MODELOVANIE KOMUNIKÁCIE V MULTIAGENTOVÝCH SYSTÉMOCH

DIPLOMOVÁ PRÁCA

JURAJ MOJÍK

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

KATEDRA APLIKOVANEJ INFORMATIKY

Študijný odbor: Informatika
Vedúci diplomovej práce: RNDr. Martin Takáč

Bratislava 2007

Abstrakt

MOJÍK, Juraj: Modelovanie komunikácie v multiagentových systémoch. [diplomová práca] – Fakulta matematiky, fyziky a informatiky Univerzity Komenského v Bratislave; Katedra aplikovanej informatiky. – Školiteľ: RNDr. Martin Takáč – Bratislava 2007. 76 strán.

Diplomová práca sa zaoberá problematikou multiagentových systémov, konkrétne modelovania komunikácie. Analyzuje súčasný stav v tejto oblasti, uvádza prehľad existujúcich modelov a ich aplikácií. Zisťuje, do akej miery sú tieto modely vhodné na rôzne situácie vyskytujúce sa pri komunikácii v multiagentových systémoch a taktiež nakoľko spĺňajú požiadavky modelárov, ktorí tieto situácie modelujú. Podrobnejšie sa venuje farbeným Petriho sieťam. Vysvetľuje modelovanie komunikácie v multiagentových systémoch pomocou farbených Petriho sietí. K práci je priložený nástroj, ktorý umožňuje pohodlné implementovanie takýchto modelov v multiagentovom prostredí JADE.

Kľúčové slová: multiagentové systémy, modelovanie, komunikácia, farbené Petriho siete, JADE

Predslov

Multiagentové systémy sú zaujímavou oblasťou umelej inteligencie. Mnoho systémov, ktoré sú v súčasnosti prakticky využívané vykazuje vlastnosti typické práve pre multiagentové systémy – mnoho samostatných jednotiek bez centrálneho riadenia, ktoré kooperujú na dosiahnutí spoločného cieľa. Preto rozšírenie znalostí v tejto oblasti môže priniesť zlepšenie fungovania a navrhovania práve takýchto systémov, ktoré, ako sa zdá, sa tešia stále väčšej obľúbenosti.

Preto mi téma spojená s multiagentovými systémami prišla zaujímavá. Obzvlášť, ak sa týka takej kľúčovej oblasti akou je komunikácia, teda „lepidlo“, ktoré drží multiagentový systém pokope. Vedel som však, že táto téma je natoľko rozsiahla, že vyčerpávajúci opis všetkých formalizmov, ktoré sa používajú na modelovanie komunikácie v multiagentových systémoch nebude možné v rozsahu práce uviesť. No ambícia priniesť základný pohľad na túto tému a poukázať na vlastnosti, ktoré sú kľúčové a určujúce pre dobrý komunikačný model bola pre mňa atraktívna.

Praktická časť bola pre mňa výzvou. Vytvoriť nástroj, ktorý umožní namodelovanú situáciu ľahko implementovať je náročná úloha, ktorá samozrejme nezohľadní nároky všetkých programátorov, no práve preto si zachová dostatočnú všeobecnosť. Veľkou pomocou je multiagentové prostredie JADE, ktoré umožňuje abstrahovať od detailov nízkej úrovne a sústrediť sa iba na to, čo je naozaj dôležité.

Téma mojej diplomovej práce je určite veľkou a samostatnou problematikou, ktorá si zaslúži podrobný výskum. Podávam tu čitateľovi jej základy, na ktorých, ako dúfam, sa bude dobre stavať.

Na záver by som sa chcel poďakovať môjmu diplomovému vedúcemu za pomoc a podporu pri písaní tejto práce.

Obsah

Úvod

Keďže táto diplomová práca si kládla za cieľ dve úlohy – nájsť najvhodnejší model komunikácie v multiagentových systémoch a následne ho implementovať - tak aj samotný text môžeme rozdeliť na dve časti – **teoretickú** a **praktickú**.

Prvou kapitolou je **úvod do komunikácie**, kde sa zaoberáme komunikáciou v širších súvislostiach ako je problematika multiagentových systémov.

V druhej kapitole uvádzame niekoľko **existujúcich modelov** – deterministické konečné automaty, Dooleyho grafy, AUML a farbené Petriho siete. Bližšie rozoberáme ich vlastnosti a snažíme sa dopátrať k tomu, čo určuje dobrý komunikačný model, resp. ktoré vlastnosti sú pre komunikačný model dôležité.

V tretej kapitole sa venujeme zvolenému modelu – **farbeným Petriho sieťam**. Začneme ich formálnou definíciou. Potom podrobnejšie opisujeme, ako sú použité na modelovanie základných stavebných kameňov komunikácie ako napr. posielania správy alebo protokolu.

Tým uzatvárame teoretickú časť a pokračujeme praktickou. Štvrtá kapitola obsahuje **opis implementácie farbených Petriho sietí** v JADE, ako aj návod, čo musí programátor spraviť aby mohol túto implementáciu využiť.

Piata kapitola rozoberá **praktické príklady**, ktoré sme vytvorili pre lepšie porozumenie naimplementovaného systému ako aj ilustráciu toho, ako sa z modelu stane fungujúci program.

1 Úvod do komunikácie

Komunikácia, ktorá je nosnou témou tejto práce, je silným nástrojom, ktorý umožňuje prenos informácií. Tento prenos sa deje medzi mnohými typmi entít (ľudia, zvieratá, hmyz či agenty) a mnohými rozličnými spôsobmi (reč, neverbálne spôsoby prejavu alebo napríklad aj feromónové stopy mravcov). Preto, keď hovoríme o komunikácii je dôležité špecifikovať o akej komunikácii hovoríme, teda hlavne aké typy informácií sú komunikované, medzi akými agentmi a s akým výsledkom.

Na komunikáciu sa môžeme pozerať teda ako na sociálnu interakciu, kde najmenej dva interagujúce agenty zdieľajú spoločnú množinu znakov a pravidiel, ktoré vyjadrujú význam týchto symbolov. Najjednoduchším modelom je informácia posielaná od odosielateľa (kódera) k príjemcovi (dekóderovi). V mierne zložitejšom modeli máme navyše aj spätnú väzbu od príjemcu k odosielateľovi, no to si však už vyžaduje symbolickú aktivitu, často na úrovni jazyka. Potom môžeme hovoriť aj o vývoji komunikácie, teda o vývoji procesu porozumenia tomu, čo sa agentovi snažia ostatní povedať.

Komunikácia ako pomenovaná disciplína sa prvý krát nachádza v Sokratovských dialógoch, ktoré sa dajú považovať ako základ aj mnohých iných oblastí, najmä skorých vied a filozofie. Snaha presne zadefinovať komunikáciu ako slovo alebo vednú disciplínu nemusí byť až taká dôležitá, ako pochopenie komunikácie, čo ukázal Ludwig Wittgenstein (1953) vo svojich mnohých definíciách. Niektoré sú všeobecnejšie a ukazujú, že aj zvieratá môžu komunikovať podobne ako ľudia, no iné sú o poznanie striktnejšie a hovoria len o ľuďoch a ľudskej reči.

Komunikácia sa najčastejšie popisuje v troch hlavných rovinách :

1. Obsah
2. Forma
3. Určenie (cieľ)

S výskytom komunikačného šumu sa však tieto tri roviny stávajú nepresnými. V rámci obsahu, ktorý tvorí komunikáciu potom hovoríme skôr o komunikačných aktoch,

ktoré vypovedajú o skúsenosti alebo vedomosti, dávajú radu alebo príkaz či sa pýtajú otázku. Tieto akty majú mnoho foriem vrátane gest (neverbálna komunikácia), písania či reči. Forma závisí na type použitých symbolov. Spoločne forma a obsah komunikácie vytvárajú správy, ktoré sú posielané na miesto určenia. Cieľom môže byť sám odosielateľ, iný človek, alebo iná entita.

Konkrétny príklad komunikácie sa nazýva rečový akt (viac o rečových aktoch nájdete v kapitole 2.3 o Dooleyho grafoch). Rečový akt zvyčajne nasleduje niekoľko spôsobov jeho doručenia. Najčastejší spôsob je dialóg, pri ktorom obe komunikujúce strany sú zapojené do komunikácie a posielajú si navzájom informácie. Sú mnohé formy komunikácie, no dôvod, prečo práve dialóg je považovaný za dobrú formu je ten, že umožňuje jasnejšiu a zrozumiteľnejšiu komunikáciu vďaka spätnej väzbe.

V akademickom svete sa v súčasnosti veľa diskutuje o tom, čo vlastne tvorí komunikáciu. Môžeme povedať, že komunikácia je vlastne proces posielania informácie od jednej entity k druhej, no mnohí vedci túto definíciu považujú za pracovnú a používajú častejšie Lasswelov výrok „kto hovorí čo komu cez aký kanál a s akým výsledkom“ ako základ teórie komunikácie. Keďže však je komunikácia hlboko zakorenená v ľudskom chovaní, mnoho štúdií má problémy oddeliť komunikáciu od ľudského chovania. Keďže teória komunikácie je relatívne mladá vedná disciplína, pričom zahŕňa do svojich poznatkov aj výsledky z iných odvetví ako napríklad z filozofie, psychológie alebo sociológie, nemôžeme ešte očakávať konsenzus medzi odvetviami v tom, čo je vlastne komunikácia.

Je užitočné sa na komunikáciu a následne komunikačnú teóriu pozerať jedným z týchto hľadísk :

Mechanistické : tento pohľad sa zameriava na komunikáciu ako na bezchybný prenos správy od odosielateľa k príjemcovi.

Psychologické : pri tom type pohľadu sa komunikácia pokladá za akt poslania správy príjemcovi, a sústredí sa na pocity a myšlienky, ktoré interpretácia správy v príjemcovi vyvolá.

Sociálne konštruktivistické : považuje komunikáciu za produkt interagujúcich entít, ktoré zdieľajú a vytvárajú význam.

Systemické : komunikácia sa zvažuje z pohľadu prechodu správy systémom, teda ako je správa interpretovaná, prípadne pozmenená ako prechádza systémom.

Prienik konkrétnej teórie s týmito oblasťami vymedzuje povahu komunikácie v danej teórii.

Teórie komunikácie môžu byť študované aj na základe *ontologického*, *epistemologického* a *axiologického* prístupu.

Ontológa sa zaoberá tým, čo je vlastnou povahou toho, čo sa v teórii študuje.

Epistemológia skúma ako teoretik študuje príslušnú oblasť. Štúdium epistemológie si zakladá na tom, že objektívna znalosť je výsledok systematického pohľadu na kauzálne vzťahy javov.

Axiológia rozoberá, ktoré hodnoty viedli autora teórie k jej vytvoreniu.

Komunikácia a aj teórie, ktoré sa ňou zaoberajú majú veľa podôb. V našom prípade budeme skúmať komunikáciu ako prenos informácie medzi agentmi – neživými samostatnými jednotkami. Správy, ktoré si vymieňajú budú mať prevažne jasný význam, ktorý buď agent rozumie a potom dokáže správu spracovať, alebo nie. Nebudeme uvažovať poruchy ani priamo strácanie správ. Zaujíma nás komunikácia ako celok, budeme sa snažiť zachytiť tie najpodstatnejšie aspekty komunikácie do modelu, aby ten čo najvernejšie reprezentoval situáciu, na ktorej je postavený.

2 Modely komunikácie

2.1 Existujúce modely

V teórii multiagentových systémov (ďalej MAS) je schopnosť agentov navzájom komunikovať veľmi dôležitá, ak nie kľúčová. Je to vlastnosť, ktorá z izolovaných autonómnych jednotiek vytvorí komplexný systém, ktorý môže byť hodnotnejší ako súčet vlastností jednotlivých zložiek, ktoré ho tvoria. Preto je samozrejmé, že sa formalizovaniu a modelovaniu komunikácie v MAS venuje zvýšená pozornosť.

Dobre navrhnutý model komunikácie uľahčí navrhovanie multiagentového systému, zlepší zrozumiteľnosť a umožní overiť vlastnosti navrhovaného systému pred jeho uvedením do praxe. Zároveň je neoceniteľný pri implementovaní systému, kde môže výrazne skrátiť čas, ktorý táto fáza zaberie. Taktiež je potrebný pri analyzovaní už existujúcich komunikácií a skúmaní ich vlastností. Preto je veľmi dôležité, ako vyzerá formalizmus na modelovanie komunikácie v MAS a aké typy informácií nám umožňuje zachytiť.

Modely, ktorým bola venovaná zvýšená pozornosť a ktoré sa v mnohých situáciách ukázali ako efektívne, boli založené na deterministických konečných automatoch, Dooleyho grafoch, UML a farbených Petriho sieťach. V nasledujúcich podkapitolách si tieto modely rozoberieme podrobnejšie a ukážeme ako sú použité na modelovanie komunikácie v MAS. Zároveň poukážeme na výhody a nevýhody týchto prístupov.

2.2 Deterministické konečné automaty

Deterministické konečné automaty (ďalej DKA) sú klasickým modelom v teoretickej informatike. Silou tohto modelu je jednoduchosť a výborne spracované teoretické pozadie, ich vlastnosti a správanie sú dobre známe. Zároveň je to vhodný matematický základ, na ktorom možno stavať. Preto nie je prekvapivé, že DKA boli jedným z prvých modelov, ktoré boli použité na formalizovanie komunikácie v MAS.

Tento typ modelu odráža stavovú informáciu, teda informáciu o momentálnom stave systému ako celku. Je samozrejmé, že iba takáto informácia nie je dostačujúca pre komplexné modely, no na navrhovanie či analyzovanie menších systémov sa môže deterministický konečný automat ukázať ako rýchly a účinný modelovací nástroj.

2.2.1. Definícia DKA

Princípom fungovania DKA je čítanie vstupu. Na základe momentálne prečítaného vstupu (vždy sa číta len jeden znak) a stavu, v ktorom sa automat nachádza, sa rozhodne, do ktorého zo stavov sa automat presunie. Pokiaľ automat prečítal všetky znaky a je v jednom zo špeciálnych, tzv. konečných stavov, tak to čo bolo na vstupe sa považuje za platné slovo (patrí do jazyka).

Definícia 1. (Formálne jazyky a automaty, 1998) Deterministický konečný automat A je pätica $(K, \Sigma, \delta, q_0, F)$, kde K je konečná množina stavov, Σ je vstupná abeceda, $q_0 \in K$ je počiatočný stav, $F \subseteq K$ je množina akceptačných (koncových) stavov a $\delta : K \times \Sigma \rightarrow K$ je prechodová funkcia.

Avšak, aby sme mohli s DKA pracovať, potrebujeme ešte nasledovné definície, ktoré špecifikujú správanie tohto teoretického modelu.

Definícia 2. (Formálne jazyky a automaty, 1998) Konfigurácia DKA A je prvok $(q, w) \in K \times \Sigma^*$, kde q je stav automatu a w je zvyšok vstupného slova.

Definícia 3. (Formálne jazyky a automaty, 1998) Krok výpočtu DKA A je relácia $\Rightarrow$ na konfiguráciách definovaná $(q, av) \Rightarrow (p, v) \Leftrightarrow p = \delta(q,a)$.

Definícia 4. (Formálne jazyky a automaty, 1998) Jazyk akceptovaný deterministickým konečným automat A je množina $L(A) = \{ w \in \Sigma^* \mid (q_0, w) \Rightarrow^* (q, \varepsilon), q \in F$, kde $\Rightarrow^*$ je tranzitívny uzáver relácie $\Rightarrow$.

2.2.2 DKA a modelovanie komunikácie

Aby mohli byť DKA použité ako model komunikácie v MAS, je treba jednotlivé abstraktné prvky, ktoré tvoria DKA stotožniť s prvkami (namapovať na prvky), ktoré tvoria multiagentový systém.

Vo všeobecnosti máme dva možné prístupy, ako modelovať komunikáciu pomocou DKA. Pri prvom spôsobe, máme pre každého agenta, prípadne pre každú rolu samostatný automat. Naopak, pri druhom prístupe máme jeden automat pre celý systém. Prvý spôsob využijeme najmä vtedy, ak chceme zdôrazniť stav jedného agenta. Keďže však my chceme modelovať multiagentový systém ako celok, využijeme druhý prístup a budeme modelovať jedným automatom celý systém.

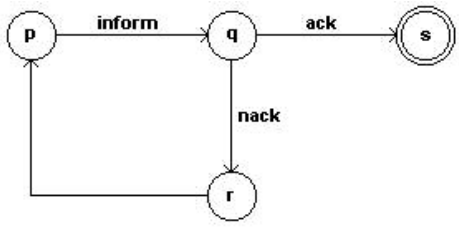
Abeceda, ktorá je vstupom DKA, predstavuje správy, ktoré si agenty v systéme vymieňajú. Jeden symbol abecedy je teda konkrétny typ správy so všetkými náležitosťami (napr. adresát, príjemca...), pričom miera konkrétnosti záleží na tom, ako presne je potrebné daný systém modelovať. Jazyk akceptovaný automatom potom predstavuje postupnosti správ, ktoré sú v systéme posielané od jeho vzniku až po jeho ukončenie.

Stavy automatu predstavujú stav celého multiagentového systému, teda nie stav jedného agenta, ale systému ako celku. Každý stav automatu je teda zložený zo stavov všetkých agentov v systéme, keď sa zmení stav aj keď len jedného agenta, musí sa zmeniť stav celého systému a automatu.

Prechodová funkcia udáva, ako sa zmení stav systému po prijatí správy daného typu.

Príklad 1. Predpokladajme nasledovnú jednoduchú situáciu. Máme dva agenty, pričom jeden sa snaží informovať druhého o niečom. Pošle mu správu a ten, na základe

toho či porozumel alebo nie pošle naspäť správu ack alebo nack. V prípade, že neporozumel, agent znovu skúša vyslať inform. Situáciu, modelovanú pomocou DKA vidíme na nasledujúcom obrázku.



Obrázok 1 : Deterministický konečný automat k príkladu.

Využívame tu štandardnú notáciu pre DKA – krúžky predstavujú stavy, šípky predstavujú prechody, nadpisy nad čiarami predstavujú prijatý typ správy. Stav, ktorý je reprezentovaný dvojitým krúžkom znamená koncový stav. Začíname v stave *p*. Po vyslaní inform prechádzame do stavu *q*. Tu, podľa toho akú správu pošle agent ako odpoveď buď ukončíme činnosť modelu v stave *s*, alebo zopakujeme vyslanie inform tak, že prejdeme do stavu *r* a následne sa dostaneme do stavu *p*. Tento prechod sa udeje bez prijatia správy a teda by sme mohli stavy *r* a *p* stotožniť (to je známy výsledok z teórie formálnych jazykov a automatov). My sme tak neurobili pre väčšiu názornosť.

2.2.3 Praktické použitie DKA v MAS

Modely využívajúce formalizmus DKA majú veľkú výhodu v jednoduchosti. Tento model je veľmi prehľadný a ľahký na pochopenie. Využíva sa najmä ako špecifikácia stavov, v ktorých sa MAS môže nachádzať. Je vhodný najmä na modelovanie menších úloh, kde stavov systému nie je až tak veľa, prípadne na modelovanie menších častí veľkého systému.

S narastaním systému, so zvyšujúcim sa počtom agentov a stavov, v ktorých sa môžu nachádzať sa však z predností tohto modelu môžu stať nedostatky. Pokiaľ by sme ale nehľadeli na narastajúcu komplexnosť systému a z toho vyplývajúcu menšiu prehľadnosť ako na prekážku, veď koniec koncov každý systém sa s narastajúcim počtom agentov

zväčšuje, tak je tu ďalší pomerne dôležitý nedostatok. Keďže tento formalizmus zachytáva iba informáciu o stave systému, neposkytuje nám informáciu o účastníkoch konverzácie. A už vôbec nie o rôznych úlohách, v ktorých môžu účastníci vystupovať. V reálnych MAS väčšina agentov vystupuje nie v jednej ale vo viac roliach (roles). Ak v jednej konverzácii agent vystupuje ako kupujúci, v inej môže byť predajcom. Avšak v tomto type formalizmu, nevieme rozlíšiť ani jednotlivých agentov, ani ich úlohy (role).

