KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Vilma Liorenšaitytė

Logistikos sistemų formalus imitacinis modeliavimas

Magistro darbas

Darbo vadovas
prof. habil. dr. H. Pranevičius

Kaunas, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Vilma Liorenšaitytė

Logistikos sistemų formalus imitacinis modeliavimas

Magistro darbas

Recenzentas
dr. S. Drąsutis
2007-05-29

Vadovas
prof.habil.dr. H.Pranevičius
2007-05-29

Atliko
IFM-1 gr. stud.
Vilma Liorenšaitytė
2007-05-29

Kaunas, 2007

TURINYS

SUMMARY

Formal simulation modeling for logistic systems

At present, a large number of modeling and simulation techniques and tools have been developed to deal with complex business systems. In this paper, we concentrate on scenario illustrating, how simulation models can be integrated in to business management system. Different infrastructure forms are possible, because services may be implemented on single machine or distributed throughout several companies' networks.

In this paper a concept will be analyzed - how to combine different simulation models. The technology for involvement of simulation models in to information systems will be created. Also the problem of simulation model transformation in to program code will be solved.

For model formalization we can use a method of Piece-Linear Aggregates (PLA) that belongs to the class of time automata model. PLA method is close to Discrete Event Simulation (DEVS) formalism, which is used to create wide purpose simulation models.

ĮVADAS

Problemos aktualumas

„Šiuolaikinių organizacijų veiklos pobūdis ir verslo aplinkos savybės – globalizavimas, industrinės ekonomikos pokyčiai, organizacijų veiklos pokyčiai – reikalauja informacinių technologijų paslaugų ir kartu, visų veiklos sričių kompiuterizuotų informacinių sistemų (IS). Globalizavimo požymiai: vadyba ir kontrolė globalioje rinkoje, konkurencija pasaulio rinkose, globalinės darbo grupės, globalinės produkto paskirstymo sistemos“[13].

Pagrindinis kompanijos tikslas yra gauti didesnį pelną, pasiekiant konkurencinių privalumų prieš savo konkurentus. Keičiantis prekybos taisyklėms, kompanijos yra priverstos keisti savo vidinius verslo procesus ir tuo pačiu kurti išorinius, į klientą orientuotus verslo metodus[8]. Taigi, pagrindinis organizacijos tikslas yra tenkinti kliento reikmes ir generuoti pelną.

Labai sunku yra tenkinti klientų poreikius vien žemesnių kainų ar didesnių produktų funkcionalumų dėka, nes šiais laikais klientai perka ne prekę, o privilegijas, tokias kaip „pristatymas laiku“(Christopher, 1992)[8].

Klientų pasitenkinimo lygį galima smarkiai pagerinti tiesiog žinant, kad pristatyti prekes kai kuriems klientams užtrunka ilgiau ir kad skirtingų pristatymų trukmė yra skirtinga. Šių žinių pagrindu galima geriau valdyti pristatymų terminus ir užtikrinti, kad kuo daugiau užsakymų būtų įvykdyta pažadėtu laiku.

Deja, įvykdyti šiuos reikalavimus, naudojantis jau egzistuojančiomis sistemomis (kaip MPR II - medžiagų planavimo sistemomis) yra sudėtinga, nes jos remiasi pasenusiomis gamybos sąlygomis[8]. Šios sistemos labiau susitelkia vidinių procesų gerinimui.

Kita problema, jog paprastai yra sunku apskaičiuoti tam tikro sprendimo įtaką į visos verslo organizacijos sistemos esminius elgsenos požymius, tokius kaip kaina, našumas, gamybos ciklo trukmė, įrenginių prastovėjimo laikas, eilės ir t.t. To priežastys yra stochastinis sistemos pobūdis, sudėtingos priklausomybės tarp sistemos komponenčių, besikeičianti išorinė aplinka ir pan. O tradiciniai problemos sprendimo metodai veda prie didesnių laiko vėlinimų komunikacijos procese.

Logistikos sistema gali būti vienas iš geriausių būdų, kaip padidinti kompanijos konkurencinį pranašumą. Efektyviai veikianti logistikos sistema kuria klientų aptarnavimo lygio elementus - prekės prieinamumą, pristatymo patikimumą ir pan.

Sprendimų priėmimas logistikos sistemoje reikalauja operatyviai naudoti verslo procesų modelius, kurie leistų įvertinti besikeičiančias sistemos vidines ir išorines sąlygas. Imitacinių modelių naudojimas, sprendžiant uždavinius šiose sąlygose, leidžia pagrįsti priimamus sprendimus. Tam, kad būtų galima panaudoti imitacinį modelį, reikia šį modelį integruoti į verslo valdymo sistemą.

Darbo tikslas

Šio magistrinio darbo tikslas – gamybos ir logistikos procesų imitacinių modelių sudarymas ir modelių integravimas į verslo valdymo informacinę sistemą.

Darbe sprendžiami uždaviniai

Sprendžiami uždaviniai:

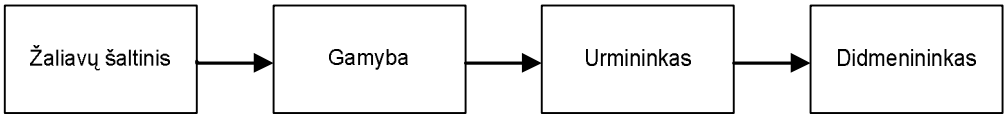
- Verslo procesų modeliavimas, formalizmas bei imitacinio modelio sudarymas.
- IS programinių komponentų sąsajų su imitaciniais modeliais sudarymas;

Imitacinių modelių integravimo į IS sąsajas dalys:

- Imitacinio modelio sąsaja su programiniais komponentais;
- Imitacinio modelio sąsaja su IS duomenų baze;

1. LOGISTIKA

Jei į veiklos sistemą įtraukiami tiekėjai bei klientai ir kiti veiklos partneriai, o jų atliekama veikla vertinama pagal tai, kiek jie padeda organizacijai gauti papildomo pelno, tai bus logistikos modelis (1 pav.).



1 pav. Tiesinis logistikos modelis

Logistika apima visas verslo operacijas, kol žaliava paverčiama gatavu produktu. Šis modelis, tobulinant organizacijos veiklą, gali būti naudojamas keliems praktiniams tikslams:

- Atlikti organizacijos veiklos analizei iš sisteminių pozicijų, t. y. veiklos proceso analizę;
- Atlikti grandinės vertės analizei, siekiant rasti tas vietas, kurios ateityje būtų pelningiausios ir efektyviausios;
- Tyrinėti organizacijos IS ir modeliuoti jos teikiamą efektą;
- Aprašyti vartotojo vaidmenį veiklos procese ir analizuoti vartotojo informacinius poreikius;
- Identifikuoti duomenis bei duomenų apdorojimo procedūras, būtinus veiklai vykdyti ir valdyti.
- Projektuoti IS duomenų bazę ir taikomuosius uždavinius.

Yra penkios pagrindinės veiklos rūšys (veiklos procesai):

- Tiekimas – naujų medžiagų ir gautų organizacijoje tvarkymas ir saugojimas (įvežamoji logistika);
- Gamyba – pagrindinio produkto (gaminio) formavimo operacijos, gamybos procesas;
- Paskirstymas – apima produkto sandėliavimo ir paskirstymo procesus (išvežamoji logistika);
- Marketingas ir pardavimas – apima reklamą, kainodarą, ryšių su klientais procesus;
- Marketingas ir paslaugos – naujos įrangos instaliavimas, atskirų padalinių produktų tiekimas.

Sistemų teorija sako, kad nepriklausomai nuo sistemos dydžio, pats principas nesikeičia. Todėl šiame darbe bendra koncepcija pritaikoma, nagrinėjant du verslo procesus: gamybos ir paskirstymo.

2. FORMALUS SISTEMOS MODELIAVIMAS

2.1. Modeliai ir modeliavimas

Šiuo metu modelis suprantamas kaip sistemos (reiškinio, proceso) žodinis, grafinis ar matematinis aprašymas, arba fizinis darinys, kuris leidžia pavaizduoti ir nagrinėti modeliuojamojo dalyko (objekto) tam tikrus ypatumus. Toks modelis (įrankis) yra patogus tuo, kad išreiškia tik tam tikras savybes, atsiribodamas nuo visų kitų. Modelis, kuris išreikštų visus modeliuojamojo dalyko požymius ir savybes būtų tolygus modeliuojamajam dalykui, o tai reikštų, jog jis yra toks pat sudėtingas ir nesuprantamas kaip ir modeliuojamasis, t.y. daiktas savyje [2].

Norėdami gauti teisę valdyti sudėtingą techniką, pvz., automobilį ar lėktuvą, turėjome nemažai laiko praleisti prie treniruoklių. Taigi visų pirma atlikdavome modeliavimo pratimus. Tai leisdavo atsikratyti galimų incidentų ir avarijų rizikos, sutaupyti nemažai laiko, pinigų ir kitų išteklių, pvz., kuro, sąnaudų. Tam realūs sudėtingi objektai buvo keičiami paprastesniais materialiais pakaitalais, manekenais, kitaip tariant, modeliais.

Modelių kūrimas leidžia besimokančiajam įsigilinti į analizuojamą problemą, išskirti esminius sistemos elementus ir ištirti jų sąveiką.

Kompiuterinis modeliavimas grindžiamas sistemų analizės principais. Tai reiškia, kad į sprendžiamą problemą žiūrima kaip į sistemą, kurią sudaro tam tikri elementai ir ryšiai tarp jų.

Taikydami sisteminę metodologiją realioms problemoms analizuoti, bandome iš daugelio galimų faktorių išskirti pagrindinius, išryškinti esminius jų tarpusavio ryšius ir atmesti kitus faktorius bei ryšius, kurie konkrečiame kontekste yra neesminiai. Palaipsniui, mus dominanti sudėtinga problema tampa skaidresnė ir suprantamesnė, įvairūs iš pirmo žvilgsnio nesusiję reiškiniai susijungia į naują loginę visumą, sistemą, o į visą pasaulį imame žiūrėti kaip į tokių sistemų rinkinį. Apibrėžti sistemas galima įvairiais būdais, bet, norint gauti konkrečius atsakymus į iškeltus klausimus, sistemą reikia formalizuoti, t.y. padaryti visiems vienareikšmiškai suprantamą. Tam ir skirtas modeliavimo metodas, kurio galutinis produktas – modelis. Jis – esminių realios sistemos savybių išraiška, kuri tam tikru būdu atspindi sistemos elgesį ir padeda ją tirti ar eksploatuoti. Reikėtų pabrėžti, kad modelis nėra tikslus ir detalus sistemos aprašymas, jis tik imituoja mus dominantį sistemos elgesį [4].

Imitacinių modelių sudarymas, naudojant sistemos formalius aprašymus, tinka modelių

sudarymui, sprendimų priėmimui. Šie metodai leidžia ne tik sukurti patį imitacinį modelį, bet ir panaudojant formalius metodus patikrinti ar modelis sudarytas teisingai. Labiausiai paplitę formalūs metodai, naudojami imitacinių modelių sudarymui yra Petri tinklai, DEVS ir PLA.

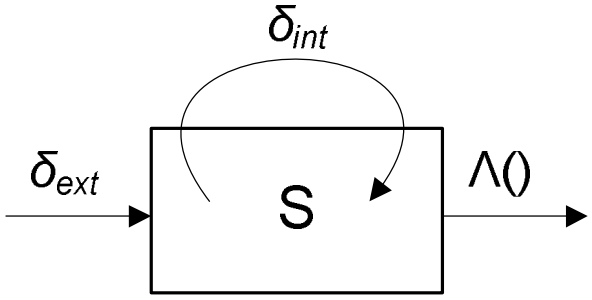
2.2. Diskrečių įvykių sistemos specifikacijos formalizmas

Sudėtingų sistemų imitaciniam modeliavimui naudojamos įvairios metodikos. Vienas iš būdų formaliai aprašyti sistemą yra diskrečių įvykių sistemos specifikacijos (DEVS - Discrete Event Simulation) formalizmas. DEVS formalizmas buvo sukurtas 1976 m. Bernard Zeigler. DEVS modelis (2 pav.) gali turėti du komponenčių tipus[7].:

- atominis DEVS - savyje neturi jokių komponenčių. Jis tik turi matematinę jo veikimo apibudinimą;
- jungtinis DEVS - sudarytas iš vieno ar kelių atominių DEVS.

Jungtinį DEVS galima naudoti, kad sudarytume sudėtingesnius DEVS komponentus, tam formalizme reikalinga nurodyti:

- bazinius modelius, kuriais remiantis konstruojami stambesni;
- kaip šie modeliai tarpusavyje sujungti hierarchinėje struktūroje.



2 pav. DEVS modelis

Diskrečiosios įvykių sistemos specifikacijos struktūra:

$$\mathbf{M = [X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta]} \qquad (1)$$

Čia:

X – įėjimo verčių sritis;

S – būsenų sritis;

Y – išėjimų verčių sritis;

δint: $S \to S$ – vidinių perėjimų funkcija;

δext: $Q \times X \to S$ – išorinių perėjimų funkcija;

$Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}$ – pilna būsenų sritis;

e – laikas, praėjęs nuo paskutinio perėjimo;

λ: $S \to Y$ – išėjimų funkcija;

ta: $S \to \mathbf{R}_{+0,\infty}$ - galima laiko kitimo funkcija.

2.2.1. Atominis DEVS

Atominio DEVS formalizmo struktūra aprašanti skirtingus diskretinio įvykio sistemos veikimo aspektus:

$$atomicDEVS \equiv \langle S, ta, \delta_{int}, X, \delta_{ext}, Y, \lambda \rangle. \qquad (2)$$

Laikas T yra nepertraukiamas ir nėra aiškiai apibrėžtas:

$$T = \mathbb{R}.$$

S būsena yra rinkinys galimų nuoseklių būsenų: DEVS dinamika susideda iš užduotos nuoseklios būsenos S. Tipinė S bus struktūrizuota:

$$S = \times_{i=1}^{n} S_i.$$

Tai formalizuoja daugkartines (n) lygiagrečias sistemos dalis. Pažymėta, kaip struktūrizuotas būsenų rinkinys dažnai sintezuojamas nuo lygiagrečių būsenos rinkinių komponenčių iki sujungto DEVS modelio. Laikas, kurio metu sistema pasilieka nuoseklioje būsenoje prieš pereinant į sekančią nuoseklią būseną, yra sumodeliuotas laiko funkcijos pokyti.

$$ta : S \to \mathbb{R}^{+}_{0,+\infty}.$$

Kadangi laikas realiame pasaulyje visada kinta, *ta* atvaizdas turi būti ne neigiamas. Kai ta=0 vyksta momentalūs perėjimai: laikas nepasikeičia prieš perėjimą į naują būseną.

