KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Stasys Razmus

Būsto inžinerinių mazgų modeliavimo metodika Ptolemy II sistemoje

Magistro darbas

Vadovas
prof. dr. E. Kazanavičius

KAUNAS 2006

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

ĮVADAS

Sparčiai vystantis kompiuterinei technikai, jos panaudojimo sfera apima vis daugiau gyvenimo sričių. Informaciniai įrenginiai montuojami daugumoje buitinės technikos prietaisų, juos galima valdyti internetu. Keletą dešimtmečių technologai kalba apie „modernų būstą“. Vizija dažniausiai nusakoma kaip būstas, užpildytas technine įranga, kuri atlieka gyventojo įsakymus ir nuobodžius darbus. Iš tiesų „modernus būstas“ dar neatitinka tokios vizijos. Plačiajuosčių duomenų magistralių, greitesnių, mažesnių ir netgi protingesnių skaičiavimo įrenginių prieinamumas, sujungtų laidais ir bevielių tinklų technologijų panaudojimas leidžia priartinti viziją prie realybės. Šios technologijos plinta po būsto infrastruktūrą, sudarydamos tinklus.

Kuriant naujus būsto inžinerinių mazgų įrenginius nėra vienodo standartizuoto metodo kaip aprašyti veikimo sąsają su kitais jau egzistuojančiais įrenginiais. Tokiais atvejais yra vertinamas tik konkretus inžinerinis mazgas, neatsižvelgiant į visumą. Todėl gali iškilti suderinamumo problemų. Norint išvengti minėtų sunkumų būtina turėti būsto inžinerinių mazgų modeliavimo metodiką, leidžiančią supaprastinti projektuotojų darbą taip pat įvertinti inžinerinių mazgų tarpusavio sąveiką.

Ptolemy II yra įrankis, kuriuo galima modeliuoti sukurtus projektus. Ptolemy II yra atviro kodo projektas. Pagrindinė priežastis jį pasirinkti ta, kad galima naudoti tinklo integraciją, kodo migravimą, gijų panaudojimą. Visi šie elementai sujungti Java sąsaja. Ptolemy II skaičiavimo modeliai palaiko projektavimą naudojant aktorius (angl. *actor-oriented design*). Tai skiriasi nuo objektiškai orientuoto projektavimo, nes naudojat aktorius galima įvertinti lygiagretumą ir komunikaciją tarp komponentų. Komponentai, vadinami aktoriais, vykdo ir bendrauja su kitais modelyje naudojamais aktoriais.

Aktorius – tai objektas, atliekantis parametrizuotus veiksmus su įvesties duomenimis tam, kad gauti išeities duomenis. Priklausomai nuo aktoriaus sandaros jis gali turėti arba neturėti būsenų. Įvesties ir išvesties duomenys perduodami per jungtis (angl. *port*). Jungtys ir parametrai sudaro aktoriaus sąsają.

Jungiant aktorius į tam tikras struktūras, galima gauti norimo inžinerinio mazgo funkcionalumą. Panaudojus hierarchinę struktūrą, inžinerinis mazgas išskaidomas į konkrečius uždavinius atliekančius posistemius. Kiekvienas posistemis turi griežtai nusakytą sąsają bei funkcionalumą. Sukurti ir ištestuoti inžineriniai mazgai jungiami į biblioteką.

Norint modeliuoti pasirinktus būsto inžinerinius mazgus, reikia iš sukurtos bibliotekos įterpti norimus mazgus. Modeliavimas vyksta keičiant minėtų mazgų parametrus. Ptolemy II sistema leidžia lengvai keisti mazgo vidinę sandarą ir parametrus. Taip galima greitai įvertinti padarytus pokyčius per gana trumpą laiką.

Magistrinio darbo objektas – būsto inžinerinių mazgų modeliavimas, naudojant Ptolemy II sistemą.

Magistrinio darbo tikslas – sudaryti modeliavimo metodiką, apimančią pagrindinius gyvenamojo namo inžinerinių mazgų procesus ir jų modeliavimą.

Darbe sprendžiami uždaviniai:

- modernaus būsto architektūrinių projektavimo metodų analizė;
- modernaus būsto naudojamų įrenginių analizė;
- namo plane esančių būsto įrenginių transformacijos į Ptolemy II modeliavimo ir kūrimo metodikos sudarymas;
- modelio elementų (aktorių) formalių specifikacijų, kurių pagrindu būtų aprašoma elemento elgsena, sudarymas;
- atlikti gauto modelio tyrimą Ptolemy II sistemoje.

Analizės dalyje analizuojami modernaus būsto architektūrinio projektavimo metodai. Čia pateikiama jau naudojamų modernaus būsto komponentų sprendimai ir aprašoma Ptolemy II sistema.

Teorinėje dalyje pateikiama namo plane esančių būsto įrenginių transformacijos į Ptolemy II sistemą modeliavimo ir kūrimo metodika.

Eksperimentinėje dalyje aprašomas konkretaus posistemio transformacijos rezultatas.

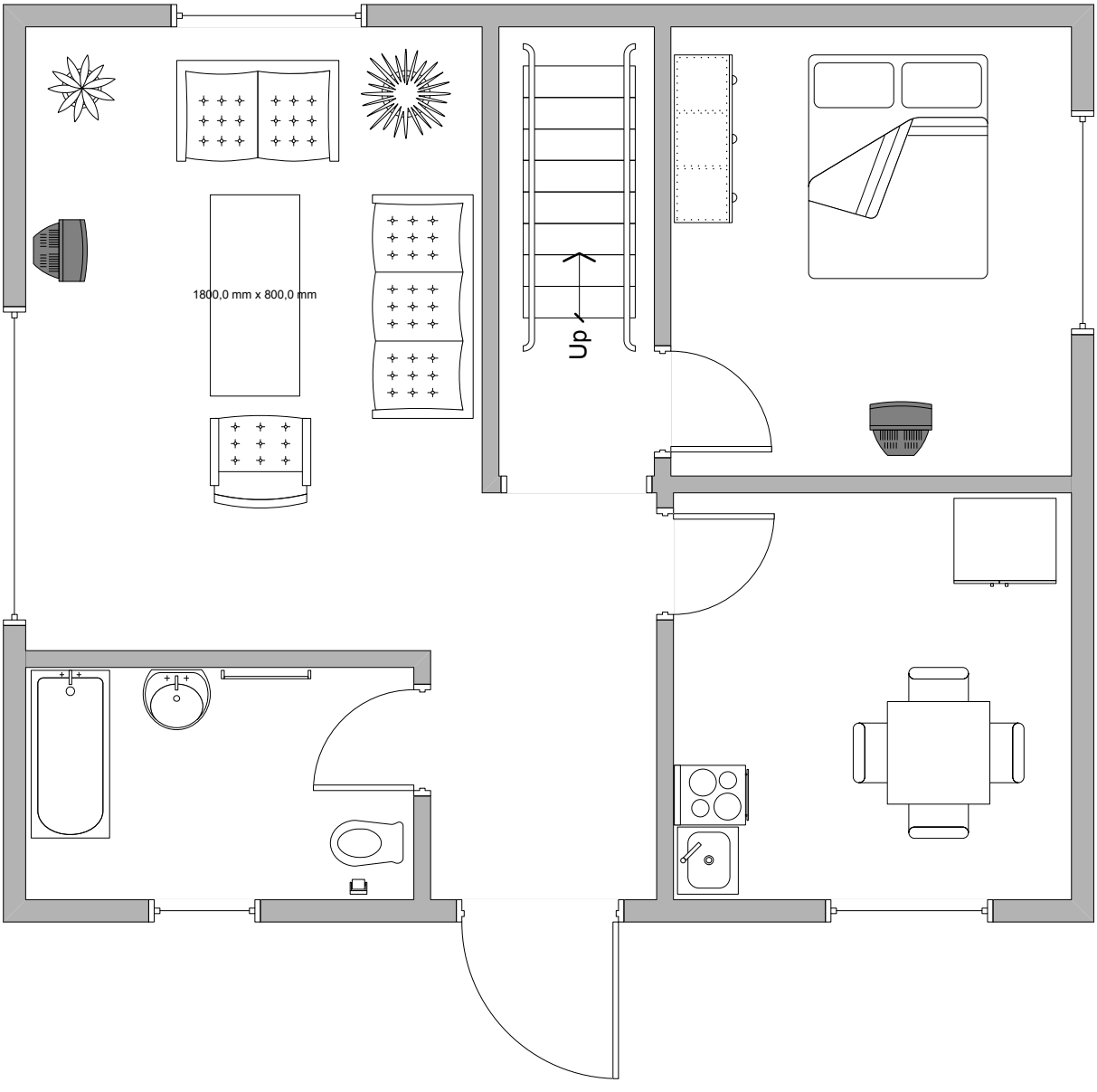
Darbas paruoštas Kauno technologijos universiteto Informatikos fakulteto Kompiuterių katedroje, kurioje vykdomas projektas „Ateities būsto aukštosios technologijos ir įranga“, remiamas Lietuvos valstybinio mokslo ir studijų fondo.

1. BŪSTO INŽINERINIŲ SISTEMŲ ANALIZĖ

1.1. Situacijos analizė

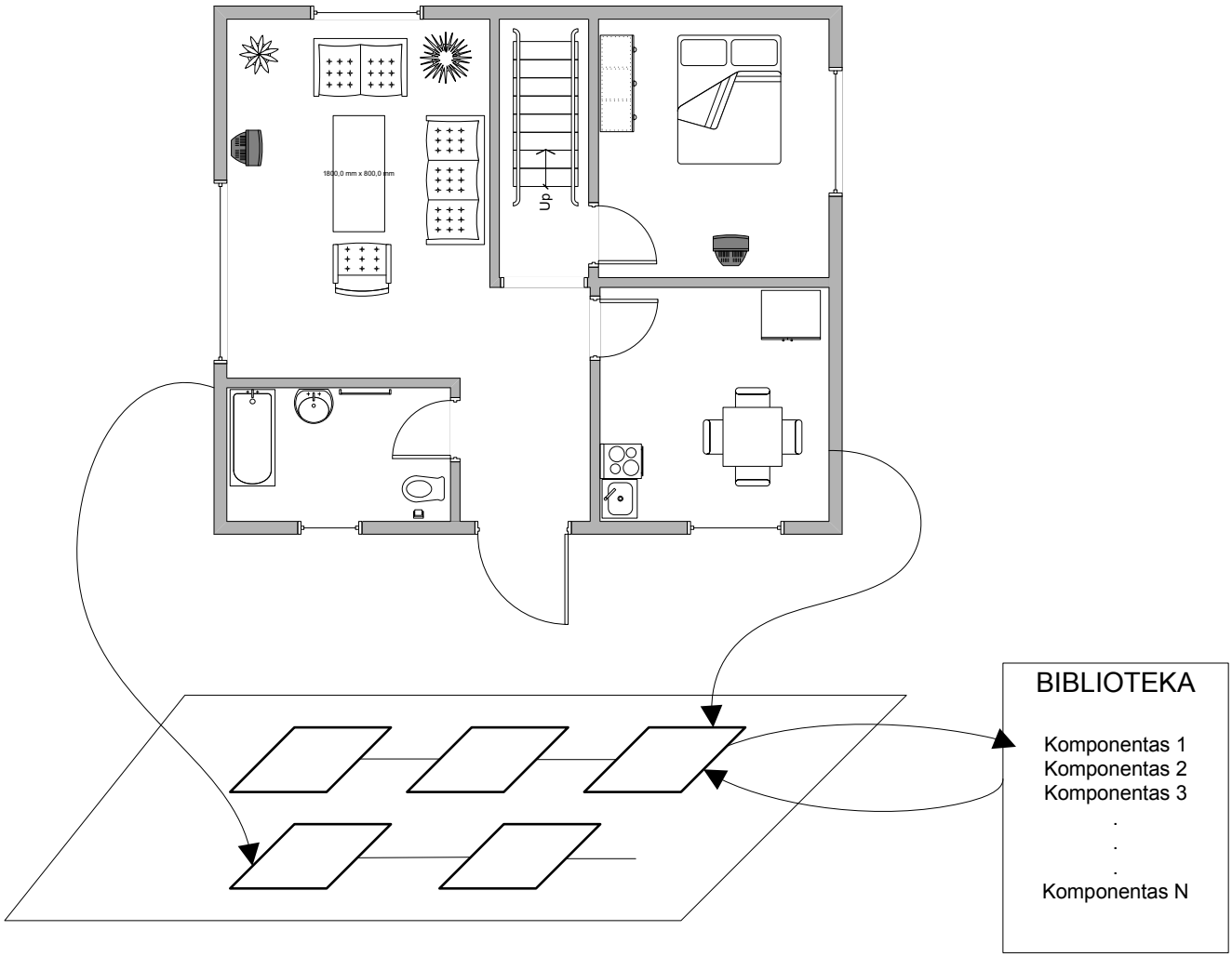
Gyvenamasis namas neįsivaizduojamas be inžinerinių mazgų, tokių kaip vandentiekis, gamtinės dujos, elektros instaliacija, šildymo mazgas, apsaugos sistema. Šiais laikais gyvenamuosiuose būstuose įrengiamas namo kompiuteris, kuris valdo daugumą šių mazgų, surenka duomenis iš apskaitos įrenginių, pagal nustatytą režimą palaiko kambarių temperatūrą.

Projektuotojas, kurdamas gyvenamojo namo planą, gali į jį įtraukti norimą montuoti įrangą (1.1 pav.). Pažymėtina tai, kad daugumą įrenginių įmanoma valdyti X10 protokolo pagalba, t. y. komunikacijai galima naudoti elektros tinklą.



1.1 pav. Namo planas

Tam, kad projektuotojas galėtų įterpti sudedamą įrangą, yra paruošta komponentų biblioteka. Šios bibliotekos komponentai gali būti aprašyti matematinėmis lygtimis arba formaliomis specifikacijomis.



1.2 pav. Namo plano transformavimas į formalų komponentinį modelį

Panaudojus projektavimo sistemą Microsoft Visio sudaromas gyvenamojo namo planas. Jame sudedami visi reikalingi komponentai bei inžineriniai mazgai. Norint modeliuoti inžinerinius mazgus reikalingas modeliavimo paketas Ptolemy II, kurio pagalba atliekami norimi tyrimai. Duomenų perkėlimui iš MS Visio į Ptolemy II naudojamas XML formatas, kuris leidžia hierarchiškai aprašyti modelį.

1.2. Modernaus būsto analizė ir architektūrinio projektavimo metodai

Tikslūs reikalavimai yra sėkmingo produkto vystymo pagrindas (Durrett; Burnell; Priest: 2002: 85). Tokius reikalavimus nustatyti/apibrėžti daugeliu atvejų yra gana sudėtinga, o ypač problematiška naujose ir sparčiai besivystančiose srityse. Modernaus būsto technologijų plėtra yra iššūkis dėl jų dinamiškos projektavimo terpės, susidedančios iš greitai besikeičiančių technologijų su keletu standartų ir vartotojų, turinčių neaiškių galimybių idėjas. Didžiausia problema – vartotojo pasitenkinimo, susijusio su įvairiais vartotojo tipais, preferencijomis ir dinamiška projektavimo terpe, optimizavimas. J. R. Durrett, L. J. Burnell ir J. W. Priest į pirminę vartotojo aplinkos plėtrą siūlo naują požiūrį, paremtą tinkamai įdiegtomis programinėmis įrangomis, valdymo teorijomis ir techninėmis įrangomis. Šie mokslininkai siūlo įvairių naudojimo atvejų ir namo analizės modelių, imitatorių, prototipų ir informacijos koordinavimo procesų technikos sintezę.

Efektyvus produkto plėtojimas yra procesas, susijęs su tam tikrais reikalavimais. Sėkmingo produkto plėtojimo esmė yra vartotojo poreikių žinojimas ir tų poreikių tenkinimas, aprūpinant vartotoją reikalingu produktu, bei lengvas produkto pritaikymas prie ateities reikalavimų: konkurencinga kaina ir tinkamas reikiamas laikas. Tenkinant vartotojo reikalavimus reikia įtraukti punktus, kurių reikalauja pats vartotojas, bei punktus, kurių vartotojas nereikalauja dėl informacijos apie naujas technologijas ar pažangias idėjas stokos. Tokie vartotojo reikalavimai, kartu su technologijomis, kainomis ir prieinamomis alternatyvomis visada evoliucionuoja.

Kai analitikas nustato sistemos reikalavimus, pirminiu rūpesčiu tampa srities eksperto ir sistemos projektuotojo komunikacija. J. R. Durrett, L. J. Burnell ir J. W. Priest sutelkė pastangas į tinkamai įdiegtų programinės įrangos inžinerijos technologijų, valdymo teorijos ir techninės įrangos sintezę, siekdami palengvinti komunikaciją tuo metu, kai nustatyti vartotojo reikalavimai. Mokslininkų tikslas – parodyti, kad:

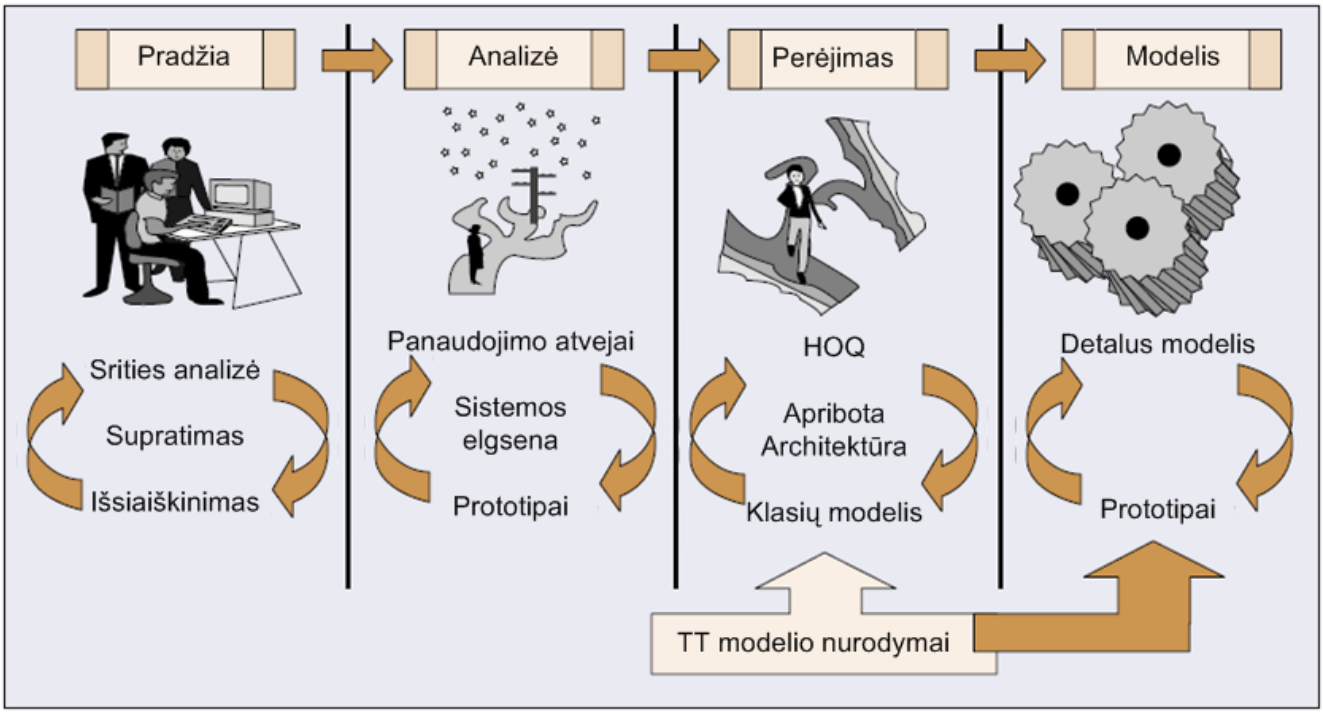
- žmoniškųjų išteklių koordinavimo technika iš informacijos apdorojimo teorijos (angl. *information processing theory IPT*);
- objektiškai orientuota analizė ir projektavimo technika, kaip panaudojimo atvejai ir prototipai;
- techninės įrangos plėtros metodai, kurie susiję su kokybišku namu (angl. *house of quality HOQ*);

padidins bet kokios sistemos projektavimo efektyvumą, o ypač modernaus būsto sistemų.