2.3 Dooleyho grafy

Prístup načrtnutý deterministickými konečnými automatmi rozširujú modely založené na Dooleyho grafoch, resp. obohatených (Enhanced) Dooleyho grafoch. Tento model sa snaží prekonať nedostatky DKA tým, že sa snaží zachytiť nielen stavovú informáciu, ale aj informáciu o účastníkoch komunikácie a ich roliach.

Dooleyho grafy sú o poznanie zložitejší model ako DKA. Sú vybudované nad Teóriou rečových aktov (Speech Act Theory) a pre pochopenie Dooleyho grafov si v nasledujúcej sekcii stručne načrtneme základy tejto teórie. Až potom pristúpime k samotnej definícii a analýze vlastností Dooleyho grafov.

2.3.1 Úvod do Teórie rečových aktov

Teória rečových aktov stavia (ako aj mnohé ďalšie teórie na poli MAS) na pozorovaní, ktoré ako prvý uviedol J.L. Austin v roku 1962 (Austin, 1975). Tvrdí, že vyjadrenia (utterances), ktoré si agenty vymieňajú nie sú jednoduché výroky (propositions), ktoré môžu byť pravdivé alebo nie, ale je to úsilie (attempt), ktoré môže uspieť alebo zlyhať. To znamená, že jednotlivé vyjadrenia sú vlastne pokusy rečníka (speaker) niečo urobiť, dosiahnuť aby sa niečo stalo, resp. aby bolo niečo urobené.

Veľa snahy sa venuje úsiliu formalizovať a usporiadať jednotlivé takéto pokusy alebo *úkony* (performatives) do hierarchickej štruktúry. Vzhľadom na to, že sa MAS používajú na veľa účelov a v rôznych doménach, tak je to náročná a rozsiahla úloha. Zatiaľ nie je známa celková hierarchia, ktorá by spĺňala všetky požiadavky vo všetkých oblastiach a bola by všeobecne akceptovaná. Posledné revízie hierarchie, ktorú zavádza KQML (Knowledge and Query Manipulation Language) ukazujú, že niektoré z úkonov sú nadbytočné, iné ani nie sú úkonmi a naopak niektoré dôležité úkony chýbajú. Pre naše potreby si uvedieme hierarchiu, ktorú vytvoril P. Cohen a kolektív, resp. jej podčasť určenú pre komerčné domény.

Úkony rozdeľujeme na *Rečové akty* (Speech Acts) a *Ne-rečové akty* (Non-Speech Acts). Rozoberme si najprv *Rečové akty*, lebo tie nás zaujímajú prednostne.

Prvým typom *Rečových aktov* je *Výzva* (Solicit). *Výzva* je pokus rečníka vyvolať v príjemcovi presvedčenie, že rečník chce aby príjemca niečo vykonal. *Výzvu* môžeme ešte jemnejšie rozčleniť na *Otázku* (Question) a *Požiadavku* (Request). *Otázka* je *Výzva*, aby príjemca *Informoval* (Inform) rečníka (odosielateľa) o nejakom výroku. *Požiadavka* je *Výzva*, aby sa príjemca *Zaviazal* (Commit), že sa zaujíma o akciu, ktorú odosielateľ navrhuje.

Druhým typom *Rečových aktov* je *Tvrdenie* (Assert). *Tvrdenie* je pokus rečníka vyvolať v príjemcovi presvedčenie, že rečník verí uvedenému tvrdeniu. Tento typ môžeme taktiež ešte zjemniť. *Informovanie* (Inform) je to isté čo *Tvrdenie*, no navyše ešte aj pokus presvedčiť príjemcu aby uveril v obsah. *Záväzok* (Commit) je tvrdenie, že odosielateľ prijal za svoj trvalý cieľ dosiahnuť niečo. *Odmietnutie* (Refuse) je tvrdenie, že odosielateľ neprijal za svoj trvalý cieľ dosiahnuť niečo.

Medzi *Ne-rečové akty* patria v komerčnej doméne najme *Platba* (Pay) a *Dodanie tovaru* (Ship). Ich význam je jasný z názvu.

2.3.2 Sekvenčné relácie medzi rečovými aktmi

Medzi úspešnými úkonmi, môžeme definovať sekvenčné relácie. Tie sú základom Dooleyho grafov a celého komunikačného modelu postaveného na nich. V. Parunak, ktorý je autorom tohto typu formalizmu, navrhuje štyri základné relácie medzi úspešnými komunikačnými úkonmi : *Odpoveď* (Reply), *Odozva* (Respond), *Vyriešenie* (Resultion) a *Dokončenie* (Completition). Tie si teraz zadefinujeme.

Definícia 5. (Parunak, 1998) Vyjadrenie i je *Odozvou* na vyjadrenie j, ak :

1. Odosielateľ vyjadrenia i (agent S_i) prijal vyjadrenie j
2. Dopad vyjadrenia j na stav agenta S_i spôsobil, že agent S_i poslal vyjadrenie i
3. Neexistuje žiadna séria vyjadrení $k_1...k_n$ taká, že vyjadrenie k_1 je *Odozvou* na vyjadrenie j, vyjadrenie i je *Odozvou* na vyjadrenie k_n a $\forall m>1$ vyjadrenie k_m je *Odozvou* na vyjadrenie k_{m-1} (teda vyjadrenie i je prvé, ktoré vyhovuje 1. a 2.)

Definícia 6. (Parunak, 1998) Vyjadrenie i je *Odpoveďou* na vyjadrenie j, ak vyjadrenie i je najnovšou (poslednou) *Odozvou* na vyjadrenie j, ktorá je zároveň aj adresovaná agentovi S_j.

Rozdiel medzi *Odozvou* a *Odpoveďou* je teda v tom, že *Odpoveď* smeruje priamo k rečníkovi, ktorý spôsobil vyslanie *Odpovede*, no *Odozva* môže smerovať aj k iným agentom. Príkladom na *Odozvu* môže byť, keď kupujúci osloví predajcu, no ten sám nevie splniť požiadavky kupujúceho a preto najprv osloví svojich dodávateľov.

Definície *Vyriešenia* a *Dokončenia* sú viazané na to, aké typy vyjadrení v teórii používame, preto sa môžu líšiť od domény k doméne. Vzhľadom na to, že presné definície si vyžadujú presnejšiu formálnu analýzu ako chceme my ponúknuť v tomto stručnom úvode, budú nasledovné definície trocha neformálne a nepresné, no pre naše účely by mali byť postačujúce.

Definícia 7. *Úkon*, ktorý vykoná adresát *Prosby*, sa nazýva *Vyriešením*, ak

1. Adresát *Informoval* odosielateľa na základe *Otázky*
2. Adresát sa *Zaviazal* alebo *Odmietol* splniť *Požiadavku*
3. Adresát vykonal *Úkon* požadovaný v *Požiadavke*.

Idea tejto definície je jednoduchá. Zaslanie istých typov Rečových Aktov očakáva zo strany adresáta niektoré akcie skôr ako iné. Pokiaľ adresát sa rozhodne reagovať na takýto typ Vyjadrenia, prijíma akési pravidlá komunikácie, ktoré určil rečník. Preto, ak adresát pokračuje v komunikácii, tak je to podľa týchto pravidiel, čo znamenajú, že aj odpoveď bude nejakého typu. Táto odpoveď môže byť formou *Vyjadrenia* (v našom prípade 2. - *Zaviazanie* sa alebo *Odmietnutie*) alebo potom priamym vykonaním *Úkonu* bez ďalšej komunikácie(bod 3. z definície). Takýto typ relácie medzi *Úkonmi* nazývame *Vyriešením*.

Ak niekto vyšle *Výzvu*, je na ostatných aby konali. Ak sa nejaký agent *Zaviaže*, tak predpokladáme, že to bude práve tento agent, kto vykoná požadovaný *Úkon*. Väzba medzi *Zaviazaním* a *Úkonom* však nie je podchytená reláciou *Vyriešenia* a preto zavádzame reláciu *Dokončenia*.

Definícia 8. Úkon sa nazýva *Dokončením*, ak

1. Agent ním splní *Úkon*, na ktorý sa *Zaviazal*
2. Je to *Odmietnutie* a agent ním odstúpi od *Úkonu*, na ktorý sa *Zaviazal*

Teraz, keď máme zadefinované všetky potrebné relácie, môžeme prejsť k definícii Dooleyho grafov.

2.3.3 Definícia Dooleyho grafu

Definícia 9. (Parunak, 1998) Dooleyho graf je generovaný usporiadanou štvoricou <E, P, M, A>, kde

E = {1, 2, …, n} je množina rastúcich indexov, ktoré sú usporiadané (alebo aspoň usporiadateľné), ktoré určujú poradie *Vyjadrení* v konverzácii.

P = $\{p_1, p_2, \ldots, p_m\}$ je množina účastníkov konverzácie.

A = $\{<p_i, p_j, k> : <p_i, k> \in S \wedge <p_j, k> \in R\}$ je množina usporiadaných trojíc, definovaná pomocou dvoch množín usporiadaných dvojíc nad E a P : množiny odosielateľov S = $\{<p_i, k>, p_i \in P \wedge k \in E \wedge$ účastník p_i odoslal k$\}$ a množiny príjemcov P = $\{ <k, p_j>, k \in E \wedge p_j \in P \wedge$ účastník p_j prijal k$\}$. Každá trojica v A tvaru $<p_i, p_j, k>$ (resp. odosielateľ, príjemca, správa) zodpovedá hrane v Dooleyho grafe.

M je relácia na množine S $\cup$ R, pričom prvky sú v relácii $<i,p_a>M<p_b,j>$ práve vtedy, keď platí $p_a = p_b$ a platí aspoň jedna z nasledujúcich podmienok

1. j je *Odpoveďou* na i
2. i je *Odpoveďou* a *Vyriešením* j
3. i je *Odpoveďou* na j a je poslednou *Výpoveďou* v konverzácii
4. i je *Dokončením* k a k je *Odpoveďou* a *Vyriešením* j

Pomocou <E, P, M, A> vytvoríme graf nasledujúcim spôsobom :

Definujme reláciu ekvivalencie N nad S $\cup$ R tak, že najprv položíme N = M a potom na N aplikujeme reflexívno-tranzitívny uzáver. Takto zadefinovaná relácia N

indukuje rozklad V = (S ∪ R) / N množiny S ∪ R. Triedy rozkladu vytvárajú jednotlivé roly a teda vrcholy grafu. Množina A vytvára hrany, pričom každej usporiadanej trojici tvaru <p_1, p_2, k> prislúchajú triedy rozkladu v_1 a v_2 také, že <p_1, k> ∈ v_1 a <k, p_2> ∈ v_2. Takúto hranu vedúcu od vrcholu v_1 k vrcholu v_2 potom označíme k.

Dooleyho grafy, ktoré budú vytvorené v súlade s definíciou, ktorú sme práve uviedli, budú často obsahovať niekoľko komponentov a teda budú nespojité aj keď obsahujú iba jednu konverzáciu. Môže to nastať napríklad vtedy, keď nejaké *Vyjadrenie* vyvolá podkonverzáciu (ako napr. požiadavka na tovar vyvolá konverzáciu s dodávateľom o dostupnosti tovaru), no táto podkonverzácia už nebude pripojená ku zvyšku grafu, pretože agenty v nej vystupujú v iných rolách a prepojenosť na pôvodné *Vyjadrenie* sa stratí (Viď príklad). Toto môže byť v niektorých prípadoch žiadúce a užitočné, no zároveň je to niečo, čo je trochu proti prirodzenej intuícii. Predpokladali by sme, že všetko, čo sa celkovej konverzácie týka je prepojené aj v grafe a teda v jednom komponente. Preto V. Parunak navrhol rozšírenie k Dooleyho grafom (Parunak, 1998).

Toto rozšírenie sa zakladá na relácii *Odozva*. Pozorný čitateľ si isto všimol, že v základnej definícii Dooleyho grafu sa táto relácia nevyužíva. Použijeme ju na pridanie ďalších hrán, no iného typu (aby sme ich odlíšili od hrán, ktoré predstavujú *Výpovede*). Tieto hrany pridáme podľa pravidla uvedeného v nasledujúcej definícii.

Definícia 10. (Parunak, 1998) Obohatený Dooleyho graf získame tak, že k už existujúcemu Dooleyho grafu pridáme neoznačené hrany iného typu pomocou tohto pravidla :

Ak <i, p> ∈ R, <p, j> ∈ S, j je *Odozvou* na i, j nie je *Odpoveďou* na i, <i, p> a <p, j> patria rozličným vrcholom (teda do rozličných tried rozkladu podľa V), ktoré ešte nie sú prepojené pomocou tohto pravidla, pridajme neoznačenú hranu z vrcholu, ktorý obsahuje <i, p> do vrcholu ktorý obsahuje <p, j>.

2.3.4 Príklad

Pre lepšie pochopenie Dooleyho grafov si pomôžeme príkladom z (Parunak, 1996). Nasledujúca tabuľka popisuje správy, ktoré vyslali jednotliví agenti (máme

štyroch agentov – označme ich A,B,C,D), pričom prvý stĺpec označuje sekvenčné číslo správy (poradie), potom nasledujú stĺpce s odosielateľom a príjemcami. Štvrtý stĺpec obsahuje typ rečového aktu, resp. správy či úkonu a prípadne aj obsah správy. Zvyšné stĺpce predstavujú relácie *Odozvy*, *Odpovede*, *Vyriešenia* a *Ukončenia*, pričom číslo, ktoré sa nachádza v riadku príslušnej správy udáva, s ktorou správou je táto správa v relácii.

Por	Odos	Príj	Vyjadrenie	Odozva	Odpoveď	Vyriešenie	Dokončenie
1	A	B,C,D	POŽIADAVKA : Pošlite mi prosím 50 jednotiek tovaru do budúceho štvrtka.				
2	B	C	OTÁZKA : Budeš reagovať na požiadavku od A?	1			
3	C	B	INFORMOVANIE : Áno	2	2	2	
4	B	A	ODMIETNUTIE	3	1	1	
5	C	A	NÁVRH (INFORMOVANIE + POŽIADAVKA) : A čo takto 40 jednotiek tovaru do budúceho piatku?	1	1		
6	A	C	POŽIADAVKA : Pošlite mi prosím 40 jednotiek tovaru do budúceho piatku.	5	5	5	
7	C	A	ZÁVÄZOK : Plánujem vám poslať 40 jednotiek tovaru do budúceho piatku.	6	6	6	
8	D	A	ZÁVÄZOK : Plánujem vám poslať 50 jednotiek tovaru do budúceho štvrtka.	1	1	1	
9	A	C	TVRDENIE : Našiel som lepšieho dodávateľa a nespolieham na váš ZÁVÄZOK	7,8	7		
10	C	A	ODMIETNUTIE : Odstupujem od svojho ZÁVÄZKU	9	9		7
11	D	A	DODANIE TOVARU : Tu je váš tovar, prosím zaplaťte.	1	1		8
12	A	D	TVRDENIE + POŽIADAVKA : Dodali ste o 5 menej, prosím pošlite zvyšok.	11	11		
13	D	A	DODANIE TOVARU : Tu je 5 jednotiek tovaru. Prosím zaplaťte.	12	12	12	
14	A	D	PLATBA	13	13	13	

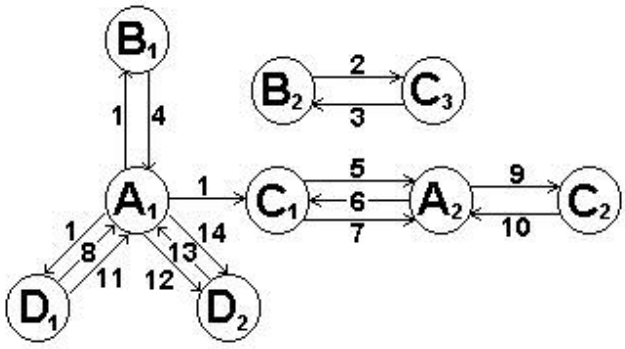
Tabuľka 1 : Konverzácia k príkladu

Táto konverzácia je samozrejme čisto hypotetická. Agent A zadáva *Požiadavku* – chce nakúpiť 50 jednotiek (nejakého, bližšie neurčeného) tovaru do nasledujúceho štvrtku. Túto *Požiadavku* rozošle všetkým zúčastneným agentom. Agenty B a C sú čo sa

týka vybavovania objednávok spojené, teda má zmysel aby reagoval len jeden z nich (napr. preto, lebo majú spoločný sklad). Preto agent B osloví agenta C s *Otázkou* – či reaguje na *Požiadavku* agenta A. Agent C *Informuje* agenta B, že reaguje na túto *Požiadavku* a preto B *Odmietne Požiadavku* agenta A. Agent C navrhne agentovi A inú ponuku – ponúka 40 jednotiek do piatku. Agent A preto pošle novú *Požiadavku* agentovi C. Agent C sa *Zaviaže* poslať 40 jednotiek tovaru agentovi A do piatku. Potom sa však ozve agent D, ktorý sa rozhodne zareagovať na pôvodnú ponuku agenta A a *Zaviaže* sa poslať 50 jednotiek tovaru do štvrtku. Preto agent A pošle *Tvrdenie* agentovi C, v ktorom mu oznámi, že našiel lepšieho dodávateľa. Agent C teda *Odstúpi* od svojho Záväzku. V zvyšnej časti už komunikujú len agenty A a D, pričom agent D najprv *Doručí* len 45 jednotiek tovaru, načo agent A pošle novú požiadavku na zvyšných 5, po ktorých doručení agent A *Zaplatí* agentovi D.

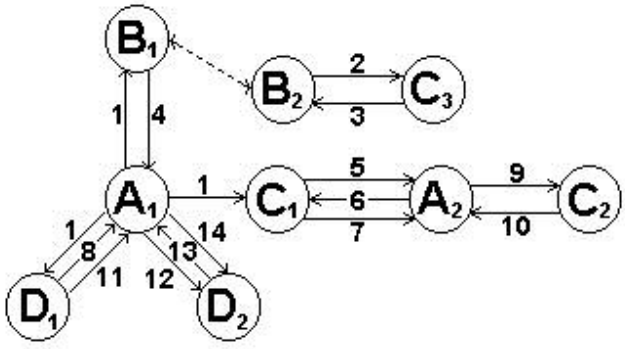
Všimnime si teraz posledné stĺpce tabuľky so sekvenčnými reláciami medzi *Udalosťami*. Je tu vidieť rozdiel medzi *Odozvou* a *Odpoveďou*. Udalosť číslo dva, teda *Otázka* od B pre C, žiadosť reaguje na ponuku A je *Odozvou* na *Udalosť* číslo jedna, nie je však *Odpoveďou* na žiadnu *Udalosť*. Je to preto, že síce *Udalosť* jedna spôsobila, že B vyslal správu číslo dva, no táto správa nie je určená pre A a teda nemôže byť klasifikovaná ako *Odpoveď*. Prvou *Odpoveďou* v našej konverzácii je až udalosť číslo tri, keď agent C *Informuje* agenta B, že na *Požiadavku* agenta A bude reagovať. Všimnime si teraz správu na riadku 10. Touto správou agent C *Odstupuje* od dohody s agentom A. Keďže však predtým už učinil *Záväzok* (riadok 7) tak nemôžeme toto *Odstúpenie* klasifikovať ako *Vyriešenie*. Je to *Ukončenie* a teda to, čo agent C robil na základe žiadosti agenta A sa týmto ukončilo.

Pozrime sa teraz na grafy, ktoré skonštruujeme na základe tejto konverzácie, najprv na Dooleyho graf a potom na obohatený Dooleyho graf.



Obrázok 2 : Dooleyho graf skonštruovaný z príkladu. Je rozdelený na dva komponenty.