Akivaizdu, kad ši abstrakcija kilusi iš tikrovės, kuri gali privesti prie modeliavimo tipo kaip akimirksninės kilpos, kurios neatitinka realios fizikinės elgsenos. Jei sistema turi pasilikti

baigtinėje būsenoje S visada, tai sumodeliuota pagal vidurkį nuo ta=+∞.

Vidinė perėjimų funkcija:

$$\delta_{int} : S \to S$$

Tai perėjimo nuo vienos būsenos iki kitos nuoseklios būsenos modelis. δ*int* aprašo elgseną baigtinio būsenų automato, *ta* prideda laiko pokyčius.

Galima stebėti sistemos išėjimą. Išėjimas Y reiškia galimų išėjimų rinkinį. Kaip taisyklė Y bus struktūrizuota visuma (išėjimų rinkinys)

$$Y = \times_{i=1}^{l} Y_i.$$

Tai formalizuojama daugkartiniai (l) portų išėjimai. Kiekvienas portas identifikuojamas jo unikaliu indeksu i. Į vartotoją orientuotos modeliavimo kalbos indeksai buvo gauti iš unikalių porto pavadinimų.

Išėjimo funkcija:

$$\lambda : S \to Y \cup \{\phi\}$$

Pažymėtos vidinės būsenos ant išėjimo. Išėjimo įvykiai tik sukurti DEVS modelio vidinio perėjimo laiku. Kai būsena prieš perėjimą naudojama kaip įėjimas λ . Visą kitą laiką ne-įvykis Ø yra išėjimas. Aprašinėjant pilną būsenų sistemą kiekviename laiko taške nuosekli būsena s priklausomumo S nepakankama. Baigtinis laikas e kaip sistema padarė perėjimą į besikeičiančią būseną s, kad būti laiko intervale tam, kad kurti pilną būsenų rinkinį.

$$Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}$$

Baigtinis laikas e turi vertes intervale nuo 0 (perėjimas ką tik padarytas) iki ta(s) (besiruošiantis daryti perėjimą į sekančią nuoseklią būseną). Dažnai laikas σ būsenoje yra naudojamas

$$\sigma = ta(s) - e$$

Dar ir dabar tik autonominė sistema buvo aprašyta: sistema negauna jokių išorinių įėjimų. Vadinasi įėjimas X nustato visų galimų įėjimų verčių reikšmes. Kaip taisyklė X bus struktūrizuota aibė.

$$X = \times_{i=1}^{m} X_i$$

Tai formalizuoja daug kartiniai (m) įėjimo portai. Kiekvienas portas yra identifikuotas jo

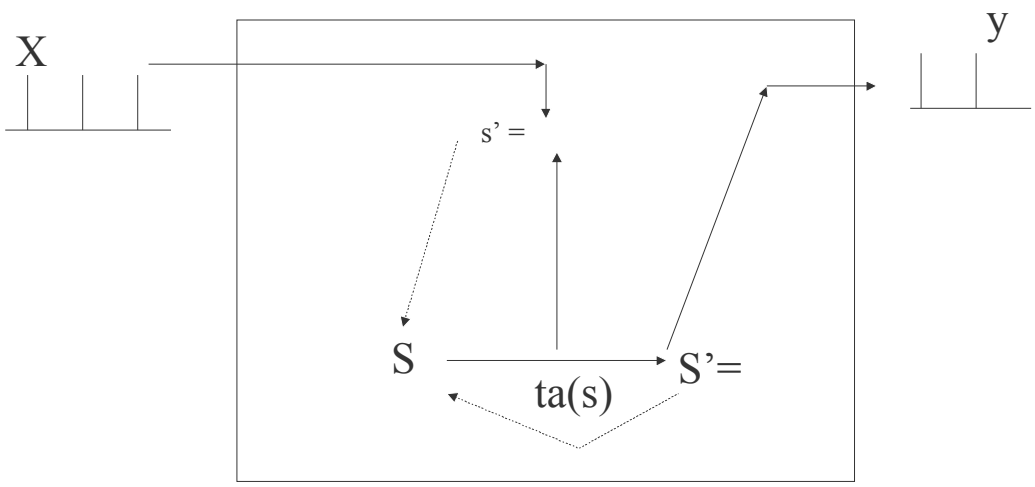
unikaliu indeksu i. Kaip ir išėjimų aibės portų indeksais galima pažymėti pavadinimus. Aibė Ω turi visus galimus įėjimus, segmentuotus ω

$$\omega : T \to X \cup \{\phi\}$$

Diskretinio įvykio modeliavimo sistemose įėjimai sukelia įėjimo įvykį, skirtingą nuo ne įvykio, tik galutiniam momentų skaičiuje apibrėžtame laiko intervale. Šie išoriniai įvykiai įėjimai x nuo X yra sistemos priežastis pertraukti autonominę būseną ir reguliuoti eigoje nurodytą išorinės perėjimų funkciją.

$$\delta_{ext} : Q \times X \to S$$

Sistemos reakcija į išorinį įvykį priklauso nuo nuoseklios būsenos, jei sistema yra ypatingame įėjime ir baigtiniame laike. Tokiu būdu δ_{ext} aprašo didelę tipinę perėjimų grupę, atrastų diskretinio įvykio modelyje (įskaitant sinchronizaciją, iškrovą, pristabdymą ir atgaivinimą). Kai įėjimo įvykis x neįvestas į atominio modelio δ_{ext} specifinį sąrašą, įvykis ignoruojamas.



3 pav. DEVS veikimas

Tarkime, jog bet kuriuo laiko momentu sistema yra kažkurioje būsenoje s. Laiko kitimo funkcija *ta(s)* nusako laiko tarpą iki sekančios būsenos kaitos. Jei išorinis įvykis neįvyksta, sistema pasilieka būsenoje *s ta(s)* laiką.

Pirmu atveju, buvimas būsenoje s toks trumpas, kad jokie išoriniai įvykiai negali įtakoti, tai yra *s* yra perėjimo būsena. Antru atveju, sistema nepasilieka būsenoje *s* visą laiką, jei išorinis

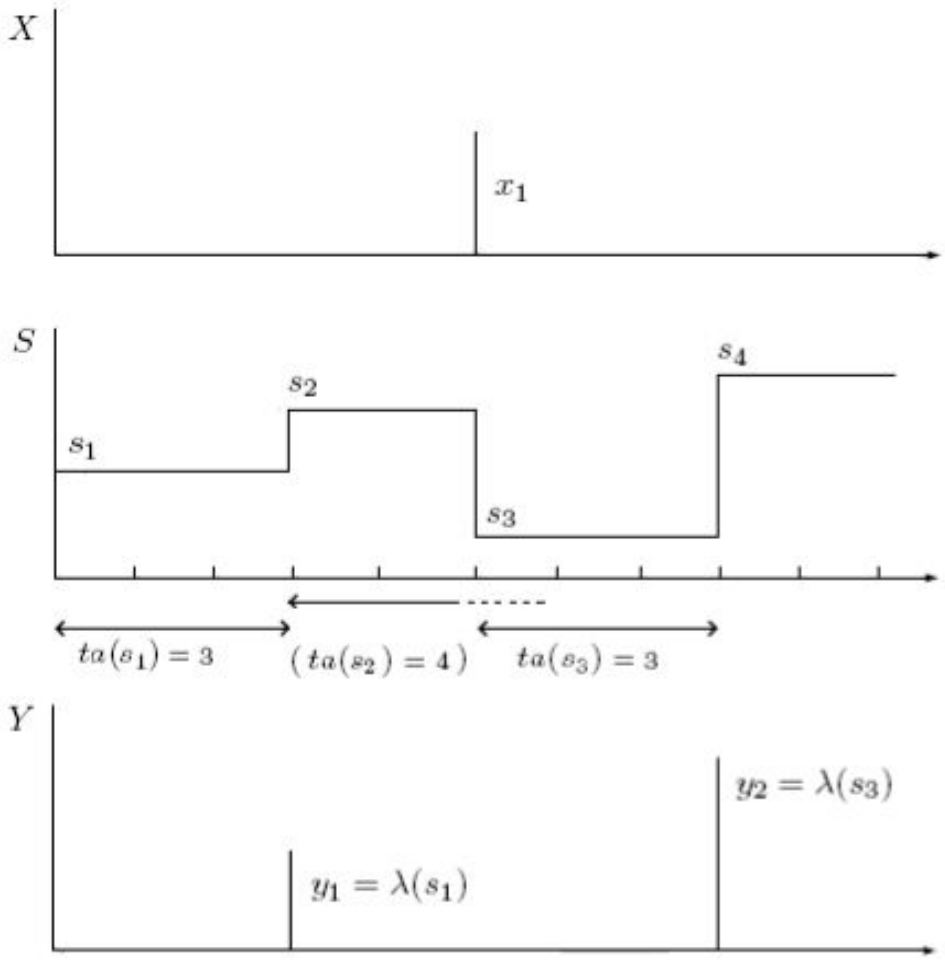
įvykis pertraukia jo budėjimą.

Sakome, kad *s* šiuo atveju yra pasyvi būsena. Kai budėjimo laikas baigiasi, t.y. kai baigtinis laikas $e = ta(s)$ (e – nusako laiko tarpą iki sekančios transakcijos), sistemos išėjimo funkcija $\lambda(s)$ siunčia išėjimo reikšmę, ir vyksta būsenos pokytis iš *s* į *s'*, kuri aprašoma $\delta_{int}(s)$ funkcija. Išėjimas galimas tik prieš vidinį perėjimą.

Jei išorinis įvykis įvyksta prieš šį laiko pasibaigimą, t.y. kai sistema yra pilnoje būsenoje (s,e) su $e \leq ta(s)$, tai vyksta išorinė būsenų kaita, kuri aprašoma $\delta_{ext}(s,e,x)$ funkcija. Ši funkcija priklauso nuo įėjimo *x*, besikeičiančios būsenos *s* ir kaip ilgai sistema buvo šioje būsenoje *e*, kai įvyko išorinis įvykis. Tokiu būdu vidinė perėjimo funkcija sukuria naują sistemos būseną, kai neįvyko jokie įvykiai nuo paskutinio perėjimo.

Abiem atvejais sistema yra kažkurioje būsenoje *s'* su kažkokiu nauju budėjimo laiku *ta(s')* ir ta pati istorija kartojasi.

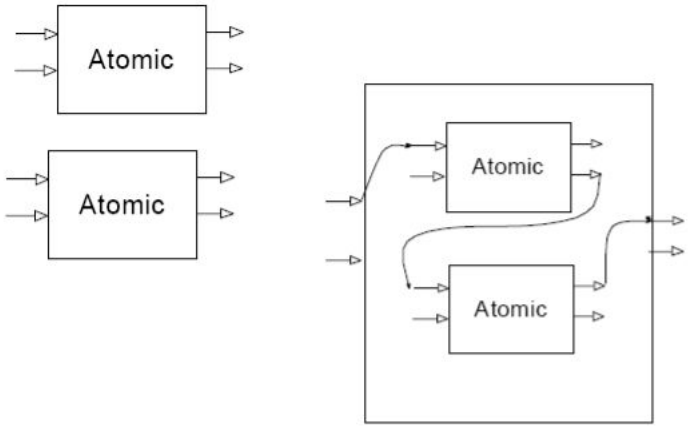
Perspėjimas: nėra jokių būdų generuoti išėjimą tiesiai nuo išorinio įvykio. Išėjimas gali tik įvykti kaip tik prieš vidinį perėjimą. Turint išorinį įvykį sukuriamas išėjimas be vėlavimo, buvo pažymėta vidinė būsena su nuliniu vėlinimo laiku.



4 pav. Ryšys tarp išorinių perėjimų, vidinių perėjimų ir išėjimų

Jei nėra išorinių įėjimo įvykių, sistema pasilieka būsenoje *S1 ta(S1)* laiką. Per šį periodą praėjęs laikas e pasikeičia nuo 0 iki *ta(s1)*, su bendra būsena *a=(s1,e)*. Kai praėjęs laikas pasiekia vertę *ta(s1)*, iš pradžių yra sugeneruojamas išėjimas: *y1=λ(s1)*, tada sistema akimirksniu pereina į naują būseną $s2=\delta_{int}(s1)$. Jei čia būtų automatinis rėžimas, sistemą pasiliktų būsenoje *s2 ta(s2)* laiką ir tada pereitų į būseną $s3=\delta_{int}(s2)$ (po išėjimo generavimo). Tačiau prieš tai kai *e* pasiekia *ta(s2)*, pasirodo išorinis įėjimo poveikis *x*. Tuo metu sistema pamiršta numatytą vidinį perėjimą ir pereina į būseną $s3=\delta_{int}((s2,e),x)$. Atkreipkite dėmesį kaip išorinis poveikis nepaveikia išėjimo. Tarkime, kad būsenoje *s3* sistema veikia automatiniu rėžimu.

2.2.2. Jungtinis DEVS



5pav. Jungtinis DEVS

DEVS suteikia galimybę aprašyti sistemos elgseną dviem lygiais. Žemesniame lygyje bazinis DEVS aprašo autonominę diskretinių įvykių sistemos elgseną kaip deterministinių perėjimų tarp nuoseklių būsenų seką, taip pat reakciją į išorinį įėjimą (įvykį) ir kaip yra generuojamas išėjimas (įvykis). Aukštesniame lygyje, sujungtas DEVS aprašo sistemą kaip komponenčių tinklą. Komponentėmis gali būti baziniai DEVS modeliai arba sujungti DEVS. Ryšiai aprašo kaip komponentės veikia viena kitą[15].