Numatytas/laukiamas rezultatas – modernaus būsto komponentų, kurie efektyviai ir veiksmingai patenkintų vartotojo poreikius, kūrimas.

Reikalavimų nustatymas yra specifinių poreikių identifikavimas, išmatavimas, įvertinimas, išdėstymas pagal svarbą ir dokumentų rašymas apie naujus produktus, procesus, paslaugas bei jų plėtrą

(Priest; Sanchez: 2001: 104). Toks procesas yra labai brangus, tačiau svarbus viso projekto sėkmei. Dėl to, kad vartotojas nėra susipažinęs su modernaus būsto technologijomis ar jų pajėgumais, modernaus būsto kūrėjams kyla vienas labai svarbus klausimas – potencialių vartotojų norų ir poreikių nustatymas. Daugelis įsivaizduojamų technologijų yra paruoštos ar beveik paruoštos tam, kad būtų įterptos į vidutinį namą. Pavojus kyla dėl to, kad projektuotojai vadovausis savo asmenine patirtimi, kurdami produktus mažai techniškai nusimanančių vartotojų namuose. Modernaus būsto realizavimo sėkmei labai svarbu, kad sprendimas atitiktų reikalavimus ir, kad projektuotojai nepakliūtų į tuos pačius ankstyvųjų informacinių sistemų su trūkumais spąstus. Pavyzdžiui, ar vartotojas tikrai nori termostato su 56 skirtingais nustatymais (Postrel: 1999: 5)? Kokios produkto savybės yra reikalingos vartotojui ir namo statytojui įtikinti, kad jis nupirktų tuos produktus? Be to siūlomi produktai turi veikti kartu su esamais ir ateities produktais, kad suformuotų integruotą namo architektūrą.



Šaltinis: DURRETT, John R.; BURNELL, Lisa, J., PRIEST, John W. A Hybrid analysis and architectural design method for development of smart home components; p 86.

1.3 pav. Modernaus būsto pirminis vystymo procesas

Netgi projektuotojai negali sutarti dėl modernaus būsto galimybių. Modernaus būsto projektuotojai privalo įvertinti vartotojų reikalavimus: vertę, panaudojimą, pritaikymą, privatumą, saugumą, kainą, palaikymas (koreguojamas ir prognozuojamas). Panaudojimo atvejai naudojami komunikacijai tarp projektuotojo ir vartotojo gerinti. Prototipai ir imitavimas yra naudojami sistemos elgsenai pavaizduoti, galutinio projekto patvirtinimui ir pradinės dokumentacijos paruošimui. HOQ yra naudojamas tolimesniam sistemos savybių tobulinimui ir vartotojo prioritetų lyginimui su esamais projektavimo modeliais. Tikimybių teorija (angl. *contingency theory CT*) ir informacijos apdorojimo teorija (*IPT*) padeda suprasti pirminę sistemos struktūrą. Šių metodų naudojimas keturiose modernaus

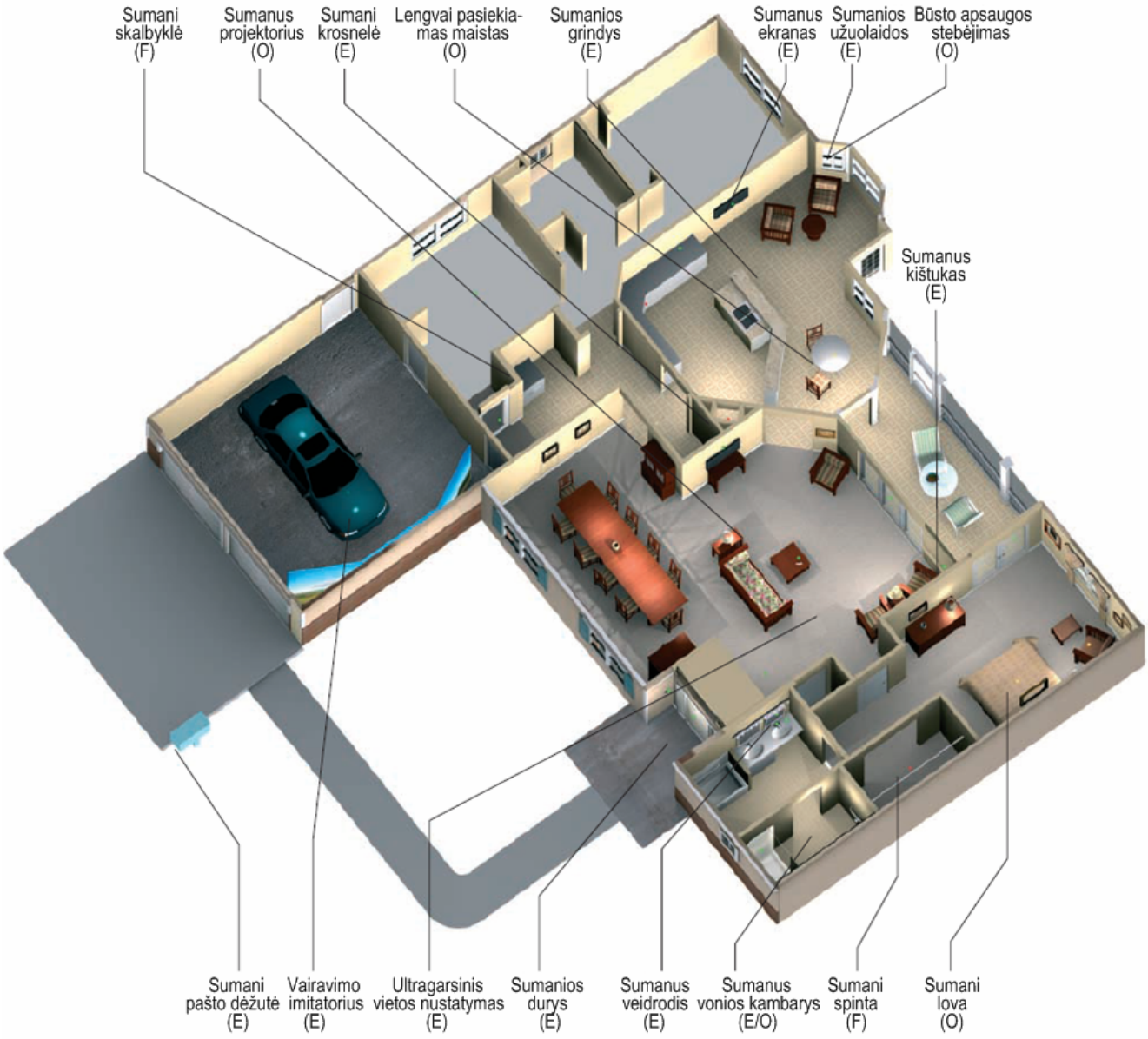
būsto programinės įrangos kūrimo proceso fazėse yra pavaizduotas 1.3 paveiksle. Pradinėje fazėje (Jacobson; Booch; Rumbaugh: 1999: 53) pagrindiniai uždaviniai yra srities ir vartotojo supratimas, srities ir reikalavimų analizė. Analizės fazės tikslas yra sistemos elgsenos apibūdinimas: ką pasiūlyta sistema turi daryti. Objektiškai orientuotoje plėtroje ryšys tarp analizės ir projektavimo nėra taip gerai apibūdintas, kaip struktūrizuotuose metoduose. Panaudojimo atvejų modeliavimas yra naudojamas sistemos veikimui užfiksuoti; klasių modeliavimas apibūdina klases, atributus ir tarpusavio ryšius; dinaminis modeliavimas apibrėžia klasių tarpusavio ryšius. Perėjimas nuo analizės prie projektavimo, arba nuo išorinės elgsenos prie vidinės struktūros, veikia su CT/IPT. Šios idėjos fiksuojamos per HOQ ir klasių modelius (sąveikos diagramos gali būti naudojamos kompleksinėm vidinėm elgsenom). Šis procesas veda į detalesnį projektavimą, kuriame prototipai naudojami taip pat kaip ir analizės fazėje, tačiau skirtingiems tikslams. Klasių metodai yra apibrėžti objektiškai orientuotame projektavime.

1.3. Modernaus būsto technologijos

Tyrėjų grupė, sudaryta iš universiteto ir pramonės atstovų, sukūrė sistemos prototipą, kuris parodo kompiuterinių įrenginių plitimą įvairių sričių aplikacijose (Helal; El-Zabadani;. King; Kaddoura; Janssen: 2005: 50). Tokiuose projektuose didžiausias dėmesys buvo skiriamas pagrindinių sistemų integracijai, t. y. jutiklių, aktyvatorių, kompiuterių ir kitų įrenginių sujungimui. Deja, dauguma pirmųjų kompiuterinių sistemų neturėjo galimybės vystytis. Norint sujungti keletą heterogeninių elementų, reikalingas specialus procesas. Naujo elemento įterpimas reikalauja jo charakteristikų ir veiksenos tyrimo, nustatymų ir integravimo instrukcijų bei varginančių ir daug laiko reikalaujančių testavimų, kurių metu išaiškinami konfliktai ar neapibrėžta veiksena bendroje sistemoje. Terpė yra siaura, todėl ribojamas vystymas. Dėl šio ribojimo Floridos universiteto Mobilių įrenginių laboratorija sukūrė programuojamą plintančią erdvę (angl. *programmable pervasive space*), kurioje sumani erdvė egzistuoja kaip bandymo terpė ir programinės įrangos biblioteka. Servisas ir tinklų sąsajos protokolai automatiškai sujungia sistemos elementus naudojant bendrą programinę įrangą, kuri palaiko serviso apibrėžimus kiekvienam erdvės jutikliui ir aktyvatoriui. Programuotojai surenka servisus į atskiras aplikacijas, kurias trečiosios šalys gali lengvai įdiegti arba praplėsti. Į servisus orientuotų programuojamų erdvių panaudojimas praplečia tradicinį programinį modelį. S. Helal, W.Mann, H. El-Zabadani, J. King, Y. Kaddoura, E. Jansen siūlymas leidžia srities ekspertams kurti ir vystyti efektyvias vartotojų aplikacijas. Minėtų autorių projekto tikslas – sukurti pagalbines aplinkas (angl. *assistive environments*) tokias kaip namas, kurios analizuos save ir gyventojų veiksmus bei vykdys sąryšį tarp fizinio pasaulio ir sistemos valdymo nuotolinio būdu.

1.3.1. Modernaus būsto elementai

Pagrindiniai Gator Tech Smart House projekto elementai pavaizduoti 1.4 paveiksle (Helal; El-Zabadani;. King; Kaddoura; Janssen: 2005: 51).



Šaltinis: HELAL, Sumi; MANN, Williams; EL-ZABADANI, Hicham; KING, Jeffrey; KADDOURA, Youssef; JANSSEN, Erwin. The Gator Tech Smart House: A Programmable Pervasive Space; p. 51.

1.4 pav. Gator Tech Smart House

Projekte jau įgyvendinta (E), toliau vystoma (O), palikta ateičiai (F)

- *Sumani pašto dėžutė*. Pašto dėžutė užfiksuoja gautą korespondenciją ir praneša modernaus būsto gyventojui.
- *Sumanios durys*. Šiose duryse yra įmontuotas radijo dažnių imtuvas (angl. *RFID – radio frequence identification*), leidžiantis gyventojams įeiti į namą, nenaudojant raktų. Šios durys taip pat turi mikrofoną, vaizdo kamerą, skystųjų kristalų ekraną (*LCD*),

automatinio atidarymo funkciją, elektrinį užraktą ir garsiakalbį, kuris padeda gyventojams komunikuoti ir priimti svečius.

- *Vairavimo imitatorius*. Garaže yra vairavimo imitatorius, padedantis įvertinti vairavimo įgūdžius ir surinkti duomenis tyrimui.
- *Sumanios užuolaidos*. Visus langus dengia automatinės užuolaidos, kurios reguliuojamos nuotolinio valdymo aparatu, kontroliuoja aplinkos apšvietimą ar privatumą.
- *Sumani lova*. Namo šeimininko miegamajame esanti lova turi specialų įrengimą, kuris fiksuoja asmens miego būseną.
- *Sumani spinta*. Šeimininko miegamojo spinta ateityje patars kuo rengtis, atsižvelgiant į oro sąlygas.
- *Sumani skalbyklė*. Ateities technologijos, paremtos RFID, praneš namo gyventojams, kada skalbti skalbinius bei kaip juos surūšiuoti.
- *Sumanus veidrodis*. Šeimininko vonios veidrodis parodys svarbius pranešimus ir priminimus, pavyzdžiui, kad reikia tam tikru laiku suvartoti gydytojo paskirtus vaistus. Ši technologija gali būti pritaikyta ir kituose kambariuose.
- *Sumanus vonios kambarys*. Vonios kambaryje yra tualetinio popieriaus daviklis, dušas, kuris reguliuoja vandens temperatūrą, muilo išdavimo įtaisas, kuris paslaugų centrui praneša apie būtiną muilo papildymą. Taip pat kitos technologijos, pritaikytos atsižvelgiant į gyventojo biometrinius duomenis (svorį, temperatūrą).
- *Sumanus ekranas*. Visame name esančių ekranų dėka, informaciją namo gyventojai gali gauti būdami bet kuriame kambaryje.
- *Sumani krosnelė*. Virtuvės mikrobangų krosnelė automatiškai nustato laiką ir temperatūrą, nurodo vartotojams, kaip paruošti bet kokį užšaldytą maistą.
- *Sumanus šaldytuvas*. Modernus šaldytuvas kontroliuos maisto produktų atsargas ir vartojimą, fiksuos maisto produktų vartojimo terminus, sudarys reikalingų produktų sąrašą, patars kaip gaminti maistą iš turimų maisto produktų atsargų.
- *Lengvai pasiekiamas maistas*. Vaizdo ir garso technologijų, įdiegtų valgomajame, dėka namo gyventojams bus lengviau dalintis, pasiekti ir prieiti prie norimo maisto.
- *Ultragarsinis vietos nustatymas*. Davikliai, įtaisyti svetainėje, fiksuos gyventojų judesius, buvimo vietą.
- *Sumanios grindys*. Davikliai, įtaisyti virtuvės ir pramogų centro patalpose, identifikuos ir stebės visų namo gyventojų buvimo vietą. Taip pat yra vystomos technologijos, kurios padėtų identifikuoti gyventojo pargriuvimą ir praneštų apie tai pagalbos tarnyboms.

- *Sumanus telefonas*. Integruojamos tradicinės telefono funkcijos su įvairių svetainės įtaisų, prietaisų nuotoliniu valdymu. Taip pat toks telefonas gali perduoti svarbią informaciją namo gyventojams, kol jų nėra namie.
- *Sumanus kištukas*. Davikliai, įmontuoti prieš svetainės, virtuvės ir miegamojo kištukinius lizdus, aptinka elektros prietaisus arba lempas ir susieja juos su nuotoliniu valdymu.
- *Sumanus termostatas*. Ateityje namo gyventojai galės reguliuoti oro sąlygas ir šildymą, pritaikydami jas prie asmeninių poreikių. Pavyzdžiui, temperatūros pakėlimas žiemą, maudantis duše.
- *Sumanus projektorius*. Kuriamas projektorius, kuris naudoja vietos nustatymo informaciją, gautą ultragarsu ir ant kambario sienos pateikia gyventojui įvairius priminimus.
- *Būsto apsaugos stebėjimas*. Saugumo sistema stebi visus namo langus, duris, taip pat informuoja namo gyventojus apie atidarytus langus ar atidarytas duris.

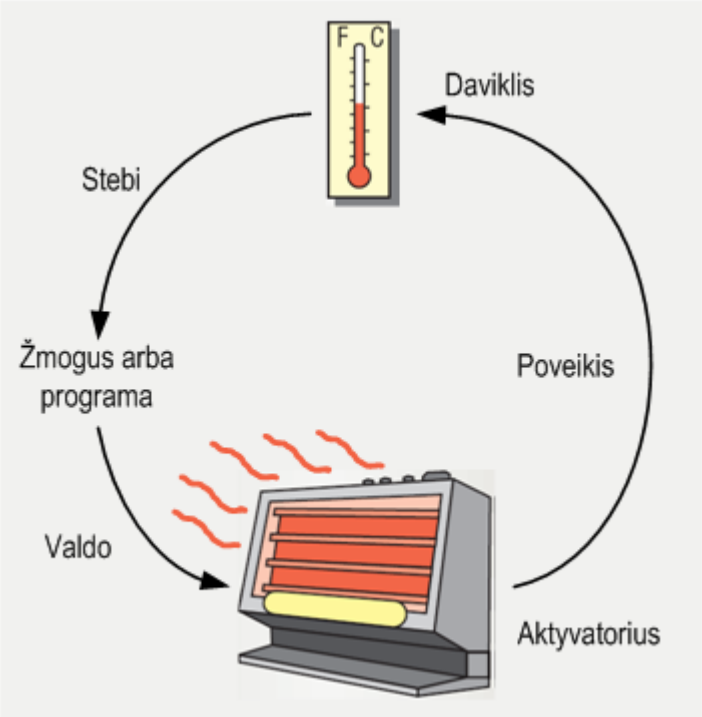
1.3.2. Situacijos supratimas

Sumanios aplinkos, t.y. tokios, kaip Gator Tech Smart House projektas, programavimas apima tris skirtingas veiklas (Helal; El-Zabadani;. King; Kaddoura; Janssen: 2005: 54):

- situacijos inžinerija – daviklių duomenų interpretavimas ir būsto gyventojų poreikių identifikavimas („karšta“, „saulėta“);
- programinės įrangos inžinerija – įvairių programinės įrangos komponentų elgsenos apibūdinimas, pavyzdžiui, šilumos atsukimas ar galimo meniu iš tam tikrų ingredientų sukūrimas;
- elgsenos ir situacijos susiejimas – tai nustatymas tų programinės įrangos komponentų, kurie gali veikti konkrečioje situacijoje, ir tų, kuriuos sistema turėtų sužadinti, esant situaciniams pokyčiams. Šio proceso lemiamas momentas yra sekti-kontroliuoti sąsaja tarp daviklių ir aktyvatorių, kaip parodyta 1.5 paveiksle.

Modernus būstas informaciją apie aplinką gaus įvairių daviklių pagalba ir galės šiuos duomenis naudoti konkrečių veiksmų inicijavimui. Tipiškas būstas priklausys nuo daviklių, kad galėtų veikti, pavyzdžiui, jei name per šalta, termostatas aktyvuos šilumos šaltinį. Tačiau tai, kuo išsiskiria tokia aplinką suprantanti sistema kaip „Modernus būstas“, yra gebėjimas gauti būsenos informaciją ir atlikti veiksmus, kurie atitinka būsto gyventojų poreikius („karšta“, „saulėta“).

Daugelis daviklių suprojektuoti konkrečios srities reikšmės aptikimui. Pavyzdžiui, temperatūros daviklis gali parodyti, kad name oro temperatūra yra 25° C, šviesos daviklis gali užfiksuoti 10000 liuksų, sklindančių per lango stiklą.



Šaltinis: HELAL, Sumi; MANN, Williams; EL-ZABADANI, Hicham; KING, Jeffrey; KADDOURA, Youssef; JANSSEN, Erwin. The Gator Tech Smart House: A Programmable Pervasive Space; p. 54.

1.5 pav. Daviklio ir aktyvatoriaus tarpusavio sąveika

Kiekvienai galimai daviklio reikšmei labai sunku užkoduoti elgseną, pašalinti defektus ar ją išplėsti.

Daug lengviau susieti veiksmus su duomenimis („karšta“, „saulėta“), kurie susiję su temperatūra ir liuminescencija.

Kai name yra karšta, sistema įjungia oro kondicionierių; jeigu lauke saulėta ir televizorius yra įjungtas, sistema užtraukia užuolaidas. Šis metodas gali būti lengvai išplėstas ir pritaikytas įvairioms situacijoms, pavyzdžiui, jeigu namo gyventojas laikosi dietos, sistema galėtų neleisti mikrobangų krosnelei kepti riebios picos.

