Multiagentové prístupy k modelovaniu ekonomických systémov

*Jaroslav ZAJAC**

Úvod

Naším cieľom je analýza a rozbor prostriedkov na vytvorenie simulačného prostredia na modelovanie a simuláciu ekonomických subjektov – agentov, napríklad podnikov, ako aj samotný návrh a implementácia konkrétnych riešení a možnosti simulácie a modelovania.

Vytvorenie modelu na simuláciu ekonomických javov, podnikov a hospodárstiev môže byť v prípade všeobecných simulačných prostriedkov náročné a zdĺhavé. Navyše, často skrývajú podstatu simulácie a neumožňujú vhodnú mieru abstrakcie od simulačných prvkov.

Ako príklad môžeme uviesť nástroj na simuláciu podnikateľa na trhu. Druhým nástrojom by mohol byť prostriedok na modelovanie nejakého trhu. Tieto dva prostriedky však medzi sebou priamo nekomunikujú, preto skĺbenie takýchto dvoch modelov do jediného môže byť náročné a v mnohých prípadoch aj nepraktické, resp. nemožné. Tento všeobecný príklad ukazuje hlavnú nevýhodu používania špecializovaných prostriedkov.

Simulátory, ktoré umožňujú tvorbu modelov určitej skupiny, sú relatívne populárne. Takéto prostriedky nie sú špecializované na jediný model, ale na jednu množinu modelov. Dá sa napríklad nájsť prostriedok na tvorbu, modelovanie a simuláciu systémov, v ktorých vystupujú agenti. Závisí od konkrétneho produktu, aká miera abstrakcie sa zvolí, resp. v akej miere sa prikláňa k všeobecným prostriedkom a v akej k špecializovaným prostriedkom. Tam, kde je potrebná jednoduchosť a zrozumiteľnosť, sa uplatnia špecializované prostriedky. Naopak, kde sa vyžaduje komplexnosť tvorby modelov, treba použiť všeobecné nástroje. Na pomyselnej čiare medzi špecializovanými a univerzálnymi prostriedkami je kompromisom hybridný systém – teda taký, ktorý umožňuje používateľovi tvoriť modely určitej skupiny.

Cieľový systém má spĺňať nasledovné požiadavky:

- základný model a simulátor na hierarchické modelovanie a simuláciu;
- podpora modelovania podnikov a hospodárstiev;

* doc. Ing. Jaroslav ZAJAC, CSc., Slovenská technická univerzita, Fakulta elektrotechniky a informatiky, Katedra ekonómie a manažmentu, Ilkovičova 3, 812 19 Bratislava 1; e-mail: jzajac@elf.stuba.sk

- rozšíriteľnosť, otvorenosť systému;
- podpora využitia evolučných algoritmov a neurónových sietí;
- podpora editácie modelu používateľom a vizualizácie výsledkov.

Vytváraný systém by mal byť nezávislý od konkrétnej implementácie editácie modelu, vizualizácie výsledkov, ukladania a obnovy stavu simulácie a použitých algoritmov neurónových sietí a evolučných algoritmov.

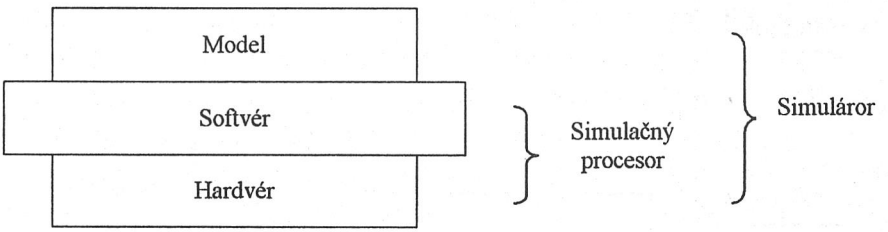
Je potrebné charakterizovať základné črty problému:

- problém ako súčasť tvorby simulačných prostriedkov;
- problém z hľadiska simulácie agentov – podnikov a prostredia;
- problém z pohľadu použitia matematického aparátu, algoritmov a ich opisu;
- možnosť prechodu na vytváraný systém, jeho rozšírení, otvorenosti a jednoduchej adaptácie používateľa.

Existuje veľké množstvo literatúry (napr. [1]), ktorá popisuje základy stavby simulačného prostriedku. V tejto časti preto spomenieme len základné princípy. Bude prevažovať objektovo orientovaný prístup ([3], ale aj [2]).

Predovšetkým je potrebné vysvetliť, čo je to simulačný prostriedok, simulátor, model a aké sú ich vzájomné vzťahy. Základná štruktúra je znázornená na obrázku 1.

Obrázok 1



Takýto simulačný procesor musí umožňovať riadenie ekonomických procesov: správanie podnikov, obsluhy ich požiadaviek, vplyvov neurčitosti a simuláciu prostredia. Na možnostiach, ktoré poskytuje simulačný procesor, je potom možné stavať model. Tento model bude zasahovať do parametrov a stavov prvkov, ktoré procesor poskytuje.

V každom simulačnom prostriedku je nutné simulovať *čas*. V prípade ekonomických simulácií je tento predpoklad samozrejmý. Veľa ekonomických procesov sa deje v časových skokoch, napríklad burza sa otvára a zatvára v stanovených hodinách, pričom jeden deň by sa považoval za jednotku času. Navyše, tieto časové skoky majú pravidelný charakter, takže ich možno simulovať použitím modelovania času, *technikou s pevným krokom*. Druhou alternatívou je *technika*

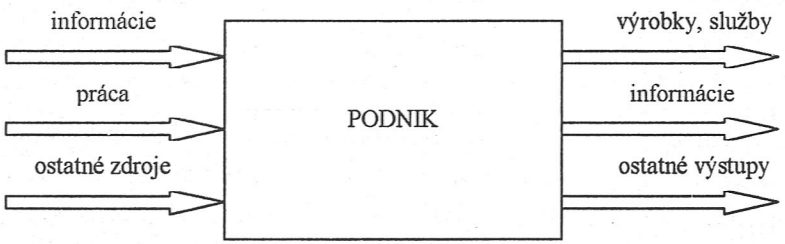
nasledujúcej udalosti, pri ktorej sú jednotlivé udalosti rozložené v čase nerovnomerne. Ako sme už uviedli, z ekonomického hľadiska je výhodné použiť prvú menovanú techniku.

Treba si predovšetkým uvedomiť, že klasický simulačný aparát nie je v prípade ekonomických javov postačujúci. V ekonómii sa nevytvára čakajúci rad, aby sa mohli požiadavky postupne vybavovať. Naopak, treba tu simulovať sociálne problémy, pohnútky, pamäť, mutualistické vzťahy a pod. Práve preto je potrebné prikročiť k menej tradičným prvkom. Príkladom môžu byť evolučné algoritmy [4; 5], alebo neurónové siete [6; 7]. Práve tieto vlastnosti môžu byť obalené objektovým prístupom [7; 2]. Prostredie, podnik, ľudia môžu byť jednotlivými objektmi – agentmi. Na tomto mieste nebudeme uvádzať možnosti evolučných algoritmov a neurónových sietí, iba rozoberieme problémy pri skúmaní zvolenia vhodnosti miery abstrakcie subjektov a ich pridelenie jednotlivým agentom.

Podnik sa dá popísať ako učiaca sa spoločnosť. Na základe vedomostí robí svoje kroky a na základe výsledkov upravuje svoje vedomosti. Ide tu o evolučný proces. Z vnútorného hľadiska možno podnik definovať ako súbor osobných, nehmotných a hmotných zložiek podnikania. K podniku patria veci, práva a iné majetkové a duchovné hodnoty. V každej simulácii je nevyhnutné zaoberať sa hmotnými hodnotami. Práve duchovnými hodnotami ako znalosťami sa podnik môže ďalej rozvíjať, a preto tieto časti nie je možné ignorovať.

Podnik, pri určitej miere abstrakcie, je možné znázorniť použitím nasledovnej schémy (obrázok 2).

Obrázok 2



Ako vyplýva z tejto schémy, dôležitou súčasťou sú *informácie*, čiže *vedomosti* – či už na úrovni vstupov alebo výstupov. Samotná činnosť podniku je transformácia vstupov na výstup. Do informácií na vstupe zahŕňame aj vlastné vedomosti podniku. Jednoducho povedané, v jednom kroku evolučného procesu sa vstupy „pretransformujú" na výstup a tieto výstupy prispievajú do vstupov pre ďalší krok. Spôsobom prezentácie samotných vedomostí sa zaoberá nasledovná časť.

Modelovanie fungovania podnikov

V celom systéme sa bude nachádzať množstvo podnikov. Každý z nich bude nezávislý od ostatných, ale v určitej úrovni budú predsa len od seba závislé – majú totiž jednotný trh prostriedkov, na ktorom si budú konkurovať.

Prostredie sa dá vo všeobecnosti popísať nasledovnou štruktúrou:

- samotné vnútro podniku – zdroje, štruktúra, vedomosti;
- konkurenčné prostredie – zákazníci, dodávatelia, konkurenti;
- interakčné prostredie – akcionári, záujmové skupiny, veritelia, priemysel a pod.;
- spoločenské prostredie – demografické, sociálne, historicko-kultúrne, politické a právne aspekty;
- globálne prostredie – globalizácia trhov.

