (c) model uloge (eng. Role Model) - predstavlja uloge uključene u svaku podorganizaciju, nasljedne odnose među njima i društveno ponašanje potrebno među ulogama za postizanje njihovih ciljeva
   (d) model cilja (eng. Goal Model) - predstavlja ciljeve povezane sa svakom uloge
2. model domene (eng. Domain Model) - predstavlja entitete koji se mogu koristiti kao resursi organizacije

U aktivnosti Specifikacije zahtjeva (eng. Requirements Specification) moguće je identificirati:

1. model ponašanja (eng. Behavior Model) - predstavlja raščlanjivanje ciljeva na zadatke i protokole kako bi se razumio unutarnji tijek uloge za određivanje njenih odgovornosti
2. model okruženja (eng. Environment Model) - predstavljanje dopuštenja uloga identificiranih u modelu uloga s obzirom na resursa modela domene
3. model organizacijskih pravila (eng. Organizational Rules Model) - predstavlja ograničenja u ponašanju organizacije



Slika 4: Definicija i specifikacija zahtjeva (Izvor: Alkhateeb, Maghayreh i Doush, 2011)

Kako bi se dobio jasan uvid u korištene modele, svaki od njih je predstavljen na sljedeći način.

Izjava o misiji definirana je na prirodnom jeziku, s preporučenim proširenjem od jednog ili dva odlomka. Budući da ta izjava identificira opći cilj unutar organizacije u cjelini, pruža nam informacije o organizacijskim i funkcionalnim perspektivama, predstavljajući korijen stabla usavršavanja koje se uzastopno usavršava dodavajući ciljeve kao čvorovi (listovi) u stablu.

U ovom procesu moguće je razlikovati tri opće razine:

1. definiranje dekompozicije sustava u hijerarhiji podorganizacija, predstavljajući tako organizacijski model (podorganizacija predstavlja dio sustava koji ima za cilj postići cilj u sustavu i slabo komunicira s drugim dijelovima sustava)
2. raščlanjivanje podorganizacije na uloge koje djelomično predstavljaju model uloge (uloga predstavlja apstraktni entitet koji ima jedan ili više ciljeva sustava)
3. identificiranje ciljeva sustava i njihovu hijerarhiju (ciljevi predstavljaju zadatke koje obavlja uloga u podorganizaciji)

Struktura *stabla usavršavanja* omogućuje nam identificiranje elemenata organizacijske perspektive kroz dekompoziciju sustava na podorganizacije, na elemente strukturne perspektive identificiranjem uloga koje čine podorganizacije i na aspekte funkcionalne perspektive identificiranjem ciljeva koje svaka uloga mora ostvariti.

Kao što je već spomenuto, *model uloge* opisuje uloge koje pripadaju podorganizacijama stabla usavršavanja. Svrha ovog modela je predstavljati različite uloge koje se nalaze u svakoj podorganizaciji i uskladiti njihove posebne odnose. Posebni odnosi među njima mogu poslužiti za identifikaciju zajedničkih svojstava između njih kako bi se stvorila hijerarhija uloga pomoću nasljednih odnosa i identifikacija odnosa društvenog ponašanja između uloga u različite podorganizacije. Rezultirajući model sadrži informacije predstavljene u stablu, jedan dijagram za nasljedne odnose i jedan (ili više) dijagrama prema potrebi za svaku podorganizaciju za odnose društvenog ponašanja među ulogama. UML Use Case dijagram koristi se za predstavljanje ovakvih podataka i nadopuniti prikaz uloga u stablu usavršavanja. Uloge su predstavljenje kao akteri koji su označeni s «uloga» (eng. role), nasljeđivanje je označeno odgovarajućim relacijskim dijagramom, a relacije između društvenog ponašanja kao «suradnja» (eng. collaboration), te imamo «narav» (eng. disposition) i «vjerodostojnost» (eng. veracity). Prema tome imenujemo odnose s odgovarajućim svojstvom (npr. za suradnju - "komunikativan", "nekomunikativan", za narav - "dobronamjeran", "vlastiti interes", "zlonamjeran", za vjerodostojnost - "istinit", neistinit", itd.). Odnos društvenog ponašanja između dvije uloga mogao bi biti od jednog ili više svojstava, budući da je odnos dinamički, tj. može se mijenjati ovisno o agentu koji će nakraju odigrati ulogu. Ova informacija nam omogućuje izražavanje elemente strukturnih, organizacijskih i perspektive društvenog ponašanja.

Model domene predstavlja entitete identificirane u domeni problema. Svrha ovoga je identificiranje ključnih pojmova i odnosa koji predstavljaju prvi strukturalni pogled. Ti se entiteti promatraju sa stajališta domene aplikacije, pa se detalji implementacije izbjegavaju na ovoj razini. Asocijacija i nasljeđivanje između entiteta domena također su prikazani. Identifikacija ovih entiteta domene i njihovih odnosa omogućuje nam izdvajanje informacija za stgrukturnu perspektivu i djelomično izdvajanje informacija za organizacijsku perspektivu. UML dijagram klasa će se koristiti za predstavljanje ovih podataka.

Model ponašanja prikazuje slijed koraka koji predstavljaju tijek potrebnih aktivnosti za postizanje ciljeva utvrđenih u sustavu. Prikaz tijeka zadataka mogao bi biti koristan za razumijevanje logičkog tijeka uloge, nadopunjavanje informacija o društvenom ponašanju identificiranih u modelu uloga te pomoći u identificiranju novih informacija kada jedna uloga mora raditi s drugima kako bi se postigao zadatak. Identifikacija tijeka aktivnosti omogućuje nam da izvučemo informacije za funkcionalnu perspektivu. Nadalje, identifikacija interakcija između različitih uloga omogućuje nam identificiranje informacija za perspektivu društvenog ponašanja. Koristi se UML dijagram aktivnosti za predstavljanje ovih podataka.

Model okruženja predstavlja dopuštenja uloga s obzirom na entitete identificirane u modelu domene. Za svaku ulogu identificiranu u modelu uloga uspostavljaju se resursi za one koji im mogu legitimno pristupiti. Identifikacija ovih dopuštenja nudi informacije o strukturnim i funkcionalnim perspektivama sustava. UML Use Case dijagram se koristi za predstavljanje ove informacije, a uloge su predstavljeni kao akteri koji su označeni sa «uloga», izvori su predstavljeni kao odnosi između uloge i entiteta s «uočiti» (eng. perceive) i «izmijeniti» (eng. modify).

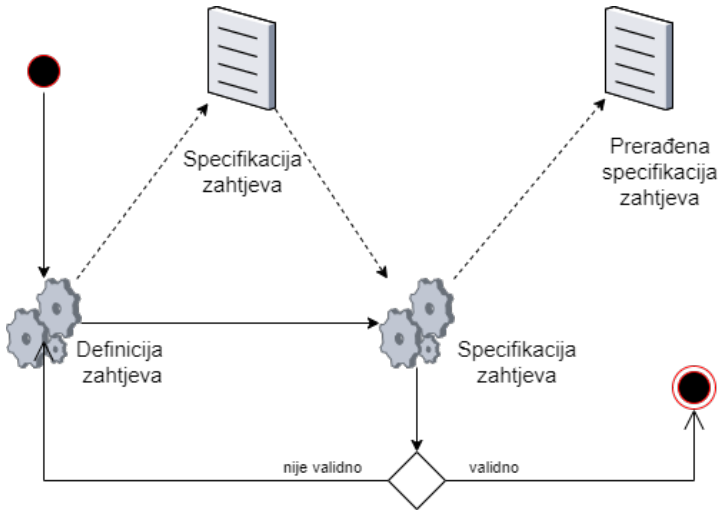
Model organizacijskih pravila identificira i predstavlja opća pravila koja se odnose za ponašanje organizacije. Ova se pravila mogu promatrati kao opća pravila, odgovornosti, ograničenja, željeno ponašanje i slijed ili redoslijed u takvom ponašanju. Skup pravila bi trebao biti predstavljen shemom tablice u kojoj je svako pravilo definirano opisom odnosa na prirodnom jeziku, tipom i odgovarajućom formulom ako je potrebno.

Svaki od ovih modela pruža informacije potrebne za pokrivanje različitih perspektiva VAS-a:

- strukturna - model domene, model uloga, model okruženja
- organizacijska - izjava o misiji, organizacijskih model, model uloga, model domene, model organizacijskih pravila
- funkcionalna - izjava o misiji, model ciljeva, model ponašanja, model okoliša, model organizacijskih pravila
- društveno ponašanje - model uloga, model ponašanja

3.3. Proces modeliranja zahtjeva

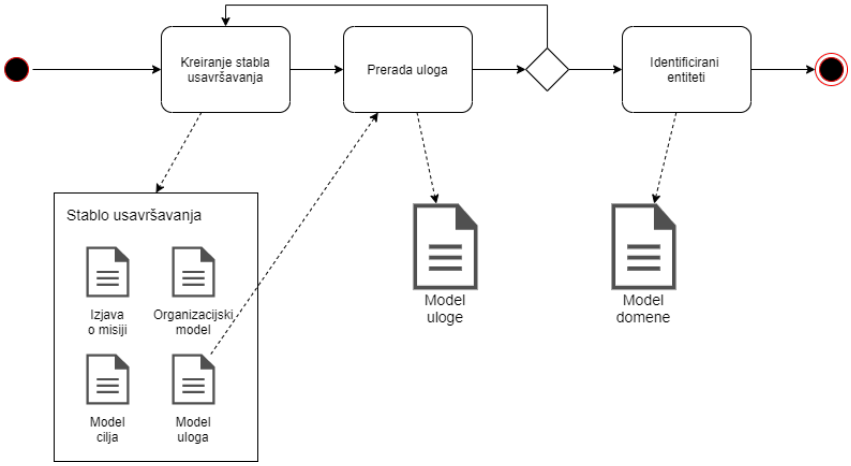
Kao što je ranije spomenuto, predloženi proces modeliranja zahtjeva uključuje dvije faze, definiciju zahtjeva i specifikaciju zahtjeva (Slika 5). Svaka aktivnost procesa proizvodi dokument koji se sastoji od skupa svih modela i dokumenata radnih definicija koja je uključena u svaku aktivnost. Prvo se izvode zadaci aktivnosti definiranja zahtjeva, čime se proizvodi specifikacija zahtjeva. Zatim se izvršavaju zadaci aktivnosti specifikacije zahtjeva, koristeći specifikaciju zahtjeva proizvedenu u prethodnom koraku kao ulaz i rezultirajući izradom precizirane specifikacije zahtjeva. U ovom trenutku se aktivnost definicije zahtjeva može ponovno izvesti u slučaju da se u specifikaciji naiđe na neku vrstu nedosljednosti ili nepotpunosti ili proces može završiti.



Slika 5: Proces modeliranja zahtjeva (Izvor: Alkhateeb, Maghayreh i Doush, 2011)

3.3.1. Definicija zahtjeva

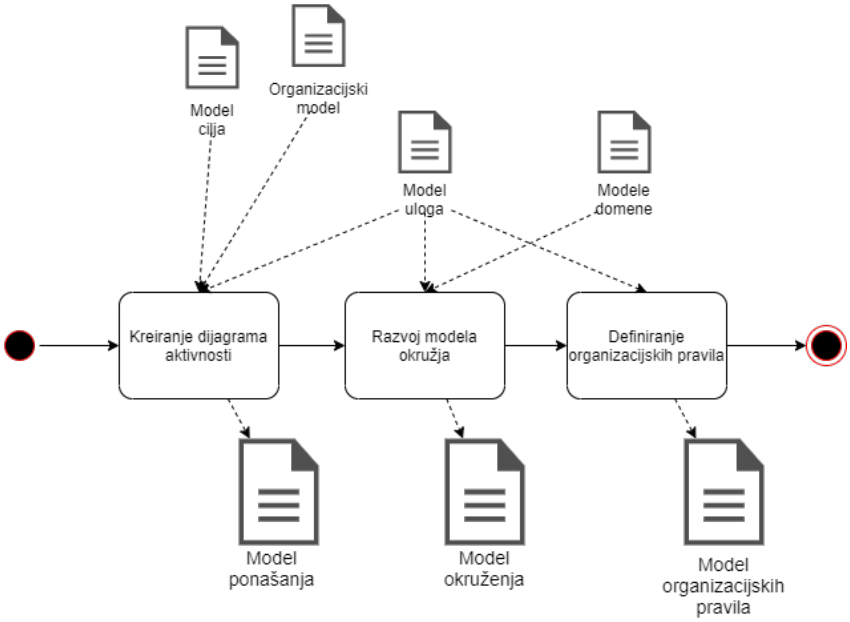
Aktivnost definicije zahtjeva sastoji se od tri koraka čiji je cilj identificirati modele faza (Slika 6). Prvi korak je kreiranje stabla usavršavanja, počevši od definicije izjave o misiji koja se zatim dijeli na podorganizacije, uloge i ciljeve. Ove informacije su dio izjave o misiji, organizacijskog modela, modela uloga i modela ciljeva. Popis uloga identificiran u prethodnom koraku će se koristiti kao ulaz za sljedeći korak, prerada uloga. Ovdje raspravljamo o mogućim strukturnim sličnostima kako bismo identificirali nasljeđivanje te analiziramo buduće ciljeve postignute svakom ulogom u svakoj podorganizaciji kako bi se identificiralo odnose društvenog ponašanja među njima. Ako se smatra prikladnim, moguće je vratiti se na prethodni korak za ažuriranje stabla usavršavanja ili se može izvesti sljedeći korak. U posljednjem koraku, identificiranje entiteta, model domene je izgrađen od identificiranih entiteta i definirani su asocijacijski, kao i nasljeđivanje među njima.



Slika 6: Definicija zahtjeva (Izvor: Alkhateeb, Maghayreh i Doush, 2011)

3.3.2. Specifikacija zahtjeva

Aktivnost specifikacije zahtjeva uključuje stvaranje tri modela, modela ponašanja, model okružja i model organizacijskih pravila, pa se stoga sastoji od tri koraka za izradu modela (Slika 7). Prvi korak je stvaranje dijagrama aktivnosti gdje se kao ulazni podaci koristi organizacijski model, model uloge i model cilja. Kao rezultat toga nastaju potrebni dijagrami aktivnosti. Kad je to dovršeno, izvršava se sljedeći korak, razvoj modela okruženja gdje se koriste model uloge i model domene iz faze definicije kao ulazni podaci. Zatim se izvršava korak definiranja organizacijskih pravila, uzimajući model uloge iz definicije i model okruženja iz trenutne faze gdje kao rezultat dobijemo model organizacijskih pravila.
Artefakti nastali tijekom procesa mogu se povezati na analizu i projektiranje artefakata iz drugih metodologija uspostavljanjem okvira za sljedivost čime će se povećati ukupna kvaliteta sustava.



Slika 7: Specifikacija zahtjeva (Izvor: Alkhateeb, Maghayreh i Doush, 2011)

4. Agent UML

Unified Modeling Language (UML) dobiva široko prihvaćanje za predstavljanje inženjerskih artefakata u objektno orijentiranom softveru. Viđenje agenta kao proširenje objekta navodi nas na istraživanje UML-a i idioma unutar njega kako bi se prilagodili posebnim zahtjevima agenata. Kako bi se postigao ovaj cilj, uspostavljena je suradnja između Foundation of Intelligent Physical Agents (FIPA) i Object Management Group (OMG). Sažet ćemo osnovne dijelove unutar agenta UML (u nastavku MA-UML), tj. mehanizme za modeliranje protokola za višeagentne interakcije i proširenja dijagrama klasa za agente, gdje ćemo uvesti nove dijagrame klasa u UML, dijagram protokola i dijagram klasa agenta.
Dijagrami protokola proširuju se i kombiniraju s UML dijagramima stanja i slijeda na različite načine. Posebna proširenja, u tom kontekstu, uključuje uloge agenata, višedretvene životne cikluse, proširena semantika, parametrizirani ugniježđeni protokoli i predlošci protokola. Dijagram klase agenata proširuje uobičajene dijagrame klasa informacijama specifičnim za agenta. [6]

4.1. UML

UML objedinjuje i formalizira metode mnogih objektno orijentiranih pristupa, uključujući Boocha, Rumbaugha (OMT), Jacobsona i Odella.