1.3.3. Situacijos valdymas

Be įvairių daviklių moderniame būste yra ir aktyvatorių – tai įtaisai, su kuriais žmonės gali sąveikauti (Helal; El-Zabadani;. King; Kaddoura; Janssen: 2005: 55). Aktyvatorius gali pakeisti aplinkos būseną. Davikliai gali sekti aktyvatorių veikimą. Pavyzdžiui, šviesos daviklis nustato, kada namas ar gyventojas įjungia lempą. Remiantis nustatyta aplinkos būsena, namas ar gyventojas gali aktyvuoti aktyvatorius. Kiekvienas moderniame būste esantis aktyvatorius turi tam tikrą iš anksto apgalvotą rezultatą (angl. *intentional effect*), kurią daviklis gali sekti. Pavyzdžiui, šilumos įjungimo iš anksto apgalvotas rezultatas yra temperatūros pakėlimas. Jeigu aktyvatoriaus iš anksto apgalvotam rezultatui yra suteikiamas aiškus apibrėžimas, tampa įmanoma apibrėžti laukiamą elgseną tam tikroje situacijoje, analizuojant visas konkrečios situacijos elgsenas, ir identifikuoti, kurie iš anksto apgalvoti

rezultatai yra nesuderinami. Tai garantuoja, pavyzdžiui, kad sistema niekada nesužadins oro kondicionieriaus ir šilumos šaltinio vienu metu.

Situacijos pokyčiai gali įvykti dėl:

- aktyvatoriaus iš anksto apgalvoto rezultato, pavyzdžiui, po to, kai įjungiamas šilumos šaltinis, namo oro temperatūra pasikeičia iš „šaltos“ į „šiltą“;
- nekontroliuojamų jėgų ar įvykių, pavyzdžiui, saulės šviesos reguliavimas ir skirtingas paros laikas „diena“ ir „naktis“.

Idealu būtų, jeigu sumani namo aplinka, patekusi į neleistiną padėtį, išeitų iš šios padėties be žmogaus įsikišimo. Suteikus aktyvatoriaus iš anksto apgalvotam rezultatui konkrečioje srityje standartizuotą apibūdinimą/aprašymą ir susiejus daviklio reikšmes su konkrečia situacija, turėtų tapti įmanoma apibrėžti tą aktyvatorių, kuris turėtų būti sužadintas, kad būtų išvengta nepageidautinos situacijos. Tačiau, jeigu nepriimtinos situacijos išvengimas yra neįmanomas, sistema gali kreiptis į išorinį dalyvį pagalbos. Pavyzdžiui, jeigu šaldytuve nėra maisto produktų ir maisto produktų pristatymo paslaugos yra neprieinamos, tada sistema galėtų informuoti išorinį dalyvį, kad laikas papildyti maisto atsargas.

1.4. Modeliavimo sistemos Ptolemy II analizė

Ptolemy II – dabartinė Ptolemy projekto programinės įrangos infrastruktūra (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 2). Šio projekto dalyviams Ptolemy II yra svarbus įrankis, kuriuo galima modeliuoti savo sukurtus projektus. Ptolemy projektas yra atviro kodo projektas dėl šių priežasčių: programinė įranga papildoma įprastiniais terpės elementais, kurie tarnauja kaip aiškūs, vienareikšmiai ir išbaigti kūrėjų rezultatų pasiekimai; atvira architektūra ir atviras kodas skatina tyrinėtojus kurti savus metodus, kurie keičia ir praplečia Ptolemy programinės įrangos branduolį.

1.4.1. Ptolemy II

Ptolemy projektas pradėtas 1996 metais (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 2). Pagrindinė priežastis jį pradėti buvo ta, kad būtų galima naudoti tinklo integraciją, kodo migravimą, gijų (angl. *thread*) panaudojimą. Visi šie elementai sujungiami Java sąsaja. Ptolemy II įvedė naują sąvoką – srities polimorfizmą (kur komponentai projektuojami taip, kad juos būtų galima naudoti daugelyje sričių) ir modalinius modelius (kur baigtinių būsenų automatai hierarchiškai sujungti su skaičiavimo modeliais). Taip pat įdiegta tęstinio laiko sritis (angl. *continuous-time domain*), kuri padeda sujungti modalinių modelių galimybes su hibridinių sistemų modeliais. Ptolemy II – moderni sistema, kurioje galima pasirinkti išvadų tipus, galimas duomenų polimorfizmas (kur komponentai gali būti projektuojami dirbti su skirtingais duomenų tipais). Visa tai jungia gerai išplėtota išraiškų kalba. Komponentai ir sritys gali turėti tam tikras sąsajas, kurios apibūdina ne tik statinę bet ir dinaminę

elgseną. Vartotojo sąsaja paremta Java kalba. Ptolemy II sukurti modeliai gali būti naudojami kaip apletai. Duomenų vaizdavimui ir komponentų perkėlimui naudojamas XML standartas.

1.4.2. Modeliavimas ir projektavimas

Ptolemy projektas tiria heterogeninius modelius, modeliavimą ir lygiagrečių sistemų modeliavimą (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 4). Didžiausias dėmesys skiriamas įterptinėms sistemoms, ypač toms, kuriose susipina skirtingos technologijos, pavyzdžiui, analoginė ir skaitmeninė elektronika, techninė ir programinė įranga bei elektroniniai ar mechaniniai įrenginiai. Taip pat pabrėžiamos kompleksinės sistemos, kurios jungia labai skirtingas operacijas, tokias kaip: tinkliniai taikymai, signalų apdorojimas, grįžtamojo ryšio valdymas, būsenų pasikeitimai ir vartotojo sąsaja.

Modeliavimas – tai formalus sistemos arba posistemio vaizdavimas. Modelis gali būti matematinis, t. y. aibė teiginių, susijusių su sistemos funkcionalumu ar fizinėmis dimensijomis. Taip pat modelis gali būti konstruktyvus, t. y. nustatytos skaičiavimo procedūros, kurios imituoja sistemos savybes. Konstruktyvus modelis dažnai naudojamas norint perteikti sistemos elgseną bei reakciją į išorinius veiksnius. Konstruktyvus modelis dažnai vadinamas vykdomuoju modeliu.

Projektavimas – tai sistemos ar posistemio apibrėžimas. Dažniausiai tai apima vieno ar daugiau sistemų modelių apibrėžimą bei modelio tobulinimą tol, kol pasiekiamas reikiamas funkcionalumas.

Projektavimas ir modeliavimas yra glaudžiai susiję. Kai kuriais atvejais modelis gali būti nekeičiamas, t. y. jis gali nurodyti posistemius, apribojimus ar elgsenas, kurie yra išoriškai priskirti projektui. Pavyzdžiui, mechaninės sistemos apibrėžimas, kuri neprojektuojama, bet turi būti valdoma elektroninės sistemos, kuri yra projektuojama.

Vykdomieji modeliai kartais vadinami simuliatoriais. Tačiau daugumoje elektroninių sistemų modelis, kuris prasideda modeliavimu pakinta į sistemoje įgyvendintą programinį produktą. Šiuo atveju skirtumas tarp modelio ir pačios sistemos tampa miglotas. Tai ypač atsispindi įterptinėje programinėje įrangoje.

1.4.3. Įterptinė programinė įranga

Įterptinė programinė įranga – tai programinė įranga, kuri patalpinta įrenginiuose (ne kompiuteriuose) (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 4). Ji paplitusi automobiliuose, telefonuose, žaisluose, lėktuvuose, apsaugos sistemose, spausdintuvuose ir kituose įrenginiuose. Aktyvus žmogus dažniau susiduria su įterptine programine įranga nei su įprastine. Pagrindinė įterptinės programinės įrangos savybė yra ta, kad ji jungia fizinį pasaulį.

Ptolemy II ypač akcentuoja metodologiją, apibrėžiančią ir gaminančią įterptinę programinę įrangą, kartu su pačiomis įterptinėmis sistemomis.

Vykdomasis modelis konstruojamas kaip skaičiavimo modelis paremtas fizikiniais dėsniais, kuriais valdomos komponentų tarpusavio sąveikos. Jeigu modelis nusako mechaninę sistemą, tada jis tiesiog nusakomas fizikiniais dėsniais. Paprastai tai yra taisyklių rinkinys, kurio pagrindu yra kuriamas modelis. Skaičiavimo modelis gali turėti daugiau nei vieną semantiką (taisyklių rinkinys, kuriomis valdomi komponentai) todėl gali būti skirtingi taisyklių rinkiniai, kurie duos tuos pačius elgsenos ribojimus.

Skaičiavimo modelio pasirinkimas labai priklauso nuo kuriamo modelio. Pavyzdžiui, grynai skaičiuojamajai sistemai, kuri transformuoja duomenų kiekį į kitą baigtinį duomenų kiekį, semantika atitiks programavimo kalbą (C, C++, Java). Modeliuojant mechaninę sistemą, semantika turi būti įgali valdyti lygiagretumą ir paskirstyti laiką.

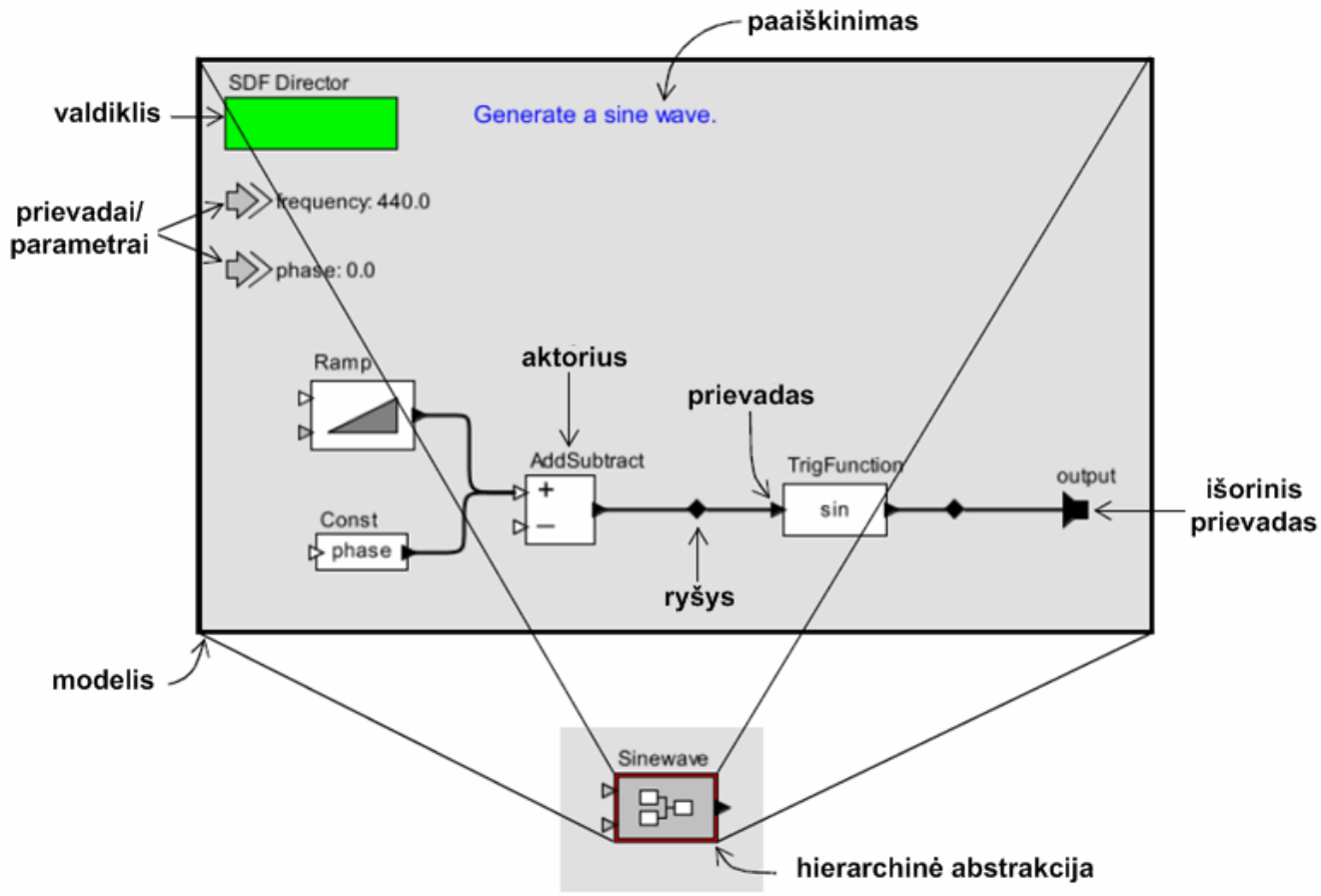
Modelio galimybė kisti į įgyvendinimą glaudžiai susijusi su naudojamu skaičiavimo modeliu. Kai kurie modeliai tinkami įgyvendinti tik specialioje techninėje įrangoje, tuo tarpu kiti sunkiai pritaikomi specialioje techninėje įrangoje dėl tam tikrų esminių pobūdžių. Netinkamo skaičiavimo modelio parinkimas gali įtakoti projekto kokybę ir privesti projektuotoją prie brangesnių ir mažiau patikimų įgyvendinimų.

Ptolemy II sistema leidžia ankstesniame žingsnyje įvertinti projektuojamos sistemos kokybę priklausomai nuo skaičiavimo modelio parinkimo.

Įterptinėse sistemose naudojami modeliai, kurie palaiko lygiagretumą ir priklausimo nuo laiko parametro. Taip yra todėl, kad įterptinės sistemos dažniausiai sudarytos iš komponentų, kurie funkcionuoja tuo pat metu ir turi sudėtinius vienalaikius sužadinimo šaltinius. Be to, jie dirba realiose sąlygose (yra įvertinamas laikas), kur atsako į dirgiklį savalaikiškumas yra labai svarbus.

1.4.4. Projektavimas naudojant aktorius

Dauguma Ptolemy II skaičiavimo modelių palaiko projektavimą naudojant aktorius (angl. *actor-oriented design*) (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 5). Tai skiriasi nuo objektiškai orientuoto projektavimo, nes naudojat aktorius galima įvertinti lygiagretumą ir komunikaciją tarp komponentų. Komponentai, vadinami aktoriais, vykdo ir bendrauja su kitais modelyje naudojamais aktoriais (1.6 pav.). Kaip objektas, aktorius turi gerai nusakytą sąsają. Ši sąsaja apibrėžia vidinę būseną ir aktoriaus elgseną bei apriboja aktoriaus sąveiką su aplinka. Sąsaja turi išvadus, kurie vaizduoja aktoriaus komunikavimo taškus, bei parametrus, kurie naudojami nurodyti aktoriaus operaciją (1.6 pav.).



Šaltinis: BROOKS, Ch.; LEE, E. A., LIU, X., NEUENDORFFER S., AHAO Y., ZHENG H. Ptolemy II heterogeneous concurrent modeling and design in java. Volume 1: introduction to Ptolemy II, p. 6.

1.6 pav. Modelis panaudojant aktorius (viršuje) ir jo hierarchinė abstrakcija (apačioje)

Projektavimo naudojant aktorius bendravimas tarp komponentų realizuotas naudojant kanalus, kuriais perduodami duomenys iš vieno išvado į kitą pagal tam tikra pranešimų schemą. Tuo tarpu objektiškai orientuotame projekte komponentai sąveikauja perduodant valdymą per metodų iškvietimus. Kanalų naudojimas užtikrina, kad aktoriai sąveikauja tik su kanalais kuriais jie sujungti su kitais aktoriais.

Modelis, kaip aktorius, taip pat gali būti apibrėžtas kaip išorinė sąsaja. Tokia sąsaja vadinama hierarchijos abstrakcija ir yra sudaryta iš išorinių išvadų bei išorinių parametrų, kurie yra atskirti nuo modelyje panaudotų aktorių išvadų bei parametrų. Išoriniai modelio išvadai kanalų pagalba gali būti jungiami su kito išorinio modelio išvadais arba su aktoriaus išvadais. Išoriniai modelio parametrai naudojami nusakyti modelį sudarančių vidinių aktorių parametrus.

Modelio idėja, aktoriai, išvadai, parametrai ir kanalai nusako abstrakčią sintaksę. Ši sintaksė gali būti atvaizduota grafiškai (1.8 pav.), XML formatu (1.7 pav.) arba specialiu programų projektu (SystemC).

```
<class name="Sinewave">
    <property name="samplingFrequency" value="8000.0"/>
    <property name="frequency" value="440.0"/>
    <property name="phase" value="0.0"/>
    <property name="SDF Director" class="ptolemy.domains.sdf.kernel.SDFDirector"/>
    <port name="output"><property name="output"/>
    <entity name="Ramp" class="ptolemy.actor.lib.Ramp">
        <property name="init" value="phase"/>
        <property name="step" value="frequency*2*PI/samplingFrequency"/>
    </entity>
    <entity name="TrigFunction" class="ptolemy.actor.lib.TrigFunction">
        <property name="function" value="sin" class="ptolemy.kernel.util.StringAttribute"/>
    </entity>
    <relation name="relation"/>
    <relation name="relation2"/>
    <link port="output" relation="relation2"/>
    <link port="Ramp.output" relation="relation"/>
    <link port="TrigFunction.input" relation="relation"/>
    <link port="TrigFunction.output" relation="relation2"/>
</class>
```

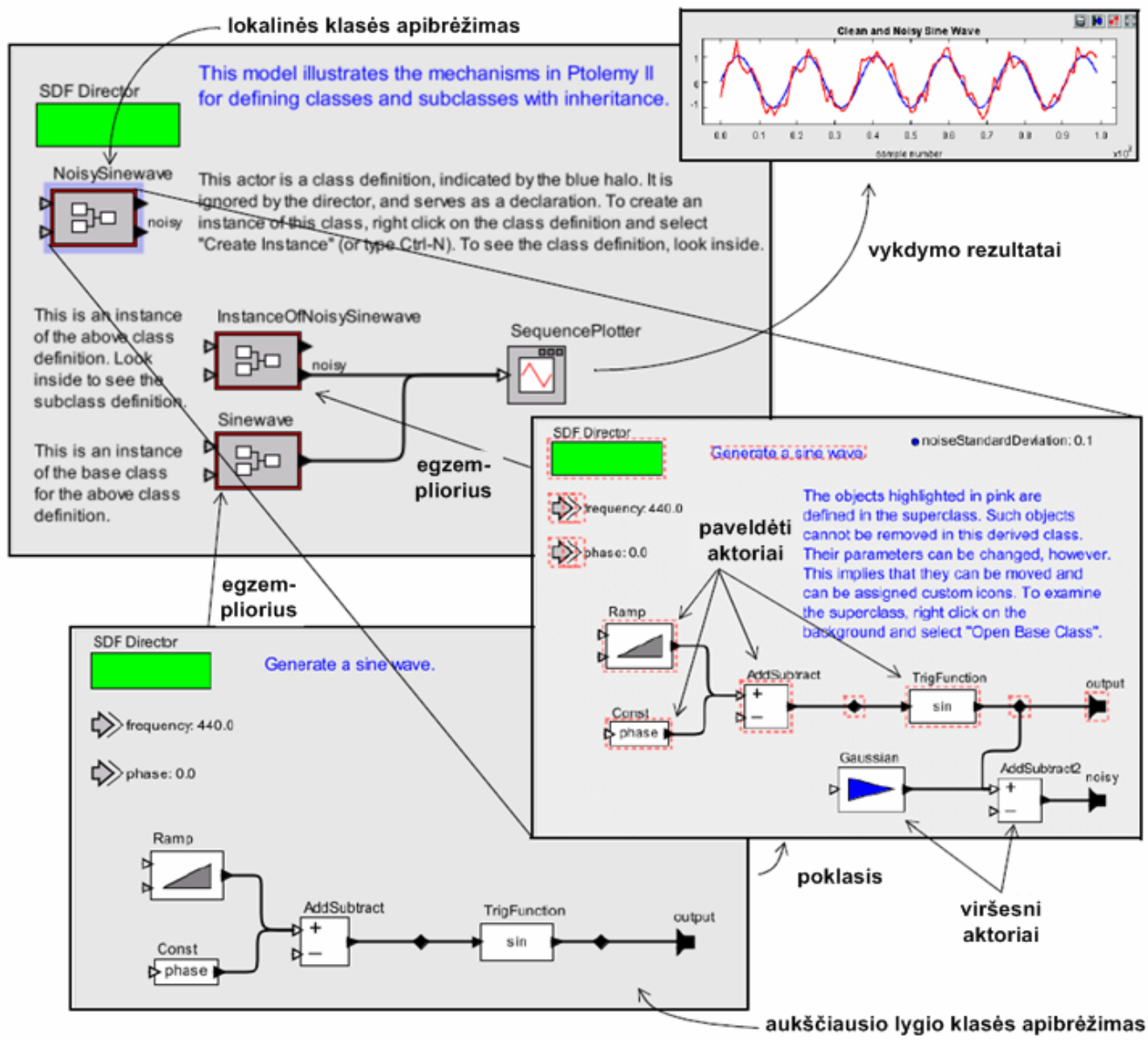
Šaltinis: BROOKS, Ch.; LEE, E. A., LIU, X., NEUENDORFFER S., AHAO Y., ZHENG H. Ptolemy II heterogeneous concurrent modeling and design in java. Volume 1: introduction to Ptolemy II, p. 7.