Vzhľadom na to, že modelovať všetky tieto zložky prostredia je veľmi náročné, je potrebné zvoliť určitú voľba abstrakciu. Musíme si uvedomiť, že nielen prostredie vplýva na samotné podniky, ale práve podniky sa podieľajú na vytváraní celkového prostredia (resp. jeho konkurenčnej časti).

Pri určitej miere abstrakcie si uvedomujeme, že samotné prostredie sa dá popísať podobne ako podnik. Jednotlivé prostredia, ktoré podniky vytvárajú, môžu mať určité variácie, ktoré si konkurujú. V tomto prípade platí, že výhodnejšie prostredie vyhráva. Podniky, ktoré boli „odchované" v starom prostredí, nemusia nutne prosperovať v novom prostredí a môžu zaniknúť. Na modelovanie takejto situácie, pri ktorej sa vyvíja aj prostredie, je potrebné vytvoriť systém, ktorý by bol schopný hľadať rôzne alternatívy prostredí a vyberať medzi nimi na základe vnútorných, čiže podnikových faktorov.

Pri modelovaní podnikov a prostredia treba brať do úvahy aj samotnú spoločnosť, ktorá vytvára aj prostredie, aj podniky. Núkajú sa tu nasledovné voľby:

- podnik zodpovedá jednému agentovi, ktorý je podmnožinou spoločnosti;
- podnik je skupina ľudí a ich vzťahov, pričom človek zodpovedá agentovi.

V prvom prípade je teda elementárnou časticou *podnik*, ktorý tvoria viacerí ľudia a ich vzájomné vzťahy. Toto chápanie znemožňuje presný význam migrácie ľudí do iných podnikov – takýto pohyb je nutné simulovať. Má to však tú výhodu, že agentov je podstatne menej ako v druhom spôsobe. Vplyv spoločnosti môže tvoriť akýsi pomyselný podnik, ktorého správanie by zodpovedalo globálnemu správaniu ľudí v krajine. Toto správanie vstupuje do podniku spolu s ostatnými sociálnymi a ekonomickými ukazovateľmi.

V prípade druhom je agent na úrovni *jednotlivca*. To znamená, že podnik tvorí skupina agentov, pričom každý z agentov má svoje pohnútky. Toto riešenie má tú nevýhodu, že je zložité určiť celkové pohnútky a zámery podniku ako celku.

Evolučné algoritmy

Evolučné algoritmy sú matematickým aparátom na hľadanie globálneho minima (resp. maxima) danej *n*-rozmernej funkcie. Globálny extrém sa väčšinou nedá nájsť analyticky. Preto vznikli rôzne postupy na hľadanie globálnych extrémov a jeden z postupov predstavujú aj *evolučné algoritmy*. Hlavná myšlienka pri použití evolučných algoritmov je inšpirovaná biologickým svetom. Ide o prirodzený výber (najsilnejších) jedincov. Sú zavedené pojmy: chromozóm, populácia, mutácia, kríženie. Princíp je jednoduchý: z celej populácie sa vyberú najsilnejší jedinci (rodičia), na základe ich genetickej informácie, použitím kríženia a následnou aplikáciou mutácie, vznikne nová generácia. Podrobný popis celého algoritmu je uvedený v [8].

Evolučné algoritmy sa dajú rozdeliť na tri skupiny: • genetické algoritmy • evolučné stratégie • evolučné programovanie;

Zásadný rozdiel medzi týmito troma algoritmami je v pohľade na kríženie a mutáciu. V súčasnom období však tieto tri princípy splynuli do jedinej podoby. Genetické algoritmy využívajú vo veľkej miere kríženie a mutáciu len ako „šum", resp. spôsob ako vymaniť systém z lokálneho extrému. Všetky tieto postupy veľmi rýchlo konvergujú do lokálnych extrémov. Pri použití nesprávneho pomeru kríženia, veľkosti populácie a miery mutácie systém nemusí konvergovať, bude vykazovať neurčité a nepoužiteľné výsledky. Preto je potrebné dôkladne preskúmať túto problematiku a vyvodiť vhodné závery. Za jednu vetvu genetických algoritmov by sa dalo považovať aj *genetické programovanie*, v ktorom sa ako chromozómy používajú programy, resp. sady príkazov. Tie môžu definovať správanie nejakého celku. Tento poznatok sa dá využiť pri modelovaní jednotlivých agentov.

V podstate je toto rozdelenie širšou otázkou. Nie je jednoduché vybrať jediný spôsob. Ak sú totiž obe populácie totožné, paralela s reálnym svetom je zrejmá, a preto sa táto možnosť javí ako samozrejmá. Opak je však pravda. Na simuláciu viacerých podnikov by sme potrebovali rovnaké množstvo agentov, ako aj prvkov populácie nesúcich chromozómy. Na účely genetických algoritmov musí byť táto populácia rozsiahla. Navyše, rodičovské prvky po krížení zaniknú, čo sa nedá povedať o agentoch. Na druhej strane, ak populácie budú rozdielne, nie je zrejmé, koľko prvkov nesúcich chromozómy má prislúchať jednému agentovi.

Neurónové siete

Bližší popis neurónových sietí možno nájsť napríklad v [6; 7; 9]. Neurónové siete sú vo svojej podstate veľmi široký okruh algoritmov. Dokážu obstáť aj vo veľmi náročných situáciách. Nie sú však všeliekom na všetko. Najprv je

nevyhnutné urobiť dôslednú analýzu problematiky, prispôsobiť algoritmy a vzťahy, a až potom môžu neurónové siete podávať uspokojivé výsledky. Na to, aby mohli neurónové siete fungovať, ich treba na začiatku trénovať. Tento proces nie je vo všeobecnosti dôkladne popísaný a líši sa prípad od prípadu. Keďže neurónové siete majú svoje vstupy a výstupy, na trénovanie sa používajú dvojice vstupov a výstupov. Na základe množiny takýchto dvojíc sa neurónová sieť nejako naučí. Ako sme už uviedli, neurónová sieť má svoje vstupy a výstupy, preto ju možno chápať z matematického hľadiska ako funkciu.

Neurónové siete nachádzajú široké pole využitia. Preto je možné urobiť nasledovné rozdelenie využitia: *klasifikáciu, predikciu, zovšeobecnenie, aproximáciu.* Dajú sa, samozrejme, nájsť aj iné spôsoby využitia, ale tie sú iba kombináciou vymenovaných metód.

Klasifikáciou sa rozumie schopnosť siete ohodnotiť, resp. zaradiť kombináciu vstupov podľa naučených parametrov. Príkladom môže byť situácia, keď na vstupe je informácia o stave podniku a sieť na základe tohto vstupu vygeneruje popis kroku, ktorý je potrebné vykonať.

Predikciou sa chápe predpovedanie budúceho vývoja na základe minulých dejov, napríklad vývoj hrubého domáceho produktu, keď sa na základe takejto informácie bude môcť podnik rozhodovať.

Zovšeobecnenie je taký typ úlohy, pri ktorom je neurónová sieť trénovaná na neúplnej množine vstupov a výstupov. Po skončení priebehu trénovania bude sieť vedieť určiť správne výstupy a aj chýbajúce prvky trénovanej množiny.

Aproximácia slúži na matematickú aproximáciu funkcií. Aproximácia sa dá použiť podobne ako predikcia.

Existujú viaceré spôsoby vnútornej organizácie neurónových sietí – od sietí ex ante, až po siete so spätnou väzbou a rekurentné siete. K trénovaniu, resp. učeniu sa neurónových sietí je potrebné povedať aj to, že doposiaľ uvádzané typy neurónových sietí bolo nevyhnutné trénovať. Musel existovať spôsob mimo systému, ktorý vedel ohodnotiť vstupy. Takýto proces sa nazýva učenie sa s učiteľom. Sieť sa teda neučila na základe svojich skúseností. Tento deficit sa dá odstrániť použitím genetických algoritmov. Spomenuli sme už, že neurónové siete sú vo svojej podstate matematickými funkciami. Tento fakt priamo predurčuje použitie genetických algoritmov na učenie sa neurónových sietí. Populáciu pre genetické algoritmy budú tvoriť jednotlivé parametre neurónových sietí. Tieto parametre budú tvorené chromozómami a budú podliehať kríženiu a mutácii. Takto sa selektívnym spôsobom nájdu „schopnejšie" parametre pre danú neurónovú sieť.

Neurónové siete významne prispievajú k tvorbe modelov správania vedomostí. Bez použitia sietí by sa tieto modely museli tvoriť použitím *fuzzy* logiky, resp. iného matematického aparátu.

Modelovanie vedomostí

Vedomosti možno rozdeliť na dve skupiny: a) informácie z minulosti; b) schopnosť rozhodovania, utvárania záverov.

Tieto dve skupiny priamo evokujú možnosť rozdeliť aj vedomostné konanie na dve skupiny:

- *reflexy* – z nejakého vstupu na základe nejakých vnútorných pohnútok dospieť k záveru (reflex);
- *schopnosť adaptácie, abstrakcie a myslenia* – z poznatkov minulých a terajších dospieť k poznaniu (adaptácia).