Podržava sljedeće vrste modela:

- use case - specifikacija radnji koje sustav ili klasa mogu izvesti interakcijom s vanjskim akterima, obično se koriste za opisivanje načina na koji kupac komunicira sa softverskim proizvodom
- statički modeli - opisuju statičku semantiku podataka i poruka u konceptualnom i operativnom načinu (npr. dijagrami klasa i paketa)
- dinamički modeli - uključuju dijagrame interakcije (npr. dijagrami slijeda i suradnje), grafikoni stanja i dijagrami aktivnosti
- implementacijski modeli - opisuju distribuciju komponenti na različitim platformama (npr. dijagrami komponenata i implementacije)
- jezici za ograničavanje objekta - jednostavan formalni jezik za izražavanje više semantike unutar UML specifikacije, koristi se za definiranje ograničenja na modelu, invarijantama, prije i poslije uvjetnih operacija i navigacijskim putevima unutar mreže objekata.

4.2. Razlozi za Agent UML-om

U principu UML ne pruža nedovoljnu osnovu za modeliranjem agenata i sustava temeljenih na agentima. U osnovi, to je zbog dva razloga. Prvo, u usporedbi s objektima, agenti su

aktivni jer mogu preuzeti inicijativu i imati kontrolu nad time obrađuju li i kako obrađuju vanjske zahtjeve. Drugo, agenti ne djeluju samo izolirano, već u suradnji ili koordinaciji s drugim agentima. Višeagentni sustavi su društvene zajednice međuovisnih članova koji djeluju individualno. Za korištenje programiranja zasnovanog na agentima, tehnika specifikacije mora podržati cijeli proces softverskog inženjeringa, od planiranja, preko analize i projektiranja, do izgradnje, prijelaza i održavanja sustava.
Fokusirat ćemo se na UML reprezentacije paketa, predložaka, dijagrame sekvence i dijagram klasa. Razlog tome je što su protokoli za interakciju agenata (eng. Agent Interaction Protocols AIP) i dijagrami klasa agenata koji predstavljaju ponašanje unutarnjeg agenta i koji ga povezuju s vanjskim ponašanjem, dovoljno složeni da ilustriraju netrivijalnu upotrebu i koriste se dovoljno često. AIP su posebna klasa uzoraka dizajna softvera jer opisuju probleme koji se često pojavljuju u višeagentnim sustavima te zatim opisuju srž rješenja problema za višekratnu uporabu. Definicija protokola interakcije dio je specifikacije dinamičkog modela agentnog sustava. U UML-u ovaj model bilježe dijagrami interakcija, dijagrami stanja i dijagrami aktivnosti.

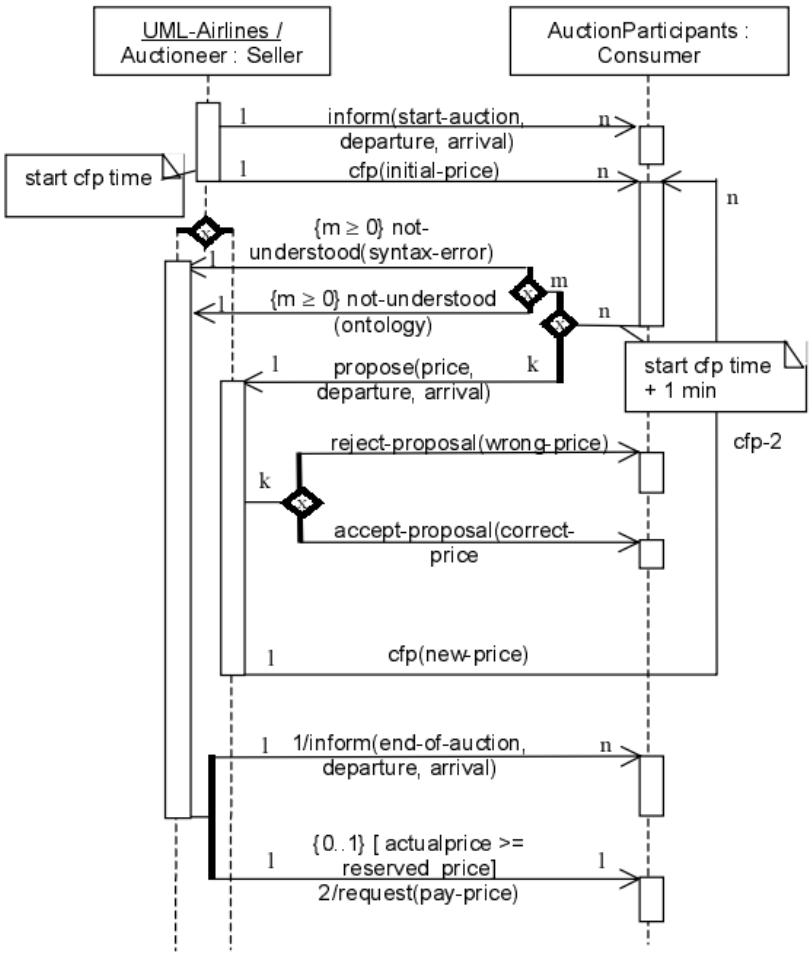
- Dijagrami interakcije (eng. Interaction diagrams) - tj. dijagrami sekvenci i dijagrami suradnje koriste se za definiranje ponašanja skupina objekata. Obično jedan dijagram interakcije bilježi ponašanje jednog slučaja uporabe. Ovi se dijagrami uglavnom koriste za definiranje osnovnih interakcija između objekata na razini pozivanja metode, nisu prikladni za opisivanje vrsta složenih društvenih interakcije u višeagentnim sustavima.
- Dijagrami stanja (eng. State diagrams) - koriste se za modeliranje ponašanja cjelovitog sustava. Definiraju sva moguća stanja do kojih objekt može doći i kako se stanje objekta mijenja ovisno o porukama koje se šalju objektu. Prikladni su za definiranje ponašanja jednog objekta u različitim slučajevima uporabe. Međutim, nisu prikladni za opisivanje ponašanja skupine objekata koji surađuju.
- Dijagrami aktivnosti (eng. Activity diagrams) - koriste se za definiranje tijeka događaja/radnji za nekoliko objekata i uporabe.
- Dijagrami klasa (eng. Class diagrams) - opisuju statičku semantiku podataka i poruka na konceptualni i implementacijski način.

4.3. Agent UML interakcijski protokoli

Definicija protokola za interakciju agenata (AIP) opisuje:

- komunikacijski uzorak
- dopušteni nizovi poruka između agenata s različitim ulogama
- ograničenja u sadržaju poruka
- semantika koja je u skladu s komunikacijskim činom (CA) unutar komunikacijskog obrasca.

Poruke moraju zadovoljavati standardizirane komunikacijske (govorne) činove koji definiraju vrstu i sadržaj poruka (npr. komunikacijski jezik agenta FIPA-e (eng. FIPA agent communication language ACL) ili KQML). Protokoli ograničavaju parametre razmjene poruka, npr. njihov redoslijed ili vrste, prema odnosima između agenata ili namjere komunikacije.
Budući da protokoli interakcije, tj. definicija suradnje između softverskih agenata, definiraju točno ponašanje grupe agenata koji surađuju, kombiniramo dijagrame sekvenci sa zapisom dijagrama stanja za specifikaciju protokola interakcije. Možemo to vidjeti na sljedećem primjeru tržišta viška karata za letove (Slika 8).
Dražba takvih ulaznica može se izvesti pomoću FIPA-inog protokola aukcije. Aukcionar (*Auctioneer*) u početku predlaže cijenu nižu od očekivane tržišne cijene, a zatim postupno podiže cijenu. Aukcionar obavještava sve sudionike da je aukcija započela (*inform(startauction, departure, arrival)*) te objavljuje detalje leta. Svaki put kada se objavi nova cijena (*cfp(initial-price) i cfp(new-price)*), aukcionar čeka do određenog roka da vidi hoće li netko od sudionika pokazati svoju spremnost platiti predloženu cijenu za kartu (*propose(price, departure, arrival)*). Ako sudionik ne razumije ontologiju ili sintaksu cfp-a, odgovara na nerazumljiv komunikacijski način. Simbol dijamanta s "x" u njemu označava odluku koja rezultira slanjem nule ili više komunikacijskih simbola. Čim jedan sudionik naznači da će prihvatiti cijenu, aukcionar objavljuje novi poziv za podnošenje ponuda (*cfp(new-price)*) s povišenom cijenom. Dražba se nastavlja sve dok nijedan sudionik dražbe nije spreman platiti predloženu cijenu i tada dražba završava. Ako zadnja cijena koju je kupac prihvatio premašuje cijenu rezervacije aukcionara, ulaznica se tom sudioniku prodaje po dogovorenoj cijeni, u protivnom dražba završava. Sudionici su obaviješteni o završetku aukcije te se od kupca traži da plati cijenu ulaznice.



Slika 8: AIP Dražba(Izvor: Bauer, Muller i Odell, 2001)

4.4. Elementi dijagrama protokola

U nastavku ćemo proći kroz osnovne elemente dijagrama protokola te što oni predstavljaju u dijagramima.

4.4.1. Uloge agenata

U UML-u, uloga je pojam usmjeren na instancu. U okviru programiranja orijentiranog na agenta, uloga agenta znači skup agenata koji zadovoljavaju istaknuta svojstva, sučelja, opise usluga ili s istaknutim ponašanjem. UML razlikuje višestruku klasifikaciju (npr. agent maloprodaje djeluje istovremeno kao kupac i agent prodavatelja) i dinamičku klasifikaciju (agent može mijenjati svoju klasifikaciju tijekom svog postojanja). Agenti mogu obavljati različite uloge u okviru jednog protokola interakcije, na primjer, na aukciji između zrakoplovnog prijevoznika i potencijalnih kupaca karata, zračni prijevoznik ima ulogu prodavatelja, a sudionici ulogu kupca. No, u isto vrijeme kupac na ovoj dražbi može djelovati kao prodavatelj na drugoj dražbi, tj. agenti koji imaju istaknutu ulogu mogu podržati višestruku klasifikaciju i dinamičku klasifikaciju. Stoga implementacija agenta može zadovoljiti različite uloge. Uloga agenta opisuje dvije varijacije koje se mogu primijeniti unutar definicije protokola. Protokol se može definirati na ra-

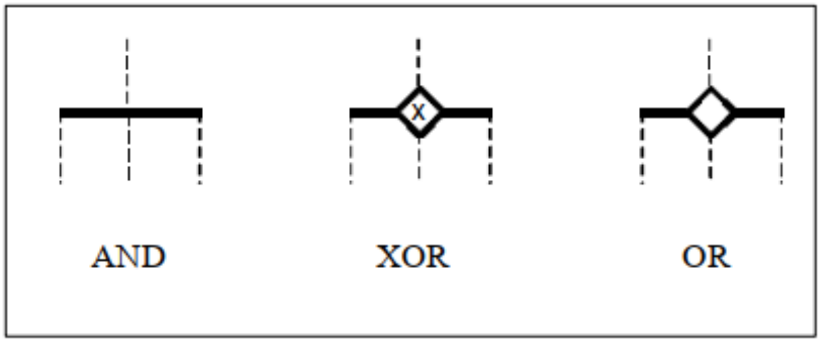
zini konkretnih instanci agenta ili za skup agenata koji zadovoljavaju istaknutu ulogu i/ili klasu. Agent koji zadovoljava istaknutu ulogu i klasu agenta naziva se *agent određene uloge* i *klase*. Opći oblik opisa uloga agenta u UML-u je:

$$instanca - 1...instanca - n/uloga - 1...uloga - m : klasa$$

označavajući istaknuti skup instanci agenta $instance - 1, ..., instanca - n$ koji zadovoljava uloge agenta $uloga - 1, ..., uloga - m$ gdje je $n, m \geq 0$ i klasi $klasa$ kojoj pripada. Na slici 8 aukcionar je konkretna instanca agenta po imenu *UML-Airlines* koji igra ulogu aukcionara klase *Seller*. Sudionici dražbe su agenti uloge *AuctionParticipant* koji su upoznati s dražbama i klase su *Consumer*.

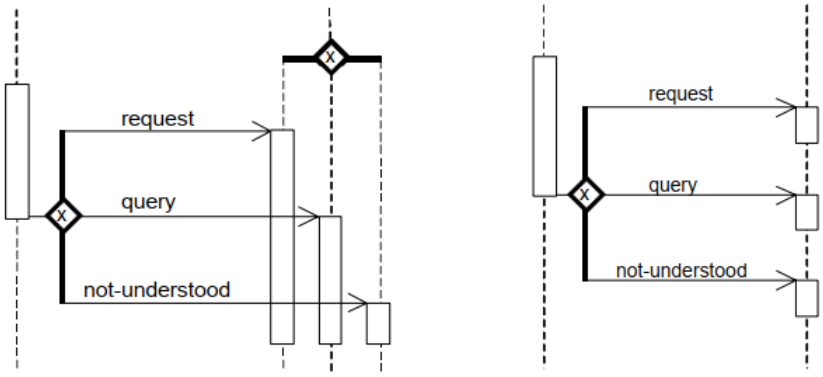
4.4.2. Životni ciklusi agenta i interakcijske dretve

Životni ciklus agenta u dijagramima protokola definira vremensko razdoblje tijekom kojeg agent postoji, predstavljen isprekidanim okomitim linijama. Ciklus počinje kad je kreiran agent određene uloge i završava kada se uništi. Na primjer, korisnički agent stvara se kada se korisnik prijavi u sustav, a uništava kada se korisnik odjavi. Ciklus se može podijeliti u dvije ili više linija kako bi se pokazala AND i OR paralele i odluke, koje odgovaraju granama u toku poruka. Ciklus se mogu spojiti u nekom sljedećem trenutku. Na slici 9 možemo vidjeti grafički prikaz logičkih operatora AND, XOR i OR.