1.7 pav. Supaprastinto sinusinio signalo šaltinio XML atitikmuo

Ptolemy II projektas sudarytas iš komponentų rinkinio. Skaičiavimo modelis valdomas semantikų sąveika. Projektavime naudojant aktorius svarbu realizuoti struktūra sintaksiškai taip, kad projektas teiktų mažai informacijos apie semantiką. Pati semantika apibrėžiama skaičiavimo modeliu. Modeliui gali būti nusakytos taisyklės pagal kurias aktoriai atlieka vidinius skaičiavimus, keičia vidines būsenas, atlieka išorinius perdavimus. Skaičiavimo modelis taip pat nusako komunikavimo sąsajas tarp komponentų.

1.4.5. Aktorių klasės, poklasiai ir paveldėjimas

Pradedant Ptolemy II versija 4.0, buvo patobulinta projektavimo panaudojant aktorius technika (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 7). Modulinis mechanizmas buvo panašus į objektiškai orientuotas programavimo kalbas. Panagrinėkime paprastą pavyzdį, pateiktą 1.8 paveiksle.



Šaltinis: BROOKS, Ch.; LEE, E. A., LIU, X., NEUENDORFFER S., AHAO Y., ZHENG H. Ptolemy II heterogeneous concurrent modeling and design in java. Volume 1: introduction to Ptolemy II, p. 8.

1.8 pav. Aktorių klasės, poklasiai ir paveldėjimas

Modelis kairiajame apatiniame kampe yra tas pats sinusinio signalo šaltinio generatorius kaip ir 1.6 paveiksle. 1.6 paveikslo blokas pavadinimu „Sinewave“ iš tikrųjų atvaizduoja klasės egzempliorių. Šio egzemplioriaus klasės aprašymas pateiktas blokine diagrama. 1.8 paveiksle ši klasė papildyta, taip sukurtas naujas poklasis – „NoisySinewave“. Toks poklasis „paveldi“ komponentus (aktorius) ir ryšius iš bazinės klasės. Paveldėti komponentai yra apibrėžti brūkšnine linija. Šis poklasis papildomas komponentais, taip gaunamas norimas funkcionalumas.

Klasė, kuri aprašyta tame pačiame faile, vadinama aukščiausio lygio klase (angl. *top-level class*). Bet kuris modelis gali būti aukščiausio lygio klase. Modelio komponento klasė, kurią galima naudoti poklasių arba egzempliorių sudarymui vadinama lokaline klase (angl. *local class*).

Vykdomas modelis suformuoja dviejų signalų grafikus (1.8 pav. viršutinis dešinysis kampas). Vienas yra paprasto sinusinio signalo, kitas – sinusinio su triukšmais signalo grafikas. Pirmąjį suformavo aktorius „Sinewave“, kuris yra Sinewave klasės egzempliorius, antrąjį – InstanceOfNoisySinewave aktorius (Sinewave poklasis).

Kuriant tokį modelį Ptolemy II sistemoje, reikia sudaryti kelis sprendimus, kurie prilygsta kalbos kūrimo sprendimams. Pirma, modelis – tai aktorių, prievadų, atributų ir ryšių aibė. Modelis gali būti atvaizduotas kaip programa, turinti vaizdinę sintaksę. Bet kuris modelis gali būti klasė arba egzempliorius. Klasė atstoja prototipą, iš kurio gaunamas egzempliorius. Klasių funkcionavimo visame abstrakčios sintaksės lygyje mechanizmui užtikrinti, Ptolemy II klasės yra grynai sintaksiški objektai, kurie neturi įtakos vykdant modelį. Jie nematomi modelio valdikliui, kuris pasirūpina variklio vykdymu. Kaip padarinys, Ptolemy II neleidžia sujungti klasės aprašo prievadus su kitais prievadais, nes tai iššaukia klaidą.

Poklasis paveldi pagrindinės klasės struktūrą. Tiksliau sakant, kiekvienas pagrindinės klasės objektas (aktorius, atributas, prievadas ar ryšys) turi atitinkamą objektą poklasyje. Tai vadinama nekintamu šaltiniu. Iš poklasio negalima pašalinti pagrindinės klasės objektų, nes tai pažeistų nekintamo šaltinio sąlygą. Tačiau poklasis gali turėti naujus objektus.

1.4.6. Sintaksė

Ptolemy II modeliai gali būti konstruojami trimis būdais (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 10). Dažniausiai naudojami vaizdiniai žymenys (1.6 pav.), bet tai, žinoma, ne vienintelis sprendimas. Dar naudojami XML failai (1.7 pav.) bei Ptolemy II branduolys ir Java programinis kodas vykdomų modelių sudarymui.

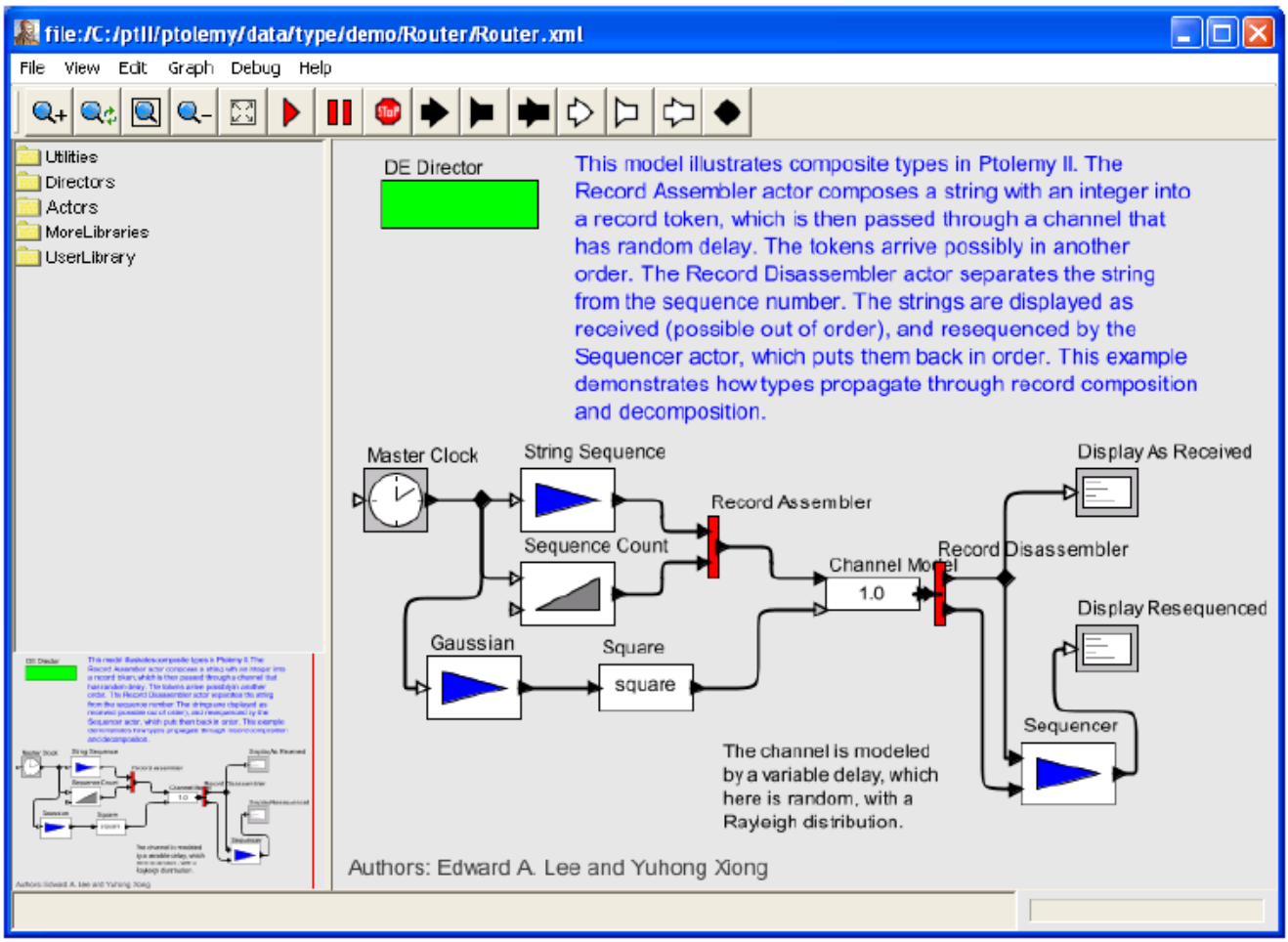
Vaizdiniai sistemos aprašymai dažnai palengvina žmogui suprasti modelio esmę. Dauguma kūrėjų Ptolemy II projekte naudoja tokius aprašus pilnai ir formaliai specifikuoti modelius. Šie vaizdiniai aprašymai leidžia alternatyvią sintaksę susieti skaičiavimo modelio semantika.

Ptolemy II modelis pavaizduotas blokine diagrama (1.9 pav.). Šiame pavyzdyje įrašai yra sudaromi iš simbolių eilučių ir skaičių, vaizduojančių eilės numerį. Įrašai paleidžiami į tinklą, kuriame imituojami atsitiktiniai užlaikymai. Įrašai gali pasiekti adresatą nesutvarkyta eile, tačiau Sequence aktorius naudojamas tam, kad surūšiuotų juos naudojant sekos numerį.

1.10 paveiksle pavaizduotas vaizdinis aprašas, kuriame komponentai atvaizduojami apskritimais, o jungtys tarp komponentų atvaizduojamos įvardintais lankais. Ši sintaksė atvaizduoja baigtinių būsenų automatą. Kiekvienas modelio apskritimas – tai būsena, o lankai – tai perėjimai tarp būsenų.

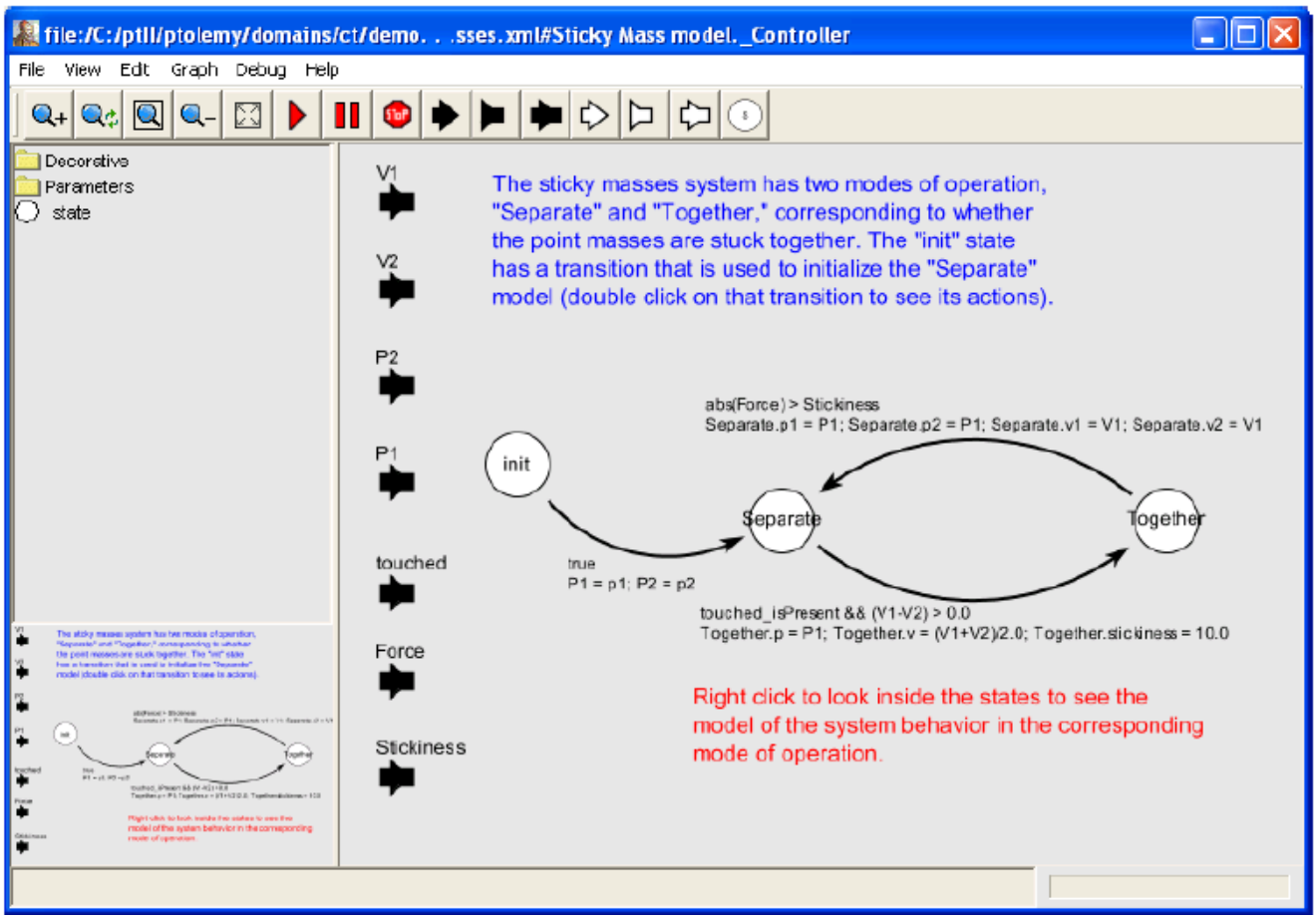
1.4.7. Architektūros projektavimas

Architektūros aprašymo kalbos, tokios kaip Wright ar Rapide paremtos formalizmu, nusakančiu komponentų tarpusavio sąveikas kylančias kuriant programinę įrangą (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 12). Priešingai, Ptolemy II gali būti vadinama architektūros kūrimo kalba, kadangi jos tikslas ne tiek aprašyti egzistuojančias tarpusavio sąveikas, bet labiau supaprastinti programinės įrangos architektūrą, įvedant kai kurias struktūras į šias sąveikas.



Šaltinis: BROOKS, Ch.; LEE, E. A., LIU, X., NEUENDORFFER S., AHAO Y., ZHENG H. Ptolemy II heterogeneous concurrent modeling and design in java. Volume 1: introduction to Ptolemy II, p. 12.

1.9 pav. Ptolemy II modelis. Blokinė diagrama



Šaltinis: BROOKS, Ch.; LEE, E. A., LIU, X., NEUENDORFFER S., AHAO Y., ZHENG H. Ptolemy II heterogeneous concurrent modeling and design in java. Volume 1: introduction to Ptolemy II, p. 13.

1.10 pav. Ptolemy II modelis. Baigtinių būsenų automatas

1.4.8. Skaičiavimo modeliai

Yra daug skaičiavimo modelių, kurie nagrinėja lygiagretumą ir laiko parametrą skirtingais metodais (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 14). Kiekvienas duoda sąveikos su komponentais mechanizmą. Skaičiavimo modelio naudingumas auga nuo modeliavimo savybių, kurios pritaikomos visiems panašiems modeliams. Daugumos skaičiavimo modelių savybės išvestos panaudojant formalią matematiką. Skaičiavimo modelis gali būti apibrėžtas, statiškai suplanuotas arba belaikis (angl. *time safe*). Priklausomai nuo savybių, parenkamas reikiamas modelis.

Ptolemy II skaičiavimo modeliai daugiausia naudojami įterptinėse sistemose. Modelį apjungia grafas, sudarytas iš mazgų (esybės) ir lankų (ryšiai). Daugumoje sričių esybės yra aktoriai (esybės su funkcionalumu) ir juos jungiantys ryšiai. Tačiau lygiagretumo modelis ir komunikavimo mechanizmas gali žymiai skirtis.

Skaičiavimo modelių tipai:

- komponentų sąveika;
- komunikuojantys nuoseklūs procesai;

- besitęsiančio laiko;
- diskrečių įvykių;
- paskirstytų diskrečių įvykių;
- diskretaus laiko;
- baigtinių būsenų automatas;
- Giotto;
- grafinis;
- hibridinės sistemos;
- procesų tinklas;
- sinchronizuotų duomenų srautų;
- sinchronizuotas/reaktyvus;
- laikinis daugiaprogramis;
- bevielis.

1.4.9. Galimybės

Ptolemy II yra trečios kartos sistema, kuri realizuota panaudojus Java programavimo kalbą (Brooks; Lee; Liu; Neuendorffer; Ahao; Zheng: 2004: 34). Pagrindinės Ptolemy II galimybės, kuriose panaudotos naujos modeliavimo ir projektavimo technologijos:

- Aukštesnio lygio lygiagretumo projektavimas naudojant Java. Java palaiko lygiagretumą žemame lygyje, naudojant gijas ir monitorius. Išlaikyti saugumą ir gyvumą yra gana sunku. Ptolemy II turi keletą sričių, kurios palaiko lygiagrečias sistemas žymiai aukštesniame abstrakcijos lygmenyje, kuris priklauso nuo programinės įrangos architektūros.
- Klasės naudojančios aktorius, poklasis ir paveldimumas. Klasės yra pagrindinis objektiškai orientuoto projektavimo agregatinis mechanizmas. Aktorių panaudojimas daro tam tikrą įtaką objektiškai orientuotam mechanizmui, panaikinti agregatiniai mechanizmai aktorių lygmenyje.
- Polimorfinių sričių aktoriai. Aktorių bibliotekos atskirtos sritimis. Aktoriai gali operuoti daugiau nei vienoje srityje. Mechanizmas, kuriuo aktoriai sąveikauja su kitais aktoriais priklauso nuo srities, kurioje jie panaudoti.
- Praplėstas XML failų formatas. XML yra nustatytas standartas atvaizduoti loginius ryšius tarp komponentų ir informacijos. Ptolemy II naudoja XML kaip pirminį formatą nuolatiniam duomenų projektavimui.
- Geresnis modulių suskirstymas panaudojant paketus. Ptolemy II suskaidyta į paketus, kurie gali būti naudoti atskirai.

- Visiškas sintaksės atskyrimas nuo semantikos.
- Saugių gijų lygiagretus vykdymas.
- Pilnai integruotų išraiškų kalba.
- Programinė įranga paremta objektiniu modeliavimu.

Atlikus analizę matyti, kad pradinėje fazėje reikalingas vartotojo poreikių išsiaiškinimas. Tai leidžia projektuotojui sukurti modernaus būsto komponentus, kurie efektyviai ir veiksmingai patenkintų vartotojo poreikius. Plėtojami produktai turi veikti kartu su esamais ir ateities produktais, kad suformuotų integruotą namo architektūrą. Naujo elemento įterpimas reikalauja jo charakteristikų ir veiksenos tyrimo, nustatymų ir integravimo instrukcijų bei testavimų, kurių metu išaiškinami konfliktai ar neapibrėžta veiksena bendroje sistemoje.

Atliekant analizę rėmiausi „The Gator Tech Smart House“ projektu, kuriame buvo aprašyti „sumanūs“ įrenginiai. Norint šiuos įrenginius įterpti į būsto projektą, reikia sukurti metodiką, kuri aiškiai apibrėžtų reikiamus atlikti žingsnius, siekiant gauti norimą rezultatą.

Kitoje darbo dalyje bus aprašoma būsto įrenginių transformacijos į Ptolemy II modeliavimo ir kūrimo metodika.