Sujungtas modelis perteikia tokią informaciją:

- komponenčių aibę;
- komponenčių poveikius;
- įėjimus, per kuriuos priimami išoriniai įvykiai;
- išėjimus, per kuriuos siunčiami išoriniai įvykiai;
- komponenčių sujungimo taisykles:
    - išorinis įėjimų sujungimas jungia modelio įėjimus su vienu ar daugiau komponenčių įėjimų;
    - išorinis išėjimų sujungimas jungia komponenčių išėjimus su vienu ar daugiau sujungto modelio išėjimų;
    - vidinis sujungimas jungia komponenčių išėjimus su kitų komponenčių įėjimais.

$$\mathbf{coupledDEVS} \equiv \langle X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select \rangle; \quad (3)$$

Čia:

X_{self} - galimų išorinių įėjimų aibė;

Y_{self} - galimų išėjimų aibė;

D – unikalių subkomponentų aibė;

$\{M_i \mid i \in D\}$ - komponenčių modelis;

I_i, $i \in D$ – i komponentės poveikių aibė;

$\{Z_{i,j}\}$ – išėjimo į įėjimą transliavimo funkcijų aibė;

select : $2^D \rightarrow D$ – pradinė funkcija, kuri nusprendžia simuliavimo įvykių praplėtimą.

Jeigu kiekviena komponentė yra bazinė DEVS komponentė:

$$\mathbf{M_i = \langle S_i, ta_i, \delta_{int,i}, X_i, \delta_{ext,i}, Y_i, \lambda_i \rangle, \forall i \in D;} \qquad (4)$$

Komponentės i poveikių aibė yra I_i. Visų poveikių aibė aprašo tinklo struktūrą.

$$\{I_i \mid i \in D \cup \{self\}\};$$

Dėl modulinės struktūros, komponentė (įskaitant *self*) negali tiesiogiai veikti jokios kitos modulio komponentės arba patį sujungtą modelį.

$$\forall i \in D \cup \{self\}: I_i \subseteq D \cup \{self\};$$

Papildomai dar įvedamas apribojimas, kad nei viena komponentė negali tiesiogiai veikti pati save, nes tai gali būti priežastimi atsirasti momentiniam ciklui, panašiai kaip tolydinėse sistemose algebrinė kilpa:

$$\forall i \in D \cup \{self\}: i \notin I_i;$$

Pastaba: kiekviena komponentė gali visada užkoduoti savo kilpą ($i \in I_i$) į vidinę perėjimo funkciją.

Kiekvienas i-tosios komponentės išėjimas prieš tapdamas j-tosios komponentės įėjimu turi būti transliuojamas. Komponentės išėjimo įvykiui paversti atitinkamu įėjimo įvykiu, naudojama funkcija $Z_{i,j}$.

$$\{Z_{i,j} \mid i \in D \cup \{self\}, j \in I_i\};$$

$$Z_{self,j} : X_{self} \rightarrow X_j, \forall j \in D;$$

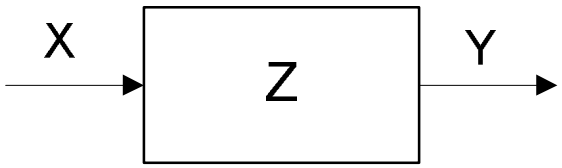
$$Z_{i,self} : Y_i \rightarrow Y_{self}, \forall i \in D;$$

$$Z_{i,j} : Y_i \rightarrow X_j, \forall i,j \in D;$$

Funkcijos I_i ir $Z_{i,j}$ pilnai aprašo sujungto modelio struktūrą ir elgseną.

2.3. Atkarpomis tiesinis agregatas

Agregatiniame sistemos specifikavimo požiūryje sistema atvaizduojama kaip tarpusavyje sąveikaujančių atkarpomis tiesinių agregatų (angl. piece-linear agregate - PLA) aibė [10]. PLA suprantamas kaip objektas su apibrėžtų būsenų aibe Z, įėjimo signalų X ir išėjimo signalų Y aibėmis (6 pav.).



6 pav. Agregato struktūra

$X = \{x_1, x_2, \ldots\}$

$Y = \{y_1, y_2, \ldots\}$

$Z = \{z_1, z_2, \ldots\}$

Agregato funkcionavimas analizuojamas laiko momentų aibėje $t \in T$. Būsena $z \in Z$, įėjimo signalas $x \in X$ ir išėjimo signalas $y \in Y$ laikomi laikinėmis funkcijomis. Be šių aibių taip pat turi būti apibrėžti perėjimo H ir išėjimo G operatoriai.

Atkarpomis tiesinio agregato būsena $z \in Z$ kiekvienu laiko momentu atitinka atkarpomis tiesinio Markovo proceso būseną:

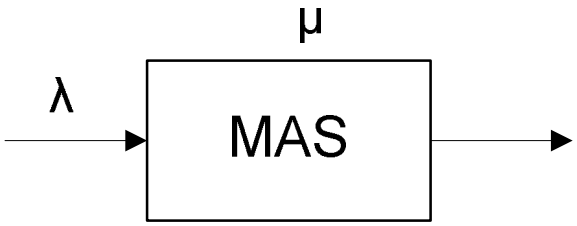
$z(t) = (y(t), z_v(t))$, kur $v(t)$ yra diskretusis būsenos komponentas, įgyjantis reikšmes iš baigtinių reikšmių aibės; ir $z_v(t)$ yra tolydžioji komponentė, susidedanti iš $z_{v1}(t), z_{v2}(t), z_{v3}, \ldots, z_{vk}(t)$ koordinačių. Ši komponentė nusako kada įvyks įvykis, galintis keisti agregato būseną.

Agregato būsena gali keistis tiktai dėl dviejų priežasčių: kai į agregatą ateina įėjimo signalas arba tolydinė komponentė pasiekia apibrėžtą reikšmę. Įėjimo signalo priėmimo faktas vadinamas išoriniu įvykiu. Nesant įėjimo signalams, agregato būsena kinta pagal sekančią priklausomybę:

$$v(t) = const, \frac{dz_v(t)}{dt} = -a_v,$$ kur $a_v = (a_{v1}, a_{v2}, \ldots, a_{vk})$ pastovus vektorius.

Tuo atveju, kai negali įi kažkoks įvykis, atitinkama tolydinės būsenos dedamosios reikšmė yra begalybė.

Kaip atkarpomis tiesinio agregato pavyzdys pateikiama vieno kanalo masinio aptarnavimo sistema (MAS) (7 pav.):



7 pav. Masinio aptarnavimo sistema (MAS)

λ - paraiškų atėjimo į MAS intensyvumas;

μ - paraiškų aptarnavimo intensyvumas.

Agregatų būsenos aprašomos taip:

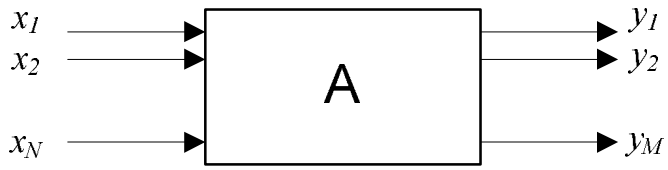
$z(t) = (y(t), z_{v1}(t), z_{v2}(t))$, kur

- $v(t)$ - paraiškų skaičius sistemoje laiko momentu t;
- $z_{v1}(t)$ - laiko tarpas, po kurio į sistemą ateis nauja paraiška;
- $z_{v_2}(t) = \begin{cases} \infty, jei\ v(t) = 0 \\ > 0, jei\ v(t) \neq 0 \end{cases}$

$z_{v2}(t)$ - tai laikas, kada pasibaigs paraiškos aptarnavimas.

Sistemos funkcionavimas apibrėžiamas fizinę reikšmę turinčiomis valdymo ir įėjimo sekomis bei sistemos valdymą nusakančiais algoritmais.

Tarkim, kad agregatas yra sistema turinti N įėjimų bei M išėjimo sąveikos taškų (8 pav.):



8 pav. Agregato įėjimų ir išėjimų schema

Įėjimo signalai $x_1, x_2, ..., x_N \in X$ atsiranda įėjimo sąveikos taške. $x_i \in X_i$, $i = 1,..N$ traktuojamas kaip elementarus signalas, o X_i - elementarių signalų aibė. Bendru atveju elementarus signalas yra vektorinės struktūros: $x_i = \{x^1{}_i, x_i^2, ..., x_i^{ri}\}$, t.y. jis gali būti apibrėžiamas keletu rinkinių, kurių kiekvienas turi reikšmę iš tam tikros aibės X_i : $x_i^j \in X_i^j$, $j = 1,..r_i$. Tokiu būdu į i-tąjį sąveikos tašką ateinančių elementarių signalų seka bus:

$X_i = X_i^1 * X_i^2 * ... * X_i^{ri}$, $i = 1,..N$ Agregato įėjimų seka yra X_i sekų sąjunga:

$$X = \bigcup_{i=1}^{N} X_i$$

Analogiškai ir su išėjimo signalais:

$Y = \{y_1, y_2, ..., y_M\}$;

$y_l = \{ y_l^1, y_l^2, ..., y_l^{si} \} \in Y_l$;

$y_l^k \in Y_l^k$, $l = 1,..M$, $k = 1,..s_1$

l-ojo sąveikos taško elementarių signalų seka:

$Y_l = Y_l^1 * Y_l^2 * ... * Y_l^{sl}$, $l = 1,..M$

Išėjimo signalų aibė:

$$Y = \bigcup_{l=1}^{M} Y_l$$

Agregato funkcionavimas nagrinėjamas diskrečiais laiko momentais $T = \{t_1, t_2, \ldots, t_m, \ldots\}$, kurias gali įvykti vienas ar keletą įvykių, iššaukiančių agregato būsenos pasikeitimą. Agregato būsenų aibė E susideda iš dviejų poaibių: $E = E' \in E''$. Į poaibį $E' = \{e'_1, e'_2, \ldots, e'_N\}$ įeina įvykių klasės/įvykiai $e'_i, i = 1,..N$, sekantys iš įėjimo signalo aibės $X = \{x_1, x_2, \ldots, x_N\}$. Įvykių klasė $e''_j = \{ e''_{ij}, j = 1,2,... \}$, kur e''_{ij} yra įvykių, įvykstančių j-uoju laiko momentu nuo momento t_0, klasės e''_i įvykis. Poaibio E' įvykiai vadinami išoriniais įvykiais. Agregato įėjimo signalų aibė atvaizduojama į poaibį E': $X \rightarrow E'$. Poaibio $E'' = \{e''_1, e''_2, \ldots, e''_f\}$ įvykiai vadinami vidiniais įvykiais, kur $e''_i = \{ e''_{ij}, j = 1,2,... \}$, $i = 1,..f$ yra agregato vidinių įvykių klasės. f apibrėžia agregate vykstančių įvykių skaičių.

Dėl įėjimo įvykių poaibio laiko momentų aibė T dalinama į du poaibius $T = T' \in T''$: T' - įėjimo signalų atsiradimo poaibis ir T'' - vidinių įvykių įvykimo poaibis. Kiekvienai įvykių e''_i klasei iš poaibio E'' valdymo seka apibrėžiama $\{ \xi_j^{(i)} \}$, kur $\xi_j^{(i)}$ - operacijos trukmė.

Agregate įvykstančių operacijų pradžiai ir pabaigai apibrėžti įvedamos kontrolinės sumos sąvokos - $\{s(e''_i, t_m)\}$, $\{w(e''_i, t_m)\}$, $i = 1,..f$, kur $s(e''_i, t_m)$ - operacijos, sekančios po klasės e''_i įvykio, pradžios laiko momentas. Šis laiko momentas yra neapibrėžtas, jei operacija neprasidėjo; $w(e''_i, t_m)$ - operacijos, sekančios po klasės e''_i įvykio, pabaigos laiko momentas. Kontrolinė suma $w(e''_i, t_m)$ apibrėžiama taip:

$$w(e''_i, t_m) = \begin{cases} s(e''_i, t_m) + \xi_{r(e''_i, t_m)+1}, & \text{jei momentu } t_m \text{ įvyksta operacija;} \\ \infty, & \text{priešingu atveju kintamojo reikšmė.} \end{cases}$$

Diskrečioji būsenos komponentė $v(t_m) = \{v_1(t_m), v_2(t_m), ..., v_p(t_m)\}$ nurodo sistemos diskrečiąją būseną. $z_v(t_m) = \{w(e''_1, t_m), w(e''_2, t_m), ..., w(e''_f, t_m)\}$ - valdančioji koordinatė, parodanti

įvykių įvykimo laiką. Ši koordinatė atitinka kiekvieną e"$_i$ iš įvykių poaibio *E″,* nes visada tenkinama sąryšis $w(e''_i, t_m) \geq t_m$.

Būsenos koordinatės $z(t_m)$ gali keisti savo reikšmes tik tam tikrais diskrečiais laiko momentais t_m, m = 0,1,2,..., kur t_0 nurodo sistemos funkcionavimo pradžios laiką.

Žinant sistemos būseną $z(t_m), m$ = 0,1,2,..., sekančio įvykio momentas t_{m+1} apibrėžiamas įėjimo signalo pasirodymu arba lygtimi:

$$t_{m+1} = \min\{w(e_i'', t_m)\}, 1 \leq i \leq f;$$

Sekančio įvykio e_{m+1} klasė nustatoma pagal įėjimo signalą, jei šis atsiranda laiko momentu t_{m+1} arba apibrėžiama valdymo koordinate, kuri įgyja mažiausią reikšmę laiko momentu t_m.

Perėjimo operatorius $H(e_i)$ nusako naują agregato būseną:

$z(t_{m+1}) = H[z(t_m), e_j], e_i \in E' \in E''$.

Išėjimo signalai y_i iš aibės $Y = \{y_1, y_2, ..., y_m\}$ gali būti agregato generuojami įvykių iš poaibių E' ir *E″* įvykimo laiko momentais.

Išvedimo operatorius $G(e_r)$ apibrėžia išėjimo signalų turinį:

$y = G[z(t_m), e_i], e_i \in E' \in E'', y \in Y$.

Pilnas agregatinės specifikacijos aprašymas susideda iš 9 punktų [1]:

1. Įėjimų aibės apibrėžimas – čia kiekvienam agregato įėjimui yra priskiriamas konkreti perduodamo signalo duomenų struktūra.
2. Išėjimų aibės apibrėžimas – analogiškai pirmajam punktui.
3. Išorinių įvykių aibė – įvykiai, įvykę agregato išorėje, bet veikiantys agregato būseną.
4. Vidinių įvykių aibė – įvykiai, įvykę agregato viduje ir keičiantys agregato būseną.
5. Valdančios sekos.
6. Diskretinės būsenos vektoriaus komponentės – aibė agregato elementų būsenų reikšmių, galinčių įgyti diskretines reikšmes.
7. Tolydinės būsenos vektoriaus komponentės – aibė agregato elementų, galinčių įgyti tolydines reikšmes.
8. Pradinė būsena – diskretinių ir tolydinių būsenos vektoriaus komponenčių pradinės reikšmės
9. Būsenos pakeitimo (H) ir signalų perdavimo (G) operatoriai – algoritmai (sakinių

sekos), pagal kuriuos keičiama agregato būsena ir išduodami signalai į agregato išorę.