Tu si môžeme všimnúť, že komponent s vrcholmi B_2 a C_3 je oddelený od zvyšku grafu. Pritom ale predstavuje súčasť komunikácie, je to rozhovor medzi agentmi B a C, kde sa agent B pýta, či bude agent C reagovať na výzvu od agenta A. Na nasledujúcom obrázku, predstavujúcom obohatený Dooleyho graf už je ale tento komponent začlenený k zvyšku konverzácie.



Obrázok 3 : Obohatený Dooleyho graf skonštruovaný z príkladu. Pridaním informácie z relácie *Odozva* sa graf stane jedným komponentom.

2.3.5 Praktické použitie Dooleyho grafov v MAS

V porovnaní s deterministickými konečnými automatmi sú Dooleyho grafy o poznanie zložitejší model, ako na konštrukciu, tak aj na pochopenie. Zachytávajú nie len informáciu o stave, ale aj o účastníkoch konverzácie a ich roliach. Dôležité je uvedomiť si, že konštrukcia Dooleyho grafu predpokladá nejakú existujúcu konverzáciu,

ktorej záznam máme k dispozícii. Dooleyho grafy potom konštruujeme na základe tohto záznamu. Je samozrejme možné vytvoriť Dooleyho graf aj bez toho, resp. vytvoriť iba abstraktný návrh, no vzhľadom na povahu týchto grafov sa tento prístup veľmi nepoužíva.

Preto sa Dooleyho grafy používajú oveľa častejšie pri analyzovaní už existujúcich komunikácií. Poskytujú celkom dobrý náhľad na interakcie, ktoré sa v komunikácii vyskytujú. Umožňujú dobrý základ pre rôzne miery, ktoré môžu byť použité ako ukazovatele kvality. Jednotlivé vrcholy zase poukazujú na kód, ktorý môže byť znovu použitý v podobnej situácii a predstavuje základ nejakého protokolu.

Pri návrhu systémov sa Dooleyho grafy dajú tiež použiť. Väčšinou sa najprv vyhodnocujú simulácie, kde sa pomocou Dooleyho grafov identifikujú dôležité komponenty a slabé miesta, ktoré sa potom buď použijú alebo znova navrhnú.

2.4 Unified Modeling Language (UML)

Doteraz sme sa zaoberali modelmi, ktoré vychádzajú z teoretických potrieb a ich aplikácie sú taktiež hlavne v teórii. Teraz sa však dostávame k modelu, ktorý vychádza z praxe – z potrieb vypozorovaných v praxi a pri praktickom používaní.

Ďalším rozdielom oproti doterajšiemu prístupu je, že UML nie je len jedna modelovacia technika, ale viacej nástrojov, ktoré umožňujú viac pohľadov na modelovanú problematiku. Každý z týchto pohľadov potom zdôrazňuje iný typ väzieb a vzťahov medzi komponentmi, ktoré tvoria modelovaný systém.

V našom texte sa zaoberáme komunikáciou v MAS, čo je vlastne akýsi podsystém v multiagentových systémoch. Preto nebudeme opisovať všetky nástroje UML, resp. AUML (Agent Unified Modeling Language), ale zoberieme si ich podskupinu, ktorá sa komunikácie priamo týka. V našom zreteli bude najmä UML Sekvenčný diagram.

2.4.1 Agentové rozšírenie UML

UML je jazyk, ktorý sa v súčasnosti v hojnej miere používa pri návrhu a analýze objektovo-orientovaných (OO) systémov. Je podporený veľkým množstvom literatúry, všeobecnou znalosťou a akceptáciou tohto jazyka a existenciou počítačových nástrojov, ktoré umožňujú jednoduché modelovanie a rýchlu implementáciu.

Pri zavádzaní novej technológie do praxe (v našom prípade sú to MAS) sa často vyskytuje požiadavka čo najlepšieho využitia nástrojov technológie bezprostredne predchádzajúcej (teda v tomto prípade OO systémy). To samozrejme vyústilo do veľkého množstva pokusov upraviť už existujúci nástroj, ktorý tak výborne podporoval proces tvorby OO systémov, pre potreby MAS.

Z veľkej množiny rôznych agentových modifikácií UML sme vybrali AUML (Agent Unified Modeling Language)(Odell, Parunak, 2000), pretože má najlepšie rozpracovanú práve problematiku komunikácie, do ktorej zavádza dôležité nové prvky (ako napr. podporu viacerých súbežných konverzácií). Treba však podotknúť, že aj

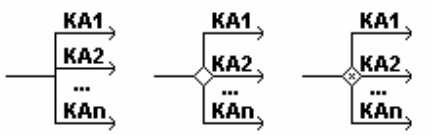
v ostatných navrhovaných rozšíreniach je použitie najsilnejšieho modelovacieho UML aparátu komunikácie - Sekvenčného diagramu – veľmi podobné.

2.4.2 Sekvenčný diagram

Sekvenčný diagram má vo všeobecnosti podobu ako na Obrázku 3. Je to diagram, ktorý sa skladá z jednoduchých geometrických útvarov – obdĺžnikov, čiar a šípok a textu. Navrchu v horizontálnej línii sú obdĺžniky, ktoré predstavujú agentov, resp. ich jednotlivé role. Tie sú uvedené vo vnútri obdĺžnikov v štandardnom formáte Agent/Rola:Trieda. Pod každým takýmto obdĺžnikom je vertikálna čiara. Z nej vychádzajú horizontálne šípky smerom k iným agentom resp. roliam. Predstavujú posielanie správy medzi agentmi v smere šípky. Môže ísť o ľubovolnú správu, alebo môžu predstavovať jeden komunikačný akt, tak ako ho poznáme z Teórie rečových aktov (viď 2.3.1). Typ komunikačného aktu je potom nadpísaný nad šípkou.

Toto je základ, ktorý môžeme nájsť v ľubovolnom agentovom rozšírení UML. AUML ešte pridáva možnosť rozhodovať sa (vybrať jednu alebo niekoľko z možných správ a tie poslať), alebo posielať viac správ naraz (možnosť súbežného posielania viacerých správ). Tieto možnosti sú naznačené na Obrázku 4 . Pokiaľ z jedného miesta vychádza viac šípok, tak to znamená že tieto správy boli poslané súčasne. Pokiaľ sa pri čiare nachádza prázdny symbol kosoštvorca, tak to znamená rozhodovanie. Niekoľko (ľubovoľný počet) správ sa na základe rozhodovacieho mechanizmu pošle súčasne v daný moment. Poslednou možnosťou je kosoštvorec so symbolom x vo vnútri, čo predstavuje exkluzívne alebo (exclusive or), a teda sa vyberie práve jedna z možností.

Obrázok 3 : Základná štruktúra sekvenčného diagramu

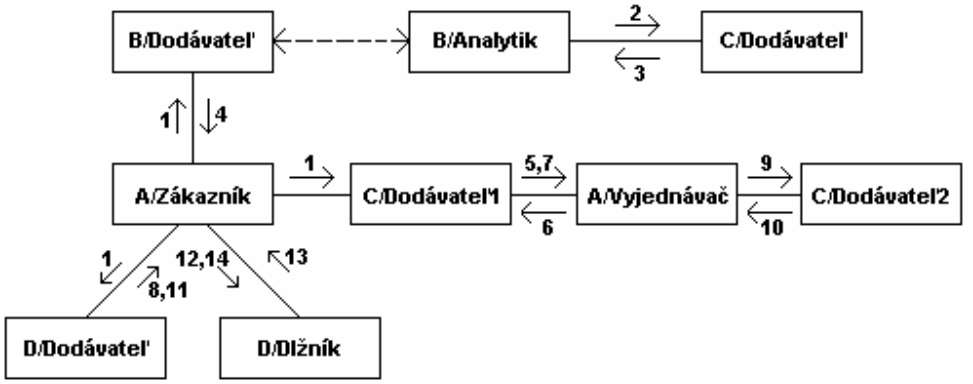


Obrázok 4 : Rôzne spôsoby zaznamenania súbežnosti a rozhodovania.

Sekvenčný diagram znázorňuje priebeh komunikácie ako sekvenciu správ vymenených medzi agentmi. Vertikálne čiary pod jednotlivými rolami vyjadrujú sekvenčnosť v zmysle, že tie udalosti (poslania či doručenia správ), ktoré sú vyššie musia nastať skôr, ako udalosti pod nimi. Nie je to však sekvenčnosť absolútna, neplatí to na celej čiare (v UML diagrame) a už vôbec nie medzi agentmi. Obdĺžniky, ktoré sa nachádzajú na týchto vertikálnych čiarach vymedzujú platnosť sekvenčnosti. Tie udalosti, ktoré sa nachádzajú vo vnútri jedného obdĺžnika (šípky ktoré z tohto obdĺžnika vychádzajú alebo vchádzajú), sú usporiadané sekvenčne za sebou. Neplatí to však medzi šípkami v rôznych obdĺžnikoch.

2.4.3 Kolaboračný diagram

Teraz si predstavíme typ AUML diagramu, ktorý je zaujímavý tým, že je izomorfný (a teda zameniteľný) s Dooleyho grafom. V Kolaboračnom diagrame, na rozdiel od sekvenčného diagramu, môžu byť agenty, resp. role umiestnené hocikde a nie iba navrchu diagramu v horizontálnej línii. Udalosti sú v tomto diagrame na rozdiel od sekvenčného diagramu očíslované. Diagram je opäť teda tvorený obdĺžnikmi, ktoré predstavujú agenty a ich role. Tie sú potom spojené čiarami, ktoré predstavujú tok správ medzi daným rolami agentov. Nad a pod týmito čiarami sú šípky a čísla, reprezentujúce jednotlivé udalosti. Ďalší, prerušovaný typ čiary reprezentuje zmenu rolí. Príklad Kolaboračného diagramu môžeme vidieť na Obrázku 5. Predstavuje situáciu z príkladu, ktorý sme modelovali v sekcii o Dooleyho grafoch. Podobnosť s Dooleyho grafom je skutočne nápadná.



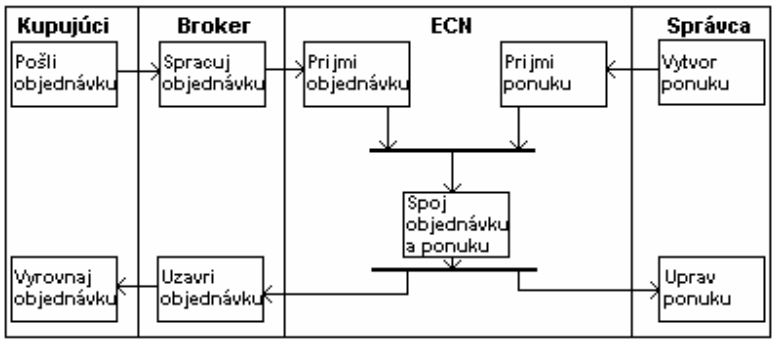
Obrázok 5 : Kolaboračný diagram

Zaujímavé je, že sémanticky je Kolaboračný diagram ekvivalentný so Sekvenčným diagramom. Kolaboračný diagram sme však uviedli práve kvôli jeho zjavnej izomorfnosti s Dooleyho grafmi.

2.4.4 Ďalšie typy diagramov – Diagram aktivít a Stavový diagram

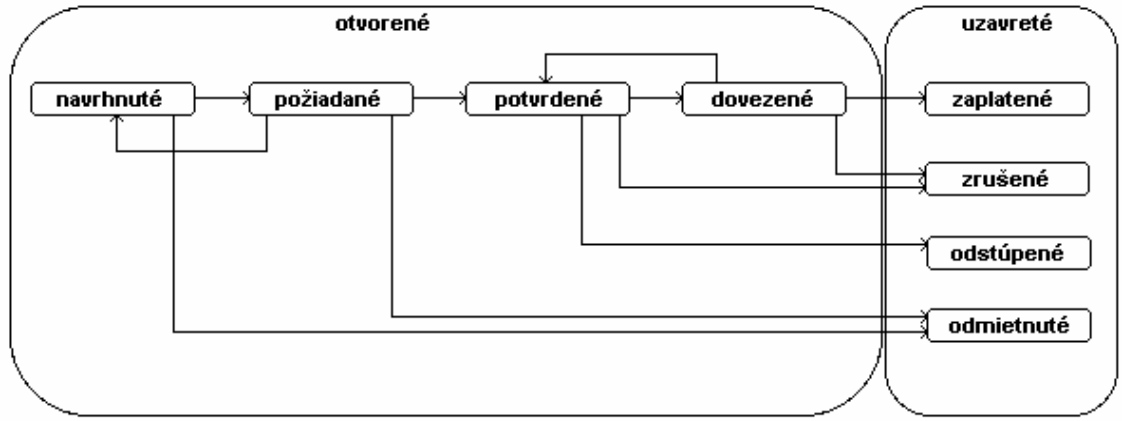
Na čo najlepšie modelovanie komunikačných interakcií a protokolov je treba často zachytiť aj ďalšie typy informácií, než aké zobrazujú Sekvenčný či Kolaboračný diagram. Sú to napr. operácie a udalosti alebo interné stavy agenta, do ktorých sa dostane po prijatí správ. Na zachytenie týchto typov informácií nám slúžia Diagram Aktivít a Stavový diagram, pričom Diagram aktivít zobrazuje procesy a Stavový diagram, ako už názov napovedá, slúži na zobrazenie stavov agenta. Tieto diagramy sú však už mimo nášho hlavného záujmu, preto iba uvedieme krátke príklady ale nebudeme ich detailne popisovať.

Na Obrázku 6 je príklad na Diagram aktivít (prevzatý z (Odell, Parunak, 2000)). Zobrazuje, ako môže byť realizovaná elektronická transakcia. Jednotlivé udalosti (ako napr. uskutočnenie objednávky) spúšťajú procesy, ktoré vyúsťujú do ďalších udalostí. Nakoniec je vykonaná transakcia a sú o nej oboznámení všetci zúčastnení.



Obrázok 6 : Diagram aktivít

Obrázok 7 zobrazuje Stavový diagram (príklad opäť prevzatý z (Odell, Parunak, 2000)). Stavy sú rozdelené na dva typy, pričom uzavreté stavy zodpovedajú vlastne finálnym stavom. Medzi jednotlivými stavmi sa prechádza po prijatí správy zodpovedajúceho typu.



Obrázok 7 : Stavový diagram

Diagramy týchto typov sú užitočné pre čo najpresnejšiu špecifikáciu modelovanej komunikácie, no je to najmä Sekvenčný, resp. Kolaboračný diagram, ktorý zobrazuje nosnú časť informácie o komunikácii. Pri modelovaní komunikácie pomocou AUML si vystačíme aj bez informácie, ktorú poskytuje Stavový diagram, alebo Diagram aktivít, no bez Sekvenčného diagramu a teda prislúchajúcej informácie o agentoch, ich roliach a sekvenciách správ, ktoré si vymieňajú nedokážeme komunikáciu zachytiť. Preto považujeme Sekvenčný, resp. Kolaboračný diagram za kľúčový pri modelovaní komunikácie.

2.4.5 Praktické použitie UML v MAS

Rozšírenia UML nám ponúkajú mnoho spôsobov, ako modelovať nielen komunikáciu, ale aj ostatné oblasti, či dokonca celé MAS. Vzhľadom na to, že tieto rozšírenia vznikli z praktických potrieb sú tieto modely ľahko použiteľné pri praktických úlohách, ako implementácia systému, znovupoužitie častí či počítačové navrhovanie (pomocou programov). Zároveň z toho vyplýva aj ich pomerne ľahká pochopiteľnosť, diagramy sú čitateľné aj keď ich študuje niekto, kto sa ešte s UML nestretol, čo sa rozhodne nedá povedať napr. o Dooleyho grafoch. Keďže sa nejedná o jedinú modelovaciu techniku, je ľahké zaradiť model komunikácie do modelu systému ako celku alebo naopak modelovať jednotlivé časti komunikácie na jemnejšej úrovni.

Sekvenčný diagram je prehľadný zápis interakcií medzi agentmi. Oproti Dooleyho grafom umožňuje ešte aj zápis súbežnej komunikácie. Dooleyho grafy si vyžadovali, aby sa dali jednotlivé udalosti usporiadať a preto vyjadrenie skutočnej súbežnosti by mohlo byť pre Dooleyho grafy problém. Keďže Dooleyho grafy konštruujeme už z existujúcich komunikácií, tak princípy rozhodovania a voľby v nich vôbec neboli zahrnuté, čo opäť Sekvenčné diagramy umožňujú. Dooleyho grafy vieme prehľadne vyjadriť vo forme Kolaboračného diagramu, čo naozaj predurčuje tento typ formalizmu na častejšie využitie v praxi.

Niekedy ale nemusí byť výhodné, mať niekoľko diagramov pre jednu komunikáciu. Predsa len, ak niečo súvisí s danou konverzáciou, mal by to model odrážať. Nie je potom nutné udržovať veľa modelov pre jeden systém, čím sa vyhneme prípadným nekonzistenciám.

Nedostatkom je aj to, že v Sekvenčnom diagrame nevidíme momentálny stav systému, resp. nevieme na ktorom mieste sa nachádzame. Podľa tohto diagramu môžeme simulovať, no model sám neodráža, ako momentálna simulácia prebieha.

2.5 Farbené Petriho siete

Farbené Petriho siete (Coloured Petri Nets) sú matematickým modelom, ktorý sa používa na popisovanie distribuovaných a súbežných procesov. Preto nie je vôbec prekvapivé, že sa Farbené Petriho siete javia ako zaujímavý model pre oblasť multiagentových systémov, ktorým sú vlastnosti ako distribuovanosť a súbežnosť vlastné.

Oproti doteraz rozoberaným modelom (DKA, Dooleyho grafy a UML) majú modely založené na Petriho sieťach niekoľko zaujímavých nových vlastností. Je to napríklad možnosť posielať správy súbežne na viacerých miestach v systéme, či priame zobrazenie akejsi stratégie agenta (podľa čoho si volí niektorú z možností) v modeli.

Keďže podrobnému rozboru farbených Petriho sietí sa venuje celá nasledujúca kapitola, v tejto podkapitole si tento model nezadefinujeme presne a iba naznačíme, ako sa využíva na modelovanie komunikácie v MAS, aby sme mohli porovnať jeho vlastnosti s ostatnými uvedenými modelmi. Presnú definíciu uvedieme až v nasledujúcej kapitole.

2.5.1 Úvod do farbených Petriho sietí

Farbené Petriho siete sú rozšírením Petriho sietí.

Definícia 11. (Nowostawski, 2001) Petriho sieť je pätica (P, T, I, O, M_0), kde P je množina miest (places), T je množina prechodov (transitions), I a O sú vstupná a výstupná funkcia (hrany), ktoré zobrazujú množinu miest na množinu prechodov a množinu prechodov na množinu miest, M_0 je označenie, vektor ktorý charakterizuje iniciačný stav systému, pričom udáva počet značiek (tokenov) v každom mieste siete.

Okrem sieťovej štruktúry Petriho siete obsahujú aj pravidlá, ktoré popisujú za akých podmienok prechody prenesú značky zo vstupných miest na výstupné miesta. Tieto podmienky sa odlišujú pri rôznych typoch Petriho Sietí.

Príklad 2 : Váhované Petriho siete

Ak majú hrany v Petriho sieti priradené čísla (teda sú tzv. váhované), potom môže prechod „páliť“ (preniesť značky), ak

1. Počet značiek v každom vstupnom mieste je väčší ako váha, priradená hranám ktoré spájajú toto miesto s prechodom
2. Súčet počet značiek nachádzajúcich sa vo výstupnom mieste a váhy hrany je menší alebo rovný ako je kapacita miesta

Ak prechod „páli“, tak

1. Počet značiek na každom vstupnom mieste je znížený o váhu hrany
2. Počet značiek na každom výstupnom mieste je zvýšený o váhu hrany

Vo farbených Petriho sieťach sú oproti Petriho sieťam jednotlivé značky rozlíšené. V klasických Petriho sieťach sú všetky značky rovnaké – majú rovnakú „farbu“ a teda sú od seba nerozlíšiteľné, sú medzi sebou zameniteľné. No vo farbených Petriho sieťach môžu byť značky rôznych typov – teda akoby mali rôzne „farby“. Väčšinou tieto typy zodpovedajú ľubovolným typom z programovacích jazykov. Toto jemnejšie rozčlenenie umožňuje lepšie pochopenie práce modelu.