Reflexívne konanie sa veľmi jednoducho dá modelovať použitím klasických neurónových sietí. Bližší popis použitia neurónových sietí je uvedený na inom mieste v tomto príspevku. *Adaptívne konanie* sa ľahko modeluje použitím genetických algoritmov. Spojenie genetických algoritmov a neurónových sietí je prirodzeným počinom, či už v oblasti modelovania vedomostí na tejto úrovni, alebo v oblasti matematického aparátu. Preto by sa pri modelovaní vedomostí malo vychádzať z tohto princípu. Výber takéhoto princípu modelovania vedomostí však môže priniesť so sebou aj určité úskalia – adaptácia môže trvať veľmi dlhý čas. Navyše, ak sa zvolí nevhodné prostredie, adaptácia nemusí nikdy nastať, čo môže viesť k nepoužiteľným výsledkom.

Samotné podniky by mali obsahovať nejaké vedomosti, vlastnosti, na základe ktorých sa budú rozhodovať, a nejaké prostriedky. Ako naložia s prostriedkami, to je úlohou rozhodovacieho procesu každého podniku. Dôraz sa musí klásť na dobrý návrh vedomostného podsystému, resp. jeho správy. To je však kompetenciou každého podniku osobitne. Simulačný prostriedok by nemal do procesu rozhodovania nijako zasahovať.

Problémom modelovania podniku zostáva iba to, ako bude podnik využívať svoje vedomosti. Táto úloha však prináleží modelovaniu vedomostí, takže modelovanie podniku nemusí podliehať prísnym pravidlám analýzy.

Pri modelovaní prostredia je nevyhnutné, aby sa postupovalo precízne. Celá simulácia je ovplyvnená výberom a charakteristikou prostredia. Tu stoja fakty hovoriace za to, aby sa prostredie modelovalo úplne. Dôvodom je dôveryhodnosť prostredia pre danú simuláciu.

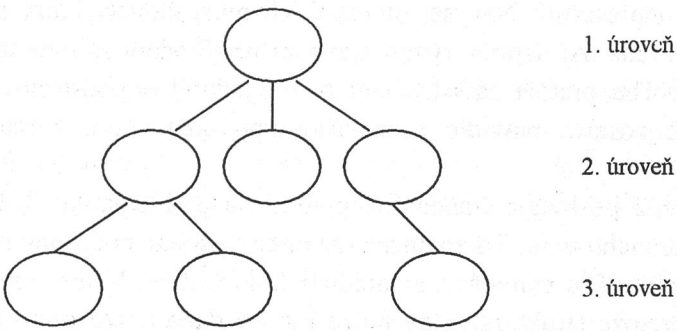
Vo všeobecnosti však treba pripomenúť, že modelovanie prostredia by malo podliehať vysokokvalitnému a podrobnému návrhu, keďže ide o esenciálnu súčasť simulácie.

Prostredie, podobne ako podniky, musí byť ovplyvniteľné používateľom. Ten do procesu modelovania prostredia bude zasahovať pridávaním, resp. menením a mazaním pravidiel, napríklad zavádzanie daní, podmienok prežitia a pod.

Agregácia v hierarchických štruktúrach

Ide o ekvivalent stromu z matematickej teórie grafov. Vystupujú tu uzly a ich orientované prepojenia. Uzly sú prepojené tak, že každý uzol môže mať svojho *rodiča* alebo *nasledovníkov*. Pre daný uzol môže existovať len jediný rodičovský uzol. Nasledovníkov môže byť viacero. Takáto forma vytvára vrstvy, pričom na najvrchnejšej vrstve je jediný, tzv. *koreňový* uzol a na najnižšej vrstve sú uzly bez potomkov, ktoré nazývame *listy*. Celý tento opis demonštruje schéma na obrázku 3.

Obrázok 3



Každý uzol nesie v sebe logiku spracovania údajov, ktoré mu poskytnú jeho nasledovníci. Po spracovaní týchto údajov poskytne výsledok svojmu rodičovi. Hlavným problémom je, ako prezentovať údaje od nasledovníkov. Podľa jednotlivých druhov informácií možno preto problematiku typu údajov rozdeliť na skupiny:

- v celom systéme existuje len jediný typ údajov;
- jedna vrstva (úroveň) poskytuje vyššej vrstve jediný typ údajov, pričom ten sa môže meniť od vrstvy k vrstve;
- jednému uzlu sa poskytuje jediný typ údajov;
- rôzne typy údajov.

V poslednom, najvšeobecnejšom prípade je situácia veľmi zložitá. Môžu tu totiž nastať prípady, keď uzol poskytne svojmu rodičovi taký typ informácie, ktorý rodič nevie spracovať. V takomto prípade je nevyhnutné informáciu transformovať z jedného typu do druhého. Touto transformáciou však vzniká ďalší problém. Premenou údaju môže vzniknúť chyby, nedostatky. Neprekonateľným problémom je, ak je nevyhnutné uskutočniť transformáciu dvoch nekompatibilných

typov. Vtedy je najvhodnejšie výsledok ani nepoužiť, aby sa do celého systému nevnášal zbytočný a ničím neopodstatnený šum, pretože dosiahnutá informácia je zbytočná. Na druhej strane môže byť žiaduce využiť aj takúto informáciu. Ak už daná informácia v systéme existuje, mala by niečo vyjadrovať. Jednoduchou filtráciou tejto informácie sa stráca presnosť celého výpočtu. Tieto dve protichodné riešenia v princípe vylučujú použitie posledného prípadu.

Tretím prípadom, keď nasledovníci jedného uzla poskytujú svojmu rodičovi jediný typ informácie, transformácia musí nastať už u samotných nasledovníkov. Tu ale existuje reálna hrozba z posledného prípadu. Samotné uzly predstavujúce nasledovníkov musia byť navrhnuté tak, aby dodávali taký údajový typ, ktorý je pre daného rodiča prijateľný. Znamená to, že uzly, pre ktoré už existuje matematický aparát počítania výstupu, je nevyhnutné prerobiť. To môže mať za následok vnášanie nepresností. Navyše, prerobiť ich musí človek, ktorý sa v danej problematike vyzná. Aj napriek týmto zjavným nevýhodám sa tretí spôsob javí ako najlepšia voľba, pretože závislosť údajov pre jednotlivé podstromy neexistuje, pričom ale zostáva pravidlo rovnakého typu údajov pre koreň každého podstromu.

Prípad číslo 2 poskytuje štandardný pohľad na problematiku. Údaje z jednej vrstvy sú jedného typu. To znamená, že treba vykonať podrobný návrh údajov, ktoré si uzly môžu vymieňať, a zaradiť ich do vrstiev. Nejde však o triviálny problém, pretože štruktúra uzlov môže byť už daná a v takomto prípade je tento spôsob nepoužiteľný. Princíp je výhodný pre novovytvárané hierarchické štruktúry.

V prvom prípade je situácia zdanlivo najjednoduchšia, keďže v celom systéme existuje jediný typ údajov. Neexistujú tu žiadne transformácie typov. Ide však o syntetický prípad, pretože len máloktorý systém spĺňa požiadavku na rovnaký typ údajov.

S počtom typov údajov úzko súvisí aj otázka významu údajov. Môžu nastať dva prípady. Prvým je situácia, keď údaje medzi uzlami sú zmysluplné, a situácia, keď také nie sú. V prvom prípade je na vstupe aj na výstupe zmysluplná informácia. V skrytých vrstvách však údaje nedávajú, resp. nemusia dávať obraz reálneho sveta. Nie je definované, ako tieto informácie súvisia so vstupnými údajmi a s výstupom. Isté je len to, že *nejako* súvisia. Preto pri prenose informácií medzi jednotlivými uzlami nemusia všetky údaje poskytované rodičovi dávať zmysel. Rodič, spolu s údajmi od ostatných nasledovníkov, ich môže premeniť na reálnu veličinu. Otázka zmysluplnosti údajov vzniká aj v prípade použitia jediného typu informácií v celom systéme. Ak by sme si napríklad určili, že jediným typom je reálne číslo, tak niektoré uzly môžu týmto reálnym číslom definovať kusovú produkciu, iné uzly môžu definovať niečo abstraktné.

Na samotnú transformáciu údajov, resp. samotnú správu transformácie je nevyhnutné urobiť rozdelenie z hľadiska miesta, kde sa spravuje transformácia informácií do požadovaných typov údajov:

- uzol (rodič) robí transformáciu na vstupe;
- uzol (nasledovník) robí transformáciu na výstupe;
- globálna transformácia cez prostredníka.