Slika 9: Grafički prikaz logičkih operatora(Izvor: Bauer, Muller i Odell, 2001)

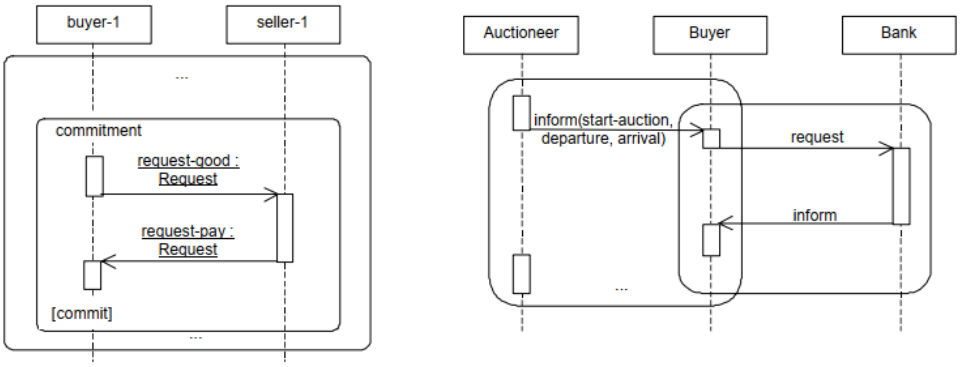
XOR se može skratiti prekidanjem dretve interakcije kao na slici 10. Dretva interakcije (obrada dolaznih poruka) je podijeljena na različite dretve interakcije jer ponašanje uloge agenta ovisi o dolaznoj poruci. Životni ciklus uloge agenta podijeljena je u skladu s tim, a dretva interakcije definira reakciju na različite vrste primljenih poruka. Dretva interakcije prikazuje razdoblje tijekom kojeg uloga agenta izvršava neki zadatak kao reakcija na dolaznu poruku. Predstavlja samo trajanje radnje, ali ne i kontrolni odnos između pošiljatelja poruke i primatelja. Dretva interakcije uvijek je povezana sa životnim ciklusom uloge agenta. Podrška paralelnih dretvi interakcije još je jedno preporučeno proširenje UML-a.



Slika 10: Notacija XOR operatora(Izvor: Bauer, Muller i Odell, 2001)

4.4.3. Ugniježđeni i isprepleteni protokoli

Budući da se protokoli mogu modificirati kao prepoznatljivi obrasci interakcije agenata, oni postaju moduli obrade za višekratnu uporabu koji se mogu tretirati kao prvorazredni pojmovi. Na primjer, slika 11 prikazuje dvije vrste protokolarnih uzoraka. Lijevi dio predstavlja ugniježđeni protokol (protokol unutar drugog protokola), a desni dio isprepleteni protokol (npr. ako sudionik dražbe zatraži neke podatke o svom bankovnom računu prije licitiranja).



Slika 11: Ugniježđeni i isprepleteni protokoli(Izvor: Bauer, Muller i Odell, 2001)

Dodatno ugniježđeni protokoli se koriste za definiciju ponavljanja ugniježđenog protokola prema zaštitama i ograničenjima. Semantika ugniježđenog protokola je semantika protokola. Ako je ugniježđeni protokol označen nekim zaštitnim znakom, tada je semantika ugniježđenog protokola semantika protokola pod pretpostavkom da je zaštita istinita, inače je semantika semantika praznog protokola, tj. ništa nije navedeno. Ako je ugniježđeni protokol označen nekim ograničenjima, ugniježđeni protokol se ponavlja sve dok je ograničenje istinito. Osim uvjeta ograničenja koji se koristi u UML-u, opis $n...m$ označava da se ugniježđeni protokol ponavlja n do m puta gdje je $n \in \{N\}, m \in \{N\} \cup \{*\}$, gdje * označava proizvoljna vremena i koristi se kao uvjet ograničenja.

4.4.4. Proširena semantika UML poruka

Glavna svrha protokola je definiranje komunikacijskih uzoraka, tj. uzoraka poruka poslanih iz jedne uloge agenta u drugu opisanih različim parametrima, kao što su različite kardinalnosti, ovisnosti o nekim ograničenjima ili pomoću AND/OR paralelizma i odluka. Slanje komunikacijskog čina od jednog agenta do drugog koji prenosi informacije i podrazumijeva pošiljateljevo očekivanje da primatelj reagira u skladu sa semantikom komunikacijskog čina. Specifikacija protokola ne govori ništa o tome kako se ova reakcija provodi. Asinkrona poruka se crta $\longrightarrow$ i prikazuje slanje poruke bez davanja kontrole. Sinkrona poruka se prikazuje kao $\Longrightarrow$ i prikazuje popuštanje dretve kontrole (semantika čekanja), tj. agent čeka dok se ne primi poruka odgovora i ništa drugo se ne može obraditi.
Obično su strelice za poruke vodoravne, ukazujući da je vrijeme potrebno za slanje poruke kratko u usporedbi s detaljnošću interakcije i da se ništa drugo ne može dogoditi tijekom prijenosa poruke. Ponavljanje dijela protokola predstavljeno je strelicom ili jednom od varijacija označenih zaštitom ili ograničenjem koje završavaju na dretvi interakcije, koja je, prema vremenu, prije ili poslije stvarne vremenske točke poput *cfp(new-price)* na slici 8. Ovo ponavljanje je još jedno proširenje UML poruka. Svaka strelica označena je oznakom poruke.

Oznaka poruke sastoji se od sljedećih dijelova, koji se također mogu naći na slici 8:

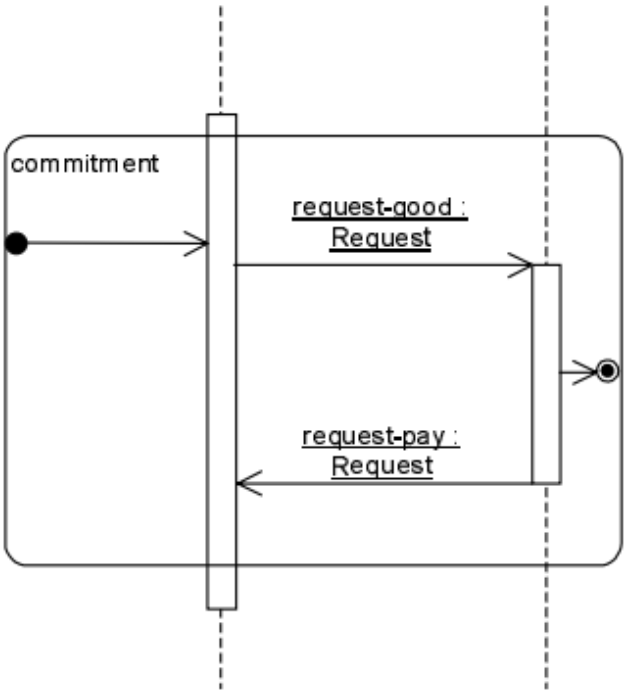
- Komunikacijski čin koji se šalje od jednog agenta drugom, poput *cfp(initial-price)* s popisom argumenata koji predstavljaju dodatne informacije za karakterizaciju komunikacijskog čina.
- Kardinalnost definira da se poruka šalje od jednog agenta do n agenata. kao u slučaju *cfp(new-price)*
- Ograničenja i zaštite, poput $\{m \geq 0\}$ i $actualprice \geq reservedprice$, mogu se dodati za definiranje uvjeta prilikom slanja poruke.
- Osim uvjeta ograničenja koji se koristi u UML-u, opis $n...m$ koji označava da se poruka ponavlja n do m puta gdje je $n \in \{N\}, m \in \{N\} \cup \{*\}$, gdje * označava proizvoljna vremena i koristi se kao uvjet ograničenja.

Poruke se mogu slati paralelno ili se može poslati točno jedna poruka iz skupa različitih poruka. Na slici 8 je ekskluzivno slanje označeno s *reject-proposal* i *accept-proposal*. *inform(end-of-auction, departure, arrival* i *request(pay-price* su poslane paralelno, ali *inform* je poslano prvo (1/*inform*-2) i *request* je poslano kao druga poruka (2/*request*). Također, *request* je poslano 0 ili jednom, ovisno o tome je li cijena rezervacije dosegnuta ili nije.

4.4.5. Ulaz i izlaz ugniježđenih protokola

Ugniježđeni protokoli mogu se definirati unutar ili izvan dijagrama protokola gdje se koristi ili izvan drugog dijagrama protokola. Ulazni parametri ugniježđenih protokola su dretve interakcije koje se odvijaju u ugniježđenom protokolu i poruke primljene iz drugih protokola.

Izlazni parametri su dretve interakcije koje su pokrenute unutar ugniježđenog protokla i prenose se izvan ugniježđenog protokola i poruke koje se šalju unutar ugniježđenog protokola ulogama agenata koji nisu uključeni u trenutnom ugniježđenom protokolu. Poruka ili dretva interakcije koja završava na ulazu ili počinje od izlaznog parametra ugniježđenog protokola, opisuje vezu cijelog dijagrama protokola s ugrađenim ugniježđenim protokolom.



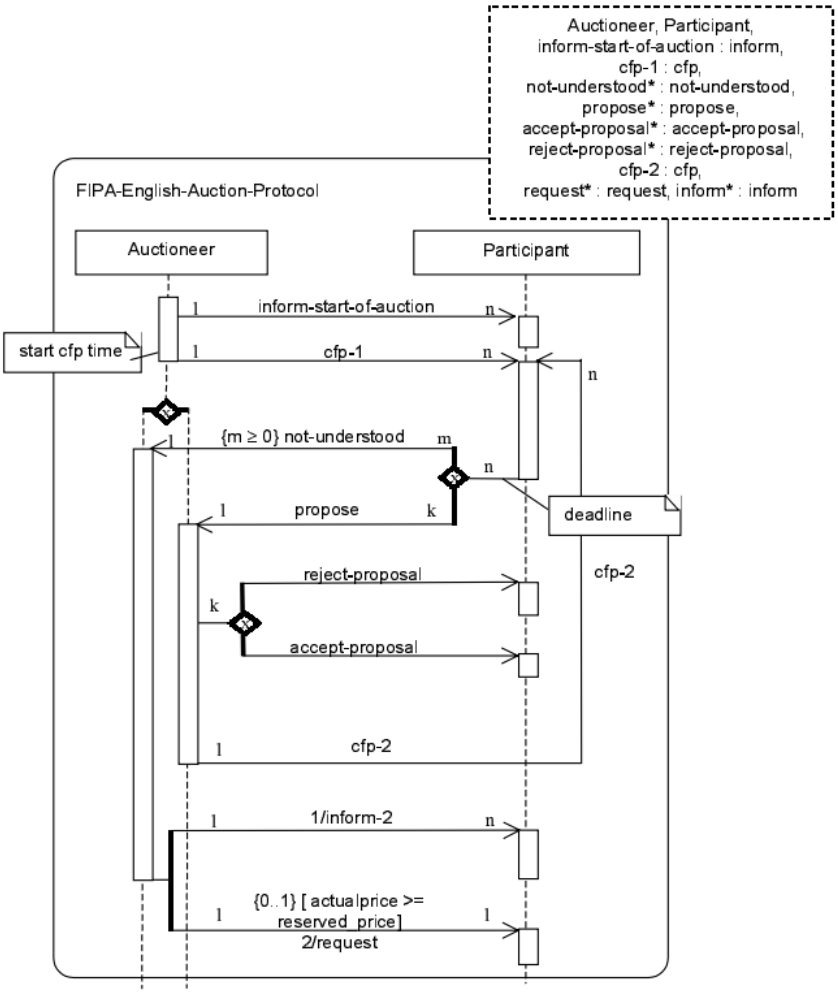
Slika 12: Ulaz i izlaz ugniježđenih protokola(Izvor: Bauer, Muller i Odell, 2001)

Ulazni i izlazni parametri za dretve interakcije ugniježđenog protokola prikazani su na slici 12. koji su nacrtani preko gornje i donje linije kvadrata ugniježđenog protokola. Parametri ulazne i izlazne poruke prikazani su kao $\circ \longrightarrow$ i $\longrightarrow \odot$. Strelice za poruke mogu se označiti poput uobičajenih poruka. Ulazno/izlazna dretva interakcije može se označiti prirodnim brojevima kako bi se definirao točan broj parametara.

4.4.6. Predlošci protokola

Svrha predložaka protokola je stvaranje obrazaca za višekratnu uporabu za konkretne instance protokola. Na primjer, slika 13 prikazuje predložak za FIPA-English-Auction protokol sa slike 8 i predstavlja dva nova koncepta predstavljena na vrhu sekvence. Prvo, protokol se u cjelini tretira kao entitet sam za sebe. Protokol se može tretirati kao uzorak koji se može prilagoditi za druga problematična područja. Isprekidani okvir u gornjem desnom kutu deklarira ovaj uzorak kao specifikaciju predloška koji identificira nevezane entitete (formalne parametre) unutar paketa koji mora biti vezan stvarnim parametrima pri instanciranju paketa. Parametrizirani protokol nije izravno upotrebljiv protokol zbog svojih nevezanih parametara. Njegovi parametri moraju biti vezani uz stvarne vrijednosti kako bi se stvorio vezani oblik koji je protokol. Komunikacijski činovi na formalnom popisu parametara mogu biti označeni zvjezdicom, označavajući različite vrste poruka koje se alternativno mogu poslati u ovom kontekstu. Ovaj se predložak

može izraditi za posebne namjene kao što je prikazano na slici 14.



Slika 13: Generički AIP kao predložak(Izvor: Bauer, Muller i Odell, 2001)

FIPA-English-Auction-Protocol <
UML-Airlines / Auctioneer : Seller, AuctionParticipants : Consumer
start cfp time + 1 min
inform(start-auction, departure, arrival),
cfp(initial-price),
not-understood(syntax-error), not-understood(ontology),
propose(pay-price),
reject-proposal(wrong-price), accept-proposal(correct-price),
cfp(increased-price),
inform(end-of-action), request(pay-price, fetch-car)
>

Slika 14: Instanciranje predloška (Izvor: Bauer, Muller i Odell, 2001)

Tamo se FIPA-English-Auction protokol primjenjuje na određeni scenarij koji uključuje određenog aukcionara UML-Airlinesa koji ima ulogu *Auctioneer* i *Seller* klase, te *AuctionParticipants* klase Consumer i određen je krajnji rok za odgovor prodavatelju. U UML terminologiji, AIP paket služi kao predložak. Predložak je parametrizirani element modela čiji su parame-

tri vezani u vrijeme modela (tj. kad se proizvodi novi prilagođeni model. Wooldridge i sur. predlažu sličan oblik definicije sa svojim definicijama protokola te definiraju pakirane predloške kao "obrazac interakcije koji je formalno definiran i apstrahiran od bilokojeg određenog slijeda koraka izvršenja". Za razliku od njihovog zapisa, mi koristimo grafički prikaz sličan UML-u, a istovremeno izražavaju istu semantiku.

4.5. UML dijagram klasa

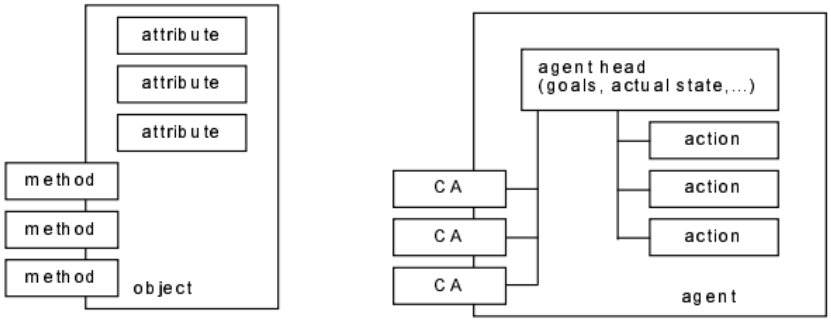
Prije svega, pogledajmo pobliže koncepte objektno orijentiranih programskih jezike, pojmove objekta i klase te ih nakon toga prilagodimo sustavima temeljenim na agentima.