Farbené Petriho siete sú teda štruktúrou podobné Petriho sieťam. Miesta, na ktorých sa ukladajú značky, sú spojené s prechodmi, ktoré za istých podmienok transportujú značky z miest, ktoré do nich vstupujú na miesta, ktoré z nich vystupujú. Okrem toho farbené Petriho siete obsahujú aj množinu dátových deklarácií, ktoré udávajú, aké typy značiek sa budú v sieti nachádzať. Navyše sú farbené Petriho siete rozšírené ešte o ďalšie elementy, ktoré umožňujú efektívnejšie riadiť tok značiek (stráže a pod.).

Neformálne môžeme povedať, že farbené Petriho siete sa skladajú z nasledovných troch komponentov :

1. sieťová štruktúra - miesta, prechody a hrany tak ako v Petriho sieťach
2. množina dátových deklarácií
3. množina popisov siete (výrazy nad hranami, stráže a inicializácie miest)

2.5.2 Farbené Petriho siete a modelovanie komunikácie

Aby sme mohli využiť Petriho siete na modelovanie komunikácie, je nutné zadefinovať sémantiku prvkov, ktoré budeme používať a teda vlastne „namapovať“ (zobraziť) elementy, ktoré tvoria farbené Petriho siete na časti, z ktorých sa skladá komunikačný podsystém MAS.

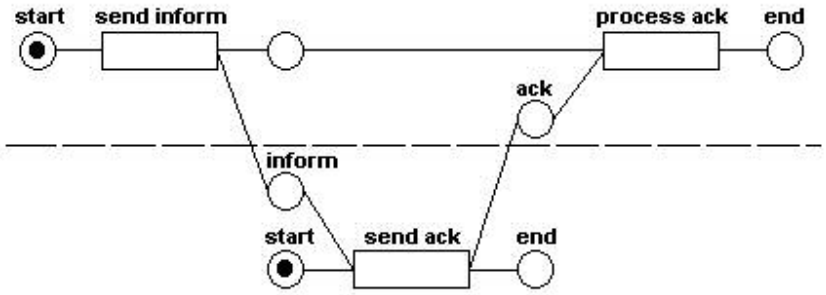
Pri zovšeobecnenom prístupe môžeme povedať, že značky reprezentujú správy, hrany reprezentujú posielanie a doručovanie správ a prechody slúžia na spracovanie správ. Miesta nebudeme zobrazovať na nič konkrétne, obsahujú vlastne len správy, ktoré sú momentálne v systéme.

Roly budú predstavovať jednotlivé podsiete, pričom ich môžeme pri grafickej reprezentácii oddeliť horizontálnymi prerušovanými čiarami. Hrany, ktoré potom prechádzajú cez prerušované čiary predstavujú fyzické prenášanie správ, teda proces vyslania a prijatia správy v MAS.

Existuje vždy práve jeden účastník konverzácie, ktorého rola konverzáciu začína vyslaním prvej správy. Je to potom práve táto rola, ktorá má štartovacie miesto (miesto Štart), ktoré umožní prvému prechodu, aby „pálil“.

Konverzáciou je potom celá Petriho sieť, ktorá pozostáva z podsietí (teda vlastne protokolov, resp. rôl), pričom aspoň jedna z nich má miesto Štart (iniciátor) a je prepojená s ostatnými účastníkmi konverzácie. Momentálnemu stavu konverzácie zodpovedá momentálne rozmiestnenie (distribúcia) značiek v sieti. To, akým spôsobom sa agent rozhoduje, je v modeli zachytené pomocou popisov a stráží, pričom sa tento spôsob môže meniť dynamicky podľa priebehu konverzácie.

Farbené Petriho siete nám jednoduchým spôsobom umožňujú skladanie väčších modelov z menších sietí. A keďže aj v komunikačných protokoloch sa často stáva, že zložitejšie protokoly vznikajú zložením jednoduchších, tak túto vlastnosť pri modelovaní vieme oceniť a naplno využiť.



Obrázok 8 : Príklad Petriho siete.

Príklad 3 : Na obrázku 8 môžeme vidieť Petriho sieť. Pri štandardnej notácii zobrazujeme miesta ako kruhy a prechody ako obdĺžniky. Čierne krúžky predstavujú značky. V tomto príklade máme dvoch agentov, ktorí sú pre väčšiu čitateľnosť oddelení prerušovanou čiarou. Horný agent sa snaží spodného agenta o niečom informovať. Vidíme, že oba agenty majú na začiatku značky iba v mieste označenom ako *start*. Horný agent môže začať, iba jeho prechod send inform môže páliť. Ten pošle správu *inform* - umiestni značku na miesto *inform*, čo spôsobí, že môže páliť prechod *send ack*. Tento umiestni okrem značky na miesto *end* aj značku na miesto *ack* – teda pošle správu *ack* (všimnime si, že hrana prešla cez prerušovanú čiaru). Nakoniec môže páliť aj posledný prechod *process ack*, ktorý predstavuje spracovanie správy *ack*.

2.5.3 Praktické použitie farbených Petriho Sietí v MAS

Farbené Petriho siete sú vhodné na modelovanie distribuovanosti – zrozumiteľná grafická reprezentácia je podporená formálnym matematickým základom. Zároveň majú mnoho dobrých vlastností, dokážu modelovať konverzácie zložené z jedného či viacerých protokolov, môžu byť konštruované dynamicky, zobrazujú priebeh komunikácie, rozlišujú jednotlivých účastníkov a podporujú súbežnosť. Jednou z ich najväčších výhod je schopnosť zobrazovať aj mechanizmus, podľa ktorého sa agenty rozhodujú, pričom touto vlastnosťou neoplýva žiadny z predchádzajúcich modelov.

Tento model sa nám zdá najvyhovujúcejší na modelovanie komunikácie. Poskytuje všetky informácie ako ostatné spomínané modely a ešte k tomu pridáva aj

ďalšie, nové. Je to model, ktorý má dobrý teoretický základ a je použiteľný v teórii, no nie je odtrhnutý od praxe a vyhovuje aj mnohým praktickým požiadavkám. Dá sa zostrojiť už z existujúcej komunikácie, ale zároveň aj ako abstraktný návrh ešte neexistujúcej komunikácie.

Farbeným Petriho sieťam je venovaná nasledujúca kapitola, v ktorej sú formálne zadefinované a podrobnejšie rozobraté.

2.6 Zhrnutie

Na predchádzajúcich stranách sme predstavili niekoľko typov formalizmov, ktoré sa používajú na modelovanie komunikácie. Tieto modely vychádzali z rôznych základov, boli postavené na odlišných princípoch a mali rozličné vlastnosti. Každý z nich bol na niektoré situácie viac na iné menej vhodný.

Popri tomto prehľade sme identifikovali nasledujúce vlastnosti ako dôležité pre komunikačný model : koľko rovín informácie dokáže zachytiť (stavová resp. účastnícka), či dokáže modelovať protokol/y, či sa dá vyjadriť súbežnosť, či dokáže zobraziť priebeh komunikácie a či sa dá vyjadriť stratégia (zámer) agenta.

Stavová informácia reprezentuje vnútorné stavy agentov v multiagentovom systéme, resp. dopady jednotlivých správ či komunikačných krokov na vnútorný stav agenta. Táto informácia najčastejšie korešponduje s históriou komunikácie, teda jednotlivé správy od ostatných agentov spôsobili zmenu stavu agenta a naopak stav agenta zodpovedá nejakej predchádzajúcej komunikačnej sekvencii. Bez tejto informačnej roviny nevieme v modeli reprezentovať dopady predchádzajúcej komunikácie na agentove súčasné rozhodovanie a správanie. Aj preto nám obohatené Dooleyho grafy ani sekvenčný diagram nepripadali vhodné.

Informácia o účastníkoch komunikácie je dôležitá najmä pri paralelných konverzáciách, teda ak jeden agent vystupuje vo viacerých rolách, ktoré sú aktívne (komunikujú) súčasne. Vtedy je vhodné rozlíšiť, ktorá časť komunikácie prislúcha ktorej role. Je možné budovať modely aj bez tejto informácie, no nie je to vhodné, obzvlášť pri väčších počtoch agentov a rôl, kde sa potom neúmerne zvyšuje komplexnosť modelu. Zavedenie rozlíšenia rôl zvyšuje zrozumiteľnosť, zprehľadňuje a zjednodušuje prácu s modelom. Je to práve absencia tejto roviny informácie, ktorá veľmi znevýhodňuje deterministické konečné automaty pre použite na rozsiahlejšie konverzácie.

Schopnosť modelovať protokol má každý spomínaný modelovací formalizmus. Považujeme to však za kľúčovú vlastnosť a preto ju v tomto prehľade uvádzame. Schopnosť zaznamenať postupnosť správ, ktoré tvoria vopred dohodnutý protokol je

podľa nás vlastnosť, ktorá oddeľuje ľubovolné modely od skutočne použiteľných modelov komunikácie. Bez tejto schopnosti nedokáže modelovací formalizmus zachytiť fundamentálnu rovinu dohody, ktorú predstavuje protokol. Je potom nepoužiteľný, nedokáže zachytiť postupnosť odoslaných správ a očakávaných odpovedí, čo predstavuje základ drvivej väčšiny konverzácií.

Súbežnosť v komunikácii predstavujú udalosti, ktoré nastávajú súčasne. Niektoré komunikačné modely si však vyžadujú, aby všetky udalosti nastali v rôznych časoch, alebo aby sa aspoň dali do rôznych časov umiestniť bez straty výpovednej hodnoty. Užitočnejšie sú tie modely, pre ktoré je súbežnosť prirodzená, ako napríklad farbené Petriho Siete. Nie je potom nutné v situáciách, pre ktoré je súčasné nastatie udalostí prirodzené hľadať riešenia, ako takú situáciu ošetriť.

Priebeh komunikácie zobrazujú formalizmy rôzne. Dooleyho grafy ho reprezentujú pomocou sekvenčných čísel pri jednotlivých správach, ktoré udávajú poradie v ktorom jednotlivé komunikačné kroky nastali. V sekvenčnom diagrame zase správy, ktoré sa nachádzajú nižšie boli odoslané alebo prijaté neskôr ako správy na začiatku diagramu. Najpraktickejší a asi aj najnázornejší je prístup farbených Petriho sietí, kde presuny jednotlivých značiek názorne zobrazujú výmeny správ. Čím je zobrazenie priebehu komunikácie názornejšie, tým užitočnejší je model napríklad pri ladení a hľadaní nedostatkov či chýb.

Schopnosť vyjadriť stratégiu (zámer) agenta je vlastnosť modelu, ktorá sa vyskytuje len pri farbených Petriho sieťach. Rozumieme pod ňou schopnosť zachytiť príčiny resp. dôvody, ktorými sa agent riadi pri svojich rozhodnutiach (ako napr. prečo sa rozhodne na nejakú požiadavku reagovať a na inú nie). Táto vlastnosť formalizmu je dôležitá pre celistvé pochopenie komunikácie a spolu so stavovou a účastníckou informáciou vytvára podrobný obraz modelovanej situácie, ktorý je neoceniteľný najmä pri praktickom použití modelu a implementácii. Táto informácia často pri formalizmoch absentovala a komunikácia tak ostávala oddelená od motivácií, ktoré ju riadia.

	Konečné Automaty	Obohatené Dooleyho Grafy	AUML – Sekvenčný diagram	Farbené Petriho Siete
Stavová info.	+			+
Účastnícka info.		+	+	+
Protokol	+	+	+	+
Súbežnosť			+	+
Priebeh kom.		+	+	+
Stratégia (zámer)				+

Tabuľka 2 : Prehľad vlastností komunikačných modelov

Modely založené na deterministických konečných automatoch sú zaujímavé svojou jednoduchosťou. Známy formálny a matematický základ umožňuje dobré využitie najmä v teórii. V praxi sa používajú najmä pre menšie modely s malým počtom aktérov a správ na špecifikovanie stavov, ktorými systém prechádza. Jeho veľkou nevýhodou je zachytenie len stavovej informácie a nezachytenie účastníckej a taktiež menšia prehľadnosť najmä pri väčšom počte účastníkov komunikácie. Aj keď výpočet konečného automatu reprezentuje priebeh komunikácie, je oddelený od grafickej reprezentácie, čo znižuje čitateľnosť, jednoduchosť a použiteľnosť. Súbežnosť ani stratégiu v tomto modeli nie je možné zachytiť kvôli architektúre tohto formalizmu. Tento model však zaostáva oproti ostatným spomínaným formalizmom najmä pre absenciu účastníckej informácie.

Obohatené Dooleyho grafy sú nadstavbou deterministických konečných automatov. Sú obľúbené v teoretickej oblasti, pretože využívajú teóriu rečových aktov, no v praxi sa veľmi nepoužívajú. Oproti deterministickým konečným automatom vymenili stavovú informáciu za informáciu o účastníkoch a ich roliach. Vznikajú už z existujúcich komunikácií a to pomerne ťažkopádnym a komplikovaným spôsobom, ktorý si vyžaduje usporiadanosť rečových aktov (poslaní správ), čo má za následok nemožnosť zachytenia súbežných udalostí. Chýbajúca stavová informácia znemožňuje komplexný pohľad na spracovanie komunikácie, resp. na dopady jednotlivých jej krokov. Preto je tento model naozaj vhodnejší pri teoretickom prístupe a pri štúdiu rečových aktov.

Sekvenčný diagram z agentového rozšírenia UML je silným modelovacím formalizmom, ktorý má veľkú podporu softwarových nástrojov, dobrú všeobecnú rozšírenosť a podporu v priemyselných odvetviach. V UML je časté použitie viacerých diagramov na jednu situáciu, aby sa zobrazili rôzne aspekty a roviny modelu. Sekvenčný diagram je najužitočnejší pre modelovanie komunikácie, zobrazuje výmeny správ medzi agentmi. Nevýhodou tohto pomerne efektívneho modelovacieho nástroja je hlavne chýbajúca stavová informácia, ktorá však môže byť zachytená iným typom diagramu. Avšak nutnosť niekoľko krát modelovať tú istú situáciu je oproti ostatným spomínaným modelom skôr nevýhodou ako výhodou. V niektorých praktických situáciách môže byť napriek tomu použitie agentového rozšírenia vhodnejšie (ako napr. tam, kde sa na multiagentové systémy prešlo z objektových systémov, ktoré boli modelované v UML), ale za iných okolností majú farbené Petriho siete výhodnejšie vlastnosti.

Z nášho prehľadu vychádzajú najlepšie farbené Petriho siete. Zachytávajú účastnícku (oddelenie rôl v sieti) aj stavovú (rozloženie značiek v jednej role) informáciu. Súbežnosť je pre farbené Petriho siete prirodzená vlastnosť, priebeh komunikácie je zobrazený veľmi prirodzeným a názorným spôsobom (presuny značiek v sieti). Vďaka svojej konštrukcii pridávajú možnosť zachytiť aj stratégiu či zámer agenta v podobe hranových popisov a stráží. Navyše je to všetko v rámci jedného modelu a diagramu. Samozrejme, pri práci s malými konverzáciami je výhodnejšie použiť konečné automaty, pri zvýšenom dôraze na teórii rečových aktov v pozadí sa oplatí viac Dooleyho graf a pri zavedenej práci s UML nástrojmi je výhodné použiť sekvenčný diagram, no vo všeobecnosti majú Petriho siete najuniverzálnejšie použitie.

3 Farbené Petriho siete

3.1 Formálna definícia farbených Petriho sietí

V predchádzajúcej kapitole sme si farbené Petriho siete uviedli len neformálne, preto teraz zavedieme formálnu matematickú definíciu. Bude nám slúžiť ako základ, ako presná špecifikácia modelu, s ktorým chceme pracovať. farbená Petriho sieť je tu (podľa (Jensen, 1994)) uvedená ako n-tica, no v praxi sa stretávame skôr s diagramami. Diagramy sú vhodné tam, kde chceme pracovať s nejakou konkrétnou sieťou, konkrétnym modelom. N-tice sú zasa vhodnejšie pri dokazovaní a analýze vlastností celej triedy (alebo podtriedy) farbených Petriho sietí. Je samozrejme možné prevádzať diagramy na n-tice a naopak, no prakticky to nie je úplne triviálna úloha.

Pred definíciou farbených Petriho sietí si však (kvôli matematickej korektnosti) musíme ešte zadefinovať niekoľko ďalších pojmov. Začnime definíciou multi-množiny. K zavedeniu tohto pojmu nás núti napríklad fakt, že na jednom mieste v sieti môžeme mať viac značiek rovnakého typu, na čo je však bežná množina nevhodná.

Definícia 12. Multi-množinou m nad neprázdnou množinou S nazývame funkciu $m : S \rightarrow N$ (N je množina prirodzených čísel), kde m udáva početnosť prvkov množiny S v multi-množine m. Reprezentujeme ju formálnym súčtom

$$\sum_s m(s)`s, \forall s \in S$$

Zápis m(s)`s znamená, že v multi-množine m sa prvok s vyskytuje m(s)-krát. Suma cez všetky prvky $s \in S$ udáva početnosť každého prvku z množiny S v multi-množine m.

Ako S_{MS} označujeme množinu všetkých multi-množín nad S. Hodnoty funkcie m $\{m(s) \mid s \in S\}$ nazývame koeficienty multi-množiny m. Hovoríme, že prvok s patrí do multi-množiny m práve vtedy, keď $m(s) \neq 0$ a označujeme $s \in m$.

Pre multi-množiny zavádzame taktiež pojem prázdnej multi-množiny $\varnothing$ obdobne, ako poznáme prázdnu množinu.

Definícia 13. (Jensen, 1994) Súčet, skalárny súčin, porovnanie a veľkosť multi-množiny definujeme nasledovne :

1. $m1 + m2 = \sum_s (m1(s) + m2(s))`s, \forall\ s \in S$ (súčet)
2. $n * m = \sum_s (n * m(s))`s, \forall\ s \in S$, pričom $n \in N$ (skalárny súčin)
3. $m1 \neq m2,\ \exists s \in S : m1(s) \neq m2(s)$ (porovnanie)
   $m1 \leq m2, \forall s \in S : m1(s) \leq m2(s)$
4. $|m| = \sum_s m(s), \forall\ s \in S$ (veľkosť)

Pokiaľ platí, že $m = \infty$ hovoríme, že m je nekonečná, inak je m konečná. Ak platí $m1 \leq m2$ tak môžeme zadefinovať aj rozdiel

5. $m2 - m1 = \sum_s (m2(s) - m1(s))`s, \forall\ s \in S$ (rozdiel)

V neformálnom opise farbených Petriho sietí z minulej kapitoly sme hovorili o akýchsi popisoch a strážach, ktoré majú charakter výrazov. Aby sme si zachovali istú mieru abstraktnosti a zároveň nemuseli presne definovať jazyk, v ktorom takéto výrazy chceme tvoriť zavedieme si nasledovné označenia , ktoré budeme v ďalšom texte používať:

- *prvky typu* T, budeme ich označovať samotným názvom typu
- *typ premennej* v – označujeme Type(v)
- *typ výrazu* expr – označujeme Type(expr)
- *množina všetkých premenných výrazu* expr – označujeme Var(expr)
- *väzba množiny premenných* V - je priradenie hodnoty každej premennej $v \in V$, pričom pre prvky väzby b platí $b(v) \in Type(v)$, teda hodnota, priradená premennej je správneho typu
- *hodnota získaná vyhodnotením výrazu* expr *vo väzbe* b – označujeme expr<b>. Získame ju tak, že každú premennú $v \in$ Var(expr) nahradíme hodnotou z väzby b(v). Určenie tejto hodnoty si teda vyžaduje nahradenie premenných vo výraze expr ich hodnotami z väzby b a následné vyhodnotenie výrazu expr. Preto požadujeme, aby premenné z Var(expr) boli

podmnožinou premenných vyskytujúcich sa vo väzbe b a teda aby bolo možné priradiť každej premennej z výrazu nejakú hodnotu.