2. MODELIAVIMO METODIKOS SUDARYMAS

2.1. Projektavimo naudojant aktorius metodika

Daugelis valdymo sistemos aspektų gali įtakoti galutinį apibrėžtą sistemos darbą (J. Liu; Eker; Janneck; J. W. Liu; Lee: 2004: 251). Pagrindinis uždavinys yra suskaidyti valdomą sistemą į daug lengvai valdomų ir į konkrečią sritį orientuotų posistemių, taip, kad projektuotojai galėtų efektyviai suskaidyti ir išspręsti problemą. Į komponentus orientuotos projektavimo metodologijos palaiko siūlymą, kad sistema skaidoma į komponentus, turinčius griežtai nusakytas sąsajas. Kiekvienas komponentas apima tam tikrą funkcionalumą, tokį kaip skaičiavimas ir komunikacija.

Yra sukurta daug komponentais pagrįstų projektavimo metodologijų pavyzdžių, kurie numato skirtingus požiūrius į komponentus, tokius kaip objektas, tarpinė komunikavimo aplinka (angl. *middleware*) ir aktorius (Szyperski: 1998: 29). Objektiškai orientuotas projektavimas paremtas objekto abstrakcija, klasių hierarchija ir metodų iškvietimais. Ši metodologija buvo pritaikyta kurti įterptinę ir realaus laiko programinę įrangą, kuri formaliai specifikuojama UML kalba.

Kadangi keletas objektų dirba kartu tam, kad gauti susijusį funkcionalumą, tarpinės komunikavimo aplinkos projektavimas palaiko vieno ar kelių objektų apjungimą į abstrakčius servisus ir šių servisų komponavimą į sistemą. Tarpinės komunikavimo aplinkos įtaka pasireiškia paskirstytose sistemose, nes komunikavimo sąvoka yra daug aiškesnė nei objektiškai orientuotame projektavime naudojami nuotolinių procedūrų iškvietimai (angl. *remote procedure call*).

Nepaisant abstrakčių skirtumų, pagrindinė struktūra objektiškai orientuotoje ir tarpinės komunikavimo aplinkos sistemose yra objektai, kurie susiję vienas su kitu ryšiais. Jų pagrindinė sąveika yra metodų iškvietimas. Metodų iškvietimas tiesiogiai perduoda valdymą iš vieno objekto į kitą. Svarbios sistemos charakteristikos, tokios kaip vienalaikiškumas ir resursų panaudojimas, yra paslėptos nuo metodų iškvietimų sąsajos. Ir objektiškai orientuotos, ir tarpinės komunikavimo aplinkos projektavimo metodikos pabrėžia kaip suskaidyti sistemą į komponentus, tačiau komponentų sudarymo teisingumas paliktas projektuotojui. Projektavimas naudojant aktorius patvirtina modelių tarpusavio sąveikos įvairovę tarp komponentų ir išreiškia šių sąveikų tipus nepriklausomai nuo komponentų funkcionalumo.

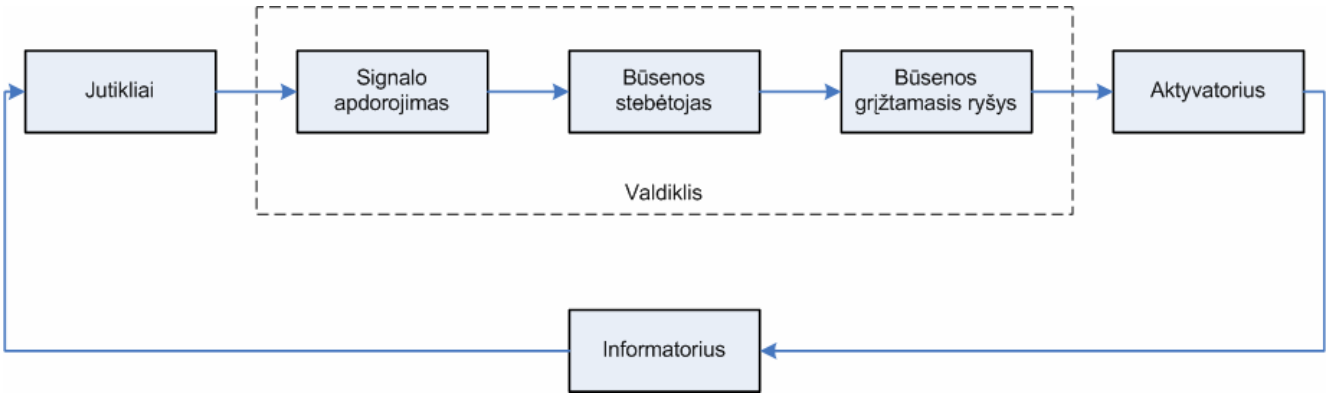
2.1.1. Aktoriaus sąvoka

Aktorius – tai objektas, atliekantis parametrizuotus veiksmai su įvesties duomenimis tam, kad gauti išeities duomenis (J. Liu; Eker; Janneck; J. W. Liu; Lee: 2004: 251). Priklausimai nuo aktoriaus sandaros jis gali turėti arba neturėti būsenų. Įvesties ir išvesties duomenys perduodami per jungtis (angl. *port*). Jungtys ir parametrai sudaro aktoriaus sąsaja. Jungtis, priešingai nei objektiškai

orientuotame projektavime, neturi kreipinio-grįžimo semantikos. Iš tikrųjų aktorius nusako vietines veiksenas ir neturi įtakos į kitus aktorius.

Yra sukurta nemažai projektavimo naudojant aktorius aplinkų, pavyzdžiui, „Simulink“ (MathWorks), „LabVIEW“ (National Instruments), „Cocentric studio“ (Synopsys), „ROOM“ (Rational Software), akademinėje bendruomenėje naudojama „Actine objects and actors“, „Moses“, „Polis“, „Ptolemy“ ir „Ptolemy II“.

Aibė aktorių elgsenų nėra gerai apibrėžta be koordinuojančio modelio. Aplinka, kuriai priklauso aktoriai ir kuri apibrėžia tarpusavio sąveikas, vadinama rėmine konstrukcija (angl. *framework*). Rėminės konstrukcijos diferencijuoja daug modeliavimo naudojant aktorius paradigmų. Pavyzdžiui, „ROOM“ aktorių modeliai siūlo kiekvieną „aktyvų“ aktorių valdyti atskiroje gijoje (angl. *thread*), „Simulink“, „LabVIEW“ neturi aktyvių aktorių. Šiose sistemose centrinis valdiklis nusako aktorių veikimo eiliškumą. Aktorių tarpusavio sąveikos tipas buvo nagrinėjamas skaičiavimo modelio (angl. *model of computation*) aspektu. Skaičiavimo modelis apibrėžia komunikavimo tarp prievadų semantiką ir veikimo eiliškumą tarp aktorių. Rėminė konstrukcija realizuoja skaičiavimo modelį. Rėminė konstrukcija kartu su aktoriais apibrėžia sistemą. Daugelis modeliavimo naudojant aktorius aplinkų turi suvienodintus skaičiavimo modelius ir realizuoja vieną rėminę konstrukciją. Pavyzdžiui, „Simulink“ turi tolydaus laiko/mišrių signalų rėmines konstrukcijas, tačiau sudėtingesnėms sistemoms dažniausiai nepakanka vieno skaičiavimo modelio.



Šaltinis: LIU, Jie; EKER, Johan; JANNECK, Jörn W.; LIU, Xiaojun; LEE, Edward A. Actor-Oriented Control System Design: A Responsible Framework Perspective. p. 252

2.1 pav. Valdymo sistemos vaizdas naudojant aktorius

2.1 paveiksle pavaizduotas paprastos valdymo sistemos skaidymas į jutiklius, signalų apdorojimo įrenginį, būsenos stebėtoją, būsenų grįžtamojo ryšio valdiklį, aktyvatorius ir informatorių. Galvojimas aktoriaus plane pavaizduotais terminais palengvina identifikuoti vienalaikiškumo ir komunikacijos problemas. Pavyzdžiui, informatorius funkcionuoja vienalaikiškai su valdikliu fiziniame pasaulyje ir viduje valdiklio esantys trys aktoriai realizuoti įterptinėje programinėje įrangoje gali funkcionuoti nuosekliai dėl duomenų priklausomybės. Tačiau sąveikos tarp komponentų tipas nėra aiškus šiame vaizde. Nėra tiesioginio skirtumo tarp informatoriaus, jutiklių, aktyvatoriaus ir valdiklio

sąveikos tolydaus laiko/mišraus signalo tipo, tačiau valdiklio viduje duomenų srautų modelis gali būti labiau apibrėžtas.

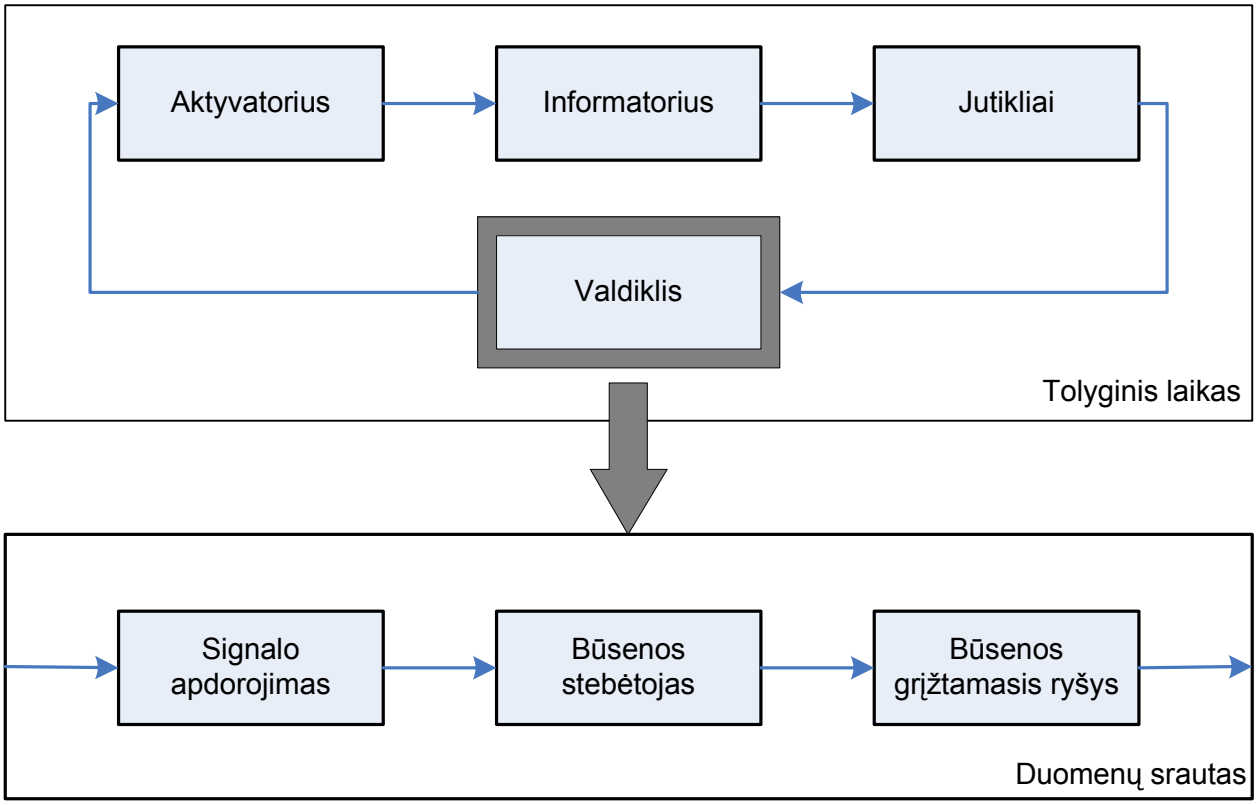
Skaičiavimo modeliai, kurie valdo aktorių tarpusavio sąveikas atspindi posistemių dinamiką, kuri galėtų būti skirtinga net duotoje paprastoje valdymo sistemoje – informatoriaus dinamika tolydi, tuo tarpu valdiklio – diskreti. Sudėtingesniuose atvejuose netgi valdiklio viduje valdymo taisyklės, perjungimo logika, realaus laiko planavimas ir komunikacijos tinklai yra taip pat skirtingi: sinchroniški arba asinchroniški, apsaugoti arba neapsaugoti, nuoseklūs arba lygiagretūs ir t. t. Nors ši teorija kiekvienai atskirai sričiai yra pakankamai gerai suprantama ir nusistovėjusi, dinamikos integravimas įneša į projektavimo uždavinį ženklų sudėtingumą. Jeigu projektavimo aplinka palaiko tik vieną skaičiavimo modelį, tai ji gali modeliuoti tik tam tikrą sudėtingos sistemos dalį arba tik tam tikrą abstrakcijos lygmenį. Galutinio projekto teisingumas gula ant galutinių testavimų, kurie dažniausiai iššaukia ilgus projektavimo ciklus ir didina kaštus.

2.1.2. Hierarchinis heterogeniškumas

Hierarchija – sąvoka, praplečianti projektavimo naudojant aktorius mastą, t. y. aktorių tinklas gali būti vaizduojamas kaip vienas aktorius (J. Liu; Eker; Janneck; J. W. Liu; Lee: 2004: 252). Naudojant hierarchiją, galima efektyviai suskaidyti sudėtingą modelį į tris sujungtus submodelius, kurie suformuoja tarpusavyje sąveikaujančių komponentų tinklą kiekviename lygmenyje. Hierarchija yra tam tikra abstrakcija, kuri paslepia posistemio detales nuo likusios sistemos. Ji taip pat apibrėžia sąveikos sritis, t. y. posistemio pakeitimai izoliuoti veikimo lygyje.

Hierarchijos gali būti panaudotos suvienodintuose modeliuose valdyti sintaksinius painumus („Simulink“). Heterogeniniams skaičiavimo modeliams valdyti efektyviau naudojamos hierarchijos (hierarchinis heterogeniškumas). Šis požiūris verčia kiekvieno lygmens tarpusavyje sąveikaujančius aktorius lokaliai būti homogeniniams, tuo tarpu skaičiavimo modeliai kiekviename hierarchijos lygmenyje gali skirtis. Gerai nusakytas skaičiavimo modelis tame pačiame lygmenyje padidina sistemos supratimą ir leidžia pritaikyti prie konstrukcijos tam tikrą sistemos dalį, kadangi formalios savybės nusakomos naudojant skaičiavimo modelį.

Aktorių apjungimas ir hierarchinis heterogeniškumas yra galingos technikos, kurios leidžia projektuotojams dirbti skirtinguose abstrakcijos lygmenyse, bet kartu išlaikant globalų sistemos vaizdą. Pavyzdžiui, 2.2 paveiksle pavaizduotas valdymo sistemos, pateiktos 2.1 paveiksle, hierarchinis modelis. Projektavimo pradžioje valdymo inžinieriai gali dirbti su modeliu tolyginio laiko srityje (2.2 paveikslo viršutinė dalis). Kai nusakomos vykdymo taisyklės, programinės įrangos kūrėjai gali gilintis ir pakeisti aktorių „Valdiklis“ duomenų srautų modeliu, kuris yra labiau tinkamas programinės įrangos įgyvendinimui. Projektavimo terpė, kuri palaiko hierarchinį heterogeniškumą, leidžia programinės įrangos kūrėjams pratęsti naudojimą aktoriaus „Informatorius“ nekeičiant jo modelio.



Šaltinis: LIU, Jie; EKER, Johan; JANNECK, Jörn W.; LIU, Xiaojun; LEE, Edward A. Actor-Oriented Control System Design: A Responsible Framework Perspective. p. 253

2.2 pav. Valdymo sistemos, pateiktos 2.1 paveiksle, hierarchinis modelis

Hierarchiškai sudaromų heterogeninių skaičiavimo modelių rėminių konstrukcijų formavimas nėra trivialus. Pirmiausia, rėminė konstrukcija, kaip aktorių koordinatorius, turi sudaryti ir valdyti kiekvieno aktoriaus veikseną taip, kad bendra modelio elgsena atitiktų skaičiavimo modelio specifiką. Svarbu tai, kad rėminė konstrukcija turi sujungti aktorių veiksenas į stambiagrūdę atominę veikseną taip, kad aukštesniame hierarchijos lygmenyje visa rėminė konstrukcija atrodytų kaip sudėtingas aktorius. Sekančiame poskyryje bus aptariamas Ptolemy hierarchinio heterogeniškumo įgyvendinimo metodas kuriant atsakingos rėminės konstrukcijos sąvoką.

2.1.3. Atsakingos rėminės konstrukcijos

Aktorių ir rėminės konstrukcijos tarpusavio sąveika gali būti formuluojama žymėtų perėjimų sistemos modeliu (angl. *labeled transition model*) (J. Liu; Eker; Janneck; J. W. Liu; Lee: 2004: 253).

Rėminė konstrukcija Φ apibrėžiama kaip žymėta perėjimo sistema $\Phi = (\Sigma, \Lambda, \Gamma, \Omega)$, kur

- Σ yra rėminės konstrukcijos būsenų aibė;
- Λ yra veiksmų aibė;
- $\Gamma \subseteq \Sigma \times \Lambda \times \Sigma$ yra perėjimų aibė priklausoma nuo veiksmų Λ; perėjimas $(\sigma_1, \lambda, \sigma_2) \in \Gamma$ dar žymimas $\sigma_1 \xrightarrow{\lambda} \sigma_2$;

- Ω yra pradinių būsenų aibė.

Jei yra svarbi rėminė konstrukcija, tai kartais rašoma $\Phi = (\Sigma_\Phi, \Lambda_\Phi, \Gamma_\Phi, \Omega_\Phi)$. Rėminės konstrukcijos būsena atspindi dalinamą informaciją tarp aktorių ir konstrukcijos, tokios kaip iteracijos sąvoka arba laikas ir komunikacijos semantika. Pavyzdžiui, jeigu komunikacijos kanalas tarp aktorių yra FIFO eilė, tai šios eilės yra rėminės konstrukcijos būsenų dalis.

Veiksmų aibėje aiškiai apibrėžiama NOP, $\varepsilon \in \Lambda$, kuri leidžia neatlikti perėjimo. Pavyzdžiui, $(\sigma_1, \varepsilon, \sigma_2) \in \Gamma \Rightarrow \sigma_1 = \sigma_2$.

Aktorius A rėminėje konstrukcijoje Φ apibrėžiamas kaip rinkinys $A = (S, T, I, Q)$, kur

- S yra aktorių būsenų aibė;
- $T \subseteq S \times \wp(\Sigma_\Phi) \times \Lambda_\Phi \times S$ yra perėjimų aibė, kur $\wp(\Sigma_\Phi)$ yra laipsninė Σ_Φ aibė (angl. *power set*).
- $I \subseteq S$ pradinių būsenų aibė;
- $Q \subseteq S$ statinių būsenų aibė. Dažniausiai $I \subset Q$.

Perėjimas $(s, G, \lambda, s') \in T$ taip pat gali būti užrašomas tokia forma: $s \xrightarrow{G/\lambda} s'$, $G \in \Sigma_\Phi$ vadinama perėjimo prižiūrėtoju (angl. *guard*), o $\lambda \in \Lambda_\Phi$ vadinama perėjimo veiksmas.