V prvom prípade ide o transformáciu vstupov. To znamená, že nasledovníci môžu na výstupe poskytovať rôzne typy údajov. Rodič si ich dokáže transformovať. Rodič pritom musí poznať jednotlivé typy údajov na výstupoch nasledovníkov a musí ich v procese transformácie akceptovať na vstupe. Samotná logika transformácie je daná v rodičovi. V druhom prípade sa o transformáciu stará nasledovník, ktorý musí poznať vstupný typ údajov svojho rodiča. Znamená to toľko, že pri zmene logiky rodiča a jeho vstupu sa musí prepísať aj transformácia robená u jeho nasledovníkov. V prvom prípade stačilo rodičovi pridať transformáciu na nový typ bez zásahu do údajov svojich nasledovníkov. Tretia možnosť prináša značné zjednodušenie celej situácie. Nasledovník svoj údaj transformuje do vopred definovaného nezávislého typu a rodič si tento údaj pretransformuje naspäť do svojho vlastného typu. V prípade zmeny logiky rodiča alebo jeho nasledovníkov nie je potrebné prerábať aj pridružené uzly a ich adaptáciu na nové typy údajov. Celá komunikácia medzi uzlami sa deje na nezávislom type údajov. Nevýhodou tohto riešenia je samotný pojem *nezávislý typ*: v prípade, že takýto typ údaju je zle navrhnutý, môže sa vyskytnúť uzol, ktorý nebude môcť vykonať konverziu svojich typov do nezávislého typu. V takomto prípade je nutné nezávislý typ pozmeniť a rozšíriť, čo ale vedie k náročnému prepísaniu, resp. údržbe logiky transformácie typov údajov pre jednotlivé uzly.

Transformáciu možno popísať ako nejakú čiernu skrinku, do ktorej niečo vstupuje a niečo z nej vystupuje. Samotná konverzia vstupu na výstup by však nemala mať trvalý charakter, čiže mala by existovať možnosť meniť parametre transformácie. Takto je potom možné dosiahnuť vytvorenie konverzie, ktorá je riadená parametrami, ktoré môžu byť výstupom inej transformácie. Táto analýza sa však nebude zaoberať vnútrom transformácie. Namiesto toho rozdelí jednotlivé spôsoby nastavovania parametrov pre jednotlivé transformácie: parametre nastavované lokálne, parametre získané od rodičov, iné získavanie parametrov.

Ak sú parametre transformácie *nastavované lokálne*, čiže každý uzol nastavuje parametre svojich vlastných transformácií bez ohľadu na okolie, môžeme dostať ťažko popísateľné výsledky. Stále tu existuje problém interpretácie nezmysluplných údajov. V podstate ani samotné parametre nemusia dávať zmysel. Proces nastavovania parametrov nemusí nikdy konvergovať k výsledku, pretože neexistuje nejaká globálna správa nad parametrami transformácií. Uzly si ich nastavujú

svojvoľne. Druhý spôsob vnáša správu, aj keď ešte nie globálnu. Pri tomto spôsobe každý *rodič informuje svojich nasledovníkov* o úspechu ich nastavenia a voľby parametrov. To znamená, že každý rodič musí obsahovať rozhodovaciu logiku, ktorej výsledkom je výber oznamu svojim nasledovníkom. Toto rozhodovanie však nemusí byť exaktne nastavené. Môže taktiež obsahovať parametre. A to je práve nevýhoda tohto riešenia. Okrem nastavovania parametrov transformácie je nutné nastaviť parametre aj pre rozhodovanie. V poslednom, treťom, prípade môžu nastať situácie, keď sú všetky parametre všetkých transformácií ovládané *centralizovane*. Môže sa to dokonca diať aj formou genetických algoritmov. Jednotlivé hodnoty parametrov tvoria jeden chromozóm. V extrémnom prípade zistíme, že samotná transformácia môže podliehať genetickému programovaniu. Je však otázne, nakoľko by tieto metódy dokázali konvergovať k výsledku a či by sa systém veľmi rýchlo nedostal k nepredvídateľnému správaniu.

Doteraz sa hovorilo len o získavaní informácií smerom od potomka k rodičovi. Istým zovšeobecnením môže byť aj prípad, keď rodič má prepojenie aj na potomkov svojich nasledovníkov. Potom rozlišujeme tieto prípady:

- jediné prepojenie nasledovník – rodič;
- prepojenie rodič – nasledovník na ľubovolnej vrstve (úrovni);
- prepojenie rodič – nasledovník na vrstve vzdialenej maximálne N vrstiev.

Tieto prípady dávajú rodičom možnosť získavania údajov nielen od bezprostredných nasledovníkov, ale aj od ich nasledovníkov na nižších vrstvách. To znamená, že rodič je ovplyvňovaný aj nižšími úrovňami. Extrémnym prípadom by mohla byť situácia, keď koreň má prepojenie na všetky uzly. Vtedy nemusí ísť o agregáciu údajov, ale o nejaké zovšeobecnenie. Takýto systém by bolo možné modelovať aj použitím prvého prípadu, čiže stromu, avšak nastavovanie parametrov a transformácií by nemuselo byť dokonalé. Požadovaný model by nemusel vykazovať vierohodné výsledky. Nevýhodou takého zovšeobecnenia je jeho nesmierna zložitosť a nároky na návrh.

Na zovšeobecnenie tejto problematiky je možné zadefinovať obojsmerné toky údajov. V takomto prípade sa maže rozdiel medzi rodičom a potomkami. Zjavnou nevýhodou je náročné zisťovanie vrstiev, určovanie obojsmerných transformácií a v neposlednom rade cykly v grafe.

Simulačný prostriedok by mal tvoriť ucelenú časť a mal by poskytovať možnosti spolupráce všetkých podsystémov. Podsystémy tvorí hlavný modul na prepojenie spracovania, simulačný procesor a modul pre model a samotné modelovanie. Voľba modulu na modelovanie závisí od možnej interpretácie agentov, čiže buď agentom budú zodpovedať podniky, alebo ľudia.

Keďže daná problematika je veľmi rozsiahla, je nevyhnutné exaktne definovať možnosti použitia systému. Cieľom systému nie je pokrytie všetkých kombinácií

systémov, štruktúr, modelov a simulačných procesorov. Namiesto toho je systém obmedzený na určitú množinu problematiky, ktorú tvoria hierarchické štruktúry a multiagentové systémy.

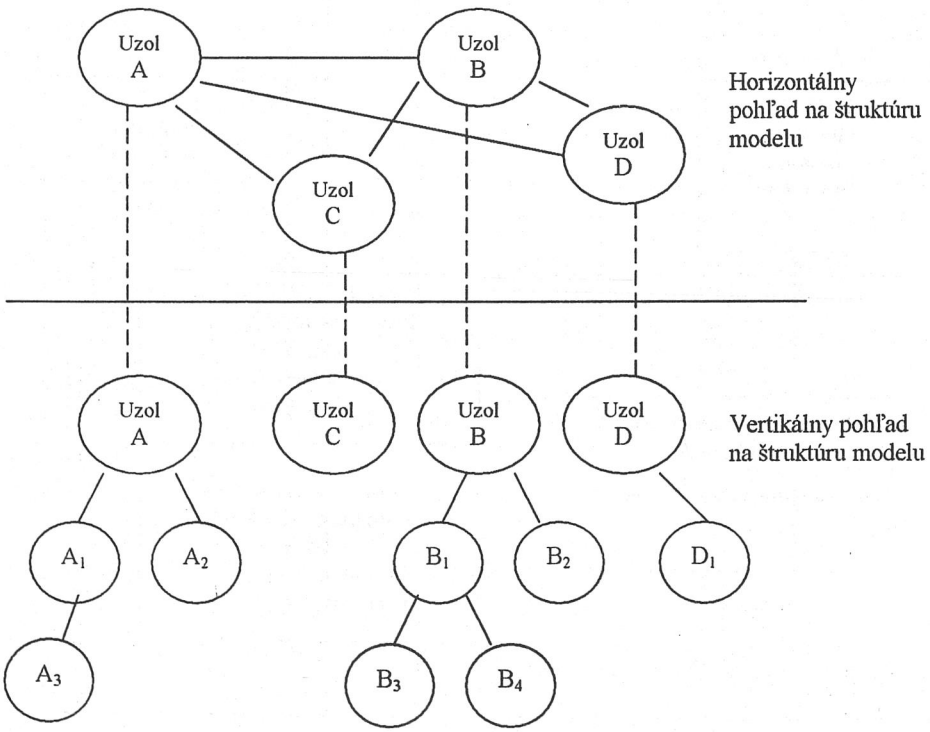
Výsledný systém má poskytovať používateľovi nástroje, nie riešenia. Ide hlavne o voľbu abstrakcie modelu, simulačného procesoru. Namiesto toho bude všeobecné a následné ekonomické súvislosti a modelovanie budú nadstavbou tohto simulačného modelu a procesoru.

Model

Model si môžeme predstaviť ako viacero stromov, pričom existujú prepojenia, resp. vzťahy, medzi jednotlivými koreňmi. Jednotlivé korene stromu nazveme *uzlami*. Každý uzol má svoj vlastný vnútorný stav. Informácie o tomto stave môže dávať k dispozícii okoliu, pomocou zatiaľ nešpecifikovaného rozhrania. Rovnakým spôsobom môže uzol získavať nastavenia svojich stavov – čiže pre každú *vlastnosť*; atribút v danom uzle môžu, ale nemusia existovať prístupové mechanizmy na zápis alebo čítanie.

Na ilustráciu vzťahov a celkovej hierarchie slúži schéma na obrázku 4.