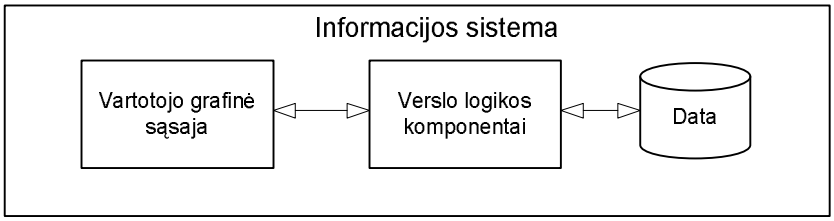
Pagal šiuos 9 punktus galima pilnai aprašyti kiekvieną sistemoje esantį agregatą, bet prieš tai reikia būti aprašius galimas perduodamų signalų struktūras.

Tam, kad būtų galima panaudoti formalizuotus imitacinius modelius, reikia šiuos modelius integruoti į verslo valdymo sistemą.

3. LOGISTIKOS IMITACINIO MODELIO INTEGRAVIMAS Į IS

Įmonių informacinės sistemos (IS) - tai reliacinių duomenų bazių, įmonės išteklių planavimo ir transakcijų vykdymo sistemų rinkinys. Tipinė informacinė sistema, veikianti keliuose kompiuteriuose, turi tokį loginį padalinimą [6](9 pav.):

- Pateikties sluoksnis – tvarko naudotojo vizualias sąsajas ir reguliuoja jo sąveiką su sistema.
- Verslo logikos sluoksnis – atlieka operacijas ir modeliuoja verslo procesus.
- Duomenų sluoksnis – yra naudojamas verslo logikos sluoksnio naudojamų duomenų saugojimui.

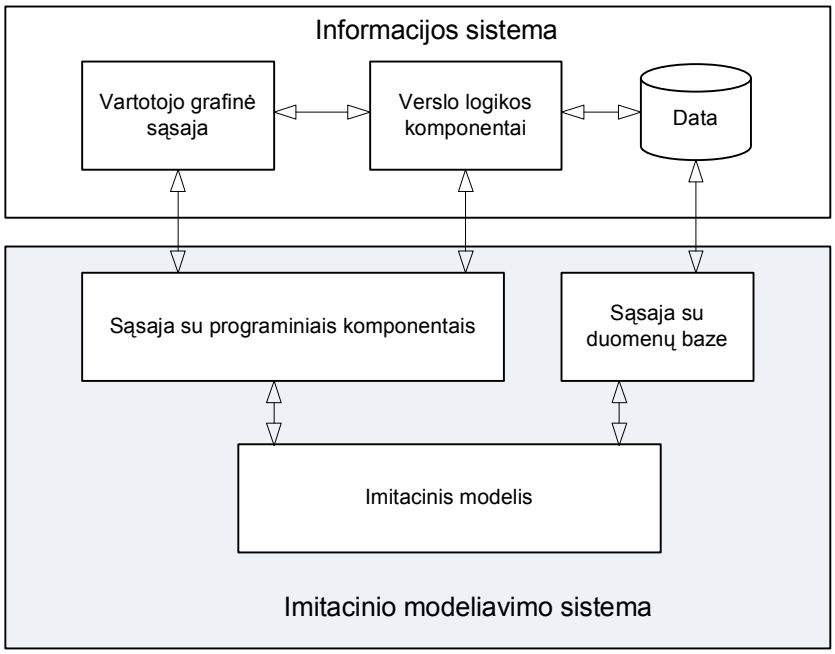


9 pav. Tipinė paskirstytos sistemos architektūra.

Tam kad būtų galima panaudoti imitacinį modelį sprendimo priėmimui tiekimo grandinės sistemoje realiame laike, būtina šį modelį integruoti į verslo valdymo sistemą (IS)[3].

Šiame darbe verslo informatikos katedros siūlomos integravimo sistemos architektūros programiniai komponentai pavaizduoti 10 paveikslėlyje[16]. Nuo tradicinės informacinės sistemos ji skiriasi šiomis dalimis:

- Imitacinio modelio sąsaja su IS programiniais komponentais:
  - Užtikrina imitacinio eksperimento valdymą iš informacinės sistemos (modelio parametrų nustatymą, modelio paleidimą vykdymui);
  - Pateikia imitacinio modeliavimo rezultatus vartotojui.
- Imitacinio modelio sąsaja su IS duomenų baze:
  - Aprūpina imitacini modelį realiais duomenimis iš duomenų bazės.



10 pav. Imitacinių modelių integravimo į IS architektūros loginis modelis.

Sprendimo priėmimui verslo valdymo sistemose reikia naudoti informaciją apie verslo procesų praeitį, dabartį ir prognozuojamą ateitį. Informaciją apie verslo procesų praeitį ir dalinai dabartį galima gauti iš verslo IS DB. Prognozuojamos ateities scenarijams įvertinti galima panaudoti verslo procesų imitacinius modelius[9]. Taigi, imitacinis modelis turi sąveikauti su IS naudojant duomenų ir funkcinę integraciją.

4. IMITACINIŲ MODELIŲ TRANSFORMACIJA Į PROGRAMINĮ KODĄ

Atskirų verslo procesų imitacinių modelių integravimui į bendrą logistikos informacinę sistemą, modelių neapjungsime, bet integruosime juos atskirai. Šio pasirinkimo priežastys:

- Apjungti visus verslo procesų imitacinius modelius į vieną sistemą yra per daug sudėtinga ir neproduktyvu:
- Galima pagerinti projektavimo rezultatus susikoncentravus ties mažesnėmis dalimis.
- Apjungti visus verslo procesų imitacinius modelius į vieną sistemą yra sudėtingas procesas.
- Susikoncentravus ties mažesnėmis dalimis galima pagerinti projektavimo rezultatus.
- Šiandien įmonėse jau egzistuoja atskiros funkcinės informacinės sistemos gamybos veiklai valdyti, atskiroms funkcijoms kompiuterizuoti. Pavyzdžiui, įrenginių kontrolės IS, gamybos planavimo IS, sandėliavimo ir apskaitos IS. Šių informacinių sistemų tarpusavio integracija būtų pigesnė ir paprastesnė, nei visų sistemų perprojektavimas į naują vientisą sistemą.
- Ne visada įmanoma realizuoti sistemą viename programiniame pakete;

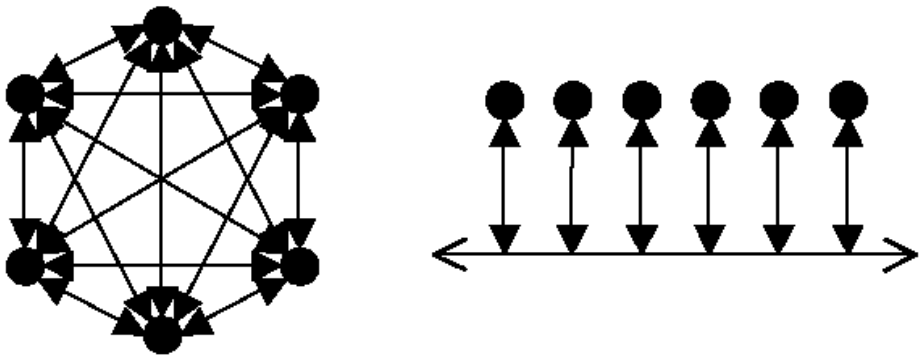
Kadangi kompanijose vyrauja skirtingi interesai, veiklos sritys, skirtingi valdymo lygiai, naudojamos ir įvairios sistemos. Nė viena pavienė informacinė sistema negali patenkinti visų informacijos ir valdymo poreikių. Jos yra skiriamos įvairiems valdymo lygiams ir skirtingoms funkcijoms [1].

Reikia automatizuoti verslo procesus integruotoje sistemoje, taip užtikrinant vieningą būdą prieiti prie informacijos, išbarstytos skirtingose sistemose.

Naudojant elektronines paslaugas, kaip bendrą sąsają su vidinėmis ir su išorinėmis įmonės programomis, išvengiama būtinybė kurti atskiras sąsajas tarp atskirų sistemų.

Kelių modelių integravimui į bendrą sistemą reikalinga infrastruktūra, užtikrinanti duomenų mainus tarp integruojamų sistemų. Tinklinėje struktūroje egzistuoja dvi galimybės:

- Paskirstytas modeliavimas (11a pav.).
- Lygiagretus modeliavimas, kuris vykdomas paskirstytos atminties multiprocesoriuose arba multikompiuteriuose (11b pav.).



a) b)

11. Paskirstytas ir lygiagretus modeliai.

Paskirstyto modeliavimo metodas yra efektyvesnis, nes lygiagretus metodas yra orientuotas į modeliavimo laiko mažinimą, o paskirstytas yra orientuotas į tarpusavio sąveikos ryšių nustatymą tarp skirtingų techninės ir programinės įrangos platformų.

4.1. Paskirstytas modeliavimas

Paskirstytos sistemos - tai programų sistema, sudaryta iš dviejų ar daugiau autonominių komponentų, kurie laikomi nepriklausomomis sistemomis, bet kartu veikiantys kaip vientisa sistema, t.y. sistemos skirtingos dalys yra vykdomos skirtinguose kompiuteriuose, atlieka skirtingus veiksmus ir tarpusavyje komunikuoja įvairiais būdais [14].

Paskirstyto modeliavimo privalumai:

- Paskirstytose informacinėse sistemose visų tipų kompiuteriai (didieji, vidutinio dydžio, asmeniniai) sujungti kompiuteriniu tinklu ir yra skirtingose, kartais labai nutolusiose vietose (kitoje šalyje ir net kitame žemyne).
- Paskirstytojo apdorojimo sistemos padalija darbą tarp dviejų ar daugiau kompiuterių.
- Paskirstytosios IS pagerina darbų koordinavimą, padeda paskirstyti duomenis ir darbo rezultatus, leidžia paskirstyti skaičiavimo sistemos išteklius:
    - galima siųsti žinutes arba failus kitiems kompiuteriams tinkle;
    - galima prieiti prie duomenų, esančių skirtingame kompiuteryje, bendrai naudoti spausdintuvus ar kitokius įrenginius.
- Paskirstytoji aplinka yra labai slanki (greitai perderinama pagal naujus poreikius), ją

naudoja vis daugiau vidutinių ir didelių organizacijų.

- Paskirstytosios sistemos apima kompiuterinius tinklus ir telekomunikacijas.

Paskirstyto modeliavimo realizacijai galima taikyti:

- HLA (aukšto lygio architektūra);
- Žiniatinklio paslaugas („web services“).

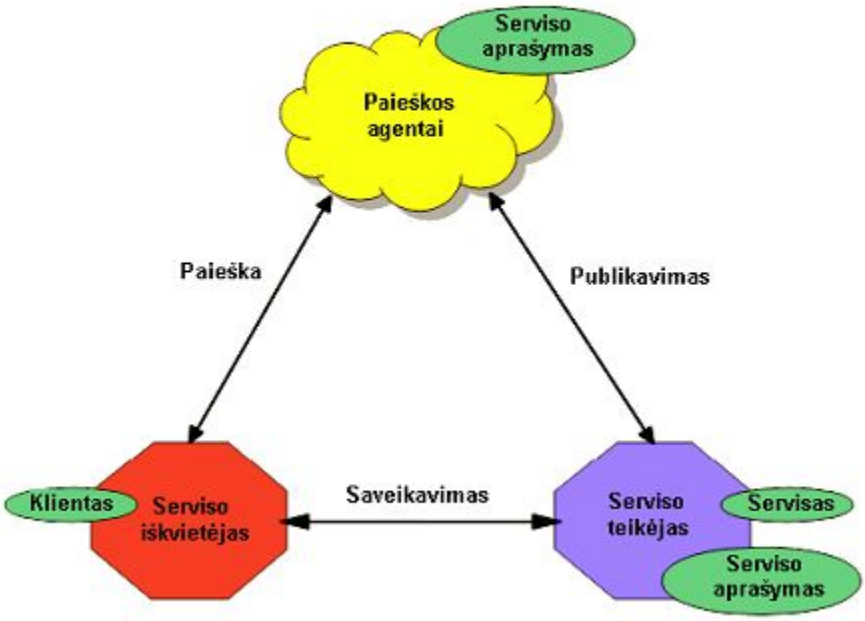
HLA užtikrina modelių tarpusavio sąveiką, pakartotiną modelių panaudojimą bei realizacija modeliuojamo laiko tarp atskirų modelių sinchronizaciją bei duomenų paskirstymą.

Kitas būdas yra žiniatinklio paslaugų („web services“) panaudojimas[5]. Ši architektūra leidžia skirtingoms programoms bendrauti nepriklausomai nuo jų sukūrimui naudotų programavimo kalbų ir technologijų.

4.2. Žiniatinklio paslaugos (WEB services)

Žiniatinklio paslaugų („WEB servisų“- *web service*) naudojimas internetinių aplikacijų kūrime sparčiai plinta. „Web servisus“ galima trumpai apibūdinti kaip funkcijas ar objektus, veikiančius internete, kuriuos galima naudoti savo programose panašiai kaip įprastines funkcijas ar objektus (Atul Bhatt, 2001) (12 pav.). Žiniatinklio paslaugų technologija sukurta interneto TCP/IP, HTML protokolų ir universalios duomenų aprašymo kalbos XML pagrindu. Ji naudoja atvirus standartus, kurie nepriklauso nuo vieno gamintojo ir yra visuotinai palaikomi:

- SOAP (Single Object Access Protocol) – bendras objekto pasiekimo protokolas, apibrėžiantis taisykles kaip turi būti struktūrizuota XML žinutė, naudojama bendravimui su elektronine paslauga.
- WSDL (Web Service Definition Language) – žiniatinklio paslaugų aprašymo kalba, naudojama standartiškam žiniatinklio paslaugos aprašymui.
- UDDI (Universal Description Discovery Integration) – universalus aprašymo, suradimo ir integravimo standartas, naudojamas sukurti žiniatinklio paslaugų registrus, kuriuose gali būti atliekama elektroninių paslaugų paieška.