Množinou B budeme označovať booleovský typ, teda množinu hodnôt {true, false}. Teraz môžeme konečne pristúpiť k definícii farbených Petriho sietí.

Definícia 14. (Jensen, 1994) Farbenou Petriho sieťou nazývame usporiadanú n-ticu $(\Sigma, P, T, A, N, C, G, E, I)$, kde :

Σ - je konečná neprázdna množina **typov**, tiež nazývaná aj množinou **farieb**

P – je konečná množina **miest**

T – je konečná množina **prechodov**

A – je konečná množina **hrán** taká, že : $P \cap T = P \cap A = T \cap A = \varnothing$

N – je **vrcholová** funkcia. $N : A \rightarrow P \times T \cup T \times P$

C – je **ofarbovacia** funkcia. $C : P \rightarrow \Sigma$

G – je funkcia **stráží**. Je definovaná z T do výrazov, pre ktoré platí :

$$\forall t \in T : [Type(G(t)) = B \wedge Type(Var(G(t))) \subseteq \Sigma]$$

E – je funkcia **hranových popisov**. Je definovaná z A do výrazov, pre ktoré platí :

$\forall a \in A : [Type(E(a)) = C(p)_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$, kde p je miesto z N(a)

I – je **inicializačná** funkcia. Je definovaná z P do uzavretých výrazov (výrazov neobsahujúcich premenné) takých, že :

$$\forall p \in P : [Type(I(p)) = C(p)_{MS}]$$

Aj keď je táto definícia matematicky postačujúca, nehovorí nám nič o charaktere jednotlivých množín. Prejdime si preto jednotlivé zložky tejto n-tice podrobnejšie.

Množina **typov** udáva dátové hodnoty (farby). Tie predstavujú jednak rôzne typy značiek, ale zároveň sú tým udané aj operácie a funkcie, ktoré môžu byť použité vo výrazoch v sieti (teda na popisoch hrán, strážach a v inicializačných výrazoch).

Miesta, **prechody** a **hrany** sú popísané trojicou množín P, T a A, pri ktorých vyžadujeme konečnosť a vzájomný prázdny prienik. Tým, že požadujeme, aby množiny miest, prechodov a hrán boli konečné, sa vyhneme množstvu technických problémov, ako napr. možnosti mať nekonečne veľa hrán medzi dvomi vrcholmi.

Vrcholová funkcia zobrazuje každú hranu na dvojicu, kde prvý komponent je zdrojový vrchol a druhý je cieľový vrchol. Tieto dva vrcholy musia byť rôzneho typu (miesto, prechod alebo naopak). Farbenej Petriho Sieti dovoľujeme mať viacero hrán medzi dvoma vrcholmi. Táto funkcia udáva štruktúru siete.

Ofarbovacia funkcia C zobrazuje každé miesto p na typ C(p). Znamená to, že toto miesto potom môže obsahovať len značky príslušného typu.

Funkcia **stráží** zobrazuje každý prechod t na booleovský výraz. Tieto výrazy umožňujú podmienené „pálenie" prechodov. Úspešné vyhodnotenie booleovského výrazu zodpovedajúceho stráži je podmienkou „pálenia" prechodu, ku ktorému je táto stráž priradená. Pri kreslení diagramov vypúšťame stráže, ktoré budú vždy vyhodnotené ako true.

Funkcia **hranových popisov** E zobrazuje každú hranu na $C(p)_{MS}$, teda na multi-množinu nad typmi, ktoré sú povolené (ofarbovacou funkciou C) pre miesto, s ktorým je hrana spojená. Pokiaľ je popis nad hranou, ktorá ide z miesta do prechodu (teda je to vstupná hrana a vstupné miesto), tak multimnožina, ktoré predstavuje vyhodnotenie popisu zodpovedá značkám, ktoré budú zo vstupného miesta odobraté, keď bude prechod „páliť". Pre výstupné miesto (hranou je spojený prechod s miestom) zasa výsledok vyhodnotenia popisu nad hranou predstavuje značky, ktoré budú do výstupného miesta pridané. Hranové popisy teda udávajú, s akými značkami prechod pracuje (aké pridáva a odoberá).

Inicializačná funkcia I zobrazuje každé miesto p na uzavretý výraz, ktorý musí byť typu $C(p)_{MS}$, teda ide opäť nad multi-množinu nad povolenými dátovými typmi pre miesto p. Táto funkcia predstavuje vloženie prvých značiek na miesta v sieti, ktoré potom spustia prvé prechody. Predstavuje teda inicializovanie siete na jej štartovací stav (úvodné rozloženie značiek na miestach v sieti). Pri kreslení diagramov vypúšťame tie funkcie, ktoré budú vždy vyhodnotené ako Ø.

Máme definíciu n-tice, ktorá tvorí farbenú Petriho sieť. Teraz si ešte definujme správanie Farbenej Petriho Siete. Začnime väzbami na prechodoch.

Definícia 15. (Jensen, 1994) Väzba na prechode t je funkcia b definovaná na Var(t), taká, že

1. $\forall$ v $\in$ Var(t) : b(v) $\in$ Type(v)
2. platí G(t)<b> (= true), čo znamená, že stráž na prechode t je vo väzbe b splnená

Základnou vlastnosťou väzby na prechode je to, že umožňuje „pálenie“ prechodu, pretože premenné v stráži G(t) po dosadení hodnôt z väzby spôsobia, že stráž je vyhodnotená ako true.

Definícia 16. (Jensen, 1994) Prvkom značenia (token element) je dvojica (p, c), kde miesto p $\in$ P a typ c $\in$ C(p). Prvkom väzby (binding element) je dvojica (t,b), kde prechod t $\in$ T a väzba b $\in$ B(t) (je z množiny všetkých možných väzieb nad prechodom t).

Množinu všetkých prvkov značenia označujeme TE, množinu všetkých prvkov väzby označujeme BE.

Prvkom značenia je teda dvojica miesto p a typ c resp. značka typu c. Táto dvojica potom predstavuje umiestnenie značky c na mieste p.

Prvkom väzby je dvojica prechod t a väzba na prechode b, ktorá zabezpečí „pálenie“ tohto prechodu.

Značením označujeme multi-množinou nad TE a predstavuje rozloženie značiek na miestach v sieti.

Krokom nazývame neprázdnu, konečnú multi-množinu nad BE. Tie dvojice prechod a väzba, ktoré patria jednému kroku určujú prechody, ktoré v jednom kroku pália a hodnoty premenných, ktoré pálenie prislúchajúcich prechodov umožnili.

Úvodné značenie M_0 je značenie, ktoré získame vyhodnotením inicializačných výrazov :

$\forall$ (p,c) $\in$ TE : $M_0(p,c) = (I(p))(c)$, teda dosadením do inicializačnej funkcie.

Množiny všetkých značení a krokov označíme M resp. Y.

Poznámka : Každé značenie M $\in$ TE_{MS} určuje jedinečnú funkciu M^* definovanú na množine miest P, takú, že $M^*(p) \in C(p)_{MS} : \forall$ p $\in$ P $\forall$ c $\in$ C(p) : $(M^*(p))(c) = M(p,c)$

Na druhej strane, každá funkcia M^* definovaná na P, taká, že $M^*(p) \in C(p)_{MS}$ pre každé miesto $p \in P$ určuje jedinečné značenie $M : \forall (p, c) \in TE : M(p, c) = (M^*(p))(c)$

Preto nebudeme medzi týmito zápismi rozlišovať a budeme podľa potreby používať obe formy.

Definícia 17. (Jensen, 1994) Krok Y je aktivovaný (enabled) v značení M práve vtedy, keď je splnená nasledujúca podmienka :

$$\forall p \in P : \sum_{(t,b)\in Y} E(p, t)<b> \leq M(p)$$

Aby bol krok Y aktivovaný, musí v každom mieste platiť nasledovné : Súčet značiek, ktoré odoberajú prechody spojené s týmto miestom a páliace v tomto kroku musí byť menší alebo rovný počtu značiek, ktoré sa v tomto mieste nachádzajú. Počet značiek v mieste p je daný M(p) a počet značiek, ktoré prechody z miesta odoberajú je daný ako suma výrazov E(p,t)<b>, teda ako suma hodnôt hranových popisov na hranách, ktoré sú spojené s týmto miestom a zároveň pália v kroku Y.

Hovoríme, že prvok väzby (t, b) je aktivovaný a tiež, že prechod t je aktivovaný. Prvky Y sú súčasne aktivované (ak platí $|Y| > 1$).

Keď je krok Y aktivovaný v značení M_1, tak hovoríme, že krok môže nastať (occur), pričom zmení značenie M_1 na značenie M_2, ktoré definujeme ako :

$$\forall p \in P : M_2(p) = (M_1(p) - \sum_{(t,b)\in Y} E(p, t)<b>) + \sum_{(t,b)\in Y} E(t, p)<b>$$

M_2 dostaneme z M_1 tak, že z odoberieme všetky značky udané príslušnými hranovými popismi. Zároveň pridáme na miesta značky, ktoré sú určené popismi na hranách, ktoré vychádzajú z prechodov aktivovaných v tomto kroku.

M_2 je priamo dosiahnuteľné z M_1, čo zapisujeme ako $M_1[Y\rangle M_2$

Kľúčom je teda vyhodnotenie výrazu E(p, t)<b>. Ten nám dá značky (tokeny), ktoré sú odobraté z miesta p, keď „páli“ prechod t s väzbou b. Sumou cez všetky prvky väzby $(t, b) \in Y$ dostaneme všetky tokeny (značky od všetkých páliacich prechodov), ktoré sú odobraté z miesta p keď nastane krok Y. Každé miesto musí obsahovať dostatok

značiek, pričom značky nemôžu jednotlivé prechody zdieľať, každý prechod potrebuje na svoje „pálenie“ vlastné značky.

Jeden krok je nedeliteľná udalosť, neexistuje (v zmysle Petriho Sietí) moment, keď nejaké značky sú už odobraté a nové ešte nie sú pridané. Celý prenos prebehne ako jedna udalosť.

Definícia 18. (Jensen, 1994) Konečná postupnosť nastatí (finite occurence sequence – slovenčina je na toto trocha prikrátka) je postupnosť značení a krokov :

$$M_1[Y_1\rangle M_2[Y_2\rangle M_3 \ldots M_n[Y_n\rangle M_{n+1}$$

taká, že $n \in N$ a $M_i[Y_i\rangle M_{i+1}$ pre každé $i \in \{1,2,\ldots,n\}$, M_1 je štartovacie značenie, M_{n+1} je konečné značenie a n je dĺžka.

Analogicky, nekonečná postupnosť nastatí je postupnosť značení a krokov :

$$M_1[Y_1\rangle M_2[Y_2\rangle M_3 \ldots$$

taká, že $M_i[Y_i\rangle M_{i+1}$ pre každé $i \geq 1$.

Značenie M'' je dosiahnuteľné zo značenia M' práve vtedy, keď existuje konečná postupnosť nastatí začínajúca v M' a končiaca v M''. Množinu značení, ktoré sú dosiahnuteľné z M' značíme [M'⟩. Značenie je dosiahnuteľné práve vtedy, keď patrí do $[M_0\rangle$.

Správanie farebných Petriho sietí je teda určené iniciálnym značením M_0. Toto udáva, ktoré prechody môžu „páliť“ (sú aktivované) ako prvé a teda aké značenia následne vzniknú. Množina značení, ktoré môžu vzniknúť ďalším „pálením“ aktivovaných prechodov potom určuje, aké značenia sa v sieti vyskytnú. Nie všetky značenia, ktoré sú prípustné (vyhovujú podmienkam) môžu naozaj nastať, práve kvôli iniciálnemu značeniu. Tie, ktoré sa teda skutočne vyskytnú udávajú správanie farbenej Petriho siete.

Formálna definícia farbených Petriho sietí uvedená vyššie hovorí o prvkoch, ktoré tvoria farbenú Petriho sieť a o jej správaní. Je to matematický základ, na ktorom budeme stavať a ktorý umožňuje analyzovanie farbených Petriho sietí na matematickej úrovni.

3.2 Farbené Petriho siete a modelovanie komunikácie v MAS

3.2.1 Základný opis

Farbené Petriho siete sú formálny matematický model vhodný na modelovanie systémov, pre ktoré je typická distribuovanosť a súbežnosť. Pre multiagentové systémy sú spomínané vlastnosti priam vlastné a preto je použitie farbených Petriho sietí na mieste.

Samotná formálna definícia však ešte nepostačuje na modelovanie komunikácie v MAS. Preto teraz opíšeme náš prístup ako jednotlivé časti komunikačného systému MAS modelovať pomocou farbených Petriho sietí.

Základnou myšlienkou modelovania komunikácie v MAS pomocou farbených Petriho sietí je reprezentácia správ značkami. Doručovanie správ je reprezentované pomocou hrán a prechody slúžia na ich spracovanie. Miesta sú určené len dočasné uloženie správ a preto nepredstavujú v rámci komunikačného systému v MAS nič konkrétne. Rolu predstavuje podsieť, časť celej Petriho siete, ktorú pre väčšiu zrozumiteľnosť oddeľujeme od ostatných rôl prerušovanými čiarami. Skutočné (fyzické) posielanie správy reprezentuje hrana, ktorá vychádza z prechodu v jednej role a končí v mieste v role inej. To, čo sa odohráva vo vnútri jednej roly, predstavuje vnútornú dynamiku jedného agenta, zmenu jeho stavov. Stav celého systému je potom zložený zo stavov jednotlivých agentov. Stav MAS je teda reprezentovaný stavom celej siete, distribúciou značiek na miestach vo farbenej Petriho sieti.

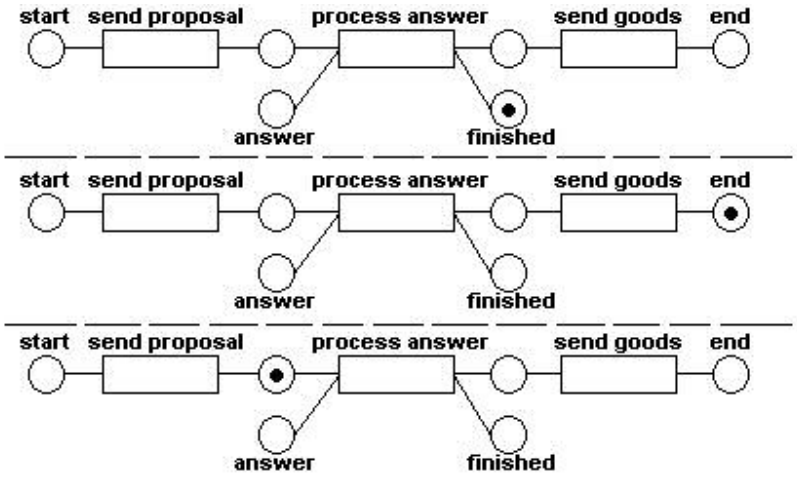
Zaujímavým prvkom v štruktúre farbených Petriho sietí sú stráže a hranové popisy. Tie umožňujú modelovať aj akési zámery, či stratégiu jednotlivých agentov, čo je rozmer, ktorý ostatné formalizmy (viď predchádzajúca kapitola) nezohľadňovali. Keďže sme však aj vo formálnej špecifikácii nechávali priestor pre jazyk, akým sú tieto elementy popísané, je na konkrétnom modelárovi, resp. modeli, akým spôsobom sa rozhodne túto informáciu zachytiť. V každom prípade máme v jednom modeli zaznamenanú informáciu nielen o tom, kto komu akú správu posiela, v akých roliach agenty vystupujú, ale aj o ich zámeroch resp. plánoch.

Časti, ktoré tvoria MAS si teraz prejdeme podrobnejšie a ukážeme aj na príkladoch, ako sa tieto zložky MAS modelujú pomocou farbených Petriho sietí.

3.2.2 Stav systému

Stav MAS je tvorený stavom jednotlivých agentov. Nemusíme predpokladať, že majú agenty tieto stavy nejako skutočne reprezentované (napr. vnútornou premennou), skôr máme na mysli stavy z modelárskeho hľadiska. Ak sa agent nejako rozhodne podľa odpovede (napr. prijatie či odmietnutie návrhu), nemusí mať toto rozhodnutie nejako interne reprezentované. Je ale vhodné toto rozhodnutie reprezentovať v modeli a považovať stav tohto agenta a potom aj celého systému za iný, podľa príslušného rozhodnutia.

Stav jedného agenta reprezentujeme rozložením značiek v miestach, ktoré prislúchajú momentálne zastávanej role. Stav celého MAS je potom reprezentovaný rozložením značiek v celej sieti.

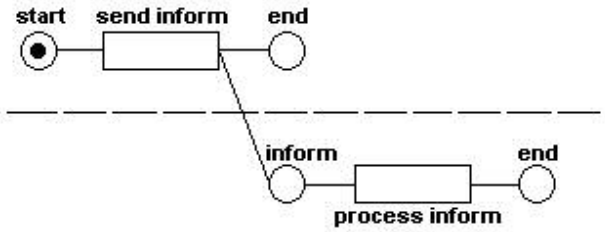


Obrázok 9 : Príklad stavu systému. Čierne bodky predstavujú značky.

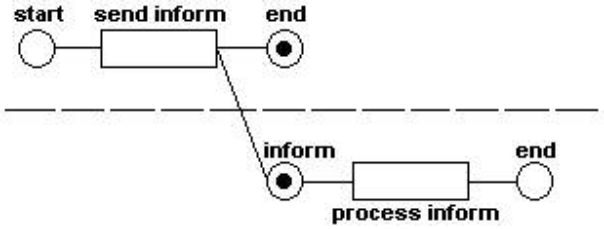
Zmena distribúcie značiek v sieti zodpovedá priebehu komunikácie. Pokiaľ sa mení rozloženie značiek iba vo vnútri rôl, tak sa jedná o zmenu interného stavu. Ak sa však vymenia aj značky medzi rolami, to už zodpovedá poslaniu správy.

3.2.3 Poslanie správy

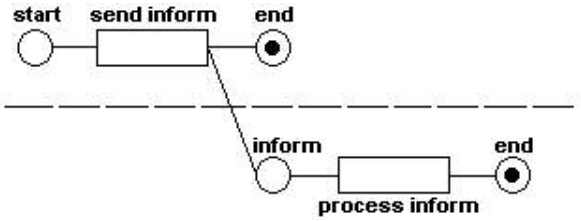
Poslanie správy reprezentujeme presunom značky z prechodu v jednej role do miesta v role inej. Pri použití deliacich prerušovaných čiar na oddelenie rôl je to potom presun značky po hrane, ktorá prechádza cez prerušovanú čiaru. Takéto hrany potom predstavujú jediné komunikačné spojenia medzi rolami.



Obrázok 10a : Iniciačný stav. Jediná značka sa nachádza v mieste start a prechod send transition bude v nasledujúcom kroku páliť – posielať správu.



Obrázok 10b : Tu je reprezentované poslanie správy, prechod send inform umiestnil značku na miesto v druhej role a teda poslal správu.



Obrázok 10c : Koniec pálenia.

Na reprezentovanie obsahu správy používame farby, resp. typy značiek, ktoré nám umožňujú s obsahom pohodlne pracovať. Na základe toho sa potom môže agent rozhodovať, meniť svoj stav, či zisťovať alebo kontrolovať počty. Odpadá potreba stavať

na teórii, ktorá určuje aké typy rečových aktov sa vyskytnú (ako pri Dooleyho grafoch), pričom toto oddelenie môže byť žiadúce najmä pri použití MAS v priemysle. Zároveň je však veľmi jednoduché stotožniť niektoré typy značiek s typmi z teórie rečových aktov a použiť tak farbené Petriho siete podobne ako Dooleyho grafy a využiť ich pri teoretickom výskume.