4.5.1. Osnovno o dijagramu klasa

U objektno orijentiranim programskim jezicima, objekt se sastoji od skupa varijabli instance, koje se također nazivaju atributima ili poljima, i njegovih metoda. Stvaranjem objekta određuje se njegov identitet objekta. Varijable instance su varijable koje drže posebne vrijednosti. Metode su operacije, funkcije ili procedure koje mogu djelovati na varijable instance i druge objekte. Vrijednosti polja mogu biti unaprijed definirane osnovne vrste podataka ili reference na druge objekte.
Klasa opisuje skup konkretnih objekata, tj. instance ove klase, s istom strukturom, istim varijablama instance, istim ponašanjem i istim metodama. Postoji standardna metoda "new" za kreiranje novih instanci klase. Definicija klase sastoji se od deklaracije polja i implementacije metode. Sastoji se od specifikacije ili dijela sučelja, kao i dijela za implementaciju. Dio sa specifikacijama opisuje koje metode i funkcije podržava klasa, ali ne i način na koji je operacija realizirana. Dio implementacije definira implementaciju/realizaciju metoda i obično nije vidljiv korisniku metode. Pristupna prava definiraju koje su metode vidljive korisniku, a koje nisu. U većini programskih jezika, klase također definiraju tipovi, odnosno, svaka definicija klase definira vrstu isto imena.
Neki programski jezici dopuštaju definiranje varijabli klase u definicijama klase koje dijele sve klase, za razliku od varijabli instance koje pripadaju jednom objektu, tj. svaka instanca klase ima svoju pohranu za svoju instancu varijable. Klasične se varijable često koriste kao zamjena za globalne varijable. Osim varijabli klase, često se koriste i metode klasa kao globalne procedure koje se mogu pozivati neovisno o kreiranom objektu.

4.5.2. Povezivanje objekata s agentima

Na idući slici (Slika 15) možemo vidjeti odnos između agenta i objekta.



Slika 15: Objekt i agent (Izvor: Bauer, Muller i Odell, 2001)

Imamo autonomiju, proaktivnost i ponovnu aktivnost, komunikacija se temelji na teoriji govornih činova (svakim izričajem govornici izvode jedan društveni čin ili više njih) (tzv. komunikacijski čin, eng. communicative act CA), a unutarnje stanje više je od samo polja s imperativnim tipovima podataka i dodatnim značajkama. Svi ti koncepti moraju biti podržani dijagramom klasa za agente.

U paradigmi programiranja orijentiranoj na agenta, moramo razlikovati klasu agenata koja s jedne strane definira vrstu pojedinačnog agenta, a s druge strane nacrt za pojedine agente, a pojedini agent je primjer klasa agenata. Stoga specificiramo shemu klase agenata koja se instancira u programima.

Agent se može podijeliti na komunikatora (odgovoran za fizičku vezu, glava) koji se bavi ciljevima, stanjima, itd. agenta, i tijelo koje izvodi radnje agenta. Za bolji pregled agenta, moramo navesti njegovu glavu i tijelo.

Cilj specifikacije unutarnjeg ponašanja agenta je pružiti mogućnosti za definiranje, npr. BDI semantike ili stalnih i stvarnih ciljeva, kao i Java agenata. Posebno se mora uzeti u obzir semantika komunikacijskih činova i reakcija agenta na neke dolazne poruke, a to može učiniti dizajner ili agent koristeći, npr. BDI semantiku. Dopuštamo definiranje proceduralnog kao i deklarativnog opisa procesa, specifikacija se može, npr. učiniti pomoću dijagrama aktivnosti ili jezika za specifikaciju UML procesa.

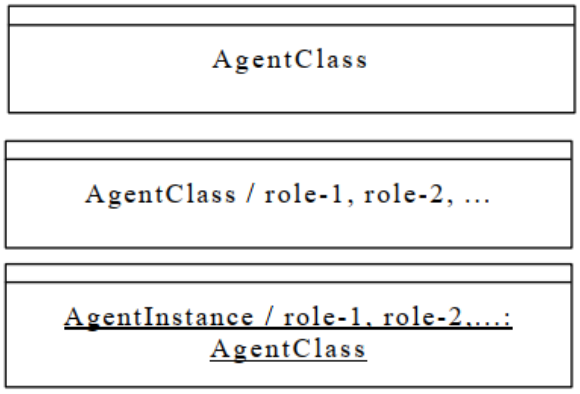
Reakcija na događaje i proaktivnost ponašanje može se definirati ili proaktivnim radnjama ili agent-head automatima za proaktivnost ponašanje. Za agenta se ne mogu definirati samo metode koje su vidljive samo samom agentu, već i radnje kojima drugi agenti mogu pristupiti. No, za razliku od orijentacije objekta, agent odlučuje mora li se izvršiti neka radnja.

Apstraktne radnje karakteriziraju preduvjeti, učinci i invarijante. Štoviše, uobičajene objektno orijentirane tehnike moraju se primijeniti na tehnologiju agenata, podržavajući učinkovit i strukturiran razvoj programa, poput nasljeđivanja, apstraktnih tipova agenata i sučelja agenata te generičkih tipova agenata. Jedno, više i dinamičko nasljeđivanje može se primijeniti za stanja, radnje, metode i rukovanje porukama. Asocijacije se mogu opisati, npr. agent A koristi usluge agenta B za izvršavanje zadatka (klijent, poslužitelj), s određenom kardinalnošću i ulogama, kao i agregacije i kompozicije, npr. usluga i nadzor parkirališta mogu biti dio agenta za parkiranje.

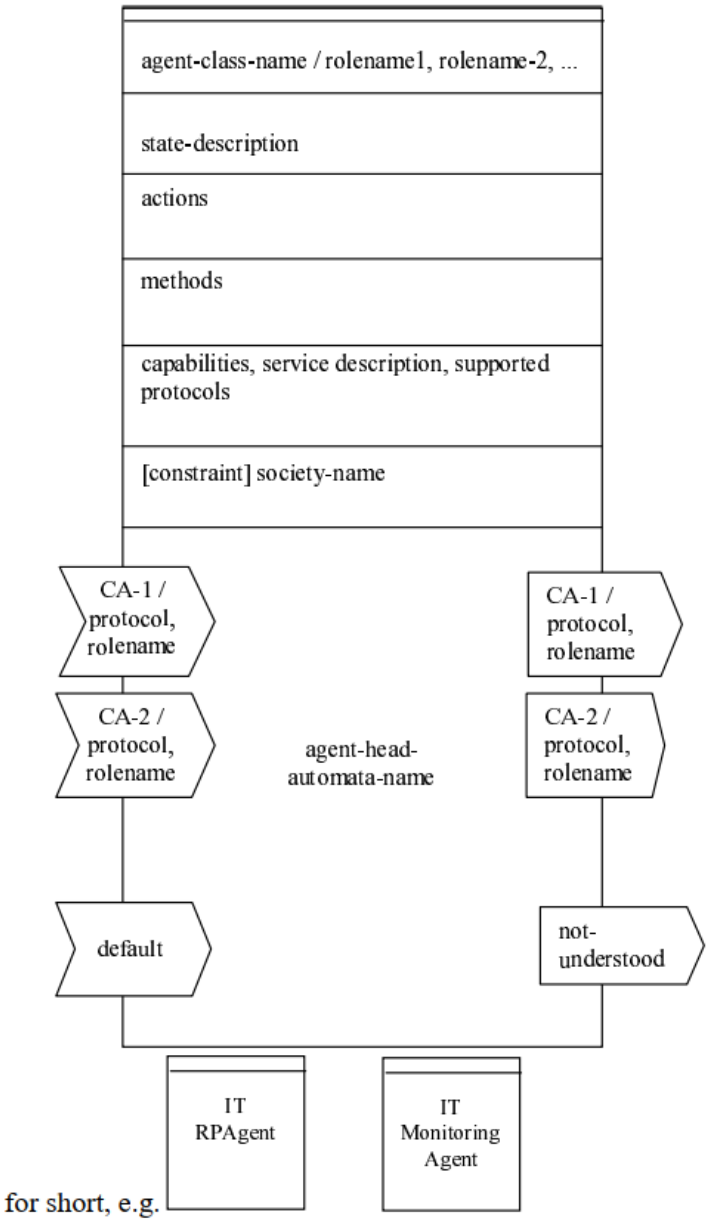
Komponente mogu biti ili klase agenata ili uobičajene objektno orijentiranje klase. Nekoliko smo puta tvrdili da su agent i objekti potpuno različite paradigme, stoga u specifikacijama moramo razlikovati agente i objekte. Posebno se agent može izgraditi pomoću nekog objekta kao dijela svog unutarnjeg stanja te se stoga različite oznake između agenata i objekata moraju koristiti izravno ili koristeći stereotipe.
Povezivanje klasa u smislu objekata s tehnologijom agenata postavlja pitanje što pojam klasa znači u kontekstu agenata, odnosno, u istom smislu kao što je klasa u kontekstu objektno orijentiranog programiranja nacrt za objekt, klasa agenata mora biti nacrt za agente. Može biti ili instanca agenta ili skup agenata koji zadovoljavaju neku posebnu ulogu ili ponašanje.
UML razlikuje različite razine specifikacija, konceptualnu, specifikacijsku i implementacijsku razinu.
Na konceptualnoj razini orijentiranoj prema agentu, agent odgovara ulozi agenta ili klasifikaciji agenta, npr. nadgledanje i planiranje rute mogu se definirati u različitim klasama agenata (tj. možemo imati agenta za planiranje pojedinačnih prometnih ruta i agenta za nadzor koji informira korisnika o nekim zastojima u prometu).
Na specifikacijskoj razini (ili razina sučelja), klasa agenta je nacrt za instance agenta, npr. nadzor i planiranje ruta dio su jedne klase agenata. No, opisana su samo sučelja, a ne i implementacija, tj. nedostaje agent-head automat koji opisuju ponašanje agenta prema dolaznim porukama. Definirana su samo unutarnja stanja i sučelje, tj. komunikacijski činovi koje agent podržava.
Implementacija razina (ili razina koda) najdetaljniji je opis sustava koji pokazuje kako instance agenata rade zajedno i kako izgleda implementacija klase agenata. Na ovoj razini agent-head automati moraju biti definirani.

4.5.3. Dijagram klasa agenata

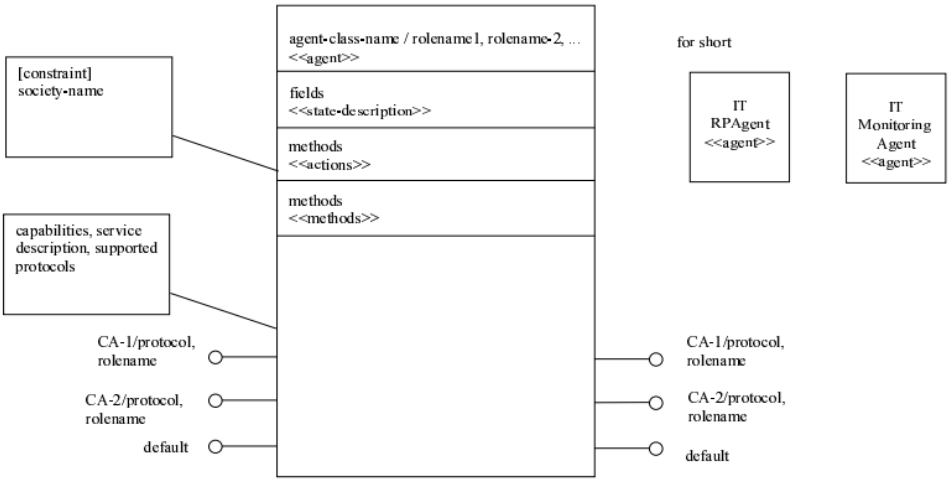
Uobičajeni UML dijagrami klasa mogu se koristiti i proširiti u okviru razvoja programiranja orijentiranog prema agentima. Na sljedećoj slici (Slika 16) možemo vidjeti notacije za razlikovanje različitih vrsta klasa agenata i instanci. Prva označava neku klasu agenata, druga klasu agenata koja zadovoljava istaknute uloge, a posljednja definira neku instancu agenta koja zadovoljava istaknute uloge. Uloge se mogu zanemariti za instance agenata. Prema tome i onome što se mora navesti za klase agenata, agente specificiramo prema dijagramu klasa agenata prikazanom na slici 17. Uobičajeni UML zapis može se primijeniti za definiranje takve klase agenata, ali iz razumljivih razloga smo uveli gornji zapis. Korištenje stereotipa za klasu agenta napisano kao dijagram klase prikazano je na slici 18.



Slika 16: Različite klase agenata(Izvor: Bauer, Muller i Odell, 2001)



Slika 17: Dijagram klase agenata i kratice(Izvor: Bauer, Muller i Odell, 2001)



Slika 18: UML dijagram klasa za specificiranje ponašanja agenta i kratice(Izvor: Bauer, Muller i Odell, 2001)

4.5.4. Opis stanja

Opis stanja izgleda slično opisu polja u dijagramima klasa, s tim što uvodimo istaknutu klasu *wff* (well formed formula) za sve vrste logičkih opisa stanja, neovisno o temeljnoj logici. S ovim proširenjem imamo mogućnost definirati i npr. BDI agente. Osim proširenja tipa za polja, dopuštamo osim atributa vidljivosti i atribut postojanosti koji karakterizira da je vrijednost ovog atributa postojanost, npr. u našem primjeru osobne pomoći pri putovanju korisnički agent može imati varijablu instance koja pohranjuje već planirana i rezervirana putovanja. Ovo polje je trajno (označeno stereotipom «persistent») ako se korisnički agent zaustavi i ponovno pokrene kasnije u novoj sesiji agenta. Opcionalno, polja se mogu inicijalizirati s nekim vrijednostima.
U slučaju BDI semantike mogu se definirati četiri varijable instance, npr. imenovana uvjerenja, želje, namjere i ciljevi, svaki tipa *wff*. Ta se polja mogu inicijalizirati početnim stanjem BDI agenta. Semantika kaže da *wff* vrijedi za uvjerenja, želje, namjere i ciljeve agenta.
U čistoj semantici usmjerenoj na cilj mogu se definirati dvije instance varijable tipa *wff*, nazvane stalni ciljevi i stvarni ciljevi, držeći formulu za trajne i stvarne ciljeve. Uobičajena UML polja mogu se definirati ako moramo navesti običnog objektno orijentiranog agenta, tj. agenta koji se implementira povrh, npr. agenta zasnovanog na Javi (npr. JADE).
Međutim, u različitim fazama projektiranja različite vrste agenata mogu biti prikladni, npr. na konceptualnoj razini mogu se navesti neki BDI agenti koje zatim implementira neka platforma agenata temeljenih na Javi, tj. izvode se neki koraci dorade od BDI agenata do Java agenata.

4.5.5. Akcije

Proaktivno ponašanje može se definirati na dva različita načina, upotrebom proaktivnih radnji i agent-head automata s proaktivnim ponašanjem. Tako se za agenta mogu odrediti dvije vrste radnji, proaktivne radnje (označene stereotipom «pro-active») pokreće sam agent, npr. pomoću timera ili se postiže posebno stanje, tj. testira se na promjenama stanja agenta (npr. mjerač vremena, ulaz senzora) ako se preduvjet radnje ocijeni s *true*. Ponovno aktivne

radnje (označene stereotipom «re-active») pokreće drugi agent, npr. primanje neke poruke od drugog agenta.
Opis radnji agenta sastoji se od potpisa radnje s atributom vidljivosti, naziva radnje i popisa parametara s pripadajućim tipovima. Semantika radnje definirana je preduvjetima, postuslovima, učincima i invarijantama kao u UML-u.

4.5.6. Metode

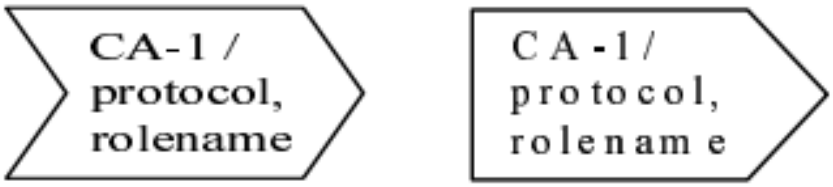
Metode se definirane kao u UML-u, eventualno s preduvjetima, post-uvjetima, učincima i invarijantima.