Obrázok 4

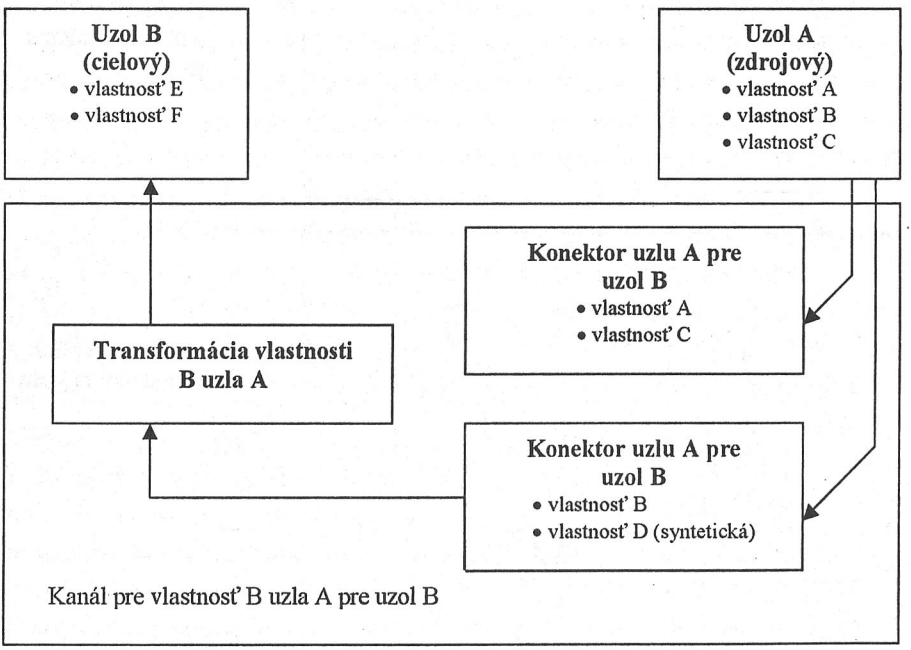


Uvedená schéma je len príkladom štruktúry systému. Znázorňuje však podstatné aspekty modelu – horizontálne rozvinutie a komunikáciu jednotlivých uzlov v jednej rovine; a vertikálnu stromovú hierarchiu. Treba podotknúť, že aj keď na jednej vrstve napríklad uzly A a B komunikujú medzi sebou, ich nasledovníci priamo komunikovať nemôžu (napr. uzly A_1 a B_1). Je to dané tým, že uzol A svojich potomkov ukrýva, zapuzdruje. Takto je možné dosiahnuť efekt abstrakcie. Na jednej strane môže mať niektorý uzol potomkov viacero, na strane druhej nemusí mať žiadneho. Naopak, medzi nasledovníkmi jedného rodiča existujú horizontálne vzťahy. (Poznámka: Na obrázku 4 sú aj samotné uzly A, B, C a D potomkami jedného rodiča, keďže sú na jednej vrstve. Ten však nie je zakreslený.)

Aj keď vo vertikálnom pohľade existujú vzťahy medzi rodičom a jeho potomkami, tieto vzťahy nie sú rovnakého typu ako v prípade horizontálnych prepojení. Vertikálne vzťahy sú určené konkrétnou implementáciou. Reprezentujú priamy prístup k dcérskym uzlom.

K horizontálnemu pohľadu je vhodné napísať, že prepojenia jednotlivých uzlov (vzťahy) sú obojstranné. Medzi dvoma uzlami existuje len jediné takéto spojenie. V prípade, že medzi dvoma uzlami vzťah neexistuje, spojenie medzi nimi je definované, aj keď sa ním neprenášajú žiadne údaje. Z dôvodov prehľadnosti schémy, nie sú zakreslené všetky prepojenia. Vzťahy medzi uzlami na jednej vrstve nie sú jednoduchými prepojeniami. Pohľad na vzťah dáva obrázok 5.

Obrázok 5



Obrázok 5 poskytuje schematický pohľad na získavanie jednej vlastnosti zo vzdialeného uzla. Pre komunikáciu a získavanie takýchto vzdialených vlastností platia nasledovné pravidlá:

1. vzdialený konektor a lokálna transformácia tvoria komunikačný *kanál*;
2. každý uzol pre jednu svoju vlastnosť definuje jedinečný konektor pre vzdialený uzol;
3. jeden konektor môže zahŕňať viacero vlastností a môže definovať aj nové vlastnosti, ktoré pridružený uzol nedefinuje;
4. pri získavaní vzdialenej vlastnosti je vzdialená vlastnosť uzla skrytá a viditeľná je len cez príslušný konektor;
5. získaná vlastnosť môže prejsť transformáciou na strane lokálneho uzla, pričom táto transformácia je jedinečná pre jeden vzťah.

Takto definovaný kanál definuje prepojenie jednej vlastnosti zo vzdialeného uzla pre lokálny uzol. To znamená, že prístup rovnakého uzla k inej vzdialenej vlastnosti je tvorený iným kanálom, ktorý pozostáva z (prípadne) iného konektora a inej transformácie. Pre prístup odlišného uzla k rovnakej vzdialenej vlastnosti bude definovaný nový kanál, ktorý bude mať určite odlišný konektor a transformáciu. Čiže dvojica uzlov tvorená zdrojovým a cieľovým uzlom obsahuje viacero kanálov, pričom v rámci tohto jednosmerného vzťahu môžu byť konektory (na strane vzdialeného uzla) prijaté. Tieto konektory sa však nesmú poskytnúť inému cieľovému uzlu. Dôvodom tohto obmedzenia je fakt, že zdrojový uzol takto definuje svoje správanie voči ostatným uzlom. A pre rôzne uzly sa správa rôznymi spôsobmi. Dá sa povedať, že konektor definuje správanie uzla navonok. Jednotlivé vlastnosti uzla majú objektívny, resp. pravdivý charakter. Konektor tieto pravdivé vlastnosti môže navonok pozmeniť. Pre rôzne uzly budú cez rôzne konektory definované rôzne typy zmien. Lokálny uzol, ktorý dostane takýto pozmenený údaj, môže použitím svojej vlastnej transformácie tento údaj zmeniť na pôvodný. Je samozrejmé, že voľba tejto transformácie je na lokálnom uzle a nemusí byť inverznou transformáciou, ktorú vykonal vzdialený konektor.

V tejto časti sme definovali pojem *uzol*, ktorý priamo korešponduje s pojmom *agent*. Takto zvolený model je pripravený na modelovaniu multiagentových systémov. Navyše, definuje aj vertikálnu štruktúru, takže je možné uskutočniť jeho agregáciu.

Preto je výhodné zadefinovať metasimulátor. *Metasimulátor* by mal mať nasledovné vlastnosti: a) poskytovať prostredie pre správu a spúšťanie simulátorov; b) poskytovať prostriedky na dynamické vytváranie simulátorov; c) vedieť spúšťať postupnosť simulátorov, resp. ich jednotlivých experimentov.

Zavedenie metasimulátora, ktorý bude zoskupovať viacero simulátorov, je výhodné z pohľadu automatizácie spúšťania simulácií a zberu údajov, pretože výsledky jednej simulácie môžu vstupovať do ďalšej. Okrem toho je možné, aby

jedna simulácia bola obsiahnutá v inej. Správa nad simuláciami pritom spadá pod metasimulátor, ktorý zabezpečí rovnaké prostredie z hľadiska spúšťania simulácií, napríklad metasimulátor sprístupňuje modul pre jazykovo nezávislé texty. Metasimulátor obsahuje aj upravovanie výnimočných stavov, ktoré nerozpoznali simulátory. Takto sa zabezpečí, že v prípade neopravenej chyby simulátora, séria simulácií môže ďalej pokračovať, resp. je možné uložiť stav simulácie na neskoršie použitie alebo analýzy príčin výskytu neupravenej výnimky.

Základnými funkciami metasimulátoru je možné definovať:

- načítanie prostredia z konfigurácie metasimulátora;
- poskytovanie globálnych informácií o prostredí;
- sekvenčné spúšťanie preddefinovaných simulácií na základe konfigurácie;
- opravenie chýb, ktoré neboli opravené priamo v simulátore;
- správa simulátorov (životný cyklus simulátora).

Samotný pojem *simulátor* má ale širšie pole pôsobnosti, preto je vhodné na tomto mieste popísať, aké vlastnosti a funkcie má simulátor:

1. v rámci času prideleného metasimulátorom vykonáva simuláciu;
2. voči ostatným simulátorom sa môže správať blokujúco, čiže neumožní beh iného blokovateľného simulátora;
3. opravuje vzniknuté chyby a spolupracuje s používateľom.

Prvý bod vyplýva z definície metasimulátora, keďže je správcom simulátorov. Samotný simulátor automaticky blokuje ostatné simulátory, ktoré sa spúšťajú sekvenčne. Ak by simulátor definoval sám v sebe nejaký iný simulátor, môže blokovať aj ten. To závisí od simulátora. Metasimulátor tomuto stavu ale nebráni.