Samotné odoslanie správy je realizované prechodom, ktorý prenesie značku na miesto v príslušnej role. Prijatie správy je potom automatické – značka zodpovedajúca správe je umiestnená v mieste, ktoré prislúcha prichádzajúcim správam. Spracovanie správy má potom v rámci tejto roly na starosti prvý prechod, ktorý spracuje značku zodpovedajúcu správe zo vstupného miesta.

3.2.4 Protokol

Môžeme neformálne povedať, že protokol je nejaká, vopred stanovená a dohodnutá postupnosť správ alebo akcií. Je to akýsi predpis, podľa ktorého sa bude konverzácia riadiť. Udáva pravidlá, aké správy či akcie sú prípustné alebo očakávané. Protokoly sú základom komunikácie, bez jasných pravidiel by sa v konverzáciách dalo len ťažko orientovať. Preto je schopnosť modelovať protokoly obzvlášť dôležitá a je základom každého dobrého modelovacieho mechanizmu.

Vo farbené Petriho sieťach sa protokoly, v zmysle postupností správ či akcií, modelujú veľmi priamočiaro, ako vidno aj na nasledujúcom príklade.

Príklad 4 : Na obrázku 10 môžeme vidieť modelovanie FIPA inform protokolu pomocou farbených Petriho sietí. Tento protokol sa skladá z poslania správy typu inform príjemcovi, ktorý túto správu spracuje. Priamočiare modelovanie pomocou farbených Petriho sietí potom pozostáva z prechodu *send inform* na jednej strane, prechodu *process inform* na druhej strane.

Keby sme mali postup, ako modelovať protokol nejako formalizovať, môžeme povedať, že každému kroku v protokole (výmene správy) zodpovedá jedna (iná) hrana medzi rolami. Každá takáto hrana, vychádza z prechodu odosielateľa do miesta príjemcu. Pre každý krok v protokole teda zostrojíme miesto v role, ktorá je príjemcom tejto správy

a prechod (prípadne prechody) z predchádzajúceho miesta do miesta príjemcu v role odosielateľa. Rozhodovanie (výber z viacerých možností odpovede) je realizovaný rôznymi prechodmi, ktoré končia v rovnakom mieste.

Samozrejme, každý konkrétny protokol ponúka viacero možností, ako ho môžeme modelovať pomocou farbených Petriho sietí. No základný princíp, kde prenos značky medzi rolami zodpovedá poslaniu správy a prechod zodpovedá jej spracovaniu, nám umožňuje takmer akýkoľvek protokol pomerne priamočiaro modelovať.

3.2.5 Roly

Už na viacerých miestach sme spomínali, že jednotlivé roly sú časti farbenej Petriho siete, ktoré pre jednoduchosť, názornosť a zrozumiteľnosť oddeľujeme od ostatných rôl prerušovanými čiarami. Túto konvenciu odporúčame zachovávať, aj keď sa dá zvoliť aj iný prístup.

Petriho siete umožňujú hierarchické štruktúrovanie. To znamená, že niektoré časti Petriho siete môžeme „vtiahnuť“ do jedného prechodu, alebo naopak, jeden prechod môžeme ešte detailnejšie namodelovať tak, že vyrobíme jemu prislúchajúcu sieť s vstupnými miestami, ktoré zodpovedajú vstupom prechodu a výstupnými miestami, ktoré zodpovedajú výstupom prechodu.

Tento prístup môžeme aplikovať aj na roly. V takom prípade môžeme jednotlivé roly pre jednoduchosť nahradiť jedným prechodom, ktorý môže byť popísaný na inom mieste. Samozrejme musíme ešte doplniť nejaké miesta, pretože pokiaľ by sme tento postup aplikovali na všetky roly, dostali by sme sieť pozostávajúcu len z prechodov, čo je podľa definície farbenej Petriho siete neprípustné. Aby sme ostali v súlade s definíciou pridáme miesta všade tam, kde je hrana medzi dvoma prechodmi. Ak však od takto pridaných miest abstrahujeme, dostaneme graf nápadne podobný Dooleyho grafu. Pri rozsiahlejších modeloch, môže byť tento prístup užitočný, obzvlášť ak sa sieť postupne zjemňuje a prechádza sa od najhrubšej štruktúry po najjemnejšiu. Takisto môže byť aj pri

menších projektoch výhodné, skryť niektoré menej významné alebo jednoduchšie roly s jasným správaním a nechať odokryté iba tie, ktoré sú pre model dôležité.

3.2.6 Stratégia a zámer agenta

Informácia o akejsi stratégii resp. zámere agenta je nový rozmer, ktorý prinášajú farbené Petriho siete oproti ostatným modelovacím formalizmom. V predchádzajúcej kapitole sme videli, že aj napr. UML, resp. agentové rozšírenie AUML dokáže zaznamenať to, že agent má na výber z viacerých možností ako odpovedať (napr. prijatie či odmietnutie), na rozdiel od Dooleyho Grafov, ktoré sú konštruované priamo z už existujúcej komunikácie a teda možnosť výberu ani nenaznačujú. No napriek tomu aj keď zachytíme, že agent má možnosť výberu, resp. nejakého rozhodnutia, nemáme možnosť v Sekvenčnom Diagrame zaznamenať dôvody, resp. motiváciu, ktorá ho pri výbere ovplyvňuje. Pri pohľade na model to je iba čierna skrinka a zámery agenta nám ostávajú skryté.

Farbené Petriho siete umožňujú túto informáciu zachytiť prostredníctvom stráží a hranových popisov. Pomocou nich môžeme určiť, ktorá transformácia má „páliť“, či sú splnené jej predpoklady a tým zachytiť, čo je pre rozhodnutie dôležité a aký zámer je ukrytý za rozhodnutím. Tak zaznamenáme ďalšiu informačnú rovinu do toho istého modelu.

Príklad 5 : Jednoduchým príkladom je obchodník, ktorý neakceptuje ponuky nižšie ako nejaká prahová hodnota. Vo farbených Petriho sieťach to môžeme reprezentovať tak, že k miestu, ktoré obsahuje všetky ponuky, priradíme prechod, ktorý ma stráž, ktorá umožní pálenie iba na ponuky ktoré majú väčšiu hodnotu ako je prahová (teda stráž by vyzerala napr. ako ponuka > 50).

Pridanie informácie o stratégii, či zámeroch agenta nám dáva komplexnejší model s väčším množstvom relevantných informácií. Takýto model je vhodnejší ako pri implementácii systému tak aj pre jeho pochopenie.

3.2.7 Súbežnosť

Jedna z vlastností, ktorá sa často vyskytuje v reálnych systémoch je súbežnosť dvoch akcií. Niektoré akcie nastanú naraz, či prebiehajú súčasne. Nie vždy je to nevyhnutné, niekedy by mohli nastať aj sekvenčne a nemalo by to na systém vplyv, no nie vždy. Niektoré formalizmy (Dooleyho grafy) sú však založené na tom, že akcie musíme vedieť usporiadať a to nie je vždy možné. Avšak farbené Petriho siete nám umožňujú skutočnú súbežnosť, akcie môžu nastať resp. prebiehať naozaj naraz.

4 Implementácia farbených Petriho sietí

4.1 Poznámky k implementácii farbených Petriho sietí.

Predtým, ako opíšeme implementáciu farbených Petriho sietí v prostredí JADE je nutné spomenúť niektoré vlastnosti tohto modelu, ktoré môžu robiť ťažkosti pri implementácii - a to nielen v programovacom jazyku JAVA či v prostredí JADE.

4.1.1 Nedeterminizmus

Jednou a asi najzásadnejšou z takýchto vlastností je nedeterministickosť. Definícia farbených Petriho sietí a ich správania umožňuje dostatočnú voľnosť pri špecifikovaní prechodov, resp. podmienok za ktorých je prechod aktivovaný. Táto voľnosť potom spôsobuje, že poradie, v ktorom prechody „pália“ môže byť rôzne a je na systéme (resp. modelárovi), ktorý konkrétnu farbenú Petriho sieť vyhodnocuje (simuluje jej správanie) aké poradie nakoniec zvolí. Ilustráciou môže byť sieť, ktorá sa počas simulácie dostane do stavu, že je možné aktivovať dva prechody v jednom kroku. Oba majú splnené podmienky, no zároveň pokiaľ páli jeden prechod, použije značky, ktoré umožňovali pálenie druhého prechodu. Nie je žiaden dôvod uprednostniť jeden alebo druhý prechod – dochádza teda k nedeterministickej voľbe. Vzhľadom na formálnu definíciu farbených Petriho sietí je takéto správanie legitímne. No pri implementovaní farbených Petriho sietí na súčasných počítačoch je nutné niektorú z možností vybrať. Rôzne systémy na simulovanie môžu k tomuto problému pristupovať viacerými spôsobmi – výber môže byť náhodný (generátorom náhodných čísel), riadený prioritne podľa nejakého zadaného kľúča (napríklad poradie v poli prechodov a pod.) alebo dokonca môže byť riadenie výberu prenechané na užívateľovi, teda simulácia sa preruší a užívateľ musí zvoliť, ktorý z prechodov bude nakoniec páliť. My sme sa rozhodli pre prioritný výber určený poradím v zozname prechodov (prvé sú prioritnejšie), pretože tak má programátor, ktorý danú Petriho sieť implementuje možnosť určiť presné poradie prechodov, či dokonca toto poradie meniť počas simulácie (ak je to žiadúce alebo potrebné). Zároveň je potom z implementácie jasné, aké možnosť bude pri výbere uprednostnená. Náhodnú možnosť výberu (pomocou náhodného generátora čísel) je

potom možné z tohto prístupu dosiahnuť tak, že pred každým testovaním sa poprehadzuje poradie prechodov v zozname.

4.1.2 Súbežnosť

Ďalšou z komplikovaných otázok je skutočná súbežnosť. Táto vlastnosť je u farbených Petriho sietí cenená, avšak pri implementácii narážame na vlastnosti počítačov, na ktorých bude takýto systém fungovať. Je samozrejmé, že naozajstnú súbežnosť ťažko dosiahneme na systéme, ktorý je čisto sekvenčný, resp. reálnu súbežnosť dosiahneme iba na počítačoch, ktoré súbežnosť hardware-ovo umožňujú. V praktických a priemyselných aplikáciách sa multiagentové systémy môžu vyskytovať aj na jednoduchších platformách ako sú mobilné telefóny alebo PDA. V našej implementácii sme sa rozhodli nepodporiť úplnú súbežnosť, pretože vo väčšine prípadov by réžia ohľadom viacerých samostatných vlákien bola skôr prekážkou ako výhodou, nakoľko prechody, ktorých množstvo môže byť teoreticky neobmedzené by museli byť v samostatných vláknach, aby mohli byť súčasne vyhodnocované. Toto obmedzenie sa týka však len súbežných prechodov v rámci jedného agenta a jedného jeho správania (jednej roly). Prechody, ktoré nastávajú u rôznych agentov, ktorí fungujú na dvoch rôznych počítačoch, alebo na jednom počítači, ktorý umožňuje súbežnosť budú vykonané naozaj súbežne, čo zabezpečuje to, že jednotlivé správania agentov sa nachádzajú v samostatných vláknach, resp. na samostatných počítačoch. Naša implementácia teda podporuje čiastočnú súbežnosť, resp. nepodporuje iba veľmi obmedzenú časť súbežných modelov a situácií.

4.1.3 Modelovanie prijímania správ

Dôležitá poznámka sa viaže aj k modelovaniu komunikácie farbenými Petriho sieťami, konkrétne k posielaniu správ. Tento jav je modelovaný presunom značky z prechodu (v role jedného agenta) na miesto v role druhého agenta. Implementačne to však nie je jednoduchá záležitosť. To čo je v modeli reprezentované presunom značky sa v reálnych multiagentových systémoch skladá z prípravy, poslania a prijatia správy, resp. pri modelovaní farbenými Petriho sieťami z odobratia značiek, prípravy, poslania

a prijatia správy a následného vytvorenia prislúchajúcej značky. Agenty sa nemusia nachádzať na jednom počítači a preto priamy prístup, keď by odosielajúci agent priamo vložil značky na miesto prijímateľa nemusí byť vhodný či dokonca ani zrealizovateľný. Okrem toho, našim hlavným záujmom je model komunikácie a teda predpokladáme, že správy majú nejaký tvar, ktorý nemusí byť zhodný s tvarom značiek. Napr. v prípade, keď implementujeme iba časť systému si nemôžeme dovoliť posielať iba reprezentáciu značky (napr. v tvare stringu) namiesto správy, resp. ešte väčší problém môže byť s prijímaním správ, kde tvar správy priamo nekorešponduje s tvarom značky. V takom prípade je najlepším prístupom transformovať správy na značky a naopak. Toto si však vyžaduje dodatočnú réžiu, ktorá nie je v modeli priamo zahrnutá.

4.2 Úvod do prostredia JADE

Java Agent Development Framework (JADE)[1] od firmy TiLab je agentový middleware pre platformu JAVA. Je to sada nástrojov na jednoduchú a efektívnu tvorbu a ladenie multiagentových systémov, ktoré sa môžu nachádzať na rôznych platformách a ktoré môžu byť distribuované na viacerých počítačoch. Hlavnou snahou tohto prostredia je ponúknuť programátorovi dostatočný základ a odbremeniť ho od nízkoúrovňových detailov.

Vytvorenie multiagentovej aplikácie pomocou JADE je jednoduché. Programátor musí vytvoriť svoje vlastné agenty odvodením z triedy `jade.core.Agent` a implementovaním niekoľkých metód. Potom stačí takto vytvorené agenty umiestniť do spusteného Agent containera resp. agentovej platformy. To zabezpečí fungovanie agentov, posielanie a prijímanie správ a ďalšie dôležité funkcie. V takejto agentovej platforme potom agenty „žijú“ – teda vykonávajú svoju činnosť až pokiaľ nie sú ukončené alebo sami neukončia svoju činnosť.

Aby bolo programovanie správania agentov v JADE jednoduchšie, agenty si môžu pridávať a odoberať správania (behaviours), čo sú vlastne inštancie tried odvodené od `jade.core.behaviour`. Takto implementované správania majú špecifický tvar, ktorý umožňuje ich jednoduchú správu a programátorovi opäť uľahčuje prácu. Autor správania musí implementovať len tri metódy - `behaviour.onStart()`, `behaviour.action()` a `behaviour.done()`. Prvá z nich inicializuje správanie, druhá vykoná akcie (je vlastne telom celého správania) a tretia slúži ako test, či už správanie ukončilo svoju činnosť (teda ak neustále vracia false tak takéto správanie nikdy neskončí). Pokiaľ je správanie pridané medzi behaviours nejakého agenta (pomocou metódy `Agent.addBehaviour())` tak sa najprv zavolá `onStart()`, ktorý toto správanie inicializuje. Potom sa vykoná metóda `action()` po prvý krát. Následne sa testuje funkcia `done()` a ak vráti true tak správanie končí svoju činnosť. Inak sa opäť zavolá `action()` a následne sa opäť testuje `done()`. Toto je cyklus jedného správania

[1] Jade je prístupné na internetovej stránke `jade.tilab.com`

agenta, pričom ten môže mať správaní niekoľko (v modeli potom takéto správania vlastne prislúchajú rolám). Autor systému vlastne vytvára správania (čo teda zodpovedá implementovaniu troch funkcií) a tie potom len pridá príslušným agentom (teda implementuje v podstate len jednu funkciu u agenta).

Posielanie a prijímanie správ rieši za programátora JADE resp. agentová platforma. Programátor musí správy len vytvoriť a v kóde zavolať `Agent.send(msg)` na odoslanie správy, či zavolať `Agent.receive()` na prijatie správy. Príjem správy môže byť aj blokujúci, no takéto prijímanie zablokuje aj ostatné správania, nie len to z ktorého bolo blokovanie zavolané. Častejšie sa preto používa receive so vzorom správy (message template), kde takýto receive vyberie z prijatých správ tú, ktorá sa zhoduje so vzorom. Hľadanie príjemcu a prenos dát už ale za programátora rieši agentová platforma. K správam ešte treba podotknúť, že sú vytvorené v súlade so špecifikáciou FIPA a majú teda možnosť nastaviť parametre ako typ rečového aktu, identifikátor konverzácie, identifikátor odpovede a mnohé ďalšie. Nie sú však povinné a programátor ich teda môže alebo nemusí vyplniť.

4.3 Implementácia farbených Petriho sietí v JADE

4.3.1 Opis implementácie

Najčastejší a zároveň najefektnejší spôsob programovania multiagentových aplikácií pomocou JADE je prostredníctvom implementácie správaní (behaviours). Tie sú potom priradené jednotlivým agentom, pre ktorých sú tieto správania určené. Preto sme sa aj my rozhodli implementovať farbené Petriho siete do JADE prostredníctvom tohto mechanizmu. Našu implementáciu nájdete na priloženom CD v adresári jade/src/jade/core/behaviours.

Vytvorili sme novú triedu správaní nazvanú `CPNBehaviour` (od Coloured Petri Net Behaviour teda správanie založené na farbených Petriho sieťach). Samotné správanie `CPNBehaviour` priradené agentovi nemá žiaden efekt, no je to preto, lebo nemá žiadne miesta ani prechody. Avšak jadro resp. engine na simulovanie farbených Petriho sietí sa nachádza práve v tejto triede. `CPNBehaviour` je regulérne správanie odvodené od jade.core.behaviour a teda implementuje zásadné metódy `onStart()`, `action()` a `done()`. V nich sa nachádza mechanizmus, ktorý simuluje farbené Petriho siete a preto neodporúčame pri správaniach odvodených z `CPNBehaviour` implementovať tieto metódy (okrem `onStart()`). Pokiaľ je to nutné, použite konštrukt `super()`, ktorý zabezpečí zavolanie týchto metód z pôvodnej `CPNBehaviour` pre zabezpečenie správneho fungovania (pri `onStart()` si na to treba dať obzvlášť pozor, lebo predpokladáme, že pri vytváraní správaní bude treba túto metódu preťažiť).

Ešte pred tým ako opíšeme princíp fungovania `CPNBehaviour` je treba spomenúť triedy, ktoré `CPNBehaviour` využíva, a ktoré sme zoskupili do balíka `CPNElements` – `CPNToken`, `CPNPlace` a `CPNTransition`. Ako už názvy napovedajú, sú to triedy, ktoré zodpovedajú značke, miestu a prechodu z Petriho sietí.

`CPNToken` teda predstavuje značku a tým pádom je nositeľom informácie. V našej implementácii nemá žiadne špeciálne vlastnosti, predpokladá sa odvodzovanie od

tejto triedy a následné doplnenie premenných, ktoré budú obsahovať konkrétne dáta a teda informácie.

`CPNPlace` zodpovedá miestu v Petriho sieťach a teda uchováva tokeny. Implementačne je reprezentovaná zoznamom (typu `ArrayList`), do ktorého môžeme tokeny pridávať, odoberať, testovať na prázdnosť a podobne. Táto trieda má implementovanú väčšinu užitočných funkcií a preto predpokladáme jej priame využívanie.

`CPNTransition` je prechod z Petriho sietí a v našej implementácii je vlastne centrom implementačnej logiky. Programátor využívajúci správanie odvodené z `CPNBehaviour` bude pracovať najmä s triedami odvodenými z `CPNTransition`. Vnútornými premennými tejto triedy sú dva zoznamy miest (objektov typu `CPNPlace`), ktoré predstavujú vstupné a výstupné miesta prechodu. Do nich je možné pridať aj odobrať miesta, odporúčame si však vstupné aj výstupné miesta odovzdávať ako parametre v konštruktore prechodu a následne ich v tele konštruktora pridať do príslušných zoznamov.