4.5.7. Sposobnosti

Sposobnosti agenta mogu se definirati ili na neformalan način ili pomoću dijagrama klasa, npr. opisi usluga FIPA-e.

4.5.8. Slanje i primanje komunikacijskih radnji

Glavno sučelje agenta s njegovim okruženjem je slanje i primanje komunikacijskih činova. Komunikacijskim činom (eng. communicative act CA) podrazumijevamo vrstu poruke kao i druge podatke poput pošiljatelja, primatelja ili sadržaja kao u FIPAACL porukama. Pretpostavljamo da su informacije o komunikacijskim činovima predstavljene klasama i objektima. Prikaz dolaznih i odlaznih poruka vidimo na idućoj slici (Slika 19). Primljeni ili poslani komunikacijski čin može biti ili neka klasa ili neka konkretna instanca.



Slika 19: Dolazne i odlazne poruke(Izvor: Bauer, Muller i Odell, 2001)

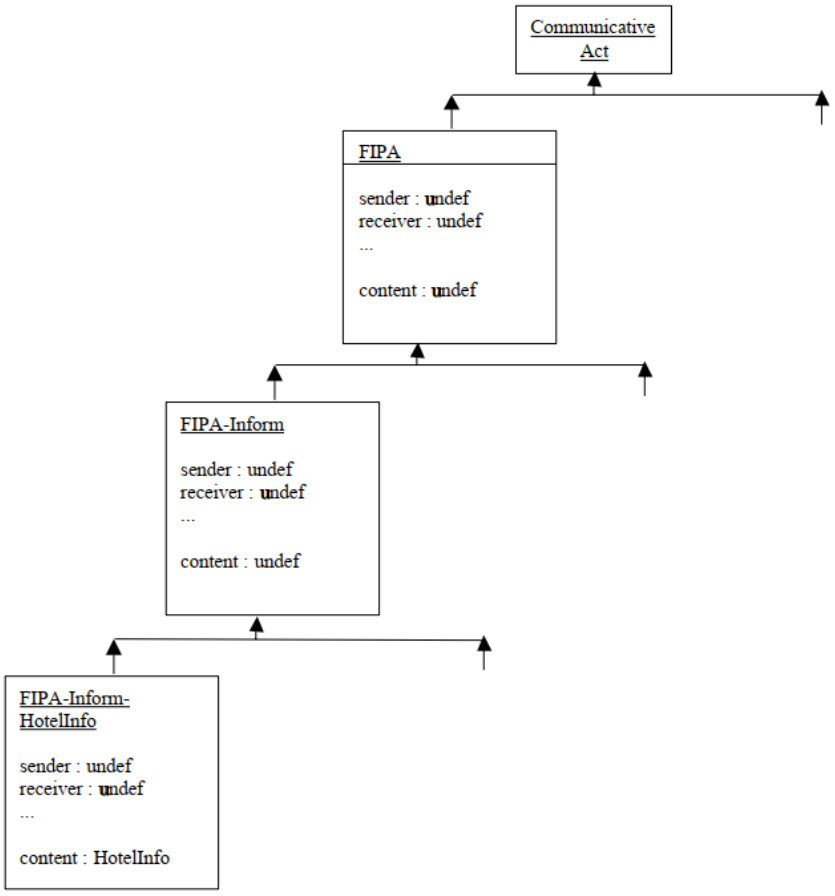
Oznaka *CA-1/protocol, rolename* koristi se ako je komunikacijski čin klase *CA-1* primljen u kontekstu protokola protokola interakcije i agenta. U slučaju instance komunikacijskog čina koristi se oznaka *CA-1/protocol, rolename*. Kao alternativu zapisujemo *protocol[CA-1, rolename]* i *protocol[CA-1, rolename]*. Kontekst */protocol, rolename* mogu se izostaviti ako se komunikacijski čin tumači neovisno o nekom protokolu i/ili ulozi. Kako bismo se ponašali prema svim vrstama primljenih komunikacijskih činova, koristimo se istaknutim zadanim komunikacijskim činom, koji odgovara svakom dolaznom komunikacijsom činu. CA koji se ne razumije se šalje ako se dolazni CA ne može protumačiti.

Razlikujemo instance i klase iz sljedećih razloga:

- instanca opisuje konkretan komunikacijski čin s fiksnim sadržajem ili drugim fiksnim poljima (npr. konkretan zahtjev "pokreni dražbu za posebno dobro") koristila bi se instanca komunikacijskog čina
- kako bismo koristili fleksibilniji ili općenitiji opis (npr. zahtjev "pokreni dražbu za bilokoju vrstu dobra"), koristi se sučelje klase agenata za komunikacijske činove

4.5.9. Usklađivanje komunikacijskih radnji

Primljeni komunikacijski čin mora se uskladiti s dolaznim komunikacijskim činovima agenta kako bi se pokrenulo odgovarajuće ponašanje agenta. Podudaranje komunikacijskih činova ovisi o redoslijedu od vrha do dna, budući da se više od jednog komunikacijskog čina može podudarati s dolaznom porukom.
Najjednostavniji slučaj je općeniti slučaj, slaže se sa svime, a nerazumljiv je odgovor na poruke koje agent ne razumije. Budući da se podudaramo s primjerima komunikacijskih činova, kao i s klasama komunikacijskih činova, moramo definirati slobodne varijable unutar instanciranog komunikacijskog čina. To je prikazano na slici 20 (dijagram klasa za komunikacijske činove gdje varijable instance imaju tip *undef*). Komunikacijski činovi definirani su klasama bez metoda.



Slika 20: Hijerarhija instanci na komunikacijskim činovima, kao instanca odgovarajuće hijerarhije klasa(Izvor: Bauer, Muller i Odell, 2001)

Primljeni komunikacijski čin CA odgovara dolaznoj poruci CA' akko:

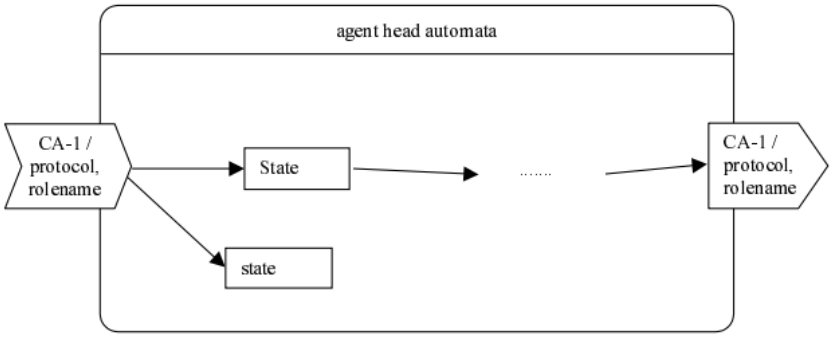
- CA je klasa, onda
  - CA' mora biti instanca klase CA ili
  - CA' mora biti podklasa klase CA ili njegova podklasa
- CA je instanca neke klase, onda
  - CA' je instanca iste klase kao CA i
  - CA.field odgovara CA'.field za sva polja *field* klase CA, definirane kao
    * CA.field odgovara CA'.field, ako CA.field ima vrijednosti *undef*
    * CA.field odgovara CA'.field, ako CA.field su jednaka CA'.field i CA.field nisu *undef* i tip *field* je osnovni tip
    * CA.field odgovara CA'.field, ako CA.field nisu *undef* i tip polja nije osnovni tip podatka i CA.field su instance klase C i CA.field.cfield odgovara CA'.field.cfield za sva polja *cfield* klase C

U slučaju komunikacijskog čina u kontekstu protokola i uloge, protokol *protocol[CA, rolename]* odgovara protokolu *protocol'[CA', rolename']*, ako CA odgovara CA' i *protocol* i *rolename'* su jednaki *protocol* i *rolename*. Analogno vrijedi za odlazne poruke, u ovom slučaju komunikacijski čin mora odgovarati rezultirajućim komunikacijskim činovima agent-head automatu.

4.5.10. Agent-head automat

Agent-head automat definiraju ponašanje glave agenta. Definirali smo agent koji se sastoji od agentovog komunikatora, glave i tijela. Komunikator agenta odgovoran je za fizičku komunikaciju agenta. Glavna funkcionalnost agenta implementirana je u tijelo agenta. To može biti npr. postojeći naslijeđeni softver koji je povezan sa sustavom s više agenata pomoću mehanizama omota (eng. wrapper mechanisms).
Glava agenta je "prekidač" agenta. Njegovo ponašanje mora biti specificirano agent-head automatima. Posebno ovi automati povezuju dolazne poruke s unutarnjim stanjem, radnjama i metodama te odlazne poruke, koje se naziva reaktivno ponašanje agenta. Štoviše, definira proaktivno ponašanje agenta, tj. automatski pokreće različite radnje, metode i promjene stanja ovisno o unutarnjem stanju agenta. Primjer proaktivnog ponašanja je učiniti neku radnju u određeno vrijeme, npr. agent migrira u unaprijed definiranim vremenskim točkama s jednog stroja na drugi ili je to rezultat nekog zahtjeva komunikacijskog čina.
UML podržava, za definiranje dinamičkog ponašanja, četiri vrste dijagrama (dijagram sekvenci, dijagram suradnje na razini objekta, dijagrame stanja i dijagrami aktivnosti). Dijagrami sekvenci i dijagrami suradnje prikladni su za definiciju ponašanja glave agenta, budući da se radi o dijagramu usmjerenom na objekt/agent. Stoga se lako može koristiti za definiranje konkretnog ponašanja, temeljen na radnjama, metodama i promjenama stanja. Primjena jednog od ovih dijagrama ovisi o željama dizajnera. Dijagram stanja i aktivnosti prikladniji su za apstraktniju specifikaciju ponašanja glave agenta.

Ako pogledamo sliku 21, vidimo kako agent reagira na dolazne poruke i njegovo reaktivno ponašanje. Oznake koje koristimo u tu svrhu su prošireni automati stanja. Za razliku od standardnih automata stanja, CA oznaka dijagrama klase koristi se za pokretanje automata (početna stanja), a konačna stanja se podudaraju s odlaznim komunikacijskim činovima. Proaktivno ponašanje se ne pokreće dolaznim porukama, već ovisi o valjanosti ograničenja ili uvjeta, tj. u automatima stanja početna stanja su označena nekim uvjetima.



Slika 21: Prošireni automati stanja(Izvor: Bauer, Muller i Odell, 2001)

5. Jezik GPLKTF

Grafički postupak modeliranja višeagentnih sustava, nazvan GPLKTF, je postupak u kojemu se za prikaz formula koristi PLKTF. Jezik PLKTF se sastoji od propozicijske logike proširene operatorima znanja, temporalnim operatorima i operatora zaboravljanja. GPLKTF temeljni dio čini lista od 25 bazična elementa kroz koje ćemo u nastavku proći. Kombinacijom tih bazičnih elemenata prikazujemo složena svojstva agenata u višeagentnim sustavima.
Ako pogledamo sustav u bilokojem trenutku, svaki od agenata je u nekom stanju. Ovo nazivamo lokalnim stanjem agenta i pretpostavljamo da sadrži sve informacije kojima agent ima pristup. Budući da svaki agent ima lokalno stanje, cijeli sustav smatramo globalnim stanjem. Globalno stanje uključuje lokalna stanja agenata i lokalno stanje okružja. Prema tome, sustav dijelimo na dvije komponente, okružje i agente, gdje okružje promatramo kao sve ostalo što je relevantno. [7]

5.1. Rezoniranje o znanju

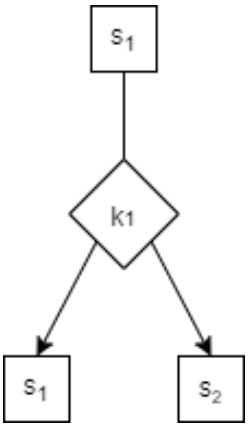
PLKTF i GPLKTF se sastoji od osnovnih notacija i koncepata kroz koje ćemo proći. Jezik PLKTF se zasniva na propozicijskoj logici i znanju agenta (jezik PLK), koji je proširen temporalnim operatorima i operatorom zaboravljanja, dok jezik GPLKTF je grafički prikaz jezika PLKTF i služi za semantičko modeliranje višeagentnih sustava.
Za početak je potrebno proći kroz propozicijsku logiku i znanje agenta, te uvesti model mogućih svjetova koji predstavlja model u kojem je stvarni svijet okružen s bezbroj mogućih svjetova. Za nekog agenta kaže se da zna neku činjenicu *F*, ako *F* vrijedi u svim svjetovima koje on smatra mogućim.
Drugim riječima, uvodimo mogućnosnu relaciju. Mogućnosna relacija predstavlja odnos između dostupnih svjetova i agenta, tj. zapisuju što u kojem svijetu agenti smatraju mogućim. Time možemo prikazati što koji agent smatra mogućim u kojem svijetu. Npr. u tablici 1 možemo vidjeti primjer mogućnosne relacije za nekog agenta k_1 i koja stanja smatra mogućima u svijetu S:

Tablica 1: Mogućnosne relacije agenta

k_1	S	S
	s_1	s_1
	s_1	s_2
	s_2	s_3
	s_3	s_4
	s_3	s_5

(Izvor: Maleković,)

Iz toga vidimo da u svijetu s_1 smatra mogućim stanja s_1 i s_2, što grafički možemo vidjeti na slici 22:



Slika 22: Agent 1 i moguća stanja (Izvor: Maleković,)

Pomoću bazičnih propozicija opisujemo agente, npr. koji agent ima koju kartu kao:

1A : Agent 1 ima kartu A
2C : Agent 2 ima kartu C

Iz toga, skup bazičnih propozicija zadan je s:

$$P = \{1A, 1B, 1C, 2A, 2B, 2C\}$$

Primjenom logičkih operatora ($\neg, \wedge, \vee, \Leftrightarrow, \Rightarrow$) na propoziciju P, dobivamo složene propozicije. Npr.

$$P_1 = 1C \wedge \neg 2A$$

Za određivanje istinitost bazičnih propozicija u mogućim svjetovima, koristimo interpretaciju I, gdje interpretacija (skupa P nad S) je funkcija

$$I : S \times P \to \{\top, \bot\}$$

Sve ovo možemo objediniti u Kripkeovu strukturu koja predstavlja uređenu n-torku $M = (S, I, k_1, k_2)$ nad P.

Definiranjem relacije $\models$ dobijemo da:

- $(M, s) \models G$ znači da formula G vrijedi u svijetu s strukture M
- $(M, s) \models p$ akko $I_s(p) = \top$, gdje je $p \in P$ bazična propozicija

Npr. $(M, (A, C)) \models 1A$ i $(M, (A, C)) \not\models 1B$.