Takýto simulátor musí spravovať celý model. Musí obsluhovať všetky uzly, priradiť im jednotlivé kanály, v rámci kanálov príslušné konektory. Simulátor pre každý uzol definuje jeho jednoznačné označenie s oblasťou platnosti pre jeden simulátor. Udržiava kanály v konzistentnom tvare, aby v prípade požiadavky na hodnotu doteraz neznámej vlastnosti nedošlo k chybe. Uzly, podobne ako aj ich vlastnosti, môžu sa dynamicky pridávať. Takisto musí zabezpečiť samotné dynamické pridávanie uzlov – či už potomkov alebo susedov.

Základné funkcie simulátora sú:

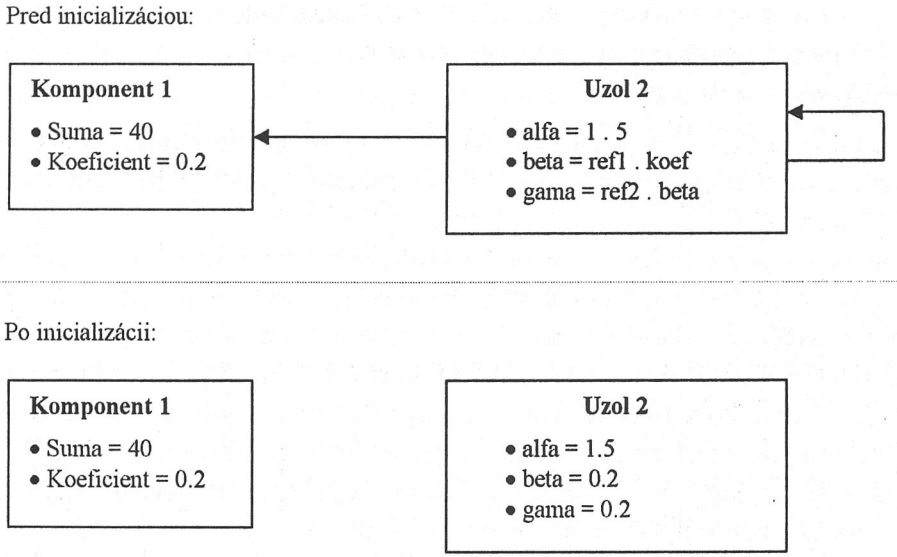
- načítanie a inicializácia uzlov, konektorov a ostatných komponentov;
- zaradenie uzlov do vrstiev;
- spustenie vykonania kroku tým uzlom, ktorých potomkovia už krok vykonali;
- zabezpečenie komunikácie medzi uzlami;
- prepojenie na metasimulátor;
- zabezpečenie dynamického pridávania uzlov;

Súčasťou simulátora je aj inicializácia *komponentov*. Komponent je objekt, ktorý má isté vlastnosti. Z tohto vyplýva, že komponentom je aj uzol, aj konektor,

ale aj lokálny transformátor vzdialených vlastností. Vlastnosti komponentu ovplyvňujú konanie komponentu, takže v prípade konektora treba rozlíšiť vlastnosti konektora a vlastnosti poskytované z uzla. Ďalšou vlastnosťou komponentu je jeho schopnosť replikovať sám seba – ide o vytvorenie identickej kópie.

Počas inicializácie komponentov a ich vlastností je dovolené, aby sa hodnota vlastnosti brala z inej vlastnosti iného komponentu, takže ide o referencie. Takýmto spôsobom je možné vytvoriť komponent obsahujúci inicializačné parametre globálneho charakteru. Ostatné komponenty môžu svoje vlastnosti inicializovať na základe hodnôt takéhoto inicializačného komponentu (obr. 6).

Obrázok 6



Ako vyplýva z obrázku 6, je nevyhnutné riešiť odkazy na odkazy: pre uzol 2 sa musí najprv inicializovať vlastnosť *beta* a až potom môže simulátor inicializovať vlastnosť *gama*. Simulátor musí riešiť aj cyklické odkazy: na obrázku 6 – napríklad keby bol odkaz z vlastnosti *gama* na *beta* a z *beta* na *gama*. Konkrétnu implementáciu algoritmu prevencie takéhoto uviaznutia simulátor rieši jednoduchým spôsobom. Tento algoritmus je ale možné rozšíriť.

Životný cyklus komponentu je nasledovný:

1. fyzická inicializácia;
2. inicializácia simulátorom z konfigurácie komponentu;
3. použitie vlastností komponentu;
4. fyzické zrušenie komponentu.

Keďže samotný komponent nedefinuje žiadne správanie, môže ho simulátor zrušiť v ktoromkoľvek čase po inicializácii komponentu.

Uzol je nadstavbou komponentu, preto je jeho životný cyklus rozšírený o niektoré etapy:

1. fyzická inicializácia;
2. inicializácia simulátorom z konfigurácie uzla;
   2.1. inicializácia vlastností uzla,
   2.2. inicializácia všetkých jeho lokálnych transformácií,
   2.3. inicializácia jeho konektorov;
3. zavolanie implementačne závislej inicializácie uzla;
4. prevádzkovanie uzla;
   4.1. zavolanie implementačne závislej časti začiatku kroku,
   4.2. zavolanie implementačne závislého vykonania kroku,
   4.3. zavolanie implementačne závislého skončenia kroku;
5. zavolanie implementačne závislej deštrukcie uzla;
6. fyzické zrušenie uzla.

Uzol je zrušený v momente ukončovacieho procesu simulátora. Počas behu simulácie uzol nemožno zrušiť, pretože iné komponenty môžu mať naň inicializačné odkazy.

Krok 4, čiže prevádzkovanie činnosti uzla, môže byť opakujúcim sa krokom. Zodpovedá jednej iterácii simulačného kvantovania času. Je rozdelený na 3 podkroky: každý uzol, ktorý sa má v danej iterácii vykonať, bude pomocou 4.1 oboznámený s touto skutočnosťou: dovolí sa mu zistiť si svoje vlastné konektory a pod. V tomto bode totiž simulátor zaručuje, že konektory budú správne inicializované. Krok 4.2 slúži na samotné vykonanie činnosti, ktorú uzol reprezentuje a definuje. Krok 4.3 je vyvolaný simulátorom a predstavuje miesto, kedy môže uzol zrušiť objekty a pod., ktoré vytvoril v kroku 4.1.

Ukladanie stavu simulácie je záležitosťou implementácie simulátora. Nejde však o elementárny problém. Spôsobov ako nainicializovať komponenty je mnoho: vlastnosti mohli byť načítané zo súboru, mohli byť vygenerované, alebo inicializované na diaľku pomocou počítačovej siete. Ukladanie stavu a prepísanie pôvodného stavu nie je možné, pretože zdroj inicializácie nemusí byť zapisovateľný alebo perzistentný. Pokiaľ ide o export stavu simulácie, čiže zápis stavu do konkrétneho tvaru, aj táto požiadavka sa necháva na konkrétnu realizáciu systému.

Základné funkcie systému by sa dali rozdeliť na nasledovné druhy: *metasimulátor*, *simulátor*, *podporné prostriedky a knižnice*.

Metasimulátor zabezpečuje chod viacerých simulátorov. Poskytuje im prvky globálneho prostredia a vytvára prepojenie na podporné prostriedky. Do tejto časti spadajú funkcie ako napríklad pridanie novej simulácie, inicializácia simulácie, poskytnutie simulačne nezávislých prvkov simuláciám.

V časti *simulátora* sa vykonáva vlastná simulácia. Nachádza sa tu aj správa uzlov, vzťahov, pod ktorú spadá aj samotná inicializácia modelu a riadenie modelu. Najvyužívanejšími funkciami sú: inicializácia simulácie, vykonanie jedného kroku simulácie, pridanie uzla, vytvorenie prepojenia.

Najdôležitejším vstupom celého systému je jeho schopnosť byť prístupný pre následné rozširovanie. To znamená, že veľká pozornosť sa musí venovať samotným rozhraniam systému, či už ide o vstupy, alebo výstupy. Vstupom vytváraného systému je aj konkrétna implementácia simulátora vytvorená používateľom.

Hlavnou formou výstupu je *komunikácia* s rozširujúcimi časťami systému, ktoré spravuje používateľ. Keďže sa predpokladá, že až sám používateľ bude definovať styk s používateľským prostredím, nie je potrebné definovať vzhľad jednotlivých dialógov a ovládacích prvkov. Postačuje, ak otvorenosť systému bude podporovať pridávanie používateľského prostredia.

Komunikácia v systéme je vytvorená na báze *objektového programovania.* Ide hlavne o princípy používania výnimiek a návratových hodnôt. Takouto formou tvorené výstupy sú pre používateľa výhodou, pretože chybové stavy, ale aj samotné výsledky simulácie sú presne definované.

Rozdelením celého systému na simulačný procesor a model je možné definovať ďalšie vstupy a výstupy.

Model, podobne ako aj celý systém, komunikuje s okolím použitím rozhraní. Toto okolie je modelu sprístupňované cez samotný simulačný procesor. Vstupy a výstupy modelu sú vzájomne prepojené na vstupy a výstupy simulačného procesoru. Jediným vstupom do modelu je samotný simulačný procesor. Na načítanie konkrétnych hodnôt a implementácie modelu sa použil mechanizmus implementovaný v simulátore. Podobne, vstupom procesoru je samotný model. Takto je možné získavať z modelu podrobné informácie bez použitia dodatočných údajových tokov.