`CPNTransition` obsahuje štyri abstraktné metódy, ktoré je potrebné implementovať a ktoré reprezentujú proces „pálenia“ prechodu. Samotné pálenie je teda rozdelené na štyri kroky – `Test()`, ktorý vráti true ak sú splnené podmienky pálenia, `RemoveTokens()`, ktorý následne odoberie príslušné značky z vstupných miest, `DoActions()`, kde sa vykonajú dodatočné kroky, ktoré nesúvisia priamo so značkami (napr. vypisovanie na obrazovku alebo interakcia s užívateľom) a konečne `AddTokens()`, ktorý pridá značky do výstupných miest. Všetky štyri tieto metódy má plne v moci programátor, pomocou nich ovláda pálenie prechodu. Keďže nie je obmedzený jazyk, v ktorom môžu byť zadané stráže a hranové popisy a ani nie sú nijako spojené vstupné a výstupné značky, ani my nekladieme žiadne obmedzenia na tieto funkcie a tie môžu byť implementované podľa potrieb autora prechodu. Odporúčame však nevkladať značky do vstupných miest a naopak nevyberať značky z výstupných miest, ale v prípade potreby pridať jedno miesto medzi vstupné a aj výstupné miesta prechodu.

`CPNBehaviour`, resp. správania odvodené z tejto triedy, využívajú spomínané triedy `CPNToken`, `CPNPlace` a `CPNTransition` v nasledovnom zmysle. `CPNBehaviour` obsahuje zoznam miest (objektov typu `CPNPlace`) a zoznam prechodov (objektov typu `CPNTransition` a odvodených tried). Miesta treba priradiť prechodom, čo vlastne znamená zadefinovať sieťovú štruktúru. Naše odporúčanie je implementovať metódu správania `onStart()` a po konštrukte `super.onStart()` vytvoriť najprv miesta a tie potom priradiť novovznikajúcim prechodom, pričom odporúčame pri písaní prechodov využiť parametre konštruktorov na tento účel. V metóde `onStart()` odporúčame taktiež vykonať aj iniciálne značenie, teda umiestniť nejaké značky (objekty typu `CPNToken` a odvodených tried) do miest (pomocou `place.AddToken()`), pričom tieto značky umožnia páliť prvým prechodom. Definovanie sieťovej štruktúry je opäť úplne v rukách programátora.

Po vytvorení sieťovej štruktúry (teda priradení miest prechodom) a iniciálnom značení (pridaní značiek niektorým miestam) sa už programátor nemusí starať o beh takto zadefinovanej farbenej Petriho siete, to už zaňho zabezpečí engine ukrývajúci sa v metóde action() pôvodnej triedy `CPNBehaviour`. Ten funguje nasledovne. Prejde cez všetky prechody uložené v zozname prechodov a zavolá `Test()`. Pri prechodoch, ktoré sú úspešne testované (a teda vrátia true) je potom následne zavolaná metóda `RemoveTokens()` a sú nastavené ako aktivované (enabled). Metóda `RemoveTokens()` sa volá hneď po úspešnom teste preto, aby sa zabezpečila výlučnosť použitia značiek, teda aby nemohli páliť dva prechody, ktoré používajú tie isté tokeny. Preto, ak jeden prechod môže byť spustený, tak sa hneď odoberú značky zo vstupných miest a tieto značky nemôže použiť iný prechod na svoje aktivovanie (úspešné volanie funkcie `Test()`). Prejdú a otestujú sa teda všetky prechody, a ktoré môžu páliť sú označené ako aktivované. V druhom prechode sa potom prejdú aktivované prechody a zavolajú sa ich metódy `DoActions()` a `AddTokens()`. Prvá vykoná programátorom zadané akcie nesúvisiace priamo s presúvaním značiek a druhá pridá nejaké (no možno žiadne) značky na výstupné miesta. Pokiaľ aspoň jeden prechod pálil, celý proces začína odznova a opäť sa začínajú testovať prechody. Správanie končí svoju

činnosť (toto má na starosti metóda správania `CPNBehaviour.done()`) vtedy, keď už žiaden prechod nepálil a teda nemohli byť pridané žiadne nové značky a v nasledujúcom kroku už nebude určite páliť žiaden prechod.

Doteraz sme však nespomenuli dôležitý aspekt programovania agentov v JADE a teda aj v našej implementácii. Posielanie správ v JADE funguje pomocou metódy agenta (volaním `Agent.send()`) a nie cez správanie, čo prakticky znamená, že každý agent musí mať nejaký mechanizmus, ktorý zistí, ktorému správaniu prislúcha ktorá správa. V našom prípade ešte navyše musíme zistiť, do ktorého miesta umiestniť dáta z doručenej správy. Na túto réžiu sme vytvorili metódu `ProcessMessages()` triedy `CPNBehaviour`. Táto metóda je súčasťou enginu, je volaná po testovaní všetkých prechodov a následnom vykonaní ich akcií. `ProcessMessages()` vracia `true` alebo `false` podľa toho, či našiel nejakú správu pre toto správanie a následne pridal značku na príslušné miesto. Kód tejto metódy však musí implementovať programátor, ktorý chce so správaním založeným na `CPNBehaviour` pracovať. My odporúčame použiť MessageTemplate, ktoré umožňujú z fronty prijatých správ vybrať také, ktoré sú zhodné v zadaných parametroch so vzorom. Ak `ProcessMessages()` vráti `true`, je to považované za rovnaký dôvod pokračovať v simulácii (v testovaní prechodov ...) ako keby pálil aspoň jeden z prechodov. Naopak, pokiaľ ani `ProcessMesagess()` nevráti `true`, záleží od nastavenia `CPNBeghaviour.WaitForMessages`, či sa správanie ukončí (`false`) alebo zablokuje (`true`). Toto zablokovanie však nie je také ako pri blokovanom prijímaní správ, pretože v tomto prípade sa dostanú ku slovu ostatné správania prislúchajúce agentovi. Správanie je odblokované pri príchode akejkoľvek správy agentovi. Preto odporúčame nastaviť `WaitForMessages` na `false` len vtedy, ak sa má správanie po ukončení pálenia ukončiť, alebo ak je to správanie, ktoré neprijíma žiadne správy.

Teda správanie odvodené od `CPNBehaviour` funguje nasledovne. Na začiatku sa vytvoria miesta a prechody. Prechodom sú priradené vstupné a výstupné miesta, čím vznikne sieťová štruktúra. Následne sú vytvorené značky, ktoré sú umiestnené do niektorých miest, čo zodpovedá iniciálnemu značeniu, ktoré umožňuje páliť prvým prechodom. Potom sa začne cyklus fungovania správania. Prejdú sa všetky prechody

v zozname. Tie, ktoré majú splnenú testovaciu podmienku odoberú zo vstupných miest značky, ktoré túto podmienku zabezpečovali. Zároveň sú tieto prechody označené ako aktivované. Následne všetky aktivované prechody vykonajú im prislúchajúce akcie a pridajú značky na výstupné miesta. Potom sa otestuje funkcia `ProcessMessages`, či nie je nejaká správa pre toto správanie v zozname správ. Ak je, tak sa pridá na príslušné miesto. To je koniec jedného cyklu. Pokiaľ pálil aspoň jeden prechod, alebo prišla správa, tak sa cyklus začína opäť testovaním všetkých prechodov. Pokiaľ nepálil ani jeden prechod a ani žiadna správa pre toto správanie nebola prijatá, správanie sa podľa nastavenia buď zablokuje (a odblokuje sa pri najbližšom prijatí správy) alebo ukončí svoju činnosť.

4.3.2 Postup programovania

Pre programátora vytvorenie správania pomocou `CPNBehaviour` pozostáva z nasledujúcich krokov. Odvodí nové triedy z `CPNToken`, ktoré reprezentujú rôzne značky v sieti a teda sú nositeľmi dát. Týmto novým triedam vyplní prevažne vnútorné premenné, ktoré budú niesť relevantné informácie. Ďalej odvodí nové triedy z `CPNTransition`, ktoré predstavujú rôzne prechody. Pri ich vytváraní implementuje najmä metódy `Test()`, `RemoveTokens()`, `DoActions()` a `AddTokens()`, ktoré definujú správanie prechodu. Nakoniec odvodí novú triedu z `CPNBehaviour`, ktorá bude reprezentovať správanie. Tejto novej triede implementuje metódu `onStart()`, kde umiestni vytvorenie miest, prechodov a prvých značiek. Zároveň tu vytvorí spojenie miest a prechodov. Okrem toho implementuje metódu `ProcessMessages()`, ktorá bude podľa nejakých pravidiel pridávať značky z informácií v správach na miesta v sieti. Takto vytvorené správanie je potom plne funkčné a možno jeho inštanciu pridať agentom. Podotýkame, že toto správanie reprezentuje iba jednu rolu a je teda len časťou celej farbenej Petriho siete reprezentujúcej model komunikácie. Na plnohodnotný multiagentový systém treba vytvoriť ešte aspoň jedno ďalšie správanie, ktoré predstavuje zvyšok siete, alebo toto správanie priradiť viacerým agentom.

5 Príklady

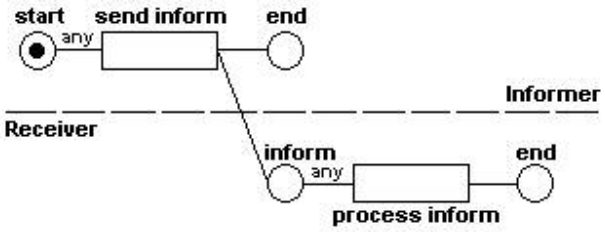
Pre lepšie pochopenie toho, akým spôsobom sa pracuje s našou implementáciou farbených Petriho sietí sme si pripravili tri príklady. Začneme najjednoduchším, ktorý podrobne rozoberieme a ukážeme na ňom základné princípy práce. Potom prejdeme na zložitejšie príklady, ktoré už nebudeme až tak detailne opisovať, ale poukážeme u nich len na tie najdôležitejšie miesta.

Všetky spomínané príklady je možné nájsť na priloženom CD v adresári jade/src/examples/CPNExamples.

5.1 Inform

Príklad sme nazvali Inform preto, lebo modelujeme FIPA Inform protokol, ktorý sme už v našej práci spomínali. Pre pripomenutie, je to kvázi najjednoduchšia situácia pri modelovaní komunikácie keď jeden agent oznámi druhému agentovi nejakú informáciu (informuje ho). Ten informáciu spracuje.

Z hľadiska modelovania farbenou Petriho sieťou máme dve roly – nazvime ich *Informer* a *Receiver*. Každá z týchto rôl má len jeden prechod. *Informer* má prechod, ktorý páli na prítomnosť značky. Pálenie znamená, že pošle správu príjemcovi, teda umiestni značku do jeho miesta, ktoré sme označili *inform* a sám si umiestni značku do miesta *end*. Príjemca – *Receiver* – má prechod, ktorý keď zdetekuje značku tak páli a umiestni značku do svojho miesta end. V reálnej situácii by zároveň spracoval informáciu, ktorá sa nachádza v značke, ktorú mu do miesta *inform* umiestnil agent *Informer*. V našom teoretickom modeli sme od toho upustili, pretože to z modelárskeho hľadiska nič nemení. Farbená Petriho sieť vyzerá ako na obrázku 11.



Obrázok 11 : Farebná Petriho sieť k príkladu Inform. Kľúčové slovo *any* nad hranami hovorí, že z miesta berieme ľubovolnú značku, ktorá umožní pálenie.

Tento model sme implementovali nasledovne. Vytvorili sme dve triedy agentov `InformAgent` a `ReceiverAgent`, obe sú triedy odvodené od `jade.core.Agent`. Z triedy `InformAgent` pri spustení prostredia JADE vytvoríme inštanciu Informer a z triedy `ReceiverAgent` inštanciu Receiver.

`InformAgent` obsahuje jediné správanie – `InformBehaviour`. Toto správanie je odvodené od `CPNBehaviour` a predstavuje hornú časť farbenej Petriho siete z obrázku 11 (teda tú nad čiarkovanou čiarou). Obsahuje dve miesta (*start* a *end*) a jeden prechod – `InformTransition`, odvodený z `CPNTransition` – ktorý zodpovedá prechodu *send inform* z modelu. V našej implementácii prechod je aktivovaný keď je miesto neprázdne (prechod testuje vstupné miesto na neprázdnosť pomocou funkcie `CPNPlace.IsEmpty()`). Vtedy z tohto miesta odoberie značku typu `AddressToken`. Táto trieda je odvodená z `CPNToken` a obsahuje jedinú premennú typu `String` – `sAddress`, ktorá obsahuje adresu agenta, ktorému budeme posielať *inform*. My dodáme do miesta start takúto značku pri vytváraní správania `InformBehaviour`, čo reprezentuje iniciálne značenie. Tým je zabezpečené, že tento prechod je aktivovaný hneď pri štarte správania. Keď prechod páli odoberie najprv značku typu `AddressToken` zo vstupného miesta. Z nej si prečíta adresu, na ktorú pomocou metódy `Agent.send()` pošle správu *inform*. Toto sa udeje v metóde `DoAction()` prechodu `InformTransition`. Potom ešte tento prechod vytvorí novú prázdnu značku typu `CPNToken` a umiestni ju do výstupného miesta.

`ReceiverAgent` obsahuje tiež len jedno správanie – `ReceiverBehaviour`. Toto správanie tiež odvodené od `CPNBehaviour` však nie je iniciované hneď pri štarte

správania, ale čaká na doručenie správy. Má implementovanú kľúčovú metódu `ProcessMessages()`, ktorá čaká na príchod správy. Využíva pri tom zhodu so vzorom, konkrétne s príznakom `ConversationId`. `InformTransition` nastaví tento príznak na `Inform` a preto aj vo vzore nastavíme `MatchTemplate.MatchConversationId("Inform")`. Keď takúto správu agent `Receiver` dostane, metóda `ProcessMessages()` vloží do vstupného miesta (ktoré zodpovedá miestu *inform* z modelu) prázdny `CPNToken`, ktorý aktivuje prechod `ReceiverTransition` (zodpovedá prechodu *process inform* z modelu). Je to jediný prechod správania `ReceiverBehaviour` a jeho jedinou funkciou je dodať prázdny `CPNToken` do výstupného miesta. Keby sme však posielali spolu so správou nejakú užitočnú informáciu, tu by sa táto informácia spracovávala.

Po úspešnom spustení prostredia s agentmi `Receiver` typu `ReceiverAgent` a `Informer` typu `InformAgent` bude výstup na obrazovke nasledovný :

```
+Informer+ Sending Inform to Receiver
-Receiver- Inform received.
```

Prvý riadok zodpovedá odoslaniu správy *inform* agentom `Informer`, druhý riadok zodpovedá prijatiu správy *inform* agentom `Receiver`.

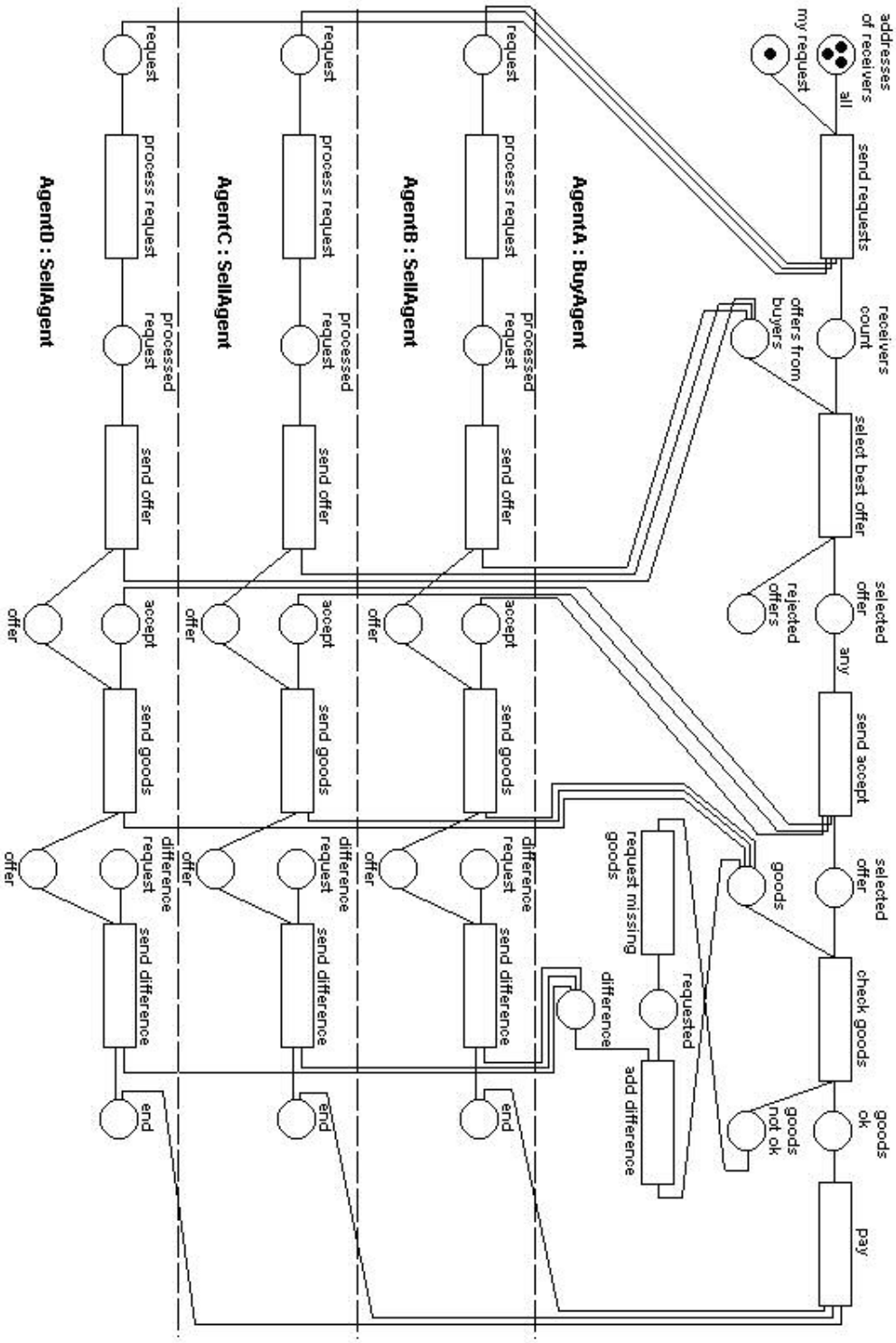
Tento jednoduchý príklad ilustruje najdôležitejšie aspekty implementovania modelu založeného na farbených Petriho sieťach – posielanie správy. Väčšinou je na jednej strane prechod, ktorý z nejakej značky zistí adresu príjemcu. Následne vyhotoví a odošle správu, čo sa deje väčšinou v metóde `DoActions()` tohto prechodu. Na druhej strane je metóda `ProcessMessages()`, ktorá čaká na správy a pridáva z nich vyrobené značky na prislúchajúce miesta.

V ďalších príkladoch trocha zmeníme optiku a nebudeme sa už venovať tak podrobne opisu implementácie. Je to aj preto, že mnoho prechodov je implementovaných

rovnakým spôsobom ako tu, no najmä preto, že zložitosť nasledujúcich príkladov je omnoho väčšia a ich opis by zabral neúmerne veľa miesta a zároveň by znížil ich pochopenie. Budeme ale na základe tohto príkladu predpokladať niekoľko vecí. Väčšina prechodov testuje vstupné miesta na neprázdnosť (slovo any v obrázku), preto ho budeme vynechávať a uvádzať popisy nad hranou budeme len ak to bude inak. Teda prechod páli ak všetky jeho vstupné miesta sú neprázdne (obsahujú aspoň jednu značku), alebo je to uvedené inak. Tak isto nebudeme explicitne v texte uvádzať, že po prijatí správy metóda ProcessMessages() príslušného správania umiestni značku s prislúchajúcimi dátami na určené miesto. Okrem toho platí, že všetky správania agenta sú odvodené od typu CPNBehaviour, všetky prechody v príkladoch sú odvodené od triedy CPNTransition a všetky značky sú odvodené od CPNPlace. Preto to nebudeme ďalej uvádzať.