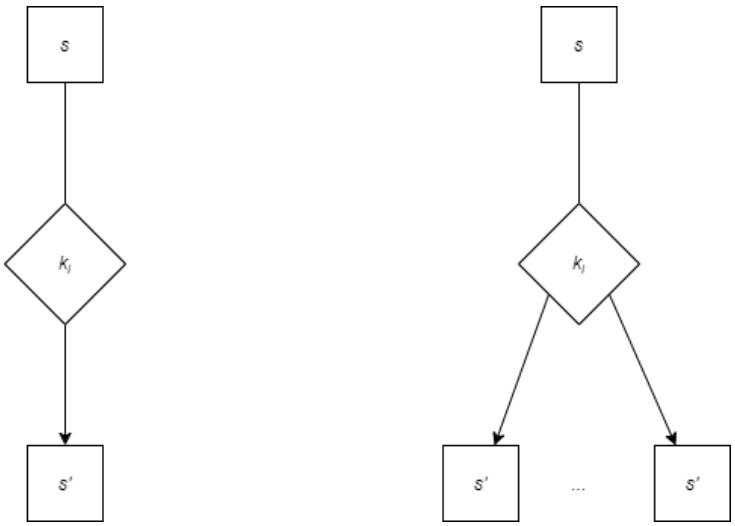
Poopćeno, Kripkeova struktura je definirana kao: Neka je $G = \{1, ..., m\}$ grupa od m agenata. Agenti rezoniraju o svijetu kojeg je moguće opisati pomoću nepraznog skupa bazičnih propozicija P. Kripkeova struktura M za G nad P definirana je kao uređena n-torka:

$$M = (S, I, k_1, ..., k_m)$$

pri čemu je:

- S skup mogućih stanja (svjetova)
- I je interpretacija $I : S \times P \to \{\top, \bot\}$ pri čemu:
  - $I_s(p) = \top$ znači da je p istinito (vrijedi) u stanju s
  - $I_s(p) = \bot$ znači da je p neistinito (nevrijedi) u stanju s
- $k_i \subseteq S \times S$ je mogućnosna relacija za agenta i

Za grafički prikaz toga kažemo da u svijetu s agent i smatra mogućim svijet s', što znači da $(s, s') \in k_i$. Odnosno, u svijetu s agent i smatra mogućim svjetove $s_1, ..., s_k$ (Slika 23.):



Slika 23: Agent k_1 i mogući svjetovi (Izvor: Maleković,)

$(M, s) \models F$ znači da formula F istinita u svijetu s strukture M. Drugim riječima, F vrijedi u stanju s u strukturi M:

- $(M, s) \models p$ ako $I_s(p) = \top$
- $(M, s) \models \neg F$ ako $(M, s) \not\models F$
- $(M, s) \models F \wedge G$ ako $(M, s) \models F$ i $(M, s) \models G$
- $(M, s) \models F \vee G$ ako $(M, s) \models F$ ili $(M, s) \models G$
- $(M, s) \models F \Rightarrow G$ ako iz $(M, s) \models F$ slijedi $(M, s) \models G$
- $(M, s) \models F \Leftrightarrow G$ ako $(M, s) \models F \Rightarrow G$ i $(M, s) \models G \Rightarrow F$
- $M \models F$ ako $(M, s) \models F, \forall s \in S$

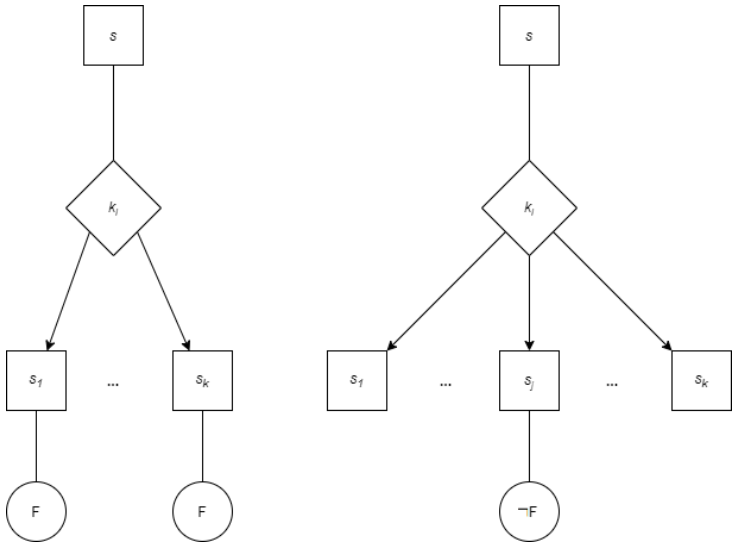
Modalni operator znanja K, koji kasnije koristimo u jeziku PLKTF je definiran kao:

$$K_i(F)$$

i čitamo: agent i zna F, odnosno:

$(M, s) \models K_i(F)$ vrijedi ako $(M, t) \models F$ za svako stanje $t \in S$ takvo da je $(s, t) \in k_i$.

Grafički (Slika 24):



Slika 24: a) $(M, s) \models K_i(F)$, b)$(M, s) \not\models K_i(F)$ (Izvor: Maleković,)

5.2. Jezik PLK

Nakon što smo prošli kroz osnovne koncepte propozicijske logike i uveli modalni operator znanja agenta, možemo definirati jezik PLK. Jezik PLK definiran je kao skup formula. Formula jezika PLK nad skupom propozicija P i grupom agenata A definirana je rekurzivno:

- Svaka bazična propozicija iz P je formula
- Ako su F i G formule, tada su to i izrazi $\neg F, (F \wedge G), (F \vee G), (F \Rightarrow G), (F \Leftrightarrow G)$ i $K_i(F), \forall i \in A$, pri čemu je K_i modalni operator znanja.

Prema jeziku PLK, znanje grupe agenata možemo podijeliti na tri vrste znanja, a to su znanje grupe agenata, opće znanje i distribuirano znanje:

- Znanje grupe agenata:
  Neka je $G = \{1, 2, ..., n\}$ grupa od n agenata. Sa $E_G(H)$ označavamo da svatko u grupi G zna H. Stoga $(M, s) \models E_G(H)$ vrijedi akko $(M, s) \models K_i(H), \forall i \in G$.
- Opće znanje grupe agenata:
  Neka je $G = \{1, 2, ..., n\}$ grupa od n agenata. Sa $C_G(H)$ označavamo da je H opće

znanje grupe G. $(M,s) \models C_G(H)$ vrijedi akko $(M,s) \models E_G^k(H)$ gdje je $k = 1,2,3,...$, $E^0 = H, E^1 = E(E^0), ..., E^{k+1} = E(E^k)$.

- Distribuirano znanje grupe agenata:
  Neka je $G = \{1,2,...,n\}$ grupa od n agenata. Neka je nadalje sa $K_i^s; i = 1,...,n$ označen skup svih znanja agenta i u svijetu s. Sa $D_G(H)$ označavamo da je H distribuirano znanje članova grupe G. $(M,s) \models D_G(H)$ vrijedi akko $(K_1^s \cup ... \cup K_n^s) \vdash H$.

Kako se agenti ponašaju u jeziku PLK opisano je u njihovim svojstvima. Svojstva agenata ovise o svojstvima pripadne relacije $k \subseteq S \times S$:

- Refleksivnost (R) - $(t,t) \in k; \forall k \in S$
  - Ako je agent i refleksivan u strukturi M, tada vrijedi
    $M \models K_i(H) \Rightarrow H$
- Simetričnost (S) - $\forall s,t \in S : (s,t) \in k \Rightarrow (t,s) \in k$
  - Ako je agent i simetričan u M, tada vrijedi
    $M \models H \Rightarrow K_i(\neg K_i(\neg H))$
- Tranzitivnost (T) - $\forall s,t,u \in S : (s,t) \in k \wedge (t,u) \in k \Rightarrow (s,u) \in k$
  - Neka je agent i tranzitivan u M, tada vrijedi
    $M \models K_i(H) \Rightarrow K_i(K_i(H))$
- Relacija ekvivalencije (E) - k je relacija ekvivalencije ako ima svojstva (R), (S) i (T)
  - Neka je za agenta i relacija k_i relacija ekvivalencije, tada za takvu strukturu $M = (S, I, k_i)$ vrijedi
    1. $M \models (K_i(G) \wedge K_i(G \Rightarrow H)) \Rightarrow K_i(H)$
    Propozicija kaže da agent 1 zna logičke posljedice svog znanja (po modul ponensu)
    2. Ako $M \models H$, onda $M \models K_i(H)$
- Euklidovo svojstvo (Eu) - $\forall s,t,u \in S : (s,t) \in k \wedge (s,u) \in k \Rightarrow (t,u) \in k$
  - Ako je agent i Euklidov agent u M, tada vrijedi
    $M \models \neg K_i(H) \Rightarrow K_i(\neg K_i(H))$
- Svojstvo serije (Se) - $\forall s \in S, \exists t \in S : (s,t) \in k$

Zaključujemo da agent ima neko svojstvo ako to svojstvo ima pripadna relacija k, odnosno:
Agent i ima svojstvo P u strukturi $M = (S, I, k_i)$ ako relacija k_i ima svojstvo P.

U nastavku ćemo uvesti temporalne operatore i operator zaboravljanja, zbog čega je bitno da definiramo prolaz višeagentnog sustava kao:
Neka je L_0 skup mogućih lokalnih stanja okoline i neka je sa L_i označen skup lokalnih stanja agenta i za $i \in A$. Definiramo $G = L_0 \times L_1 \times ... \times L_n$ skup globalnih stanja višeagentnog sustava. Globalno stanje višeaentnog sustava opisuje sustav u nekom trenutku vremena.

Pretpostavljamo diskretno vrijeme izomorfno skupu prirodnih brojeva $\mathbb{N}$. Prolaz višeagentnog sustava nad skupom globalnih stanja G je funkcija $p : \mathbb{N} \to G$. Možemo reći da je p sekvenca globalnih stanja u G, gdje p opisuje dinamiku višeagentnog sustava.

Za globalno stanje u trenutku t, $p(t) = (s_0, s_1, ..., s_n)$ definiramo lokalna stanja okružja i agenata u trenutku $tp_0(t), p_1(t), ..., p_n(t)$. Uređeni par (p, t) u kojem je p prolaz, a t trenutak u vremenu, nazivamo točkom. Kažemo da je globalno stanje $p_0(t), p_1(t), ..., p_n(t)$ globalno stanje sustava u točci (p, t).

5.3. Temporalni operatori i operator zaboravljanja

Za davanje vremenskih izjava agentima, proširujemo PLK jezik dodavanjem novim modalnim operatorima koji govore o temporalnim (vremenskim) aspektima VAS-a i operatorom zaboravljanja. Ovaj jezik ćemo u nastavku označavati s PLKTF i koristit ćemo ga za zaključivanje o događajima koji se događaju u jednom prolazu višeagentnog sustava. Definiramo pet operatora za budućnost, pet operatora za prošlost i operator zaboravljanja. [7]

Operatori za budućnost su:

- Operator *sljedeće* (next) - $N(F)$ se definira kao: $(M, p(t)) \models N(F)$ akko $(M, p(t+1)) \models F$.
  - $N(F)$ vrijedi u stanju $p(t)$ ako F vrijedi u sljedećem stanju $p(t + 1)$
- Operator *uvijek* (always) - $A(F)$ se definira kao: $(M, p(t)) \models A(F)$ akko $(M, p(t')) \models F; \forall t' \geq t$.
  - $A(F)$ vrijedi u stanju $p(t)$, ako F vrijedi vrijedi u $p(t)$ i u svim nadolazećim stanjima
- Operator *eventualno* (eventually) - $Ev(F)$ se definira kao: $(M, p(t)) \models Ev(F)$ akko $(M, p(t')) \models F; \exists t' \geq t$.
  - $Ev(F)$ će vrijediti ako F vrijedi sada ili u nekom budućem stanju
- Operator *do* (until) - FUG se definira kao: $(M, p(t)) \models FUG$ akko $(M, p(t')) \models G; \exists t' \geq t$ i $(M, p(t'')) \models F; \forall t'' : t \geq t'' > t'$.
  - Formula FUG predviđa eventualno pojavljivanje formule G i tvrdi da će F vrijedi kontinuirano do prvog pojavljivanja formule G
- Operator *čeka* (waiting for) - FWG se definira kao: $(M, p(t)) \models FWG$ akko $(M, p(t)) \models F$ U G ili $(M, p(t)) \models A(F)$.
  - Formula FWG tvrdi da F vrijedi ili do prvog pojavljivanja G ili kroz čitav prolaz

Operatori za prošlost su:

- Operator *prethodno* (previously) - $N^p(F)$ se definira kao: $(M, t(p)) \models N^p(F)$ akko $(M, p(t-1)) \models F$.

  - Formula $N^p(F)$ vrijedi u stanju $p(t)$, ako formula F vrijedi u prethodnom stanju $p(t-1)$. Jasno je da formula nikad neće vrijediti u stanju $p(0)$

- Operator *uvijek bješe* (has always been) - $A^p(F)$ se definira kao: $(M, p(t)) \models F$ akko $\forall t', 0 \leq t' \leq t : (M, p(t')) \models F$.

  - $A^p(F)$ vrijedi ako F vrijedi u trenutnom i svim prethodnim stanjima

- Operator *nekad bješe* (once) - $Ev^p(F)$ se definira kao: $(M, p(t)) \models Ev^p(F)$ akko $\exists t', 0 \leq t' \leq t : (M, p(t')) \models F$.

  - $Ev^p(F)$ vrijedi u stanju $p(t)$, ako je F vrijedilo u nekom prijašnjem stanju

- Operator *otkad* (since) - FU^pG se definira kao: $(M, p(t)) \models FU^pG$ akko $\exists t', 0 \leq t' \leq t : (M, p(t')) \models G$ i $\forall k, t' \leq k \leq t : (M, p(k)) \models F$.

  - Formula FU^pG tvrdi da se G pojavio u prošlosti te da F kontinuirano vrijedi od zadnjeg pojavljivanja G

- Operator *nazad do* (back to) - FW^pG se definira kao: $(M, p(t)) \models FW^pG$ akko $(M, p(t)) \models FU^pG$ ili $(M, p(t)) \models A^p(F)$.

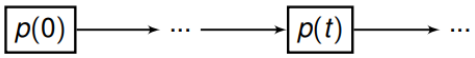
  - FW^pG vrijedi ako je F vrijedilo od zadnjeg pojavljivanja G ili ako takvog pojavljivanja nije bilo F je vrijedilo od početka prolaza

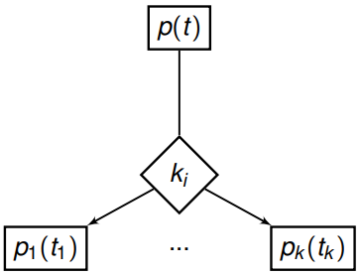
Operator zaboravljanja $F_i(G)$:
$(M, p(t)) \models F_i(G)$ akko $(M, p(t-1)) \models K_i(G)$ i $(M, p(t)) \models \neg K_i(G)$, pri čemu je $t \geq 1$. Posebno definiramo da $(M, p(0)) \not\models F_i(G)$.