12 pav. „WEB servisų“ naudojimas interneto taikomosiose programose.

WEB servisai naudojami programinės įrangos integracijai per internetinį tinklą. Daugelis organizacijų naudoja WEB servisus sistemų integracijai intranete. Ši technologija leidžia kurti naujas sistemas bei tobulinti senas nepriklausomai nuo esamų sprendimų bei turimos įrangos, neįtakojant vartotojų darbo. Sistemų tarpusavio sąryšiai yra apsprendžiami ne sistemų, o protokolų lygmenyje. WEB servisai veikia nepriklausomai nuo įrangos, operacinės sistemos ir pan. XML kalba buvo panaudota kaip duomenų perdavimo standartas.

Vienas iš WEB servisų trūkumų yra XML kalbos naudojimas: siunčiant duomenis XML formatu, jie gali užimti didelę dalį ryšio kanalo. Kita vertus ryšys kiekvieną dieną tik gerėja, be to, XML gali būti efektyviai suspaudžiamas, naudojant įvairius algoritmus. Alternatyvios kalbos nepaplito dėl per didelio sudėtingumo. Taigi, buvo pasirinktas paprastumas – XML, kaip duomenų perdavimo standartas.

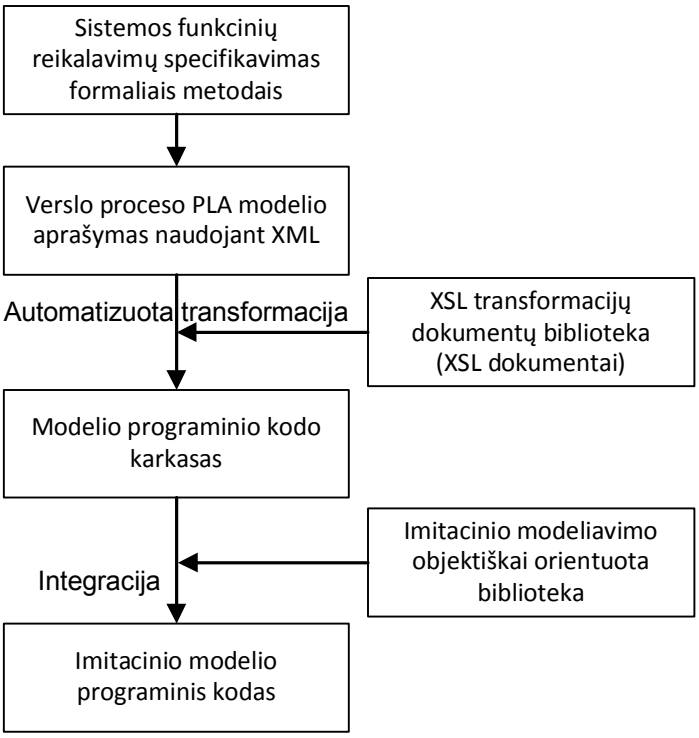
Šio metodo privalumai:

- Sutrikus vienai sistemai, kitos sistemos gali toliau be problemų funkcionuoti.
- Visa struktūra yra lengvai konfigūruojama ir praplečiama, tai yra labai svarbu dirbant su dinaminiu (nuolat besikeičiančiu) tiekimo grandinės tinklu.
- Imitacinio eksperimento valdymo modulis gali būti pasiekiamas tiek tiesiogiai iš vartotojo programos arba, bendresniu atveju, per atitinkamą žiniatinklio paslaugą.

4.3. Imitacinio modelio transformacijos technologija

Kauno technologijos universitete verslo informatikos katedroje buvo sukurta technologija, kuri PLA specifikacija formalizuotus modelius automatiškai transformuoja į programinį kodą[9]. Išnagrinėsime šią technologiją (13 pav).

- Siūloma metodika apima tokius projektavimo etapus:
- Verslo procesų formalaus PLA modelio sudarymas.
- Verslo procesų PLA formalios specifikacijos transformacija į imitacinio modelio programinį kodą.
- Imitacinio modelio programinio kodo integravimas į verslo valdymo sistemą.

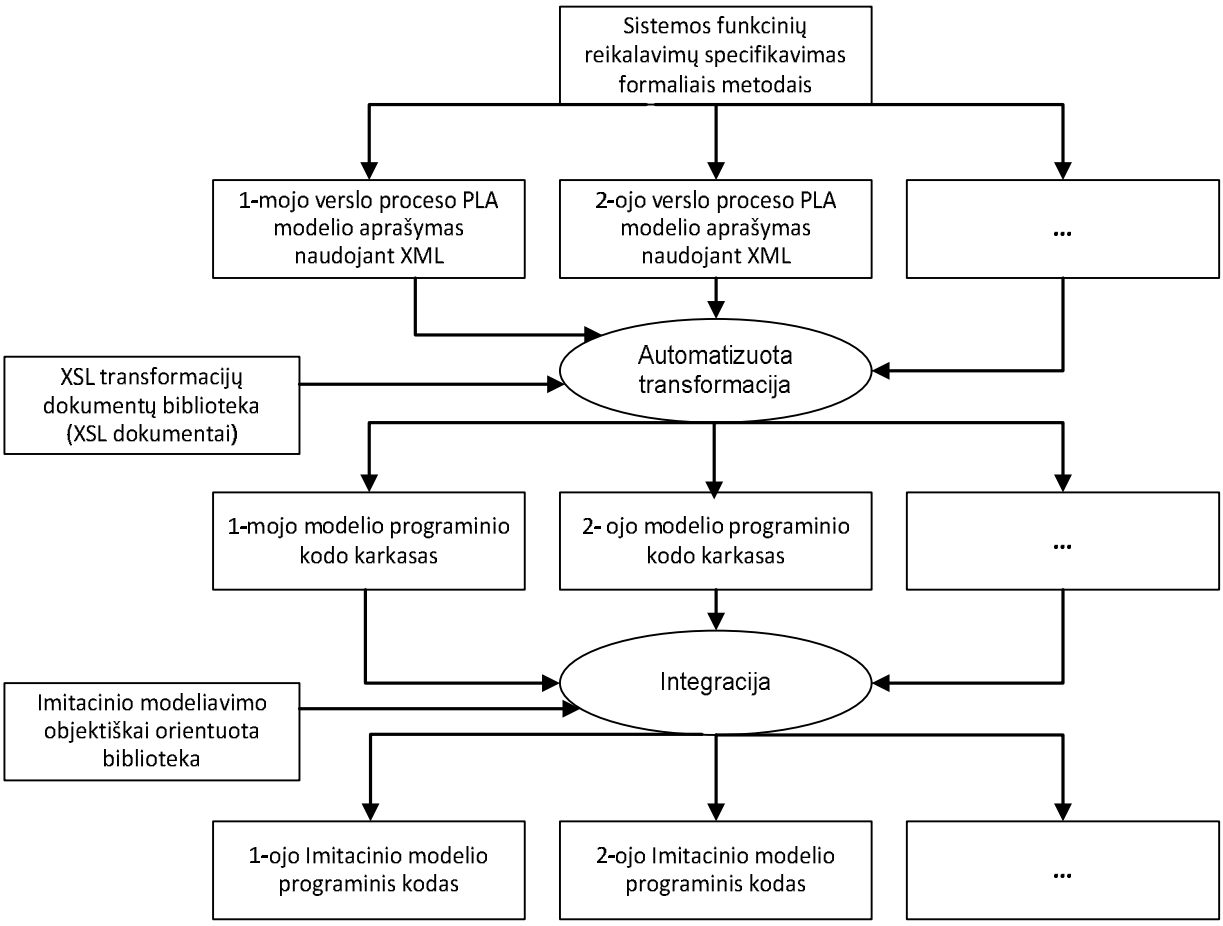


13. pav. Imitacinio modelio programinio kodo generavimo seka

Sudaromas verslo procesų formalus PLA modelis ir jo aprašymas išsaugomas XML formate. Pagal šį modelio aprašymą generuojamas imitacinio modelio programinis modelis. Kadangi sugeneruotas kodas atitinka tik verslo procesų modelio aprašymą, jis yra apjungiamas su imitacinio modeliavimo biblioteka, realizuojančia imitacini modeliavimo algoritmą PLA formaliems aprašymams.

Toks programinio kodo kūrimo procesas tinkamas programų sistemoms, kurios nėra

išskirstytos, o veikia, kaip viena vientisa sistema. Todėl kūrimo procesas turi būti padalintas į kelias atskiras dalis, o tos dalys kuriamos atskirai, vėliau apjungiamos į vieną produktą. Taigi, paskirstyto modeliavimo atveju naudojamas transformavimo procesas yra sudėtingesnis (14 pav.)



14 pav. Transformacijos technologinė schema

5. Imitacinių modelių specifikavimas

Pateikiame gamybos ir paskirstymo imitacinių modelių matematinius aprašus, naudojant DEVS ir PLA specifikacijų metodus.

5.1. Gamybos proceso imitacinio modelio sudarymas naudojant DEVS

Kaip gamybos proceso imitacinio modelio sudarymo pavyzdį, sumodeliuosime ir formalizuosime automobilių gamyklos modelį (imitacinis automobilių gamyklos modelis pateiktas priede 1).

Šis modelis sudarytas iš trijų atominių modelių ir vieno jungtinio modelio "Variklio gamykla". Variklio gamyklos modelis sudarytas iš trijų atominių modelių. Įėjimas paduodamas į "Važiuoklės gamyklą", "Kėbulo gamykla", "Pavarų dėžės gamykla" ir "Variklio gamykla", kad pradėti gamyba. Šių padalinių produktai siunčiami į galutinę "Galutinio surinkimo gamykla" ir yra surenkama galutinis produktas - mašina. Kadangi gamyklos padaliniai turi nevienodą gamybos lygį, tai kai kurie padaliniai turi laukti kol kiti pabaigs gamybą. Reikia rasti geresnį sprendimą.

Jungtinis modelis:

CMAutFact=<X,Y,{Chassis,Body,Trans,CMEngFact,FinalAss},EIC,EOC,IC,SELECT>

Čia:

X = {in}

Y = {out}

EIC = { (CMAutFact.in, Chassis.in), (CMAutFact.in, Body.in), (CMAutFact.in, Trans.in), (CMAutFact.in, CMEngFact.in) }

EOC = { (FinalAss.out, CMAutFact.out) }

IC = { (Chassis.out, FinalAss.in_chassis), (Body.out, FinalAss.in_body), (Trans.out, FinalAss.in_trans), (CMEngFact.out, FinalAss.in_engine) }

CMEngFact = <X, Y, {Piston, EngineBody, EngineAss}, EIC, EOC, IC, SELECT>

Čia:

X = {in}

Y = {out}

EIC = { (CMEngFact.in, Piston.in), (CMEngFact.in, EngineBody.in) }

EOC = { (CMEngFact.out, FinalAss.out) }

IC = {(Piston.out, EngineAss.in_piston), (EngineBody.out, EngineAss.in_eingineBody)}

Atominis modelis:

Chassis = <X, S, Y, δint, δext, λ, D >

Čia:

X = {in, done}

Y = {out}

S = {elements, phase, sigma, manufacturingTime}

```
δext (S, e, X) {
   if( msg.port() == in ) {  //nauja užduotis
        int n=(int) msg.value();          //dalių skaičius įdedamas į eilę
           elements.push_back( 1 ) ; //pradedama gaminti viena dalis
            if( elements.size() == 1 )
                 holdIn( active, manufacturingTime );
        //įdėti likusius vientus į eilę, jei gauta daugiau nei 1.
        for(int i=2;i<=n;i++)
           elements.push_back( 1 ) ;
    }
    if( msg.port() == done ) { //nustatoma,kad žinutė išsiusta
        elements.pop_front();//jei eilė ne tužčia imama sekanti dalis
    if( !elements.empty() )
        holdIn( active, manufacturingTime );
 }
            }
δint (S){ //pažymima, kad žinutė išsiusta ir laukiama naujo įvykio
   passivate();
}
λ{ // siunčia dalis per išėjimo portą
   sendOutput( msg.time(), out, elements.front() ) ;
```

}

Kiti modeliai - Body, Trans, Piston, EngineBody tu panašias struktūras ir elgsenas kaip Chassis.

EngineAss = <X, S, Y, δint, δext, λ, D >

X = {in_piston, in_engineBody, done}

Y = {out}

S = { elements_piston, elements_engineBody, phase, sigma, manufacturingTime}

δext (S, e, X) {

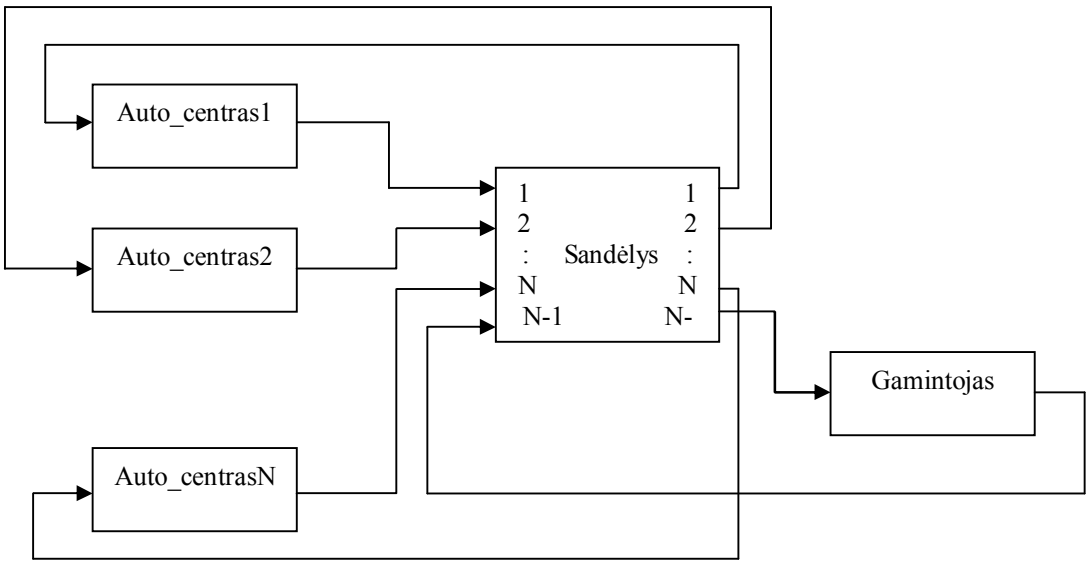
```
   int n=(int) msg.value();
   if( msg.port() == in_piston){ // dalys paimamos viena po kitos
        elements_piston.push_back( 1 ) ;
        if(elements_piston.size()==1&&elements_engineBody.size()>=1)
        //gaminama viena dalis
           holdIn( active, manufacturingTime );
           //jei daugiau nei viena atejo, tai dedama į eilę
        for(int i=2;i<=n;i++)
           elements_piston.push_back( 1 ) ;
    }
    if( msg.port() == in_engineBody ) {
        elements_engineBody.push_back( 1 ) ;
        if(elements_engineBody.size()==1&&elements_piston.size()>=1)
           holdIn( active, manufacturingTime );
        //jei daugiau nei viena atejo, tai dedama į eilę
           for(int i=2;i<=n;i++)
                elements_engineBody.push_back( 1 ) ;
    }
    if( msg.port() == done ) {  //nustatoma, kad žinutė išsiusta
        elements_piston.pop_front() ;
        elements_engineBody.pop_front() ;
        //jei eilė ne tužčia imama sekanti dalis
```

```
        if( !elements_piston.empty() &&  !elements_engineBody.empty())
            holdIn( active, manufacturingTime );
    }
}
δint (S){ //pažymima, kad žinutė išsiusta ir laukiama naujo įvykio
    passivate();
}
λ{ // siunčia dalis per išėjimo portą
   sendOutput( msg.time(), out, elements.front() ) ;
 }
```

Model FinalAss turi panašią struktūrą kaip ir EngineAss, tik turi daugiau įėjimo portų.