Model definuje aj spôsob komunikácie uzlov. Ohraničenia simulačného procesora vychádzajú z ohraničení modelu. Z hľadiska rozšíriteľnosti simulátora je systém pripravený aj na takúto možnosť. Jeho otvorená štruktúra a možnosti obohacovania systému používateľom, spolu s faktom, že ide o programovaciu knižnicu, prispievajú k výraznému zníženiu celkových ohraničení systému.

Vytvorený model je možné použiť na simulovanie neurónových sietí. Možno zadefinovať vzťah, keď jeden uzol zodpovedá jednému perceptrónu. Jednotlivé kanály medzi uzlami budú tvoriť prepojenia medzi perceptrónmi. Napríklad ak je neurónová sieť založená na váhovaní vstupov, tieto váhy možno pohodlne realizovať použitím lokálnych transformácií. Vzdialené konektory budú len jednoducho poskytovať výstup perceptrónu, pričom výstup bude tvoriť jedna vlastnosť uzla.

Neurónová sieť realizovaná týmto modelom má svoje výhody, ale aj nevýhody. Medzi výhody patrí existujúci aparát na správu uzlov a ich prepojení, životný cyklus uzlov a pod. V rámci jednej horizontálnej roviny je možné definovať takmer ľubovolnú neurónovú sieť – či už ide o sieť ex ante, alebo rekurentú, alebo aj iné typy sietí. V prípade, že uzol by predstavoval agenta, ktorý sa má rozhodovať na základe výsledkov neurónovej siete, neurónovú sieť možno realizovať ako jeho potomkov. Lepším prípadom by však bolo, ak by takýto agent zadefinoval novú simuláciu, ktorá by obsahovala len neurónovú sieť. V prípade potreby zmeny neurónovej siete by bolo možné nahradiť takúto simuláciu inou, bez nutnosti zmeny samotného agenta. Výhodou tohto riešenia je aj to, že jednotlivé kroky počas behu neurónovej siete nezávisia od krokov agenta. Keby neurónová sieť, aj agent boli v rovnakej simulácii, museli by vykonávať kroky synchronizovane, takže neurónová sieť by nemohla byť „rýchlejšia" pre prípady učenia sa, vyhodnocovania a pod.

Ak sa však použije nezávislá simulácia, je možné, aby agent zvolil vykonávanie viacerých krokov takto vnorenej simulácie. Výhodou tohto riešenia je to, že sa môže použiť už existujúca realizácia perceptrónov a siete. Jediné, čo je nevyhnutné vykonať, je vytvorenie rozhraní na prístup k objektom. Takto je možné, aby v systéme bolo viacero neurónových sietí, pričom časové nároky na výpočet sa znížia. Simulátor nemusí spravovať jednotlivé prvky siete ako uzly. Výhodou definície rozhraní je aj nezávislosť realizácie siete a jednotlivých perceptrónov a definovanie nezávislého prístupu k prvkom neurónovej siete. Navyše je prípustné, aby agent obsahoval takúto externú neurónovú sieť aj bez použitia vnorenej simulácie.

Z hľadiska evolučných algoritmov je možné vyvodiť podobný záver, ako pri neurónových sieťach. Konkrétne realizácie evolučných algoritmov sa dajú vykonať aj použitím hierarchického modelu, resp. jeho horizontálnej roviny. V takomto prípade je jeden prvok nesúci chromozóm definovaný ako jeden uzol. Ďalej je definovaný rodič všetkých takýchto uzlov – konkrétna realizácia môže byť založená aj na horizontálnom, aj na vertikálnom pohľade. Tento rodič sa bude správať ako správca jednotlivých prvkov. Bude zabezpečovať výber, vyvolanie kríženia a mutácie, a udržiavať populáciu prvkov v primeranom počte. V tomto prípade sa s výhodou využije skutočnosť, keď simulátor môže dynamicky pridávať jednotlivé uzly, prípadne aj ich prepojenia. Treba podotknúť, že prepojenia medzi uzlami tvoriacimi prvky populácie sa nevyužívajú. Simulátor ich udržiava a spravuje, takže ide o zbytočný, a navyše aj časovo náročný proces. Keďže populácia prvkov sa relatívne rýchlo mení, táto správa prvkov je nežiaduca. Simulátor navyše nedefinuje proces rušenia uzlov, takže po čase je zrejmé, že populácia zahltí pamäť a ostatné systémové prostriedky. Preto je výhodné, podobne ako v prípade neurónových sietí, definovať rozhrania na externé použitie

evolučných algoritmov. S týmto činom sa spájajú podobné charakteristiky ako v prípade neurónových sietí – nižšia správa množiny prvkov tvoriacu populáciu a podobne.

Agent je jedinec, ktorý má svoje vedomosti a skúsenosti, svoje správanie voči prostrediu a okoliu. Agentom môže byť napríklad človek alebo podnikateľ. Podobne agentom môže byť aj samotný podnik, alebo aj spoločnosť, resp. celý štát.

Agent zodpovedá jednému uzlu. Každý agent má svoje správanie voči ostatným agentom. Tu sa dá s výhodou využiť mechanizmus konektorov a lokálnych transformácií. Takto môže každý agent prechovávať odlišné správanie k rôznym agentom a môže im o sebe poskytovať rôzne informácie. Do systému sa môže zavádzať neúplná informovanosť, alebo aj dezinformácie a šum. Ku globálnym znalostiam je možné pristupovať cez mechanizmus uzlov. Každému agentovi sa poskytne rovnaký typ konektora a je na samotnom agentovi, aby si tú informáciu, resp. vedomosť pretransformoval podľa seba. V rámci globálnej vedomosti sa môžu vyskytovať aj šumy, ktoré sa môžu realizovať konektormi k vlastnostiam. Rovnakým spôsobom sa dajú do sveta vypúšťať aj falošné správy a pozorovať ich vplyv napríklad na dianie na burze.

V prípade, že agentom by bol napríklad štát, je možné vytvoriť rôzne realizácie agentov – niektoré z nich môžu ísť do hĺbky, iné môžu zachytávať len elementárne správanie. Na podrobne vypracovaných agentov však nemá vplyv, ako sa realizujú iní agenti. Agent predstavujúci štát by mal pozostávať z populácie, riadenia štátu, hmotných a nehmotných prostriedkov. Riadenie štátu pozostáva z definovania vlastností uzla predstavujúceho štát. Populáciu môže tvoriť jeden agent simulujúci celé ľudstvo, alebo sa môže zvoliť riešenie, kde jeden agent predstavuje jeden „typ" populácie, napríklad podľa sociálneho rozdelenia. Medzi nehmotné prostriedky môžeme zahrnúť aj podniky, pričom jeden agent zodpovedá jednému podniku. Tieto podniky komunikujú so spoločnosťou, takže musia byť na jednej vrstve.

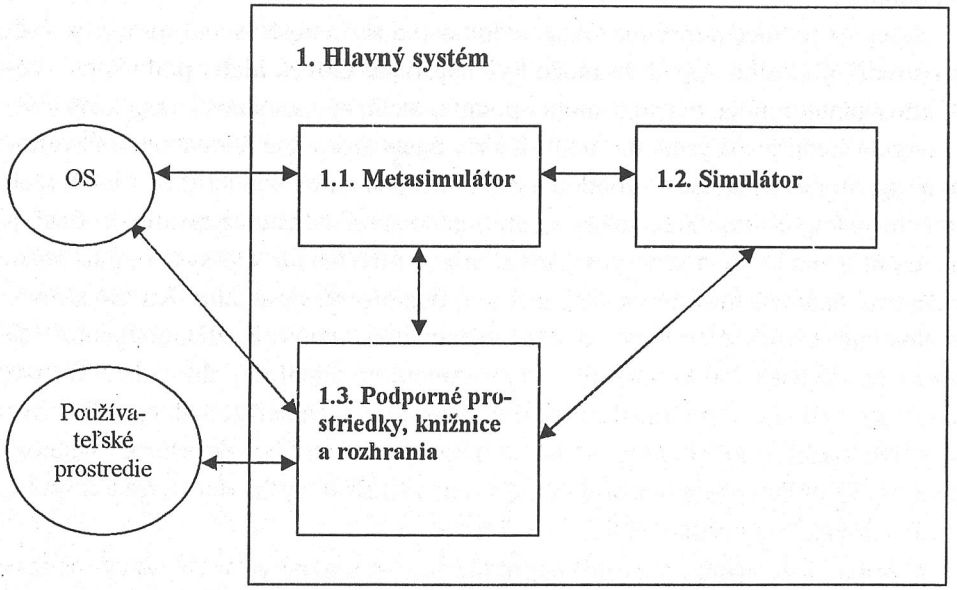
Podobne aj ostatné hmotné a nehmotné prvky simulácie – ak má existovať vzťah medzi uzlami – musia byť (fyzicky, vzhľadom na model a ním definovanú štruktúru) na jednej vrstve.