- Agent i zaboravlja formulu F u stanju $p(t)$, ako on ne zna F u tom stanju, a znao ju je u prethodnom $(p(t-1))$

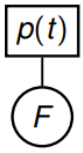
5.4. Grafički jezik GPLKTF

Jezik GPLKTF je grafička metoda koja koristi elemente PLKTF formula za semantičko modeliranje višeagentnih sustava. Sastoji se od propozicijske logike, znanja, temporalnih operatora (za budućnost i prošlost), operatora zaboravljanja, odnosno, sastoji se od sljedećih elemenata [9]:

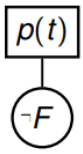
Formula F 
Stanje $p(t)$ 
Trenutno stanje $p(t)$ 
Mogućnosna relacija k_i agenta i 
Prolaz p 
Agent i u svijetu $p(t)$ smatra mogućim svjetove $p_1(t_1), ..., p_k(t_k)$, odnosno, $(p(t), p_1(t_1)) \in k_i, ..., (p(t), p_k(t_k)) \in k_i$



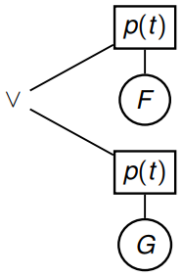
U svijetu $p(t)$ vrijedi formula F, odnosno, $(M, p(t)) \models F$



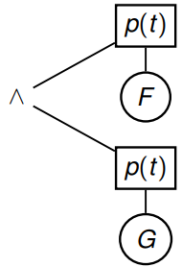
U svijetu $p(t)$ ne vrijedi formula F, odnosno, $(M, p(t)) \models \neg F$



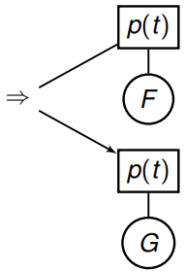
U svijetu $p(t)$ vrijedi formula F ili formula G, odnosno, $(M, p(t)) \models F \vee G$



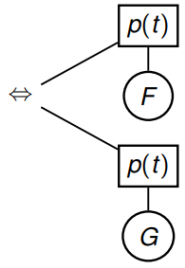
U svijetu $p(t)$ vrijedi formula F i formula G, odnosno, $(M, p(t)) \models F \wedge G$



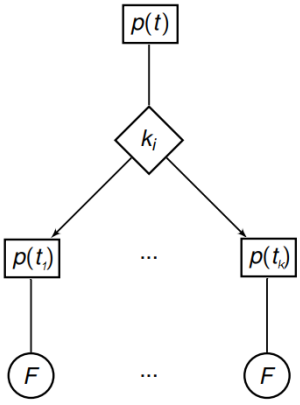
Ako u svijetu $p(t)$ vrijedi formula F, tada vrijedi i formula G, odnosno, $(M, p(t)) \models F \Rightarrow G$



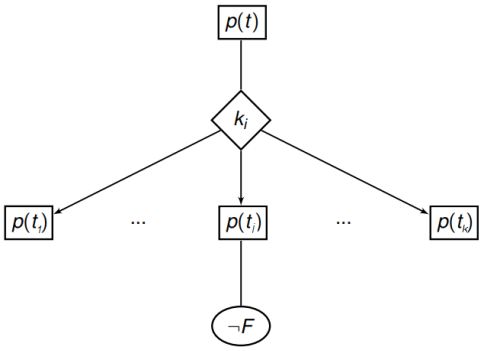
Ako i samo ako u svijetu $p(t)$ vrijedi formula F, tada vrijedi i formula G, odnosno, $(M, p(t)) \models F \Leftrightarrow G$



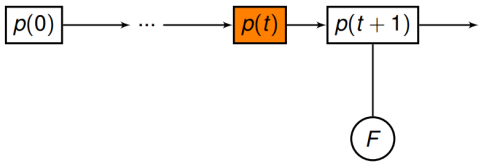
Agent i zna F u stanju $p(t)$, $(M, p(t)) \models K_i(F)$, odnosno, $(M, p(t')) \models F$ za svako stanje $p(t') \in S$ takvo da je $(p(t), p(t')) \in k_i$



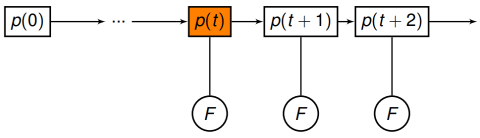
Agent i ne zna F u stanju $p(t)$, $(M, p(t)) \not\models K_i(F)$, odnosno, za neki $p(t_j) \in S$ takvo da je $(p(t), p(t_j)) \in k_i$ i $(M, p(t)) \models \neg F$



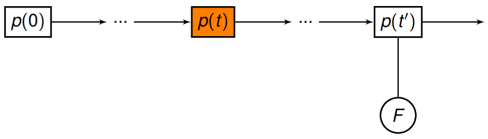
Operator $N(F)$ (u sljedećem stanju F): $(M, p(t)) \models N(F)$ akko $(M, p(t+1)) \models F$



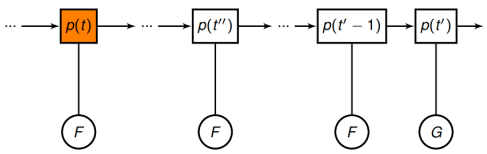
Operator $A(F)$ (uvijek će vrijediti F): $(M, p(t)) \models A(F)$ akko $(M, p(t')) \models F; \forall t' \geq t$



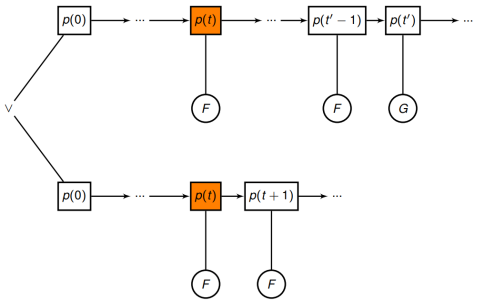
Operator $Ev(F)$ (eventualno će vrijediti F): $(M, p(t)) \models Ev(F)$ akko $(M, p(t')) \models F; \exists t' \geq t$



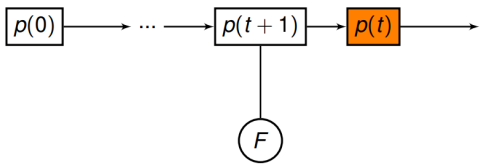
Operator FUG (F vrijedi barem do G): $(M, p(t)) \models FUG$ akko $(M, p(t')) \models G; \exists t' \geq t$ i $(M, p(t'')) \models F; \forall t'' : t \geq t'' > t'$



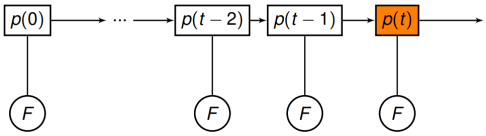
Operator FWG (F čeka na G): $(M, p(t)) \models FWG$ akko $(M, p(t)) \models F$ U G ili $(M, p(t)) \models A(F)$



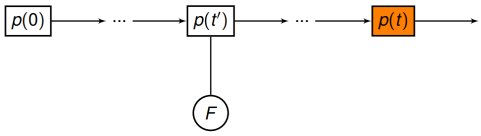
Operator $N^p(F)$ (u prethodnom stanju F): $(M, t(p)) \models N^p(F)$ akko $(M, p(t-1)) \models F$



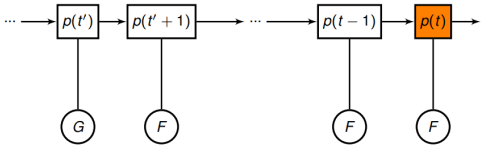
Operator $A^p(F)$ (F je uvijek vrijedilo): $(M, p(t)) \models F$ akko $\forall t', 0 \leq t' \leq t : (M, p(t')) \models F$



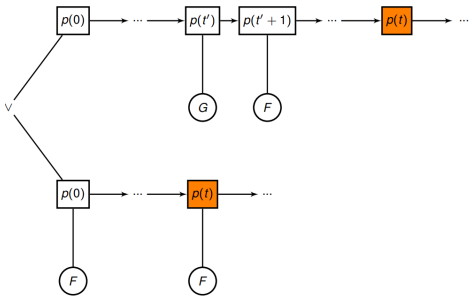
Operator $Ev^p(F)$ (nekad je vrijedilo F): $(M, p(t)) \models Ev^p(F)$ akko $\exists t', 0 \leq t' \leq t : (M, p(t')) \models F$



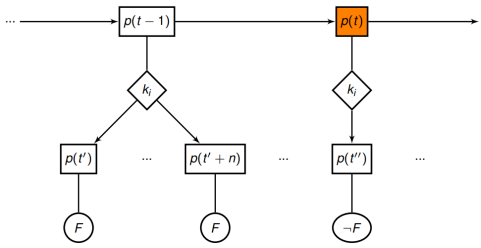
Operator FU^pG (F vrijedi od kad se pojavio G): $(M, p(t)) \models FU^pG$ akko $\exists t', 0 \leq t' \leq t : (M, p(t')) \models G$ i $\forall k, t' \leq k \leq t : (M, p(k)) \models F$



Operator FW^pG (F vrijedi nazad do G): $(M, p(t)) \models FW^pG$ akko $(M, p(t)) \models FU^pG$ ili $(M, p(t)) \models A^p(F)$



Operator $F_i(G)$: $(M, p(t)) \models F_i(G)$ akko $(M, p(t-1)) \models K_i(G)$ i $(M, p(t)) \models \neg K_i(G)$, $t \geq 1$



5.5. Primjeri modeliranja

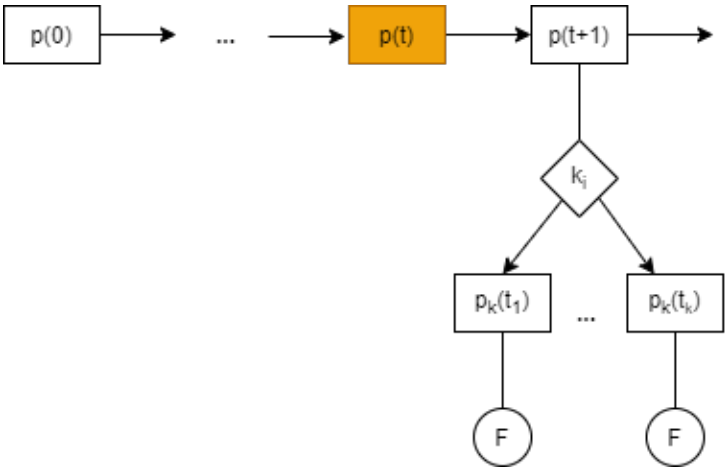
U nastavku ćemo proći kroz primjere modeliranja koristeći GPLKTF kako bismo izrazili kompleksna svojstva agenata.

Struktura rješavanja primjera će biti ista:

- a) PLKTF formula i kako ju čitamo
- b) definicija (semantika) PLKTF formule
- c) prikaz PLKTF formule pomoću GPLKTF.

Primjeri:

1. $(M, p(t)) \models N(K_i(F))$
   (a) $N(K_i(F))$ vrijedi u stanju $p(t)$ ako F vrijedi u sljedećem stanju $p(t+1)$, tj. agent i zna F u sljedećem stanju $p(t+1)$
   (b) $(p(t+1), p(t')) \in k_i \Rightarrow (M, p(t')) \models F; \forall p(t') \in S$

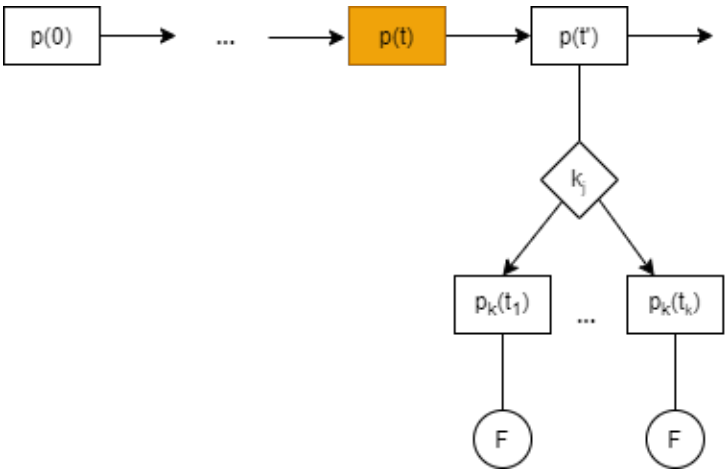


(c)

2. $(M, p(t)) \models Ev(K_j(F))$

   (a) $Ev(K_j(F))$ će vrijediti ako F vrijedi sada ili u nekom budućem stanju, tj. agent j će eventualno znati F u stanju $p(t')$

   (b) $(p(t'), p(t_i)) \in k_j \Rightarrow (M, p(t_i)) \models F; \exists t' \geq t, \forall p(t_i) \in S$

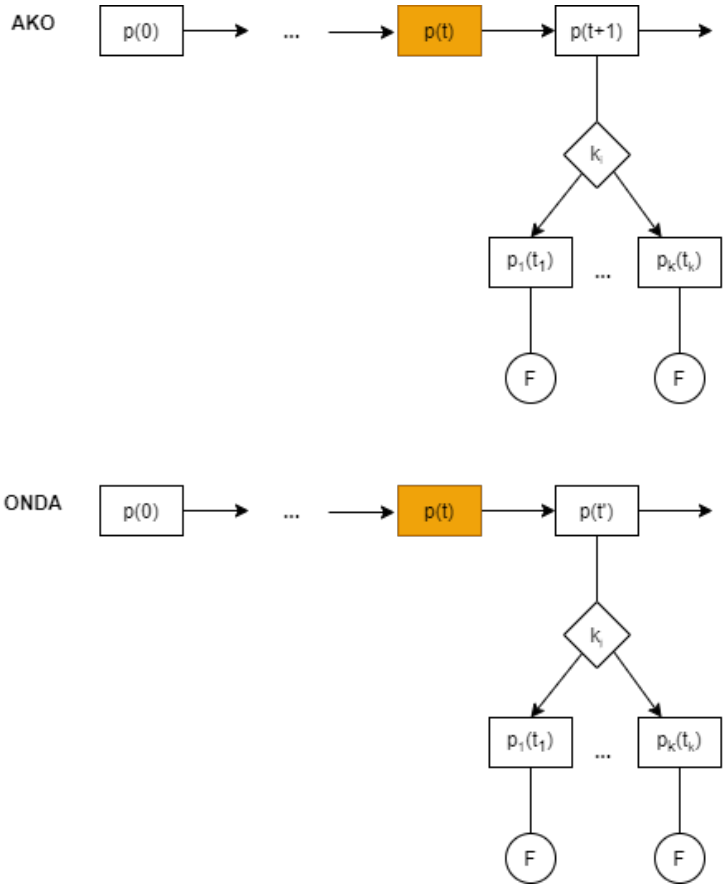


(c)

3. $(M, p(t)) \models N(K_i(F) \Rightarrow Ev(K_j(F)))$

   (a) $N(K_i(F))$ implicira $Ev(K_j(F))$, tj. ako agent i zna F u sljedećem stanju $p(t+1)$, onda eventualno agent j zna F u stanju $p(t)$

   (b) ako $(p(t+1), p(t')) \in k_i \Rightarrow (M, p(t')) \models F; \forall p(t') \in S)$,
   onda $(p(t'), p(t_i)) \in k_j \Rightarrow (M, p(t_i)) \models F; \exists t' \geq t, \forall p(t_i) \in S$

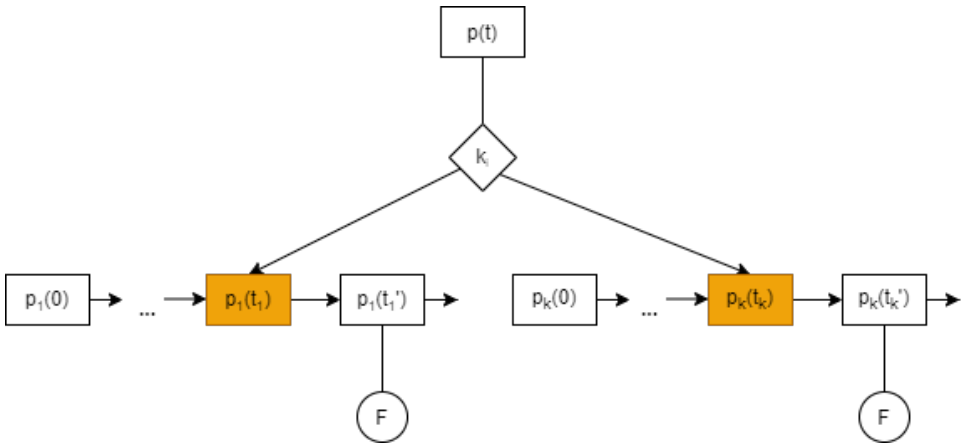


(c)