5.2. Logistikos sistemos imitacinio modelio sudarymas

Tarkime, kad sistema sudaryta iš N automobilių salonų, sandėlio ir gamintojo. Kai automobilių kiekis pardavimo centre pasidaro mažesnis už nustatytą ribą, pardavimų centras pateikia sandėliui prekių užsakymą. Kiekvienas užsakymas sandėlyje yra apdorojamas ir, jei čia esančių prekių pakanka, jos yra pristatomos į automobilių saloną. Kitu atveju, užsakymas yra statomas į užsakymų eilę. Agregatų specifikacija sudaryta iš *Prekybos_centras_i, i=1,N*; *Sandėlys ir Gamintojas* (15 pav.).



15 pav. Agregatų sujungimo schema

Auto_centrask turi vieną įeinantį ir vieną išeinantį pranešimų srautą. Kai centrų yra 2, konteineris *Sandėlys* turi 3 įėjimus ir 3 išėjimus. *Gamintojas* atitinkamai turi vieną įėjimą ir vieną išėjimą. Bendravimas vyksta pranešimų srautais.

***Auto_centrok* aprašymas:**

1. Įėjimų signalų aibė $X=\{x_1\}$, čia $x_1=(x_{11}, x_{12})$, $x_{11}=k$, x_{i2} – pristatytų prekių kiekis.
2. Išėjimo signalų aibė $Y=\{y_1\}$, kur $y_1=(y_{11}, y_{12})$ ir $y_{11}=k$, y_{12} – užsakytų prekių kiekis.
3. Išorinių įvykių aibė $E'=\{ e'_1\}$; kur e'_1 – prekės buvo pristatytos.
4. Vidinių įvykių aibė $E''=\{ e''_1\}$; kur e''_1 – laikas, kai centre prekių kiekis yra įvertintas.
5. Valdymo sekos: $e''_i \rightarrow \{\xi_j\}$, $j=1,\infty$, kur ξ_j – laiko tarpas, per kurį prekių kiekis automobilių centre yra įvertinamas.
6. Parametrai: Δs – užsakytų prekių kiekis, s_0 – pradžia. Kai $s(t_m)$ pasidaro mažesnis nei s_0, yra išduodamas prekės užsakymas.
7. Diskrečioji agregato būsenos dedamoji $\nu(t_m)=\{s(t_m), s_0(t_m)\}$; čia $s(t_m)$ – prekių atsargos automobilių centre, $s_0(t_m)$ – nustatyta riba.
8. Tolydžioji agregato būsenos dedamoji $z_\nu(t_m)=\{w(e''_1, t_m)\}$, kur $w(e''_1, t_m)$ – prekių kiekio įvertinimo pabaigos momentas.
9. Agregato pradinė būsena: $s(t_0)\geq s_0$, $w(e''_1, t_m)=t_0+\xi_1$, $s(t_0)=s_0$.
10. Perėjimo ir išėjimo operatoriai:

    $H(e'_1)$:

    $$s(t_{m+1})=s(t_m)+x_{12}.$$

    $H(e''_1)$:

    $$s(t_{m+1})=\max(0, s(t_m)-\eta_j),$$ kur η_j – prekių paklausa.

    $$w(e''_1, t_{m+1})=t_m+\xi_j$$

    $G(e''_1)$:

    $$y_1=(k, \Delta s),$$ jei $s(t_m)\eta \geq s_0 > s(t_{m+1})$.

Bendravimas tarp konteinerių vyksta pranešimų srautais („Message Flow“). 9 punkte

nurodyta, kad pradinė būsena yra nustatoma laiko momentu t_0, kuriam įvykus apibrėžiama $w(e''_1, t_m)=t_0+\xi_1$ ir yra nustatoma pradinė prekių atsargų reikšmė s. Specifikacijos 10-tame punkte yra aprašyti perėjimo ir išėjimo operatoriai. Operatorius $H(e'_1)$ nusako, kad, priėmus signal x_1, atsargų kiekis automobilių centre padidėja. Operatorius $H(e'_1)$ parodo, jog reikia suskaičiuoti prekybos centre prekių likutį ir nustatyti laiką iki kito prekių atsargų įvetinimo. Operatorius $G(e''_1)$ nurodo, kad mes išduodame signalą apie naują prekių užsakymą tuo atveju, jei prekių kiekis peržengė nustatytą ribą s_0.

Gamintojas **aprašymas:**

1. Įėjimų signalų aibė $X=\{x_1\}$, čia x_1 – prekių pristatymo paraiška.
2. Išėjimo signalų aibė $Y=\{y_1\}$, kur y_1 - pristatytų prekių kiekis.
3. Išorinių įvykių aibė $E'=\{e'_1\}$; kur e'_1 – signalo x_1 atėjimas.
4. Vidinių įvykių aibė $E''=\{ e''_1\}$; kur e''_1 – laiko tarpas, per kurį prekės išvežamos iš gamintojo.
5. Valdymo sekos: $e''_i \rightarrow \{\xi_j\}$, $j=1,\infty$.
6. Parametrai: $z_v(t_m)=\{w(e''_1, t_m)\}$.
7. Diskrečioji agregato būsenos dedamoji $v(t_m)=\{d(t_m)\}$; čia $d(t_m)$ – prekių kiekis pristatytas į sandėlį.
8. Agregato pradinė būsena: $w(e''_1, t_0)= \infty$, $d(t_0)=0$.
9. Perėjimo ir išėjimo operatoriai:

   $H(e'_1)$:

   $w(e''_1, t_{m+1})=t_m+\xi_j$

   $d(t_{m+1})=x_1$.

   $H(e''_1)$:

   $w(e''_1, t_{m+1})= \infty$,

   $d(t_{m+1})=0$.

   $G(e''_1)$:

   $y_1=d(t_m)$.

8 punkte parašyta, kad pradinė būsena yra nežinoma. Operatorius $G(e''_1)$ nurodo, kad užsakytos prekės išsiunčiamos užsakytu dydžiu.

***Sandėlys* aprašymas:**

1. Įėjimų signalų aibė $X=\{x_1, x_2, ..., x_N, x_{N+1}\}$, čia $x_1=(x_{i1}, x_{i2})$, x_{i1} – parduotuvės numeris, x_{i2} – užsakytų prekių kiekis, x_{N+1} – prekių kiekis iš gamintojo.
2. Išėjimo signalų aibė $Y=\{ y_1, y_2, ..., y_N, y_{N+1}\}$, kur $y_1=(y_{i1}, y_{i2})$ ir $y_{11}=k$, y_{12} – užsakytų prekių kiekis.
3. Išorinių įvykių aibė $E'=\{ e'_1, e'_2, ..., e'_N, e'_{N+1}\}$; kur e'i, i=1,N+1 – atėjo signalas x_1.
4. Vidinių įvykių aibė $E''=\{ e''_1, e''_2, ..., e''_N, e''_{N+1}\}$; kur e"i, i=1,N – laikas, kai pristatytos prekės į i-tąjį prekybos centrą, e''_{N+1} – prekių pristatymo paraiškų apdorojimo pabaigos laikas.
5. Valdymo sekos: $e''_i \rightarrow \{\xi_{ij}\}$, i=1,N; j=1,∞, $e''_{N+1} \rightarrow \{\eta_j\}$, j=1,∞, kur ξ_{ij} – laiko tarpas per kurį prekės pristatomos į i-taji centrą, η_j – vienos paraiškos aptarnavimo trukmė.
6. Parametrai: Δs – užsakytų prekių kiekis, s_0 – pradžia.
7. Diskrečioji agregato būsenos dedamoji $\nu(t_m)=\{s(t_m), d_1(t_m), ...d_N(t_m), \chi(t_m), q(t_m), Q(t_m), s_0(t_m)\}$; čia $s(t_m)$ – prekių atsargos sandėlyje, $d_i(t_m)$ – pristatytų prekių kiekis į i-tąjį prekybos centrą, $s_0(t_m)$ – pradinė vertė.

$$\chi(t_m)=\begin{cases}1, & \text{prekių užsakymas yra apdorotas}\\ 0, & \text{kitu atveju}\end{cases}$$

   $q(t_m)$ – prekių kiekis paraiškų eilėje.
8. Tolydžioji agregato būsenos dedamoji $z_v(t_m)=\{w(e''_1, t_m), ..., w(e''_N, t_m), w(e''_{N+1}, t_m)\}$, kur $w(e''_i, t_m)$ – i-tosios paraiškos aptarnavimo pabaigos momentas.
9. Agregato pradinė būsena: $s(t_0)\geq s_0$, $Q(t_m)=Ø$, $s(t_0)=s_0$.
10. Perėjimo ir išėjimo operatoriai:

    $H(e'_i)$: i=1,N /Atėjo signalas x_1/

$$s(t_{m+1})=\begin{cases}s(t_m)-x_{i2}, & \text{jei } s(t_m) > x_{i2} \wedge w(e''_{N+1}, t_m)=\infty,\\ s(t_m), & \text{kitu atveju}\end{cases}$$

$$Q(t_{m+1})=\begin{cases}Q(t_m), & \text{jei } s(t_m) > x_{i2} \wedge w(e''_{N+1}, t_m)=\infty,\\ ENQ(Q(t_m),(x_{i1},x_{i2})), & \text{kitu atveju}\end{cases}$$

$$q(t_{m+1})=\begin{cases}x_{i1}, & \text{jei } s(t_m) > x_{i2} \wedge w(e''_{N+1}, t_m)=\infty,\\ q(t_m), & \text{kitu atveju}\end{cases}$$

    $G(e'_1)$:

y_{N+1}= Δs, jei $s(t_m) \geq s_0 > s(t_{m+1})$.

H(e'$_{N+1}$): /Atėjo signalas x_{N+1}/

$$s(t_{m+1}) = \begin{cases} s(t_m) + x_{N+1} - n_2, & \text{jei } Q(t_m) > 0 \wedge w(e''_{N+1}, t_m) = \infty \wedge \exists k > 0 \\ s(t_m) + x_{N+1}, & \text{kitu atveju} \end{cases}$$

$$Q(t_{m+1}) = \begin{cases} DEQ(Q(t_m), k), & \text{jei } Q(t_m) > 0 \wedge w(e''_{N+1}, t_m) = \infty \wedge \exists k > 0 \\ Q(t_m) & \text{kitu atveju} \end{cases}$$

$$\chi(t_{m+1}) = \begin{cases} n_1, & \text{jei } Q(t_m) > 0 \wedge w(e''_{N+1}, t_m) = \infty \wedge \exists k > 0 \\ \chi(t_m), & \text{kitu atveju} \end{cases}$$

$$q(t_{m+1}) = \begin{cases} n_2, & \text{jei } Q(t_m) > 0 \wedge w(e''_{N+1}, t_m) = \infty \wedge \exists k > 0 \\ q(t_m), & \text{kitu atveju} \end{cases}$$

$$w(e''_{N+1}, t_{m+1}) = \begin{cases} t_m + \eta_m, & \text{jei } Q(t_m) > 0 \wedge w(e''_{N+1}, t_m) = \infty \wedge \exists k > 0 \\ w(e''_{N+1}, t_m), & \text{kitu atveju} \end{cases}$$

$$k = \min_{1 \leq j \leq \#Q(t_m)} \{ j \mid Q_j(t_m) = (n_1, n_2), n_2 \leq s(t_m) + x_{N+1} \}$$

G(e'$_{N+1}$):

y_{N+1}= Δs, jei $Q(t_m) > 0 \wedge w(e''_{N+1}, t_m) = \infty \wedge \exists k > 0 \wedge s(t_m) + x_{N+1} - n_2 < s_0$

H(e''$_i$):

i=1,N

$d_i(t_m)$=0.

G(e''$_i$):

$y_i = d_i(t_m)$.