5.2 Contract

V tomto príklade sme sa inšpirovali príkladom, ktorý sme predstavili v kapitole o Dooleyho grafoch. Tam jeden agent vypisuje kontrakt – požiada ostatné agenty, aby mu dodali isté množstvo tovaru do istého dňa. Tie mu odpovedajú – v našom prípade rovno protiponukou, teda koľko tovaru sú mu schopní dodať do akého dátumu. Agent, ktorý kontrakt vypisoval si vyberie najlepšiu ponuku a s agentom, ktorý túto ponuku poslal zaháji vymieňanie tovaru a peňazí. Ostatným agentom oznámi, že ich ponuku odmieta. Potom komunikuje už len s vybratým agentom, pričom najprv ho požiada o zaslanie tovaru. Ten mu pošle tovar, no môže sa stať, že mu pošle menej než sľúbil. Preto ho agent, ktorý vypisoval kontrakt musí požiadať najprv o dodanie zvyšku tovaru a až po jeho dodaní mu zaplatí. Model tohto príkladu môžeme vidieť na obrázku 12.



Obrázok 12 : Farbená Petriho sieť k príkladu Contract.

Z implementačného hľadiska máme dve triedy – BuyAgent a SellAgent, pričom agent, ktorý vypisuje kontrakt je reprezentovaný triedou BuyAgent a agenty, ktoré tovarom disponujú sú reprezentované ako trieda SellAgent. V našom príklade máme jedného agenta typu BuyAgent – AgentA a tri agenty typu SellAgent – AgentB, AgentC, AgentD. Agenty typu SellAgent sú parametrizovateľné, no inak sú tvorené rovnakými sieťami. Ako parametre majú množstvo a dátum, ktoré budú ponúkať ako svoju ponuku. AgentA obošle troch agentov – AgentB, AgentC a AgentD so svojou ponukou, na čo oni zareagujú poslaním svojich ponúk. Tieto AgentA usporiada podľa množstva a v prípade rovnosti aj podľa času. AgentA vie, že oboslal tri agenty a preto čaká na tri ponuky. Až po ich prijatí začne ponuky porovnávať. Víťazná ponuka potom pokračuje do podčasti siete, kde už sa uskutočňuje dodanie tovaru, kontrola množstva a následná platba. Zaujímavé na tejto podčasti je to, že sa v nej nenachádzajú žiadne globálne premenné, všetky informácie sú reprezentované pomocou značiek v sieti.

Na začiatku teda AgentA pošle agentom, ktorých adresy má vo vstupnom mieste kópiu svojej ponuky, ktorú takisto dostane ako vstupný parameter (v druhom vstupnom mieste). Agenti, ktorí ponuku prijali ju spracujú (porovnajú so svojimi vlastnými ponukami) a na základe porovnania odošlú svoj protinávrh späť odosielajúcemu agentovi. Ten vyhodnotí ponuky (prechod *select best offer* v kóde reprezentovaný inštanciou triedy `SelectBestOfferTransition`) a vyberie najvhodnejšiu, ktorú odovzdá do výstupného miesta *selected offer*. Odtiaľ ju vyberie prechod *send accept*, ktorý je touto ponukou aktivovaný a ten odošle správu accept práve jednému agentovi – víťazovi. Aj keď na obrázku 12 sú z prechodu *send accept* hrany do každého miesta *accept* v každej sieti prislúchajúcej každému agentovi typu `SellAgent`, značka sa dostane len na jedno z týchto miest a teda len jednému agentovi. Ten to berie ako výzvu na odslanie tovaru (do miesta *goods*). Potom už nasleduje len mechanizmus porovnania množstva tovaru s dohodnutým množstvom. Pokiaľ nebolo prijaté dohodnuté množstvo je požadovaný zvyšok a až po jeho doručení sa uskutoční platba.

Výstup príkladu po jeho spustení a inicializácii platformy JADE vyzerá približne takto :

```
Message for agent AgentB ready.
Message for agent AgentC ready.
Message for agent AgentD ready.
Offer send : AgentA 50 7
Offer send : AgentA 40 9
Offer send : AgentA 45 8
Selected offer is from agent : AgentD
Sending goods > 11 to agent AgentA
-- Goods not ok --
Requesting goods : only 11 received from AgentD
Sending 39 goods to AgentA
Adding difference 50
> Goods checked ok <
Sending payment to AgentD : 100
```

Prvé tri riadky zodpovedajú príprave a odoslaniu žiadosti pre SellAgenty od agenta A – kupujúceho. Nasledujúce tri riadky predstajú odoslanie ponuky (offer) od SellAgentov agentovi A. Následne je vybratá jedna ponuka, v našom prípade je to ponuka od agenta D, čomu zodpovedá siedmy riadok. AgentD odosiela tovar. Tu prichádza v našom prípade do hry náhodnosť a AgentD odošle iba 11 jednotiek namiesto dohodnutých 50. Preto AgentA si vyžiada od AgentaD zvyšok (riadok 10). Riadok 11 potom predstavuje doručenie rozdielu. Riadok 13 by kľudne mohol nahradiť riadok 9 v prípade, že by generátor náhodných vygeneroval iné číslo a AgentD by odoslal dohdnutých 50 jednotiek tovaru hneď na začiatku. Takto nastane platba (riadok 14) až teraz.

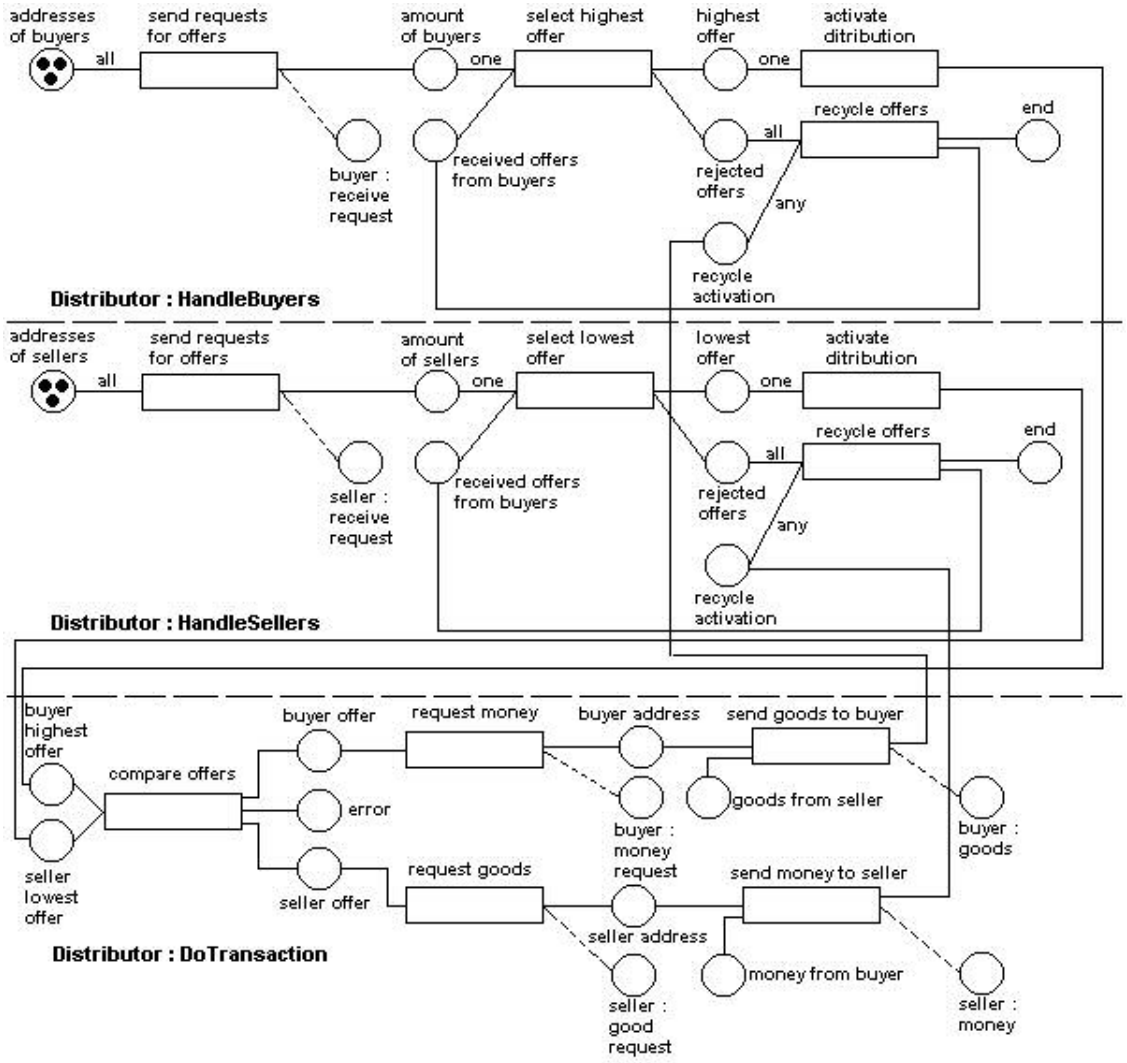
Každá z tried BuyAgent a SellAgent má implementované len jedno správanie. V nasledujúcom príklade uvedieme agenty, ktoré majú niekoľko správaní a teda rôl.

5.3 Distributor

Posledný príklad sa síce môže zdať podobný predošlému, no je o poznanie zložitejší aj keď rieši podobný problém. Tu máme hlavného agenta – Distributora, ktorý sprostredkováva obchod medzi predajcami a kupujúcimi. No nie len tak hocaký obchod, ale chce sprostredkovať taký, pri ktorom on sám najviac získa. Ide na to tzv. greedy technikou, teda vyberá z kupujúcich toho, kto chce najviac zaplatiť a medzi predajcami toho, kto pýta najmenej. Následne medzi nimi zrealizuje obchod, teda príjme z jednej strany peniaze, z druhej tovar a tie potom doručí novým majiteľom. Po zrealizovaní obchodu sa však všetko začína odznova. Pozrie sa na ponuky, ktoré neboli najlepšie a opäť sa snaží zrealizovať obchod s najväčším ziskom. Tento mechanizmus končí, pokiaľ boli zrealizované všetky obchody, alebo ak kupujúci je ochotný zaplatiť menej ako žiada predajca.

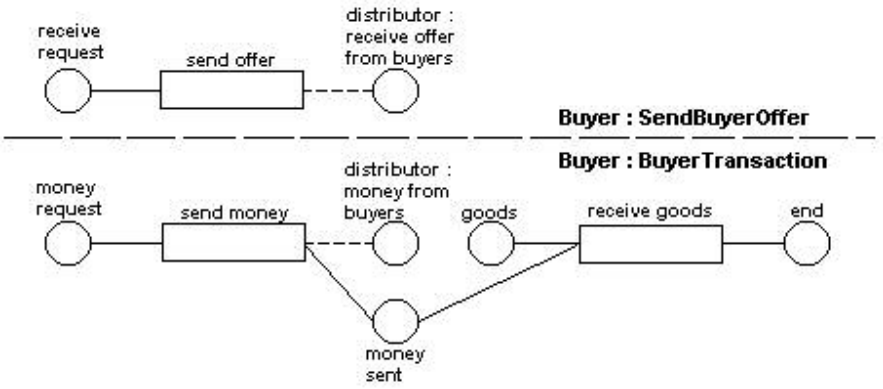
Implementačne je tento príklad najzaujímavejší. Obsahuje tri triedy – DistributorAgent, BuyerAgent a SellerAgent. Každá z nich, na rozdiel od predchádzajúcich príkladov obsahuje minimálne dve správania. DistributorAgent má tri roly. Prvá komunikuje s kupujúcimi, zozbiera od nich ponuky a vyberie najvyššiu. Tú potom odošle tretej. Druhá rola podobne pracuje s predajcami, zozbiera od nich ponuky a vyberie najnižšiu a tú potom odošle tretej. Tieto dve role fungujú paralelne, teda súbežne. Tretia rola začne, až keď predchádzajúce dve jej odošlú výsledky. Tá potom zrealizuje obchod medzi vybraným kupujúcim a predajcom. Zaujímavé je, že na komunikáciu medzi rolami sme použili posielanie správy s adresátom, ktorý je zároveň aj odosielateľom správy. Tento prístup zabezpečuje väčšiu robustnosť modelu. BuyerAgent a SellerAgent majú dve roly. Prvá slúži len na odoslanie ponuky, ktorá je generovaná generátorom náhodných čísel. Druhá zabezpečuje odoslanie peňazí resp. tovaru a následné prijatie tovaru resp. peňazí.

Vzhľadom na to, že tento príklad je najrozsiahlejší, neuvádzame tu celú farbenú Petriho sieť, ktorá ho tvorí, ale iba podsiete, ktoré predstavujú vednotlivé typy agentov. Obrázok 13 zobrazuje agenta Distributora, obr. 14 agenta Buyera a obr. 15 agenta Sellera.

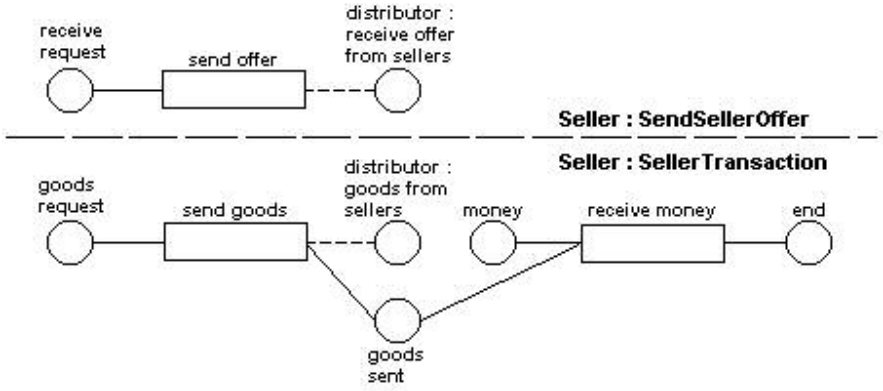


Obrázok 13 : Model agenta typu Distributor

Keďže sme nemohli uviesť sieť celú museli sme nejakým spôsobom naznačiť posielanie správ medzi agentmi. Rozhodli sme sa pre nasledujúcu konvenciu. Čiarkovaná čiara predstavuje posielanie správy inému agentovi. Agent príjemca je naznačený pri mieste, jeho meno je uvedené pred dvojbodkou. Potom nasleduje názov miesta v sieti agenta, kam bude značka reprezentujúca správu doručená. Celá sieť obsahuje až sedem agentov, pričom model každého z nich je dosť rozsiahly a preto by znázornenie celej siete bolo neprehľadné a zabralo veľa strán.



Obrázok 14 : Model agenta typu Buyer



Obrázok 15 : Model agenta typu Seller

Keďže aj výstup, ktorý tento príklad produkuje je pomerne rozsiahly, nájdete ho na priloženom CD v adresári s príkladom. Súbor má názov output.txt. Je tam zachytený typický priebeh tohto príkladu (keďže úplne presný prípad zachytiť nemôžeme, pre veľkú mieru náhodnosti – ponuky predajcov aj kupjúcich sú náhodne generované). Takmer každému páleniu prechodu zodpovedá riadok na výstupe. Najprv sú vyžiadané ponuky od predajcov a kupujúcich a následne sú vyhodnotené, čo je možné vidieť aj na výstupe. Zaujímavým momentom je recyklácia – teda opakovanie obchodovania (distribúcie) až dovtedy, kým sa ešte nejaký obchod uzatvoriť dá.

Záver

V našej práci sme ponúkli prehľad existujúcich formalizmov na modelovanie komunikácie v multiagentových systémoch. Tento prehľad nám mal slúžiť ako odrazový mostík pre identifikovanie dôležitých vlastností komunikačných modelov. Prešli sme cez deterministické konečné automaty, Dooleyho grafy a UML aby sme nakoniec predstavili farbené Petriho siete, ako formalizmus, ktorý sa nám zdal najvhodnejší na túto úlohu. Zistili sme, že dôležité je množstvo a rôzne typy informácií, ktoré dokáže model zachytiť, teda stavová a účastnícka informácia. Okrem toho sa nám zdala dôležitá schopnosť dobre reprezentovať priebeh komunikácie. Výborná reprezentácia priebehu a zachytenie informácie o zámere a stratégii agenta ako u jediného z predkladaných modelov boli najväčšie výhody nami zvoleného modelu – farbených Petriho sietí.

Po podrobnejšom rozbore sme ukázali, ako pomocou farbených Petriho sietí modelujeme komunikáciu v multiagentových systémoch. Predstavili sme náš spôsob implementácie farbených Petriho sietí pomocou agentového prostredia JADE a ilustrovali sme to na príkladoch.

Poskytli sme nástroj na účinné implementovanie modelov, ktoré sú na farbených Petriho sieťach založené. Takto je možné rýchlo a jednoducho dostať navrhnutý systém do podoby programu fungujúceho na multiagentovej platforme JADE, čo sme ilustrovali na príkladoch.

Ďalší výskum by mohol byť smerovaný na dynamické vytváranie komunikačnej siete počas ich existencie. Veríme, že aj v tejto oblasti majú farbené Petriho siete čo ponúknuť.

Použitá literatúra :

AUSTIN J. L. 1975. *How to do things with words* (2nd ed.) Cambridge MA: Harvard University Press, 1975.

BARBUCEANU, M. 1995. *COOL: Language for Describing Coordination in Multi Agent Systems*. In: V. Lesser, L. Gasser (editors), Proceedings of the First International Conference on Multi-Agent Systems, pp. 17-24, San Francisco, USA, AAAI Press, 1995.

COST, R. S. et al. 1999. Modeling agent conversations with colored petri nets, In: Working notes of the Autonomous Agents '99 Workshop on Specifying and Implementing Conversation Policies, 1999.

FIPA (Foundation for Intelligent Physical Agents) Last update 2007 [Cit 24-03-2007] : <http://www.fipa.org/>

Formálne jazyky a automaty. 1998. Skriptá, Univerzita Komenského, Fakulta Matematiky fyziky a informatiky, Bratislava, 1998.

JENSEN, K. 1994. *An Introduction to the Theoretical Aspects of Coloured Petri Nets*. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): A Decade of Concurrency, Lecture Notes in Computer Science vol. 803, Springer-Verlag, pp. 230-272, 1994.

JENSEN, K. 2000. *Petri Nets 2000*. 21st International Conference on Application and Theory of Petri Nets, Aarhus, Denmark, June 26-30, 2000.

NOWOSTAWSKI, M. 2001. *Modelling and visualising agent conversation*, In: Proceedings of the fifth international conference on Autonomous agents, pp. 234-235, Montreal, Quebec, Canada, 2001.

ODELL, J.; PARUNAK, H. V. D. 2002. *Representing social structures in UML*, In: G.Weiss Agent-Oriented Software Engineering II, pp. 1-16, Springer, Berlin, 2002.

ODELL, J.; PARUNAK, H. V. D. 2000. *Extending UML for Agents*, Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, TX, pp. 3-17. 2000.

PARUNAK, H. V. D. 1996. *Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis*, In: Proceedings of the Second International Conference on Multi-Agent Systems, 1996.

PARUNAK, H. V. D. 1998. *An Introduction to Speech Acts and Dooley Graphs*, In: Proceedings of the second international conference on Autonomous agents, pp. 100-107, ACM Press, 1998.

REED, C. 1998. *Dialogue Frames in Agent Communication*. In: Proceedings of the 3rd International Conference on Multi-Agent Systems, pp. 246-253, Paris, France, 1998.

SINGH, M. P. 1998. *Developing Formal Specifications to Coordinate Heterogeneous Autonomous Agents*. Third International Conference on Multi Agent Systems, Paris, France, 1998.

WITTGENSTEIN, L. 1953. *Philosophical Investigations*, Macmillan, New York, 1953.