4. $(M, p(t)) \models K_i(Ev(F))$

   (a) $K_i(Ev(F))$ vrijedi u stanju $p(t)$, tj. agent i zna Ev(F) u $p(t)$

   (b) $(p(t), p(t_i)) \in k_i \Rightarrow (M, p(t')) \models F; \forall p(t_i) \in S, \exists t' \geq t_i$

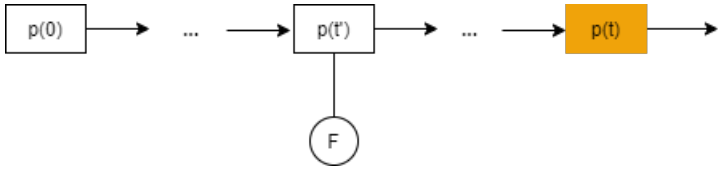


(c)

5. $(M, p(t)) \models Ev^p(F)$

   (a) $Ev^p(F)$ vrijedi u stanju $p(t)$ ako je F vrijedilo u nekom prijašnjem stanju

   (b) $(M, p(t')) \models F; \exists t' : 0 \leq t' \leq t$



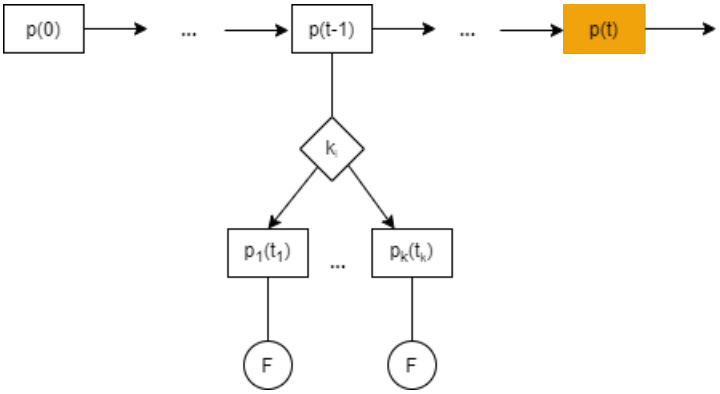
(c)

6. $(M, p(t)) \models N^p(K_i(F))$

   (a) $N^p(K_i(F))$ vrijedi u stanju $p(t)$ ako formula F vrijedi u prethodnom stanju $p(t-1)$, tj. agent i je znao F u prethodnom stanju $p(t-1)$

   (b) $(p(t), p(t_i)) \in k_i \Rightarrow (M, p(t_i)) \models F$; $t > 0$ i $\forall p(t_i) \in S$

   (c)

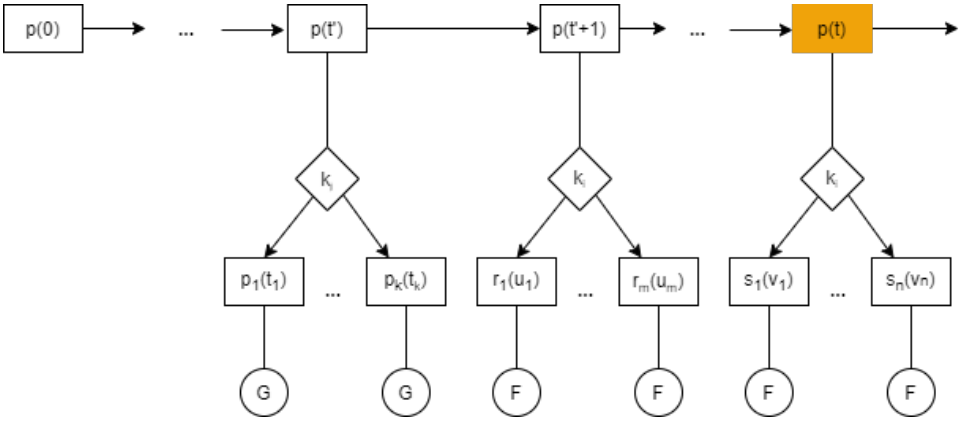


7. $(M, p(t)) \models K_i(F) U^p K_j(G)$

   (a) $K_i(F) U^p K_j(G)$ tvrdi da se G pojavio u prošlosti te da F kontinuirano vrijedi od zadnjeg pojavljivanja G, tj. agent i zna F od kad agent j zna G

   (b) $(p(t'), p(t_i)) \in k_j \Rightarrow (M, p(t_i)) \models G$; $\exists t' : 0 \leq t' \leq t, \forall p(t_i) \in S$
   i $(p(k), r(u)) \in k_i \Rightarrow (M, r(u)) \models F$; $\forall k : t' < k \leq t, \forall r(u) \in S$

   (c)

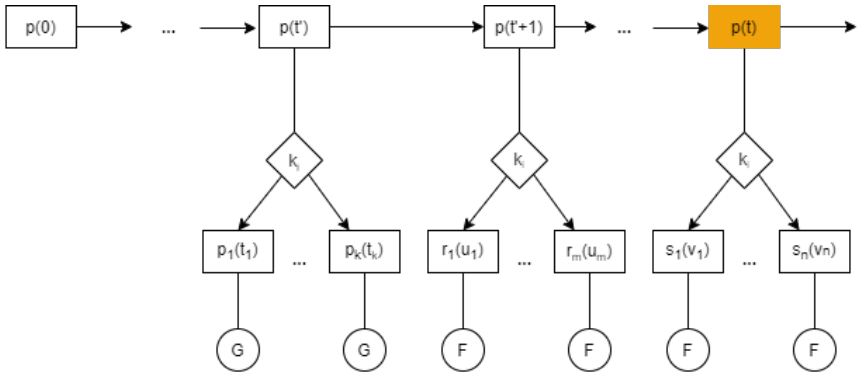


8. $(M, p(t)) \models [K_i(F) U^p K_j(G)] \Rightarrow [Ev(K_l(G)) \vee N^p(K_l(F))]$

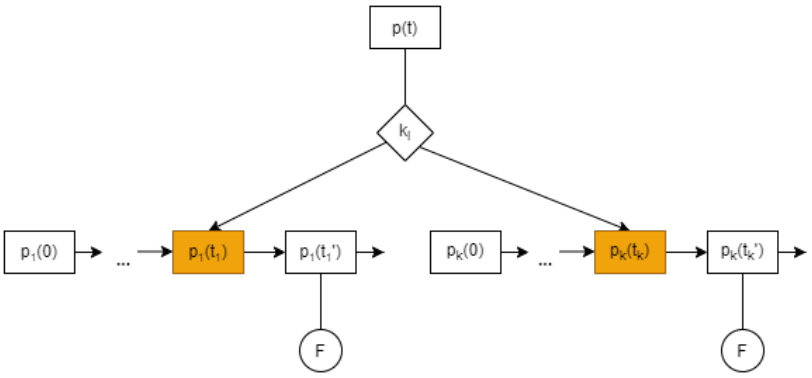
   (a) $K_i(F) U^p K_j(G)$ implicira $Ev(K_k(G)) \vee N^p(K_k(F))$, tj. u stanju $p(t)$ ako agent i zna F odkako agent j je znao G, onda agent l zna eventualno G ili agent l je znao F u prethodnom stanju $p(t-1)$

   (b) ako $(p(t'), p(t_i)) \in k_j \Rightarrow (M, p(t_i)) \models G$; $\exists t' : 0 \leq t' \leq t, \forall p(t_i) \in S$
   i $(p(u), r(v)) \in k_i \Rightarrow (M, r(v)) \models F$; $\forall u : t' < u \leq t, \forall r(v) \in S$,
   onda $(p(t), p(t_i)) \in k_l \Rightarrow (M, p(t')) \models F$; $\forall p(t_i), \exists t' \geq t_i$
   ili $t > 0$ i $(p(t-1), p(t_i)) \in k_l \Rightarrow (M, p(t_i)) \models F, \forall p(t_i) \in S$.

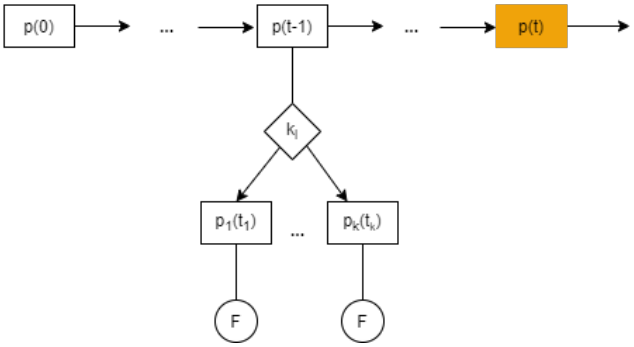
AKO



ONDA



ILI



(c)

6. Kritički osvrt

Jezici koje smo spomenuli nisu jedini koji pokušavaju riješiti problem modeliranja VAS-a. Zahtjevi koji pojedini sustavi imaju i karakteristike koje je potrebno pokriti zahtjevaju razumijevanje jezika i što nam on omogućuje i u kojoj količini. Spomenuli smo jezike poput GBRAM, KAOS i CREWS, uz Agent UML i GPLKTF koje smo pobliže pojasnili.

GBRAM je usmjeren na ciljeve i daje proceduralni vodič za identifikaciju i razvoj ciljeva poznavajući tehnike koje pomažu u njihovoj analizi i jasnoći. Sastoji se od smjernica kojih se treba pridržavati čime ovaj proces čini pouzdanijim i moguće je pratiti zahtjeve odražavajući sljedivost. Ono što mu nedostaje su modeli, formalne oznake i alati koji podržavajući modeliranje za ovu metodu.

i* framework je prilagođen za stjecanje i modeliranje aktera u sustavu i njegovom okruženju, dok ne dopušta potpuni prikaz ograničenja niti predlaže okruženje za modeliranje.

KAOS sadrži ciljeve i zahtjeve IS-a, kao i očekivanja okruženja sustava, sukobe između ciljeva, prepreka, entiteta, itd. Ideja ograničenja korisna je u identificiranju nekih vanjskih problema integriteta čime pridonosi robusnosti sustava. Uspješna implementacija uvelike ovisi o iskustvu programera i koliko je dobro definiran problem koji treba riješiti.

CREWS se temelji na objektnim modelima sustava koji su apstrakcije ključnih značajki različitih kvaliteta domene problema. Koristi modele za generiranje normalnih scenarija, a zatim koristi teorijska i empirijska istraživanja kognitivne znanosti. Potencijalna slabost je njegovo generiranje scenarija usmjereno na domenu. Ako se namjerava scenarijima potvrditi zahtjeve, ti zahtjevi trebaju biti usmjereni na stvaranje scenarija.

Agent UML je pokazao dobre rezultate u predstavljanju funkcionalnih zahtjeva, a također je i dobar alat za komunikaciju sa sudionicima. Koristimo protokol interakcije agenata (AIP) koji predstavlja središnji aspekt za VAS, iako use case dijagram ima ograničenja u bilježenju kvalitativnih aspekata okoliša i interakcija s njim.

Budući da jezik PLKTF ima vrlo složenu semantiku, koristi se grafička metoda GPLKTF. Sintaksa GPLKTF metode je grafička jednostavno zato što su mnogi stručnjaci počeli razmišljati i komunicirati na taj način o temeljnim problem domenama. Složena PLKTF semantika se može jednostavnije izraziti ovim grafičkim jezikom zbog svoje jednostavne razumljivosti. Agent je svjestan svoga stanja i globalnog stanja u sustavu. Ono što ovoj metodi pak ne ide u prilog je što jezik postaje kompliciraniji što je sustav kompliciraniji, dok s druge strane agenti nisu u stanju stupiti u interakciju, kao ni prenijeti znanje jedan drugome, bilo ono naučeno ili zaboravljeno. Također je bitno napomenuti nedostatak alata za modeliranje pomoću GPLKTF te potreba za razumijevanjem PLKTF kako bi se što uspješnije modelirao sustav.

Svi ovi nedostaci pojedinih metoda modeliranja još su uvijek predmeti otvoreni za istraživanje, ali potrebe stvarnih sustava uvelike pomažu u pronalaženju i otklanjanju takvih problema.

7. Zaključak

Kao što smo mogli vidjeti u ovome radu, još uvijek ne postoji općeprihvaćeni način modeliranja višeagentnih sustava. Pomoću formalizacije agenta u višeagentnom sustavu, identificirali smo njegove karakteristike na koje smo obraćali pažnju pri opisivanju pojedinih jezika. Različite perspektive, poput organizacijske, strukturne, funkcionalne i društvenog ponašanja, pomažu nam pri prepoznavanju i modeliranju opisa vanjske i unutarnje perspektive višeagentnih sustava. Pomoću tih perspektiva smo raščlanili zahtjeve modeliranja na definiciju zahtjeva i specifikaciju zahtjeva. Svaka od tih aktivnosti se sastoji od nekoliko modela za analizu i razvoj sustava.

Agent UML je sve raširenij i prihvaćeniji jezik za modeliranje. Prvo, olakšava korisnicima koji su upoznati s objektno orijentiranim razvojem softvera, ali su tek počeli razvijati sustave, da razumiju o čemu se radi u VAS-u i razumiju načela gledanja na sustav kao na društvo, a ne distribuiranu zbirku predmeta. Drugo, vrijeme provedeno na dizajn se može značajno smanjiti. Nadograđujući i prilagođavajući postojeće, dobro razumljive tehnike, nastojimo iskoristiti njihovu zrelost za razvoj modela i metodologije koju će oni koji su upoznati s razvojem lako naučiti i razumjeti. Ovo je važno ako se projektiranjem, implementacijom i održavanjem sustava bave softverski analitičari i inženjeri koji bi uspješno primijenili modeliranje u značajnoj mjeri na komercijalne i industrijske aplikacije.

Popis literature

[1] M. Schatten, „08 - Metodologije,” *nastavni materijali na predmetu Višeagentni sustavi [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin*, 2018.

[2] *Multiagent systems*, Second edition, serija Intelligent robotics and autonomous agents. The MIT Press, 2013., ISBN: 978-0-262-01889-0.

[3] ——, „01 - Uvod,” *nastavni materijali na predmetu Višeagentni sustavi [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin*, 2018.

[4] ——, „03 - Inteligentni agenti,” *nastavni materijali na predmetu Višeagentni sustavi [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin*, 2018.

[5] F. Alkhateeb, E. A. Maghayreh i I. A. Doush, *Multi-Agent Systems - Modeling, Control, Programming, Simulations and Applications*. InTech, 2011., ISBN: 978-953-307-174-9. DOI: 10.5772/611. adresa: http://www.intechopen.com/books/multi-agent-systems-modeling-control-programming-simulations-and-applications.

[6] B. Bauer, J. P. Muller i J. Odell, „Agent UML: A Formalism for specifying Multiagent Software Systems,” *World Scientific Publishing Company*, International Journal of Software Engineering and Knowledge Engineering, sv. 11, str. 25, 2001.

[7] M. Maleković, „GPLKT: A Graphic Method for Multi-Agent Systems,” str. 28,

[8] ——, „Rezoniranje o znanju,”

[9] M. Schatten, „07 - Rezoniranje o znanju,” *nastavni materijali na predmetu Višeagentni sustavi [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin*, 2018.

Popis slika

Popis tablica