H(e''$_{N+!}$):

$d_{\chi(tm)}(t_{m+1}) = q(t_m)$,

$w(e''_{\chi(tm)}, t_{m+1}) = t_m + \xi_{\chi(tm),j}$,

$$k = \min_{1 \leq j \leq \#Q(t_m)} \{ j \mid Q_j(t_m) = (n_1, n_2), n_2 \leq s(t_m) \}$$

$(n_1, n_2) = Q_k(t_m)$,

$$\chi(t_{m+1}) = \begin{cases} n_1, & \text{jei } \exists k > 0 \\ 0, & \text{kitu atveju} \end{cases}$$

$$q(t_{m+1}) = \begin{cases} n_2, & \text{jei } \exists k > 0 \\ 0, & \text{kitu atveju} \end{cases}$$

$$w(e''_{N+1}, t_{m+1}) = \begin{cases} t_m + \eta_m, & \text{jei } \exists k > 0 \\ w(e''_{N+1}, t_m), & \text{kitu atveju} \end{cases}$$

$$Q(t_{m+1}) = \begin{cases} ENQ(Q(t_m), k), & \text{jei } \exists k > 0 \\ Q(t_m), & \text{kitu atveju} \end{cases}$$

$$s(t_{m+1}) = \begin{cases} s(t_m) - n_2, & \text{jei } \exists k > 0 \\ s(t_m), & \text{kitu atveju} \end{cases}$$

G(e''$_{N+1}$):

$y_{N+1} = \Delta s$, jei $s(t_m) \geq s_0 > s(t_{m+1})$.

6. Komentarai apie imitacinių modelių programinio kodo realizaciją ir integravimą į verslo valdymo sistemas

Sistemos realizacijai būtų galima pasirinkti Microsoft Visual Studio 2003 įrankį. Šis įrankis „skirtas kitos kartos interneto programinės įrangos kūrimui, greitam ir efektyviam sudėtingų aplikacijų kūrimui ir pritaikymui bet kuriai aparatūrinei bazei“ [12].

Ištirta metodika apie imitacinio modelio panaudojimą projektuojant ir diegiant logistikos sistemas apima tokius projektavimo etapus:

- Verslo procesų formalaus modelio sudarymas, naudojant PLA ir DEVS specifikacijas.
- Verslo procesų PLA formalios specifikacijos transformacija į programinį kodą.
- Imitacinio modelio programinio kodo integravimas į verslo sistemas.

Verslo proceso PLA modelio aprašymas naudojant XML

Agregatinės sistemos modelio specifikacijos tarpiniam saugojimui modelių transformacijos procese naudojamas XML dokumento formatas. Failo struktūra griežtai apibrėžta, todėl tai leidžia tuos pačius duomenis panaudoti ir kitose posistemėse (skaitmeninio modeliavimo, testų generavimo, validavimo). Imitacinio modelio XML formato saugojimui naudosime Kauno technologijos universitete verslo informatikos katedroje sukurtą „AGDraw“.

XML failo struktūra atrodo taip:

```
<?xml version=”1.0“ encoding=“UTF-8“?>
<agregate>
      <name>Stotis</name>
      <discretevariables>
            <discrete>
                  <name>n</name>
                  <value>0.2</value>
            </discrete>
            ...
       </discretevariables>
       <continousvariable>
            <continous>
                  <name>e_1</name>
                  <value>3.0</value>
```

```
                        <H>...</H>
                        <G>...</G>
                    </continous>
                    ...
                </continousvariable>
                <outputs>
                    <output>y1</output>
                    ...
                </outputs>
                <inputs>
                    <input>
                        <name>x1</name>
                        <G>...</G>
                        <H>...</H>
                    </input>
                    ...
                </inputs>
            </agregate>
```

Modelio programinio kodo karkasas

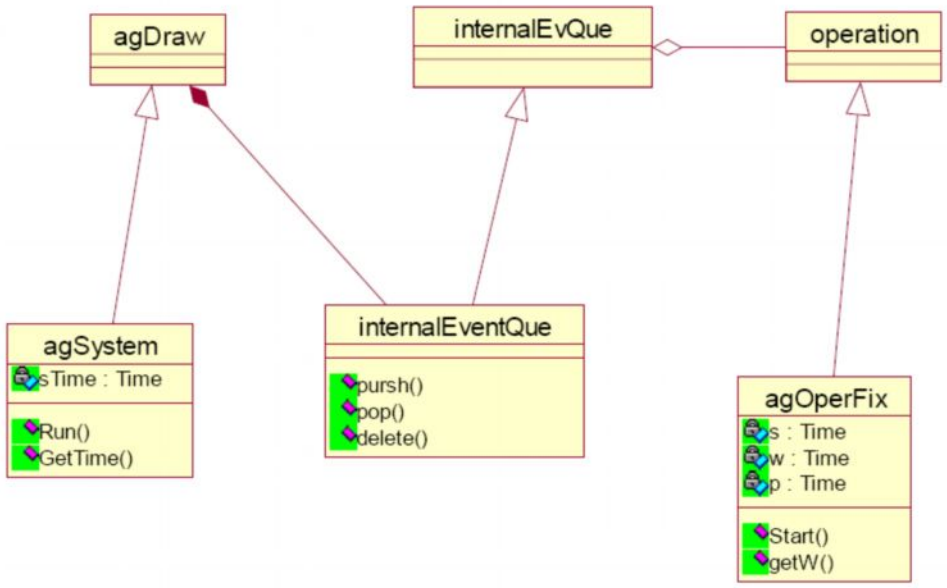
Egzistuoja priemonės kurios leidžia automatiškai generuoti agregatinio imitacinio modelio programinį kodą. Pavyzdžiui, CodeDOM modelis realizuoja transformaciją iš modelio specifikacijos XML formatu į Microsoft.NET C# ar kitas kalbas.

Automatiškai sugeneruoto modelio programinio kodo karkasas (Java, C# ar kitomis kalbomis) apjungiamas su atitinkama objektiškai orientuota imitacinio modeliavimo bibliotekos versija, realizuojančią imitacinį modeliavimo algoritmą PLA formaliems aprašymams.

Kiekvienas modelio agregatas yra traktuojamas kaip atskiras objektas, todėl jis atitinka atskirą klasę objektinėje kalboje, agregatų sistema taip pat sudaro atskirą klasę.

Kadangi agregatai yra traktuojami kaip objektai, o sujungimai atstoja ryšius tarp objektų, tai visa agregatinė specifikacija objektiniu požiūriu veikia kaip objektų visuma. Per sujungimus objektai keičiasi vienas su kitais pranešimais, arba jei reikia perduoda reikalingą informaciją.

Imitacinio modeliavimo klasės ir jų galimi ryšiai pavaizduoti klasių diagramoje (16 pav.)



16 pav. Imitacinio modeliavimo posistemės klasių diagrama

Imitacinio modelio integravimas į verslo valdymo sistemą

Kelių sistemų integravimui į bendrą sistemą reikalinga infrastruktūra, užtikrinanti duomenų mainus tarp integruojamų sistemų. Bet paprastai, šios infrastruktūros paskirtis – ne tik duomenų perdavimas. Tokiu atveju virš esamų taikomųjų programų galima pridėti papildomą funkcionalumo sluoksnį. Šio sluoksnio paskirtis – automatizuoti verslo procesus integruotoje sistemoje užtikrinant vieningą būdą prieiti prie informacijos išbarstytos skirtingose sistemose.

Duomenų integravimas

Duomenų ir informacijos manipuliavimui ir saugojimui naudojamas duomenų bazių klientas. Duomenų bazių kliento technologijos teikia vienodą sąsają naudojant skirtingas duomenų bazes (SQL Server, Oracle, Access). Duomenų bazių klientų kūrimo technologijos apima ODBC (Open DataBase Connectivity), OLE DB (Object-Linking and Embedding DataBase), ADO (ActiveX Data Objects) ir ADO.NET.

Funkcinis integravimas

Naudojant elektronines paslaugas kaip bendrą sąsają su vidinėmis ir su išorinėmis įmonės programomis, išvengiama būtinybė kurti atskiras sąsajas tarp skirtingų sistemų.

Žiniatinklio paslaugų architektūros realizacijai Microsoft siūlo naudoti nemokamus ir

laisvai platinamus programinių sistemų standartinių modulių kodų šablonus ir jų realizacijas.

IŠVADOS

- Išanalizavus atskiras verslo valdymo sistemas, buvo nustatyta, jog jos neapima visų logistikos procesų, kurie svarbūs priimant sprendimus verslo valdymo sistemose.
- Nustatyta, jog yra paprasčiau atskirai integruoti imitacinius modelius į informacines sistemas ir vėliau jas apjungti, naudojantis paskirstytų sistemų architektūra.
- Išanalizavus literatūrą ir sukūrus formalius gamybos ir logistikos modelių aprašymus, nustatyta, kad formalūs metodai DEVS ir PLA tinka verslo procesų imitacinių modeliu specifikavimui.
- Išanalizuotas imitacinio modelio integravimo į informacinę sistemą sprendimo būdas, pasiūlytas verslo informatikos katedroje [16]. Nustatyta, kad jis tinka verslo procesų integracijai į bendrą logistikos sistemą.
- Šiame darbe išanalizuoti keli imitacinių verslo modelių integravimo į tiekimo grandinės valdymo sistemą būdai paskirstytose informacinėse sistemose ir pasiūlyta naudoti žiniatinklio paslaugas, kurios yra nepriklausomos nuo platformos ir kurias naudojant sistemos yra lengvai išplečiamos.

LITERATŪROS SĄRAŠAS

1. ALTER, S. *Information Systems: a management perpective*. Addison – Wesley publishers Inc.1999. p. 122.
2. BOOTH, B., COX, M. *Model Builder 2:* Tutorial Guide. The Modus Project. Harpenden, Herts, 1997, p. 53.
3. DALAL, M. ; GROEL, B.; PRIEDITIS, A. *Intelligent Decision Making Using Real-Time Simulations of Continuous Processes*, LookAhead Decisions Inc.
4. DENISOVAS, V. *Mokomasis kompiuterinis modeliavimas. Modeliavimo programa Model Buider*. Klaipėda: Klaipėdos universiteto leidykla. 2002, p. 85.
5. GOTTSCHALK, K.; GRAHAM, S.; KREGER, H.; SNELL, J. *Introduction to Web services architecture,* IBM SYSTEMS JOURNAL, VOL 41, NO 2, 2002
6. KOUNEV S. *Performance Prediction, Sizing and Capacity Planning for Distributed E-Commerce Applications*. Information Technology Transfer Office, Germany. 2001.
7. MEDEIROS, D.J.; WATSON, E.F.; CARSON, J.S.; MANIVANNAN, M.S.; EDS. *A heterogeneous simulation framework based on the devs bus and the high level architecture:* Proceedings of the 1998 Winter Simulation Conference. Department of Electrical Engineering Korea Advanced Institute of Science and Technology, KOREA
8. NAFTHAL, M.; STOCKTON, D.; LAYDEN, J. *Reducing time delays in supply chains using advanced planning and scheduling.*
9. PILKAUSKAS, V.; PRANEVIČIUS, H.; EDC. *Applications of Simulation and IT Solutions in the Baltic Port Areas of the Associated Candidate Countries*, JUMI Ltd., Ryga, 2003June, ISBN 9984-30-057-9, 146-264.
10. PRANEVIČIUS H.; PILKAUSKAS V.; CHMIELIAUSKAS A. *Aggregate approach for specification and analysis of computer network protocols.* Kaunas: Technologija, 1994, 152 p.
11. PRANEVIČIUS H.; PILKAUSKAS, V.; MAKACKAS, G.; KAMINSKIENĖ, R. *The use of simulation models for real time decision support in business management systems*, KTU, 2006
12. Product Overview for Visual Studio .NET 2003 [interaktyvus]. Prieiga per internetą: *http://msdn.microsoft.com/vstudio/productinfo/overview/default.aspx*.
13. SIMANAUSKAS, L. *Informacinės sistemos*. Vilnius: VU leidykla. 2000. p. 50 – 56.

14. TANENBAUM A., STEEN M. *Distributed Systems: Principles and Paradigms*. Prentice Hall. 2002, p. 21-32, ISBN 0130888931.
15. ZEIGLER, B.P. *Object-Oriented Simulation with Hierarchical, Modular Models Intelligent Agents and Endomorphic Systems Revised to include source code for DEVS-C++*. Department of Electrical and Computer Engineering University of Arizona Tucson, 33-59psl.
16. Aukštųjų technologijų programa, Informacinės technologijos. *Imitacinių modelių panaudojimas sprendimų priėmimui verslo sistemose realiame laike*. Programos vadovas H.Paranevičius, Kaunas 2007.

TERMINŲ ŽODYNĖLIS

DEVS – diskrečių įvykių sistemos specifikacijos.

PLA - atkarpomis tiesinis agregatas (Piece-Linear Agregate). Vienas iš sistemos formalaus aprašymo metodų, besiremiantis atkarpomis tiesiniais procesais.

MAS - masinio aptarnavimo sistema.

AgDraw – „Agregatų Braižyklė“ programa, skirta sistemų imitaciniams modeliams sudaryti.

XML – duomenų aprašymo standartas (eXtensible Markup Language).

IS - informacinės sistemos.

DB – duomenų bazė.

HLA - aukšto lygio architektūra.

Web servines - žiniatinklio paslaugas.