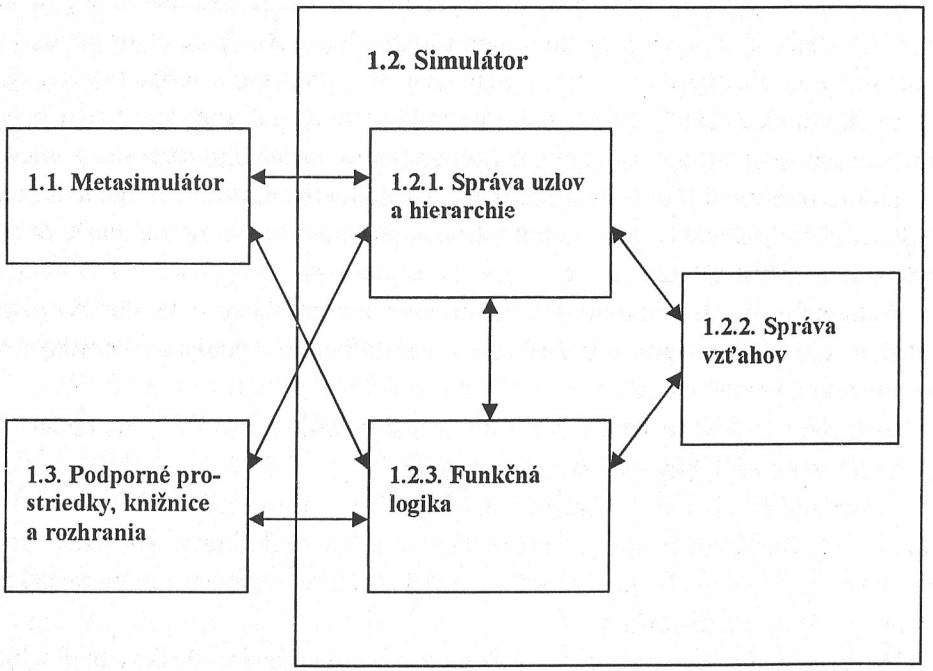
Hierarchia uzlov je skrytá pre ostatné uzly, takže je možné uskutočňovať rôzne abstrakcie a hĺbky realizácie modelu.

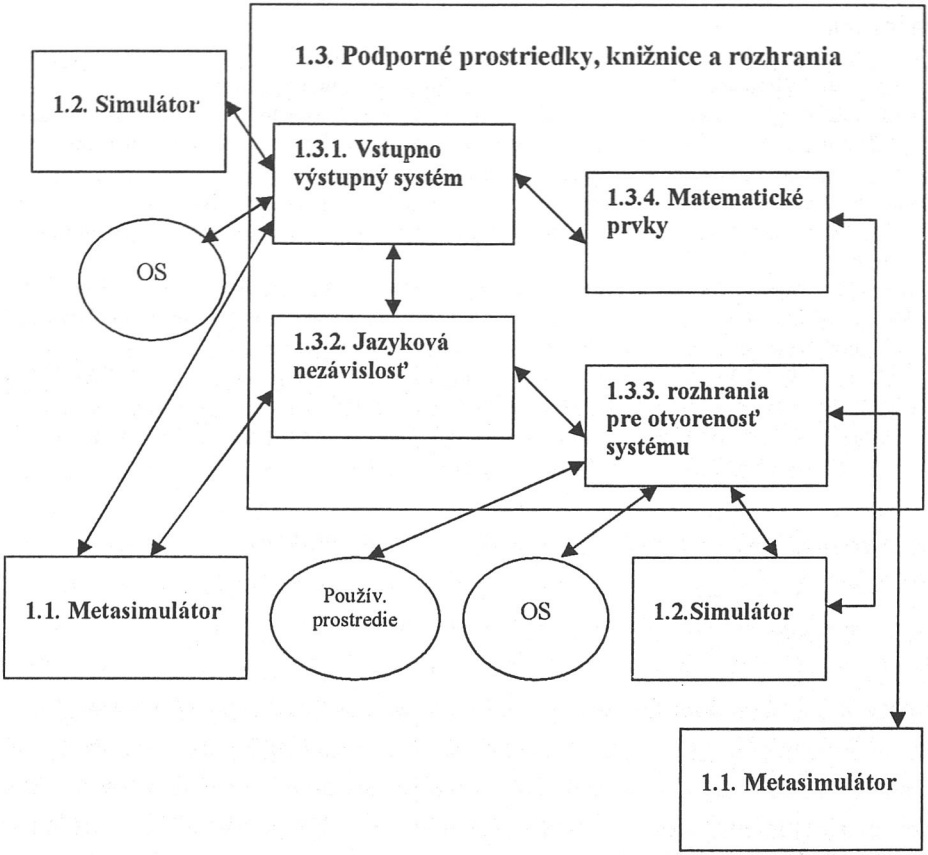
V horizontálnej rovine pohľadu sú vzájomné vzťahy uzlov definované kanálmi. Použitím tejto formy je možné, aby každý uzol definoval správanie voči iným uzlom, a pritom by používateľ nemusel realizovať samotnú implementáciu výmeny údajov medzi uzlami.

Ide napríklad o samotný proces agregácie a interpretácie výsledkov buď v hierarchických štruktúrach, alebo realizáciou konkrétnych neurónových sietí.

Diagramy toku údajov







Záver

Keď už je vytvorená takáto hierarchia štátu, je možné pristúpiť k výsledkom simulácie. V takomto prípade je nevyhnutné zbierať informácie z nižších vrstiev a vhodným spôsobom ich interpretovať. Napríklad na získanie hrubého domáceho produktu sa musia zozbierať informácie od úplne všetkých uzlov v systéme. Samotná interpretácia jednotlivých údajov závisí od pozície, kde sa tieto údaje nachádzajú. Naskytá sa tu aj otázka zmysluplnosti niektorých údajov, ich vzťahu k celému výpočtu. Táto interpretácia však závisí od konkrétne zvolenej realizácie a miery abstrakcie modelu, ktorý určí používateľ. K modelovaniu trhu treba povedať toľko, že trh nemusí byť explicitne definovaný nejakým uzlom. Môžu ho tvoriť vzťahy medzi uzlami. Takže podniky, ktoré sú súčasne aj uzlami, môžu komunikovať buď priamo medzi sebou, alebo aj s použitím prostredníka, v tomto prípade trhu. Výhodou prostredníka je to, že ide o uzol. Môže mať vlastnosti a nastavovanie parametrov. Zber informácií podlieha štandardnému postupu pre komponenty. Navyše, uzly môžu priamo ovplyvňovať tento trh – či už ide o uzly typu vláda a zákony, alebo ľudia a zvyklosti.

Literatúra

[1] System and Simulation links. http://www.isima.fr/ecosim/simul/simul.html
[2] GOKTEKIN, T.: Moose Overheads – Slides. http://www.cise.ufl.edu/~fishwick/moose/tolga.ps
[3] FISHWICK, P. A. Simpack: Getting stared with simulation programming in C and C++. ftp://ftp.cis.ufl.edu/cis/tech-reports/tr92/tr92-022.ps
[4] Genetic Algorithm Tutorials. http://www.isima.fr/ecosim/simul/simul.html
[5] WHITLEY, D.: A Genetic Algorithm Tutorial. http://www.science.uva.nl/~mes/psdocs/ga-tr103.ps.gz
[6] Neural Networks Repository. http://www.brain.web-us.com/brain/neural.html
[7] KINDERMANN J. – LINDEN A. A Tutorial in Object Oriented Simulation of Advanced Neural Networks. http://www.brain.web-us.com/brain/neural.html
[8] TOMASSINI, M. Evolutionary Algorithms. http://lslwww.epfl.ch/~marco/ga-ehw-final-SV.ps
[9] CALRESCO. Various Simulation Methods and Projects. http://www.calresco.org/links.htm
[10] NWANA, H.: Software Agents: An Overview. http://agents.umbc.edu/introduction/ao.ps
[11] XML.ORG – The XML Industry Portal. http://www.xml.org

MULTI-AGENT APPROACH ON ECONOMIC SYSTEMS MODELLING

Jaroslav ZAJAC

Multi-agents simulation reports a project in agents-based computer simulation of processes of economics in a population of boundedly rational learning agents. The same family of models will be simulated under different assumptions about the nature of the learning process and details of the production and economics process. The purpose of this procedure is to establish a relationship between the assumptions and the simulation results.

The simulation techniques with baseline simulations of boundedly rational learning processes, and do not involve the complications of dealing with economies. Can simulation „explain" the puzzles of finance regulation and particularly the key puzzle of growth and learning processes that produce the puzzling results? And just what assumptions of the simulation are not predictable associated with puzzling results?

Typical problems are multi-agents case can also offer opportunities for inference of hidden action and agent monitoring agent. Extraordinary creativity remains a mystery, no one make dependable long term predictions of what inventions will occur and when, or what their effects will be.

Agency theory drops the assumption that most individuals treat conformity as instrumental to the achievement of their personal goals. Agents can be dependent upon to conform only when held property accountable to effective incentives.

Agents need to limit the recipients of their messages as much as possible and often need to share information across both time and space and learning this becomes a case of information sharing. Mechanism such as genetic programming can improve the behavior of a species of agents over successive generation and construct markets in the environment can enable an agent community as a whole to learn.