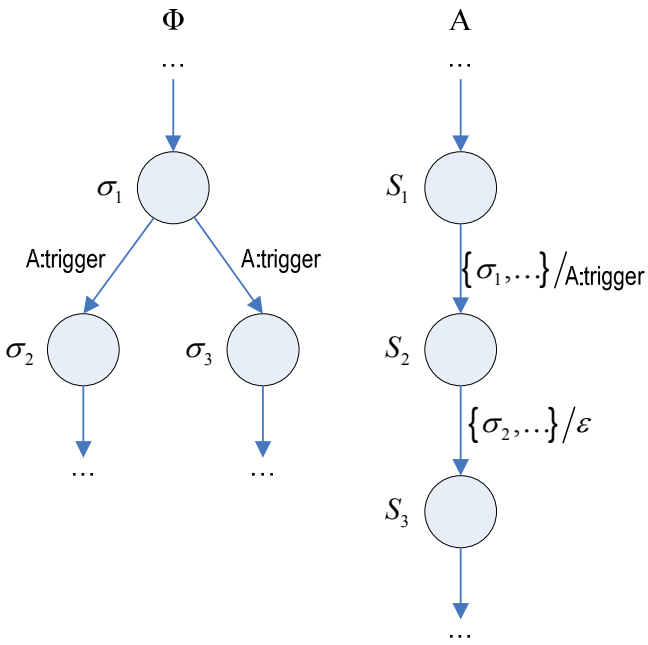
Aktorių aibės ir rėminės konstrukcijos kompozicijos vykdymas turi sinchroninę semantiką. Bet kuriame žingsnyje aktorius daro perėjimą iš vienos būsenos į kitą, jei esama rėminės konstrukcijos būsena yra perėjimo priežiūroje. Aktoriaus perėjimo veiksmas gali pakeisti rėminės konstrukcijos būseną, t. y. perėjimas $s \xrightarrow{G/\lambda} s'$ vykdomas tik tada, jeigu esama rėminės konstrukcijos būsena $\sigma \in G$. Be to, kiekvienam $\lambda \in \Lambda$, kai $s \xrightarrow{G/\lambda} s'$, egzistuoja $\sigma \in G$ ir keli $\sigma' \in \Sigma$, tokie, kad $\sigma \xrightarrow{\lambda} \sigma'$. Bendru atveju, nei rėminė konstrukcija, nei aktorius neturi būti deterministiniai. Tais atvejais, kai rėminės konstrukcijos būsena leidžia kelių aktorių perėjimus, veiksena nusakoma nedeterministiškai.

Bet kuriam $t = (s, G, \lambda, s') \in T$ apibrėžtos funkcijos $Src(t) = s$ ir $Des(t) = s'$, kurios grąžina šaltinio ir imtuvo perėjimo būsenas. Aktorių perėjimo iš būsenos s į būseną s' vykdymo kelias $P_{s,s'}$ yra perėjimų grandinė, t. y. baigtinė seka $\{t_1, t_2, \ldots, t_n\} \subseteq T$ tenkinanti šias sąlygas: $Src(t_1) = s$, $Des(t_n) = s'$, ir $\forall 1 \le i \le n-1$, $Des(t_i) = Src(t_{i+1})$. Dėl rėminės konstrukcijos ir aktorių nedeterministiškumo, gali būti daugiau nei vienas kelias iš būsenos s į būseną s'. Teigiama, kad $P_{s,s'}$ eina per būsena s'', jeigu egzistuoja du perėjimai vykdymo kelyje tokie, kad s'' yra vieno imtuvas ir kito šaltinis.

Hierarchinio heterogeniškumo tikslui pasiekti reikia paslėpti tarpines aktoriaus veiksenas ir rodyti tik „reikšmingas būsenas“ (J. Liu; Eker; Janneck; J. W. Liu; Lee: 2004: 254). Dėl šios priežasties apibrėžta statinių būsenų (angl. *quiescent state*) aibė, kuri yra poaibis iš aktorių būsenų aibės. Teoriškai, statinių būsenų aibė gali būti bet koks poaibis. Praktiškai, dažniausiai tai yra apibrėžta taip: funkcinio bloko pabaigos būsena, perėjimo pabaigos būsena, posistemio tam tikro laiko būsena arba būsena, kuri gali būti atvaizduota su minimaliu kintamųjų skaičiumi.

Aktorių vykdymas vykdomas dviem – statinėmis ir vykdymo fazėmis. Vykdymas įeina į statinę fazę, kai aktorius pasiekia statinę būseną. Pirmasis perėjimas iš statinės būsenos parodo, kad aktorius įėji į naują vykdymo fazę. Dėl šios priežasties perėjimui suteikiamas pavadinimas: statinei būsenai $q \in Q$ perėjimas t, kai $Src(t) = q$ vadinama trigeriu (angl. *trigger*) ir žymima t_q. Visi trigeriai iš q sudaro aibę T_q.

Būtų idealu, jeigu vykdymo fazė pasiektų kitą statinę būseną. Tada, žiūrint iš aukštesnio lygmens, vykdymas būtų išskirtas į atominius perėjimus iš vienos statinės būsenos į kitą. Formaliai, tiksli reakcija $P_{q,q'}$ (angl. *precise reaction*) yra vykdymo kelias iš $q \in Q$ į $q' \in Q$ neinant per bet kokias statines būsenas. Taigi, jei rėminės konstrukcijos būsenų trajektorija leidžia visus $P_{q,q'}$ perėjimus, tai gali būti pasiekta tiksli reakcija. Tačiau rėminės konstrukcijos būsenų vystymas priklauso ne tik nuo pačios konstrukcijos, bet ir nuo joje esančių visų aktorių, todėl tiksli reakcija gali būti netriviali.



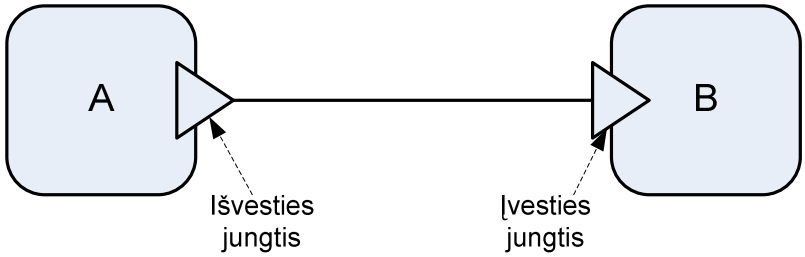
Šaltinis: LIU, Jie; EKER, Johan; JANNECK, Jörn W.; LIU, Xiaojun; LEE, Edward A. Actor-Oriented Control System Design: A Responsible Framework Perspective. p. 254

2.3 pav. Atsakingą rėminę konstrukciją vaizduojantis modelis.
Rėminė konstrukcija informuoja aktorių apie potencialų atsakingą trigerį

Bendru atveju bendradarbiavimas tarp aktorių ir rėminės konstrukcijos yra esminis siekiant gauti tikslias reakcijas. Trigeris t_q vadinamas atsakingu (angl. *responsible*), jeigu jis gali garantuoti, kad visoms galimoms rėminės konstrukcijos ateities būsenoms, kai skaičiavimo modelis įgyvendintas rėminėje konstrukcijoje, vykdymas pradedant nuo q pasieks bet kurį $q' \in Q$. Rėminė konstrukcija vadinama atsakinga, jei duoti atsakingi trigeriai ir ji gali garantuoti visų aktorių tikslias reakcijas.

Vienas konkretus perėjimo atvejis tarp aktorių ir rėminės konstrukcijos pateiktas 2.3 paveiksle, kuriame tiksli trigerio veikla, pavadinta „A:trigger“ liepia aktoriui A įspėti rėminę konstrukciją Φ dėl tikslios reakcijos. Rėminė konstrukcija gali pasirinkti priimti trigerį ir daryti perėjimą, kuris leidžia aktoriui atlikti veiksmus. Pasirinkimas parinktas nedeterministinis, todėl rėminė konstrukcija taip pat gali pereiti į tokią būseną, kurioje bus uždrausti aktoriaus veiksmai. Šis modelis plačiai naudojamas praktiškai visuose skaičiavimo modeliuose.

Pilnam atsakomybės pasiekimui, kai rėminė konstrukcija priima atsakingą trigerį, reikia suvaldyti rėminės konstrukcijos būsenų perėjimus ir išlaikyti sąlygas, kurias perdavė aktorius atsakingam trigeriui (J. Liu; Eker; Janneck; J. W. Liu; Lee: 2004: 254). Šią savybę labai sunku analizuoti, tačiau tai yra daug paprasčiau jei rėminė konstrukcija įgyvendina formalų skaičiavimo modelį ir apriboja aktorių tarpusavio sąveikas. Aktoriaus perėjimas $s \xrightarrow{G/\lambda} s'$ vadinamas *vidiniu perėjimu* (angl. *internal transition*), jei $G = \Sigma$ ir $\lambda = \varepsilon$, t. y. perėjimas gali įvykti bet kuriuo metu ir tai neturi jokios įtakos rėminės konstrukcijos būsenai. Kitu atveju perėjimas vadinamas *I/O perėjimu* (angl. *I/O transition*). Komunikacijos modelis apibrėžia prižiūrėtojo ir veiksmų I/O perėjimus. Pavyzdžiui, tarkime, kad komunikacija iš aktoriaus A į B (2.4 pav.) yra FIFO begalinė eilė. Tada aktoriaus A išvesties perėjimas visada bus įjungtas. Taigi, išvesties perėjimas t turi Σ kaip prižiūrėtoją bei vykdo veiksmus, kurie keičia rėminės konstrukcijos būseną pridėdamas papildomus duomenų simbolius prie komunikacijos eilės galo. Atitinkamas įvesties perėjimo prižiūrėtojas turi rėminės konstrukcijos būsenas, kuriose yra bent vienas simbolis eilėje. Pažymima, kad simbolių pridėjimas į eilę neapriboja perėjimo, taigi dviejų sujungtų eile aktorių vykdymai yra iš esmės atskiri. Tarkime, kad $P_{q,q'}$ yra B tiksli reakcija bei yra n įvesties perėjimų. Jeigu trigeris t_q užtikrina, kad eilėje yra bent vienas simbolis, tada vykdymas priveda prie situacijos, kad būtinai pasieksime būseną q'. Toks q' yra atsakingas trigeris. Jeigu rėminėje konstrukcijoje visos jungtys yra FIFO eilės bei trigeriai yra atsakingi, tada rėminė konstrukcija yra atsakinga paprasčiausiai įdiegiant modelį pavaizduotą 2.3 paveiksle. Ši rėminė konstrukcija įgyvendina dinaminių duomenų srautų skaičiavimo modelį. Ne visos rėminės konstrukcijos yra atsakingos. Jeigu minėtame pavyzdyje aktoriaus B trigeris vietoje patikrinimo ar egzistuoja n įvesties simbolių turėtų prižiūrėtoją $G = \Sigma$, tada tai nebūtų atsakingas trigeris.



Šaltinis: LIU, Jie; EKER, Johan; JANNECK, Jörn W.; LIU, Xiaojun; LEE, Edward A. Actor-Oriented Control System Design: A Responsible Framework Perspective. p. 254

2.4 pav. Du aktoriai komunikuoja per jungtis ir rėminę konstrukciją

Pavyzdys, tarkime, kad komunikacija tarp aktorių *A* ir *B* (2.4 pav.) turi tokią semantiką, kurioje įvesties perėjimas aktoriuje *B* turi prižiūrėtoją, kuris reikalauja, kad išvesties jungtis būtų pasiruošusi atiduoti duomenis bei išvesties perėjimas aktoriuje *A* turėtų prižiūrėtoją, kuris reikalauja, kad įvesties jungtis būtų pasiruošusi priimti duomenis. Tačiau nėra būdų trigeriui iš anksto žinoti ar aktorius pasiruošęs duomenų apsikeitimui. Todėl bent vienas iš trigerių aktoriuje *A* arba *B* turi būti neatsakingas tam, kad galėtų būti vykdomas modelis. Ši rėminė konstrukcija, kuri gali įgyvendinti nuoseklius procesus, negali būti atsakinga.

Pažymėtina tai, kad atsakingos rėminės konstrukcijos neapsaugo nuo aklaviečių. Faktiškai yra įmanoma, kad kai kuriose rėminės konstrukcijos būsenose nėra atsakingų trigerių, todėl nėra vykdomi nei vieno aktoriaus veiksmai. Tačiau tai garantuoja, kad netgi esant aklavietės būsenoje visi aktoriai yra statinėse būsenose, todėl sistema, kaip visuma, turi statinę būseną.

2.1.4. Sudėtinės tikslios reakcijos

Rėminės konstrukcijos ir aktoriaus kompozicija vadinama kombinuotu aktoriumi (angl. *composite actor*) (J. Liu; Eker; Janneck; J. W. Liu; Lee: 2004: 255). Kombinuotiems aktoriams komunikuoti su kitais aktoriais aukštesniame lygmenyje turi būti pridėta į rėminę konstrukciją daugiau būsenų ir perėjimų. Tai vadinama atvira rėmine konstrukcija (angl. *open framework*) bei žymima $\hat{\Phi} = \left(\Sigma_{\hat{\Phi}}, \Lambda_{\hat{\Phi}}, \Gamma_{\hat{\Phi}}, \Omega_{\hat{\Phi}}\right)$. Atvira rėminė konstrukcija turi būti suderinama su originalia konstrukcija. Formaliai, tarkime Σ_{Φ}^{+} yra papildomos būsenos atvirai rėminei struktūrai $\hat{\Phi}$ perdengtai I/O kintamųjų aibėje, Ω^{+} yra pradinės šių kintamųjų būsenos, tada $\hat{\Phi}$ tenkina:

- $\Sigma_{\hat{\Phi}} = \Sigma_{\Phi} \times \Sigma_{\Phi}^{+}$;
- $\Lambda_{\Phi} \subset \Lambda_{\hat{\Phi}}$;
- $\Gamma_{\hat{\Phi}}$ yra suderinama su Γ_{Φ}. Tai yra, jei $\sigma_{\Phi} \xrightarrow{\lambda} \sigma'_{\Phi}$ konstrukcijoje Φ, tai bet kuris $\sigma_{\Phi}^{+} \in \Sigma_{\Phi}^{+}$, $\left(\sigma_{\Phi}, \sigma_{\Phi}^{+}\right) \xrightarrow{\lambda} \left(\sigma'_{\Phi}, \sigma_{\Phi}^{+}\right)$ konstrukcijoje $\hat{\Phi}$;
- $\Omega_{\hat{\Phi}} = \Omega_{\Phi} \times \Omega_{\Phi}^{+}$.

Sudėtiniam vykdymui, sąveikai su išorine rėmine konstrukcija, reikalingi papildomi I/O perėjimai. Tarkime C yra kombinuotas aktorius įgyvendintas konstrukcijoje $\hat{\Phi}$ bei aibė aktorių $A_i = (S_i, T_i, I_i, Q_i)$, $i \in \{1, \ldots, n\}$ ir $\Phi' = (\Sigma_{\Phi'}, \Lambda_{\Phi'}, \Gamma_{\Phi'}, \Omega_{\Phi'})$ yra išorinės rėminės konstrukcijos aktoriui C. Tada, $C = (S_C, T_C, I_C, Q_C)$ tenkina šiuos reikalavimus:

- S_C yra būsenų aibė. $S_C = \Sigma_{\hat{\Phi}} \times \Pi_{i=1}^{n} S_i$;
- T_C yra perėjimų aibė, tokių, kad
  - visos tarpusavio sąveikos tarp $\hat{\Phi}$ ir A_i yra vidiniai perėjimai aktoriui C. Tai yra, jei $\exists s_i$, $s_i' \in S_i$, $G \subset S_{\Phi'}$ ir $\sigma_{\hat{\Phi}} \in G$ toks, kad $(\sigma_{\hat{\Phi}}, \lambda, \sigma_{\hat{\Phi}}') \in \Gamma_{\hat{\Phi}}$ ir $(s_i, G, \lambda, s_i') \in T_i$, tada bet kuriam $s_c = (\sigma_{\hat{\Phi}}, s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n)$ ir $s_c' = (\sigma_{\hat{\Phi}}', s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n)$, $(s_c, \Sigma_{\Phi'}, \varepsilon, s_c') \in T_C$.
  - I/O perėjimai papildyti tam, kad pakeistų $\Sigma_{\hat{\Phi}}$ būsenas neturėtų keisti S_i arba Σ_{Φ} būsenų. Tai yra, jeigu $(s_c, G_{\Phi'}, \lambda_{\Phi'}, s_s') \in T_C$ ir $\lambda_{\Phi'} \in \Lambda_{\Phi'}$, $G_{\Phi'} \neq \Sigma_{\Phi'}$ bei $s_c = (\sigma_{\Phi}, \sigma_{\Phi}^{+}, s_1, \ldots, s_n)$, tada $s_c' = (\sigma_{\Phi}, \sigma_{\Phi}'^{+}, s_1, \ldots, s_n)$ kai kuriems $\sigma_{\Phi}'^{+} \in \Sigma_{\Phi}^{+}$.
- $I_C = \Omega_{\hat{\Phi}} \times \Pi_{i=1}^{n} I_i$;
- $Q_C \subseteq \Sigma_{\hat{\Phi}} \times \Pi_{i=1}^{n} Q_i$;

Taigi, kombinuotų aktorių I/O perėjimai yra pilnai atskirti nuo rėminių konstrukcijų ir aktorių tarpusavio sąveikos. Šiuo atveju kombinuotų aktorių vidus atitinka skaičiavimo modelį. Tarpusavio sąveikos gali būti tiriamos tarp rėminės konstrukcijos neįtraukiant atskiro aktoriaus elgsenos.

Pagrindinė atsakingų rėminių konstrukcijų savybė yra ta, kad tikslios individualaus aktoriaus reakcijos gali būti sujungtos į kombinuoto aktoriaus tikslią reakciją. Kaip anksčiau minėta, visi kombinuoto aktoriaus vidiniai aktoriai turi būti savo statinėse būsenose. Atsakingai rėminei konstrukcijai tai lengva pasiekti, kadangi visi vidinių aktorių vykdymai iš vienos statinės būsenos pasieks kitą statinę būseną po baigtinio skaičiaus transakcijų.

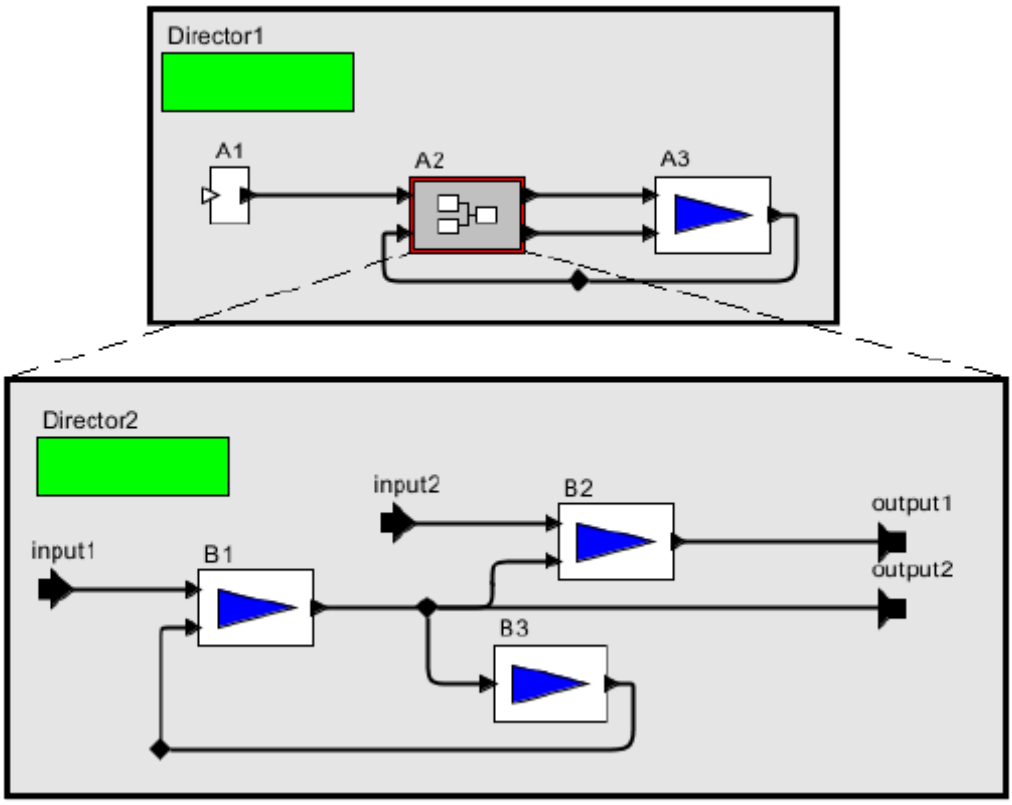
Dar vienas svarbus žingsnis kuriant kombinuotus aktorius – tai apibrėžti atsakingus kombinuotų aktorių trigerius.

Kiek tikslių reakcijų apjungti į kombinuotą tikslią reakciją priklauso nuo srities. Praktiškai laiko ir duomenų priklausomybė vaidina didžiausią vaidmenį.