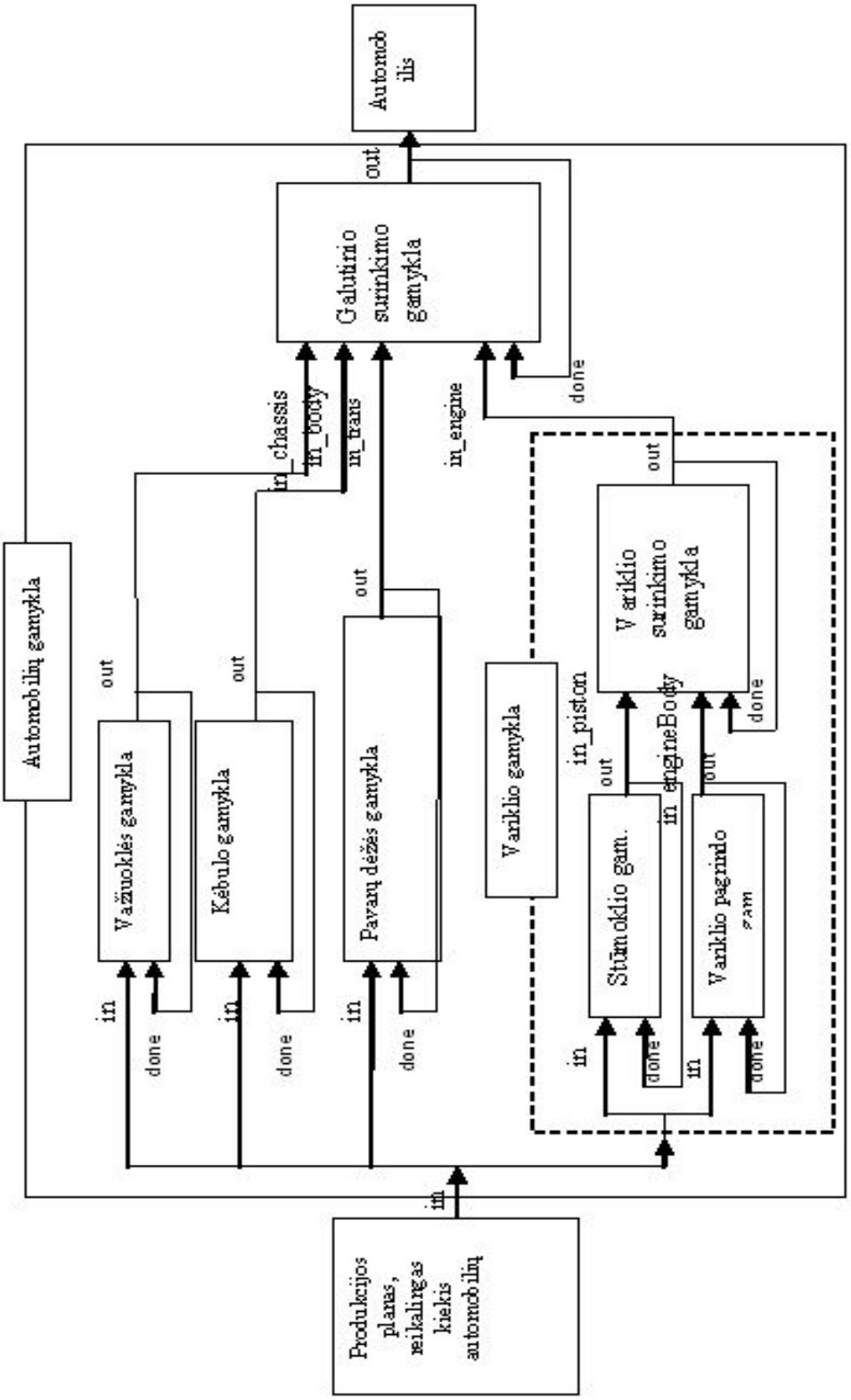
SOAP - Single Object Access Protocol.

WSDL - Web Service Definition Language.

UDDI - Universal Description Discovery Integration.

PRIEDAI

1 priedas. Automobilių gamyklos modelis



2 priedas. Paskelbtos publikacijos

“Integration of business process simulation model to business management systems” buvo pristatytas vykusioje doktorantų vasaros mokykloje „Formalūs sistemų analizės metodai informatikoje“, Druskininkuose, gegužės 13-19 d 2007:

Patvirtinu, kad straipsnis „Integration of business process simulation model to business management systems“ buvo priimtas į doktorantų vasaros mokyklos, kurios tema „Formalūs sistemų analizės metodai informatikoje“, leidinį.

Mokyklos vadovas prof. H. Pranevičius

INTEGRATION OF BUSINESS PROCESS SIMULATION MODEL TO BUSINESS MANAGEMENT SYSTEMS

Vilma Liorenšaitytė, supervisor Prof.habil.Dr Henrikas Pranevičius
Kaunas University of Technology, Faculty of Informatics
Studentų str. 50, LT - 51368 Kaunas

Abstract. A concept of the use of simulation models in business management systems in real time. At present, a large number of modeling and simulation techniques and tools have been developed to deal with complex business systems. In this paper, we concentrate on scenario illustrating, how simulation models can be integrated in to business management system. Different infrastructure forms are possible, because services may be implemented on single machine or distributed throughout several companies' networks. The concept of networked systems interoperability will be introduced.

1. Introduction

It's hard to imagine today's enterprise without Internet site and without information system within enterprise, which supports its processes. Database and web resources in enterprise allows quickly to access information collected in the past, but does not give information about business processes evolution in the future. Traditional methods lead to longer time delays in the communication process. To gain control over enterprise system behavior, new collaboration and integration approaches have to be developed to address time delays in the communication process.

Simulation based Real-time Decision-Making (RTDM) is a dynamic goal-directed decision making processes, useful for systems that continuously make decisions in real-time [1]. RTDM implies that all parts of the enterprise (complex system) can respond to events as soon as they become known to any one part of the enterprise. [2]

RTDM advantages:

- It represents a very common decision problem in realistic manufacturing systems.
- Its simplicity allows us to clearly understand the effects of various strategies. The simplicity also provided several experimental conveniences regarding debugging, simulation duration, etc.
- It can be made increasingly more complex in a systematic manner, say, by adding more layers of machines.

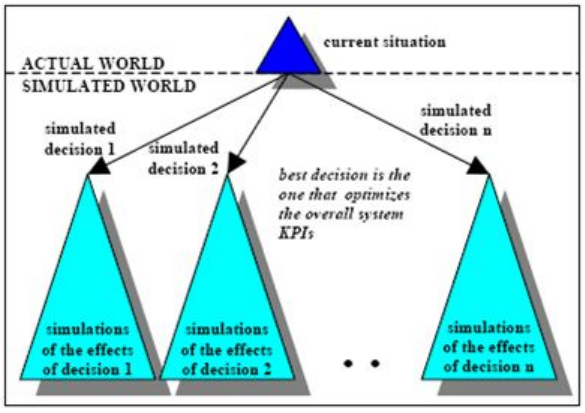


Figure 1. Simulation based real time decision making

For systems formalization we can use a method of Piece-Linear Aggregates (PLA) that belongs to the class of time automata model. PLA method is close to Discrete Event Simulation (DEVS) formalism, which is used to create wide purpose simulation models. In this paper, a PLA will be used.

A technology, which could use simulation modeling in projecting information systems, must be created. Systems must be able to adapt to volatile environment, e.g. start communicating with new information system of new enterprises partner, accept data in different format from new source etc. System should be able to change its behavior and structure.

Creating this technology some problems must be solved:

- Involvement of simulation models in to information systems?

- Simulation model transformation in to program code.
- How to combine different simulation models?

2. Simulation models integration in to information systems

Enterprise Information System (EIS) provides a single information system for organization-wide coordination and integration of key business processes. To use simulation model for real time decision making systems, it is important to integrate this model to IS.

Simulation model integration in to EIS architecture (see Figure 2) [3]:

- Simulation model interface with IS programming components:
  - Simulation experiment model control from information system (setting model parameters, controlling model execution);
  - Represents simulation modeling results for user.
- Simulation model interface with IS database:
  - Provides simulation model with real data from database.

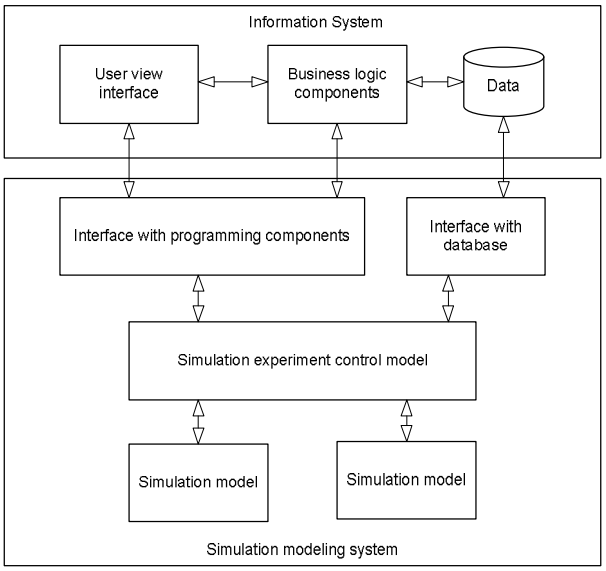


Figure 2. Simulation model integration in to EIS architecture

During design, the model is transformed into a component that represents the problem domains [4]:

- Present state and previous history;
- Forecasting of future state.

Information about business process history and present can be received from ISDB. Forecasting can be evaluated by means of simulation. That's why simulation model must interact with IS by using data and function integrations.

3. Simulation model integration with database

Database client will be used for communication with database. Database client technologies affords the same interface with different databases (SQL Server, Oracle, Access). Database client technologies includes ODBC (Open DataBase Connectivity), OLE DB (Object-Linking and Embedding DataBase), ADO (ActiveX Data Objects) ir ADO.NET.

4. Functional simulation model integration

Programming code for simulation model will be made in this order (Figure 3) [3].

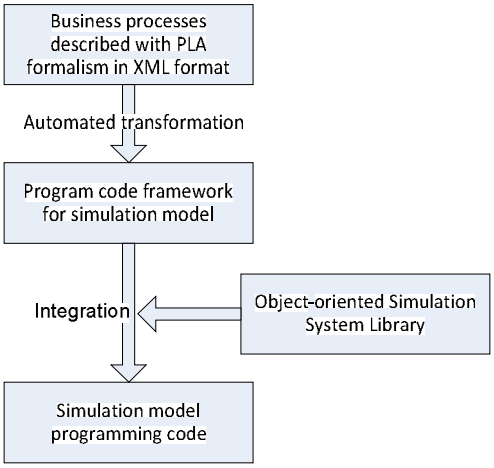


Figure 3. Functional simulation model integration

1. Aggregate system model specification will be saved as XML document format, that's why for meta-model XML document schema XSD format is used. For graphical view „Altova XMLSpy" tool can be used and simulation model saving in XML format „AGDraw" tool, created in Kaunas University of Technology, will be used.

2. Automatically to generate simulation model programming code some tools are created. For example, CodeDOM model, that implements transformation from XML format to Microsoft.NET C# or others.

3. Automatically generated programming code framework of simulation model is implemented with object-oriented simulation modeling library.

5. Interoperability between simulation models

Systems must be able to adapt to volatile environment, e.g. start communicating with new information system of new enterprises partner, accept data in different format from new source etc. System should be able to change its behavior and structure [5] – i.e. adapt. Developing methods and technologies to support interoperability among different simulations must be created.

Systems interoperability is achieved when data from one database can be effectively shared with another by:

- interrogating different databases via single entry points (one -stop shopping)
- retrieving and exchanging data from different sources
- assembling and translating data into different formats
- adding technologies that remain unobtrusive to users

Using web services as interface between inside and outside enterprise information systems, it is not necessary to create different interfaces between different systems. When deciding on network structure, basically two options exist (see Figure 4) [2]:

- Distributed simulations
- Parallel simulations, which is executed on shared memory multiprocessors or multicomputer. It reduces the time delays directly through faster communication.

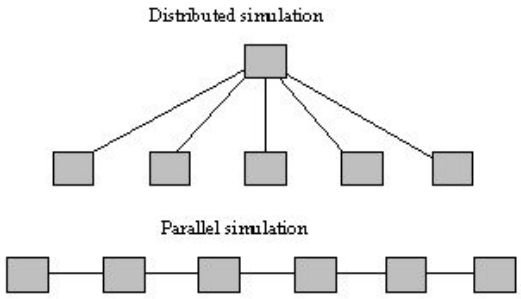


Figure. 4. Distributed vs. Parallel simulations

Distributed simulation method has proved to be the more effective method. Because it is more concerned with the issue of interoperability itself, i.e., bridging gaps between different hardware and software platforms.

HLA (The High Level Architecture) is accepted for broadcast communication method. In order to facilitate

interoperability and reusability, HLA differentiates between the simulation functionality provided by the members of the distributed simulation and a set of basic services for data exchange, communication and synchronization.

The other possibility is to use messaging system Web Service [6]. It is a software system designed to support interoperable Machine-to-Machine interaction over a network through standardized XML messaging. Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. Web services built on existing and emerging standards such as Hyper Text Transfer Protocol (HTTP), Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and the Universal Description, Discovery, and Integration (UDDI) project.

- SOAP (Single Object Access Protocol) – An XML-based, extensible message envelope format, with "bindings" to underlying protocols. The primary protocols are HTTP and HTTPS, although bindings for others, including SMTP and XMPP, have been written.
- WSDL (Web Service Definition Language) – An XML format that allows service interfaces to be described, along with the details of their bindings to specific protocols. Typically used to generate server and client code, and for configuration.
- UDDI (Universal Description Discovery Integration) – A protocol for publishing and discovering metadata about Web services, to enable applications to find Web services, either at design time or runtime.

Distributed simulation has a number of advantages [2]:

- If one system suffers a breakdown then other systems can still work.
- The whole framework is easily configurable and extendable, which is particularly important in dealing with dynamic complex system network.
- There is no extensive communication between sites.
- Enables a smooth transition between different hardware or software platforms

Simulation experiment control model can be reached through customer interface, or from web service.

6. Conclusions

For a real-world business scenario to present the current status of its processes is not enough. To perform forecasting, a simulation model has to be created and integrated in the information system. These systems are generally large and there control architecture is distributed among many computers possibly in different geographical areas. At present, more and more Web services will be used for systems interoperability.

Effective application integration can provide your organization with the following important business benefits:

- Allowing applications to be introduced into the organization more efficiently and at a lower cost
- Allowing you to modify business processes as required by the organization
- Allowing you to add automated steps into business processes that previously required manual intervention

References

[1] **M. Dalal**, **B. Groel**, **A. Prieditis**, Intelligent Decision Making Using Real-Time Simulations of Continuous Processes, *LookAhead Decisions Inc.*

[2] **M. Nafthal**, **D. Stockton**, **J. Layden**. Reducing time delays in supply chains using advanced planning and scheduling.

[3] **H. Pranevičius**, **V. Pilkauskas**, **G. Makackas**, **R. Kaminskienė**, The use of simulation models for real time decision support in business management systems, *KTU, 2006*

[4] **V. Pilkauskas**, **H. Pranevičius** and others, Applications of Simulation and IT Solutions in the Baltic Port Areas of the Associated Candidate Countries, *JUMI Ltd., Ryga, 2003June, ISBN 9984-30-057-9, 146-264.*

[5] **A.M. Uhrmacher**. A system theoretic approach to constructing test beds for multi-agent systems. Discrete event modelng and simulation technologies: a tapestry of systems and AI-based theories and methodologies, *Springer-Verlag New York, Inc., 2001, 315-339.*

[6] **K. Gottschalk**, **S. Graham**, **H. Kreger**, **J. Snell**, Introduction to Web services architecture, *IBM SYSTEMS JOURNAL, VOL 41, NO 2, 2002*

[7] **D. Chandra**, **A. Liu** and others, Guidelines for Application Integration, Microsoft Patterns & Practices, *December 2003.*