2.1.5. Modeliavimas Ptolemy II sistemoje

Ptolemy II yra grafinė modeliavimo ir projektavimo sistema įgyvendinanti projektavimą naudojant aktorius metodologiją ir hierarchinį heterogeniškumą.

Baziniai kūrimo blokai Ptolemy II modelyje yra atominiai aktoriai (angl. *atomic actors*). Atominiai aktoriai apjungia skaičiavimus nuo paprasčiausių aritmetinių operacijų iki sudėtingų (pvz, Furje transformacijos). Aktoriai turi įvesties ir išvesties jungtis, kuriuos sujungus gaunamas norimas modelis. Aktorių kompozicija prižiūri valdiklis (angl. *director*), kuris nusako skaičiavimo modelį. Ptolemy II sistemoje skaičiavimo modelis taip pat vadinamas *sritimi* (angl. *domain*). Valdiklis gali valdyti aktorių veiklas per *vykdomąją* (angl. *executable*) sąsają. Komunikacijos tarp aktorių mechanizmas išspręstas naudojant įvesties jungtyse esančiais imtuvais (angl. *receiver*). Valdiklis kartu su visais imtuvais apibrėžia rėminę konstrukciją. Skaičiavimo modelio valdiklis ir imtuvai turi sutapti.



Šaltinis: LIU, Jie; EKER, Johan; JANNECK, Jörn W.; LIU, Xiaojun; LEE, Edward A. Actor-Oriented Control System Design: A Responsible Framework Perspective. p. 254

2.5 pav. Ptolemy II hierarchinis modelis

Modelis, sudarytas iš hierarchiškai sukomponuotų aktorių, pateiktas 2.5 paveiksle. Atominiai aktoriai (*A1* ir *B1*) išdėstyti hierarchijos apačioje. Kombinuoti aktoriai (*A2*) apjungia kitus aktorius, todėl hierarchija yra atsitiktinai sujungta. Hierarchinis heterogeniškumas pasiekiamas parenkant skirtingus valdiklius skirtinguose modelio lygmenyse.

Kai yra įmanoma, valdikliai ir imtuvai įgyvendinti kaip atsakingos rėminės konstrukcijos. Tiksli atominių aktorių reakcija pasiekiama apibrėžiant daugiafazę aktorių veikseną. Nors aktoriai turi pagrindinį funkcionalumą aprašytą metode *fire()*, atsakingiems trigeriams patikrinti imtuvų pagalba naudojamas metodas *prefire()*. Įgyvendinant trigerio modelį (2.3 pav.), valdiklis pirmiausia iškviečia metodą *prefire()*, tada sužadina aktorių, jeigu *prefire()* metodas grąžina reikšmę *true*. Sudėtinė tiksli

reakcija pasiekiama per iteracijos sąvoką, apibrėžtą kiekvienoje srityje. Kombinuotų aktorių vykdymas sužadina susijusio posistemio iteraciją ir minėto aktoriaus *prefire()* metodas perduoda lokaliam valdikliui signalą, pagal kurį suskaičiuoja ar iteracija gali būti užbaigta.

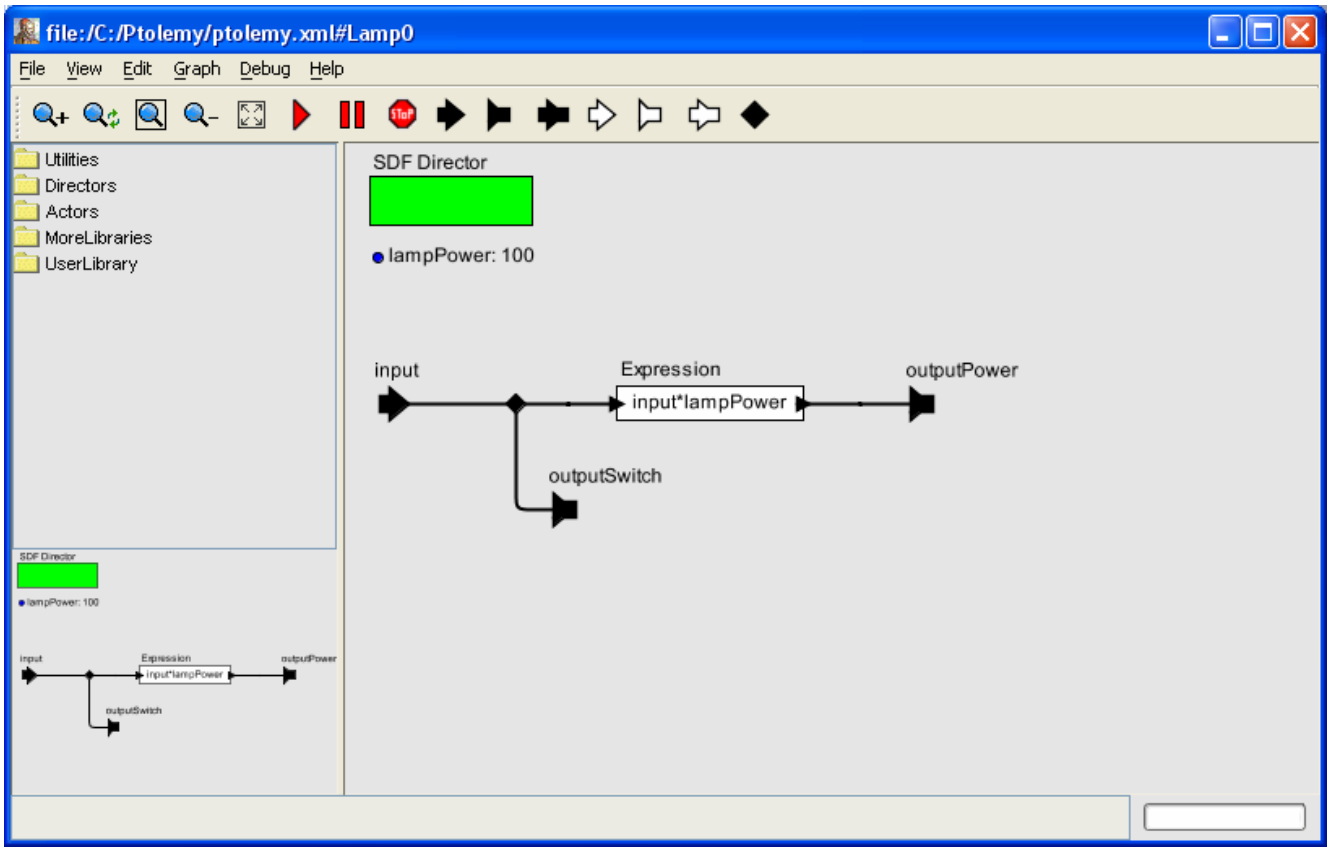
2.2. Būsto inžinerinių sistemų modeliavimo metodika

Būsto inžinerinių sistemų modeliavimo metodika susideda iš 2 pagrindinių etapų: konkretaus inžinerinio mazgo prototipo aprašymo Ptolemy II sistemoje bei projektuotojo inžinerinių mazgų patalpinimo plane.

2.2.1. Inžinerinio mazgo aprašas Ptolemy II sistemoje

Kuriant naują gyvenamojo namo inžinerinį mazgą, reikia apibrėžti jo vidinę sandarą. Ptolemy II sistemoje pasitelkiant aktorius galima sudaryti modelį naudojant formalias specifikacijas arba būsenų automatą. Kiekvienas mazgo modelis turi turėti aiškiai apibrėžtas įėjimo bei išėjimo jungtis. Kai kuriems mazgams gali būti nurodomi vidiniai parametrai. Tai priklauso nuo to, kaip yra organizuotas konkretaus inžinerinio mazgo modelis.

Apšvietimo lempos modelis pavaizduotas 2.6 paveiksle. Jo jungtys aprašytos 2.1 lentelėje.



2.6 pav. Apšvietimo lempos modelis

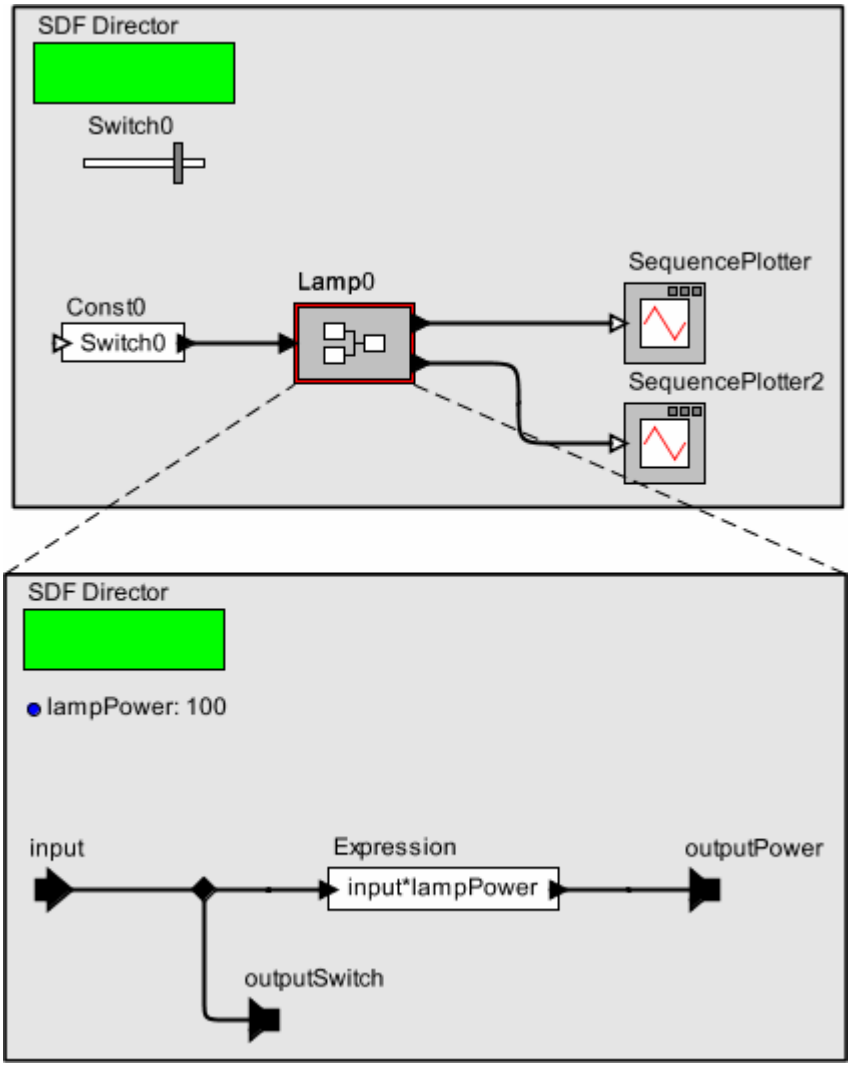
Šis modelis turi 2 išėjimo jungtis: viena tiesiogiai sujungta su įėjimo jungtimi ir atvaizduoja įėjimo signalą, antra sujungta su aktoriumi „Expression“, kuris įėjimo signalą padaugina iš apšvietimo lemputės galios (*input*lampPower*) ir suformuoja lempos sunaudojimo galingumo signalą.

2.1 lentelė

Apšvietimo lempos modelio jungtys

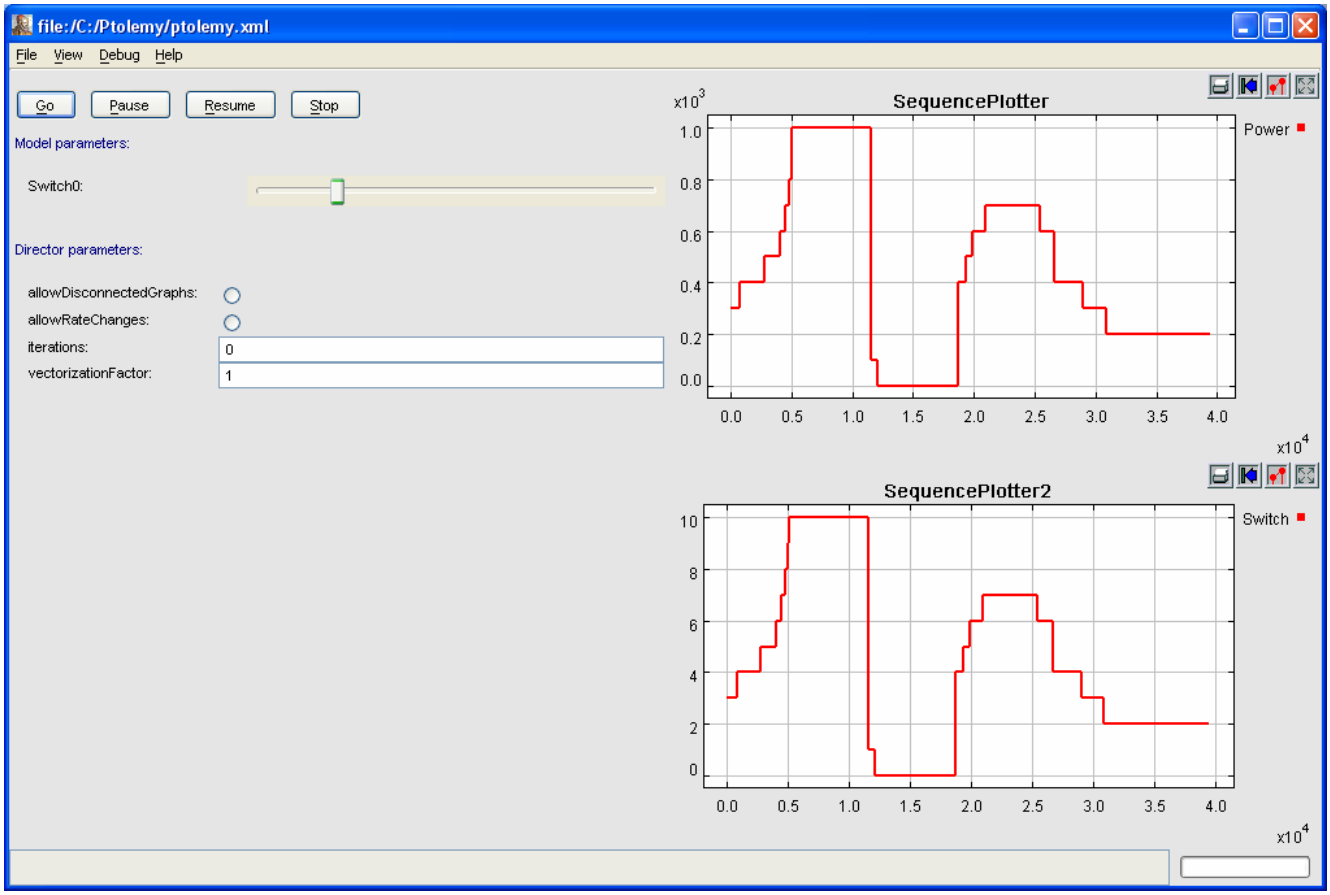
Pavadinimas	Tipas	Aprašymas
input	įvedimo	Jungiklio padėties įvedimo jungtis
outputSwitch	išvedimo	Jungiklio padėties išvedimo jungtis
outputPower	išvedimo	Sunaudojamo galingumo išvedimo jungtis

Nagrinėjamas modelis sudarytas iš 3 jungčių (*input*, *outputSwitch*, *outputPower*), 1 parametro (*lampPower*), 1 valdiklio (*SDF Director*) ir 1 aktoriaus (*Expression*). Esant modelyje keliems elementams patogu apjungti vieno modelio komponentus į kombinuotą aktorių (2.7 pav.).



2.7 pav. Kombinuotas aktorius

Prie kombinuoto aktoriaus (*Lamp0*) prijungę aktorių jungiklis (*Const0*), kuris perduoda parametro *Switch0* reikšmę bei išvedamų signalų atvaizdavimo aktorius (*SequencePlotter*, *SequencePlotter2*) galima vykdyti sukurtą modelį. Vykdymo metu šliaužiklio *Switch0* pagalba galima keisti mygtuko padėtį, taip įtakojant išvedamus signalus. Modelio vykdymo rezultatai pateikti 2.8 paveiksle.



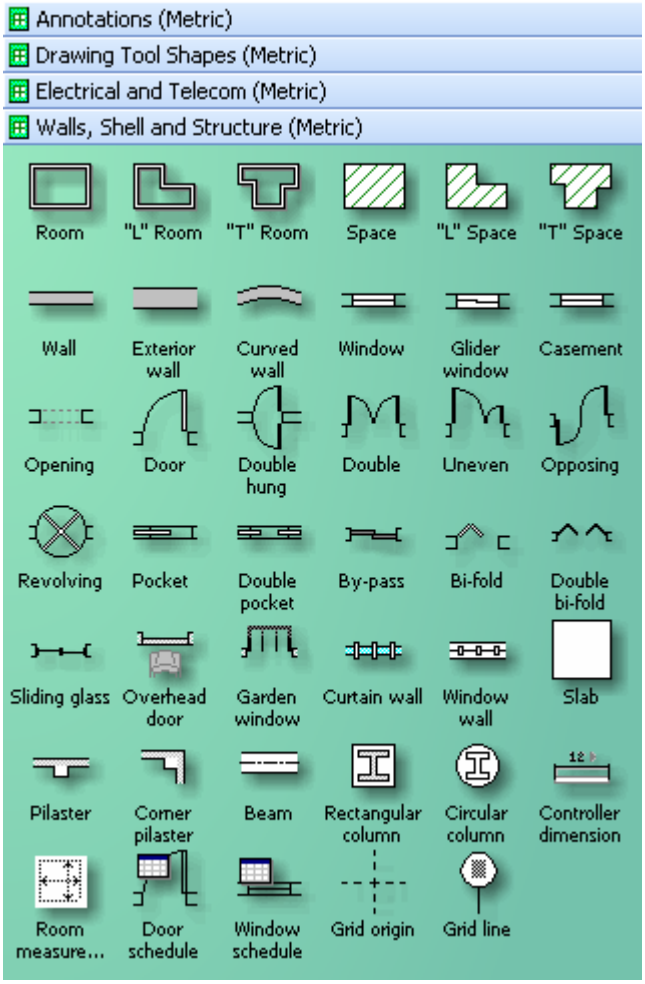
2.8 pav. Apšvietimo lempos modeliavimo rezultatai

Iš gautų modeliavimo rezultatų matome, kad lempos suvartojamas galingumas tiesiogiai proporcingas jungiklio padėčiai (*input*lampPower*). Iš to galima teigti, kad modelis dirba korektiškai.

Jungiklis ir kombinuotas aktorius sudaro būsto apšvietimo lempos inžinerinį mazgą. Išsaugoję šį modelį XML formatu, gauname struktūrizuotą aprašą, kurį naudosime sekančiame modeliavimo metodikos žingsnyje.

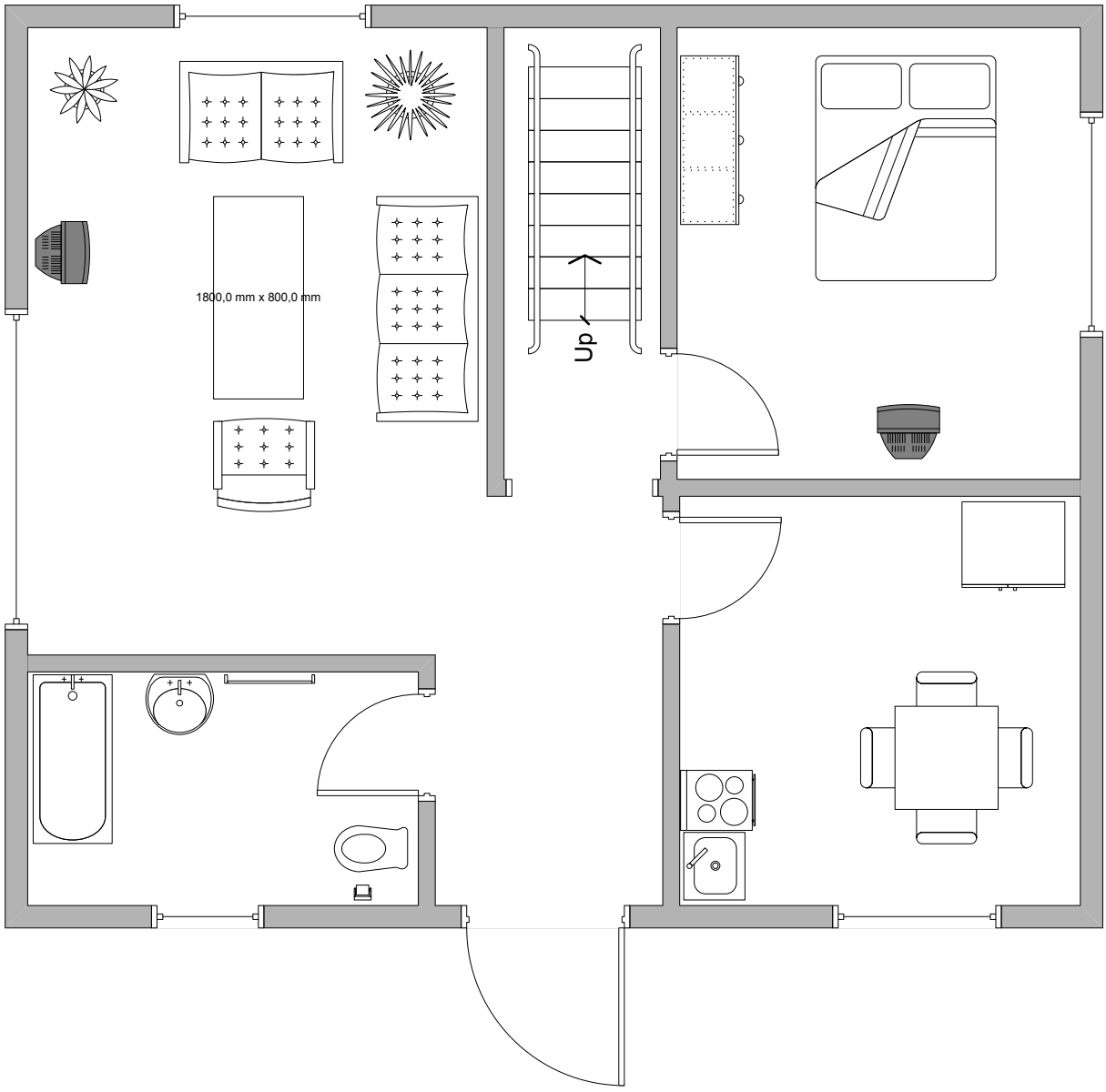
2.2.2. Inžinerinio mazgo panaudojimas MS Visio programoje

Gyvenamojo namo planui kurti naudojamas Microsoft Visio paketas. Šis paketas pasirinktas todėl, kad jame jau yra integruotas gyvenamųjų patalpų planavimo posistemis. Šiame posistemyje objektai suskirstyti į bibliotekas. Kiekvienas objektas turi savo grafinį žymenį, kurio pagalba įterpiamas objektas į kuriamą planą (2.9 pav.).



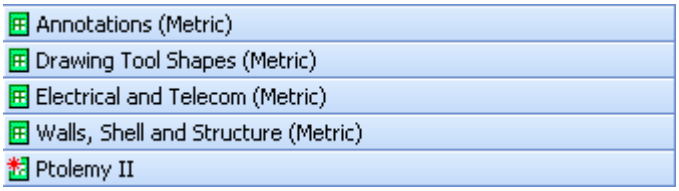
2.9 pav. Microsoft Visio gyvenamųjų patalpų planavimo įrankių bibliotekos

Naudojant reikiamus bibliotekos komponentus (sienos, langai, durys, laiptai, baldai, namų apyvokos daiktai ir kt.) suformuojamas norimo būsto planas (2.10 pav.).



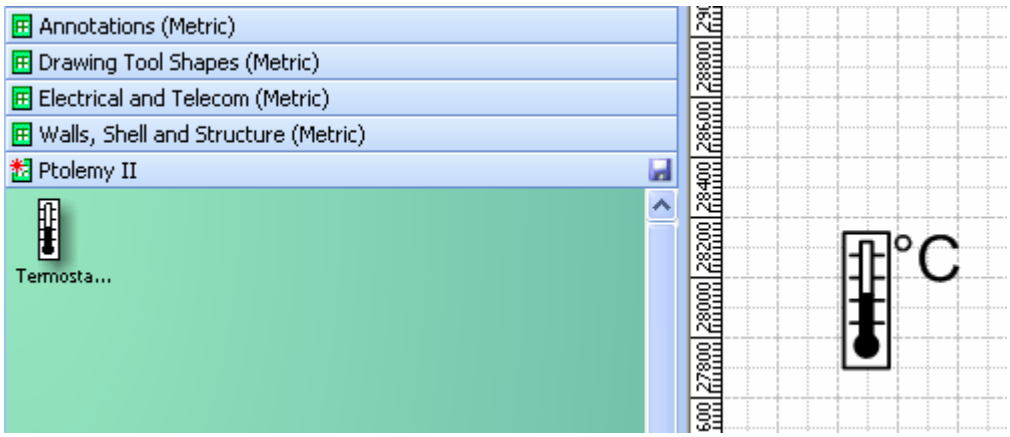
2.10 pav. Namo planas naudojant Microsoft Visio

Minėtas gyvenamųjų patalpų planavimo posistemis papildomas nauja biblioteka – „Ptolemy II“ (2.11 pav.).



2.11 pav. Naujos komponentų bibliotekos sukūrimas

Sukurta nauja biblioteka užpildoma komponentais, kurie vėliau, panaudojus XML transformaciją, bus perkelti į Ptolemy II sistemą. Kaip jau minėta, kiekvienas komponentas turi savo grafinį žymenį bibliotekoje (2.12 pav.).

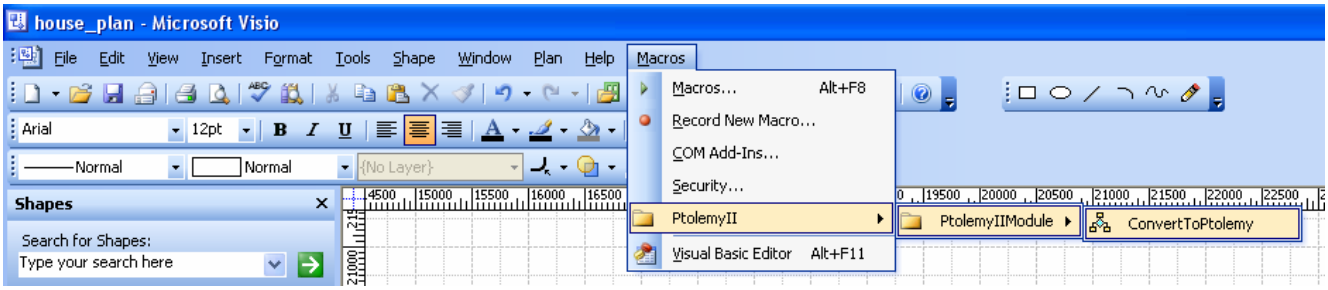


2.12 pav. Komponento grafinis žymuo (kairėje) ir tikrasis vaizdas (dešinėje)

Visual Basic programavimo kalba parašytas paprogramis, kuris susieja MS Visio grafinį žymenį su Ptolemy II sistemoje suformuotu inžinerinio mazgo modeliu. Minėtas paprogramis veikia tokiu principu:

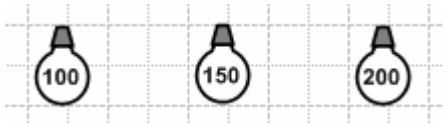
- imamas MS Visio projekto komponentas;
- lyginamas komponento pavadinimas su paprogramėje suformuotu komponentų pavadinimų sąrašu;
- radus aprašytą komponentą įterpiamas į Ptolemy II sistemos modelį komponento XML aprašas;
- kartojamas ciklas iš pradžių su kitu komponentu.

Paprogramio programos tekstas pateiktas 1 priede. Perėjimo iš MS Visio programos į Ptolemy II paprogramis iškviečiamas naudojant meniu punktus: Macros → PtolemyII → PtolemyIIModule → ConvertToPtolemy (2.13 pav.).



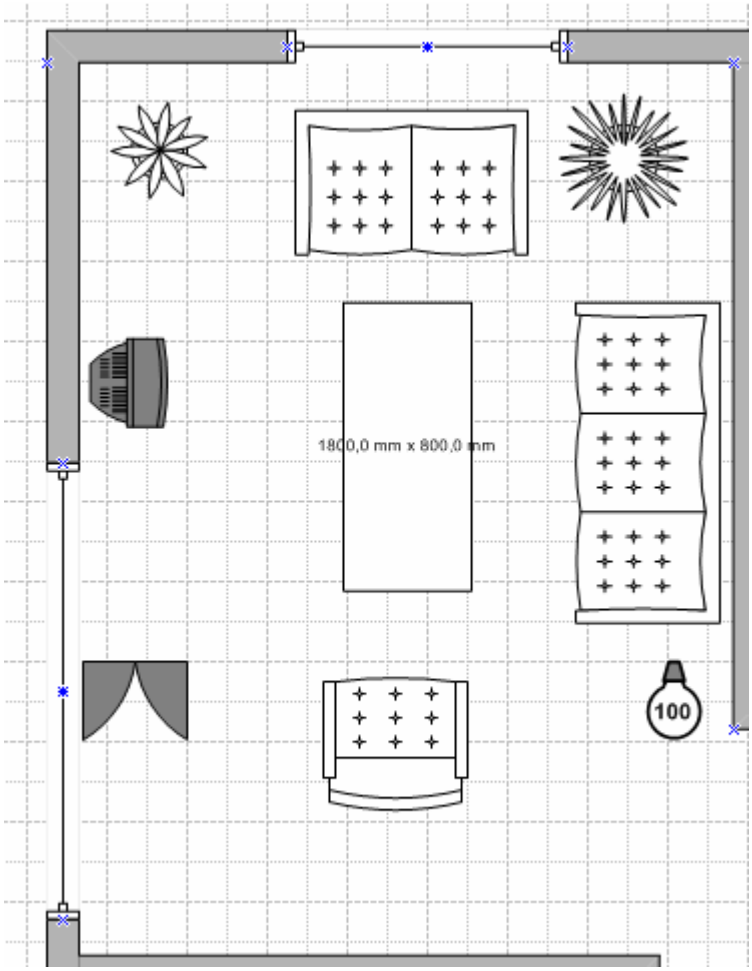
2.13 pav. Pervedimo paprogramio iškvietimas

Kai kurie MS Visio komponentai turi parametrus, kuriuos galima perduoti į Ptolemy II modelį (2.14 pav.).



2.14 pav. Komponentai su skirtingomis parametrų reikšmėmis

2.15 paveiksle pateikto namo plano pervedimas į Ptolemy II sistemą pavaizduotas 2.7 paveiksle.



2.15 pav. Suprojektuotas planas su Ptolemy II sistemos komponentais

Sudarant būsto inžinerinių mazgų modelius pagrindinis uždavinys yra suskaidyti valdomą sistemą į daug lengvai valdomų ir į konkrečią sritį orientuotų posistemių, taip, kad projektuotojai galėtų efektyviai suskaidyti ir išspręsti problemą. Į komponentus orientuotos projektavimo metodologijos komponentai turi griežtai nusakytas sąsajas. Kiekvienas komponentas apima tam tikrą funkcionalumą, tokį kaip skaičiavimas ir komunikacija.

Ptolemy II sistemoje komponentas yra aktorius – tai objektas, atliekantis parametrizuotus veiksmai su įvesties duomenimis tam, kad gauti išeities duomenis. Aibė aktorių elgsenų nėra gerai apibrėžta be koordinuojančio modelio. Aplinka, kuriai priklauso aktoriai ir kuri apibrėžia tarpusavio sąveikas, vadinama rėmine konstrukcija.

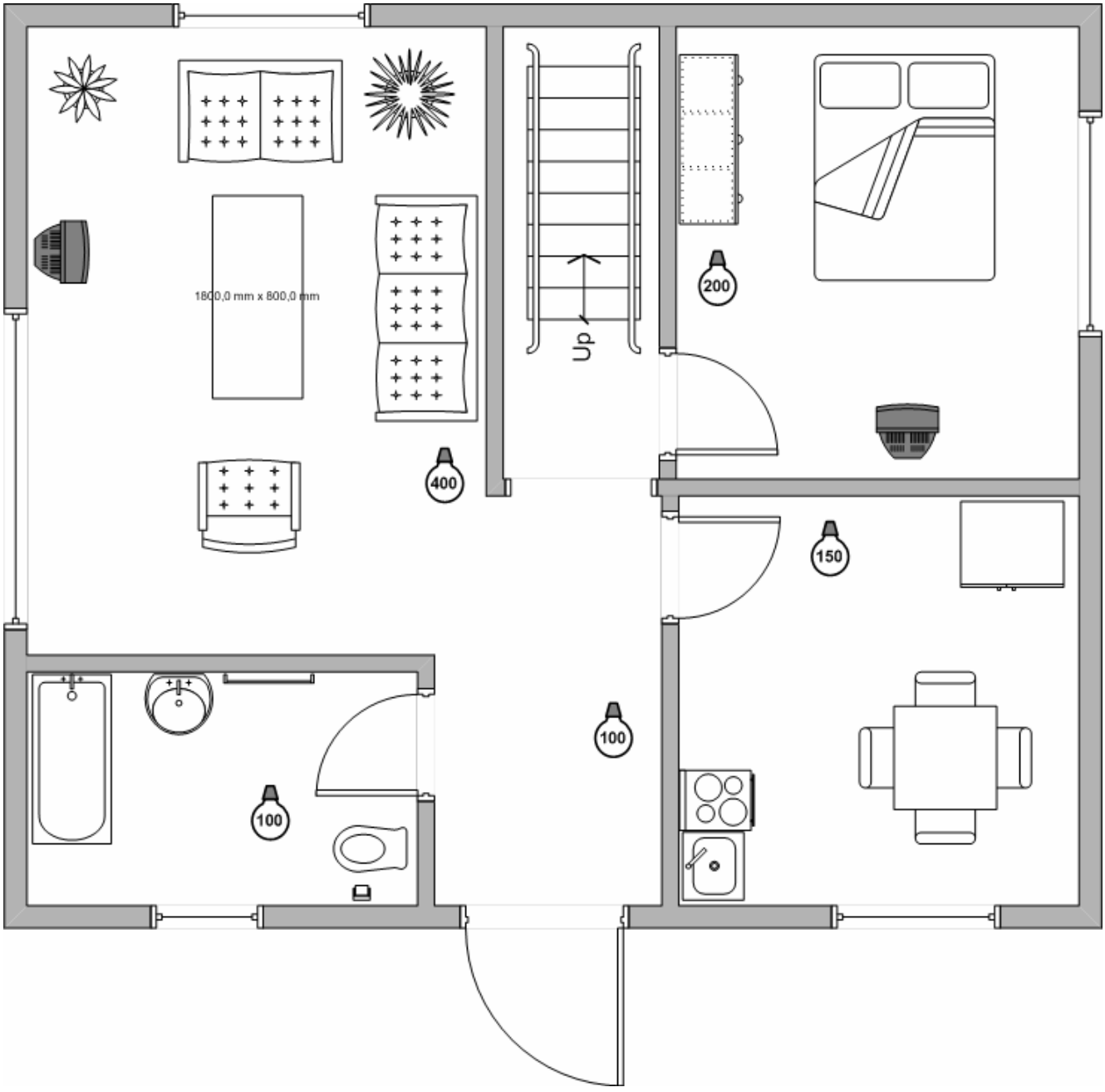
Būsto inžinerinių sistemų modeliavimo metodika susideda iš 2 pagrindinių etapų: konkretaus inžinerinio mazgo prototipo aprašymo Ptolemy II sistemoje bei projektuotojo inžinerinių mazgų patalpinimo plane.

Gyvenamojo namo planui kurti naudojamas Microsoft Visio paketas. Šis paketas pasirinktas todėl, kad jame jau yra integruotas gyvenamųjų patalpų planavimo posistemis.

3. BŪSTO INŽINERINIŲ MAZGŲ MODELIAVIMO EKSPERIMENTAI

3.1. Apšvietimo sistemos modeliavimas

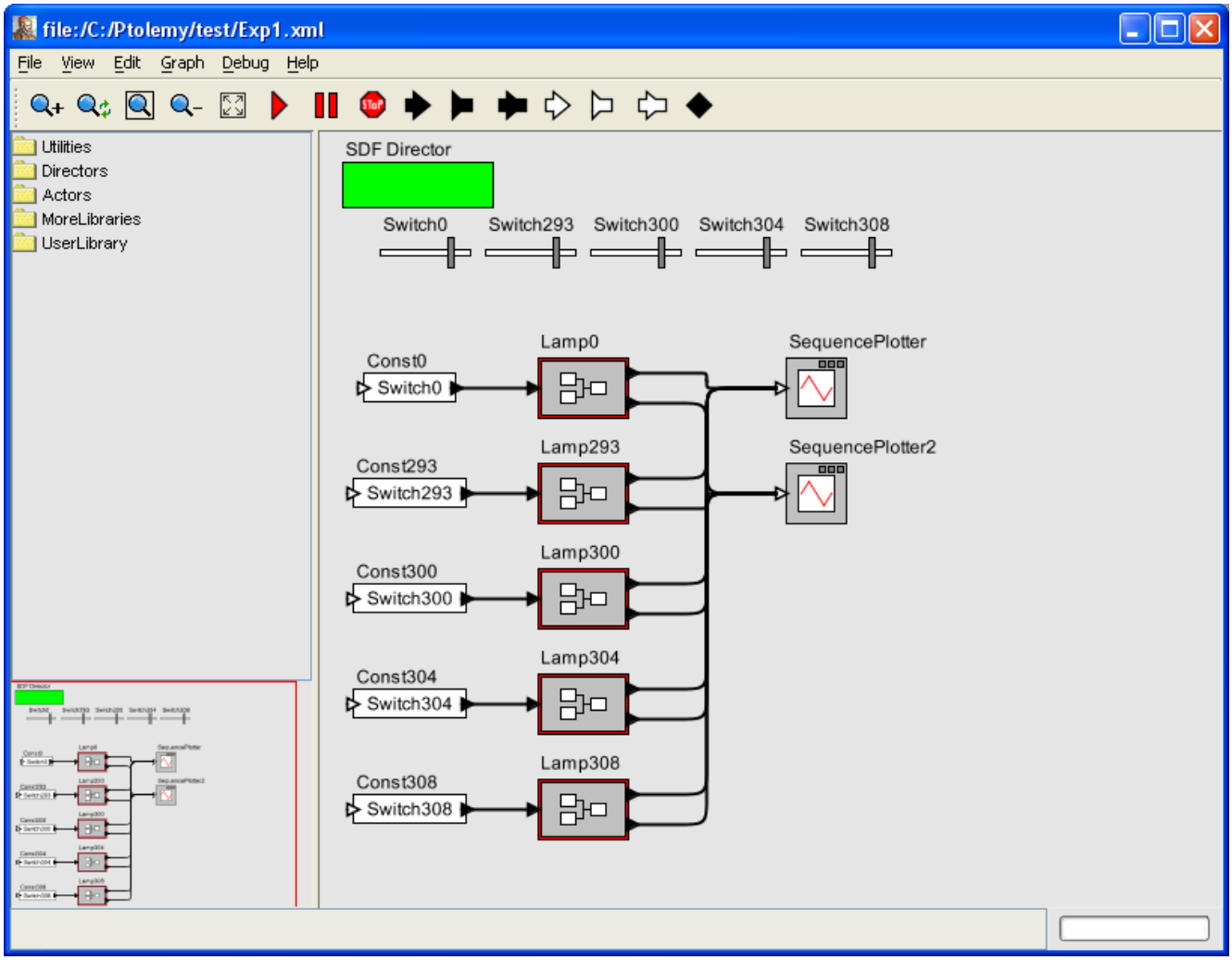
Turime suprojektuotą namo planą. Kadangi jau yra sukurtas apšvietimo lempos modelis, MS Visio programos pagalba sudedame reikiamus komponentus (apšvietimo lempas) ant plano. Apšvietimo lempa turi savo parametrą – suvartojamą galingumą, kurį galima priskirti skirtingą kiekvienai lempai (3.1 pav.).



3.1 pav. Namo planas su įterptais apšvietimo elementais

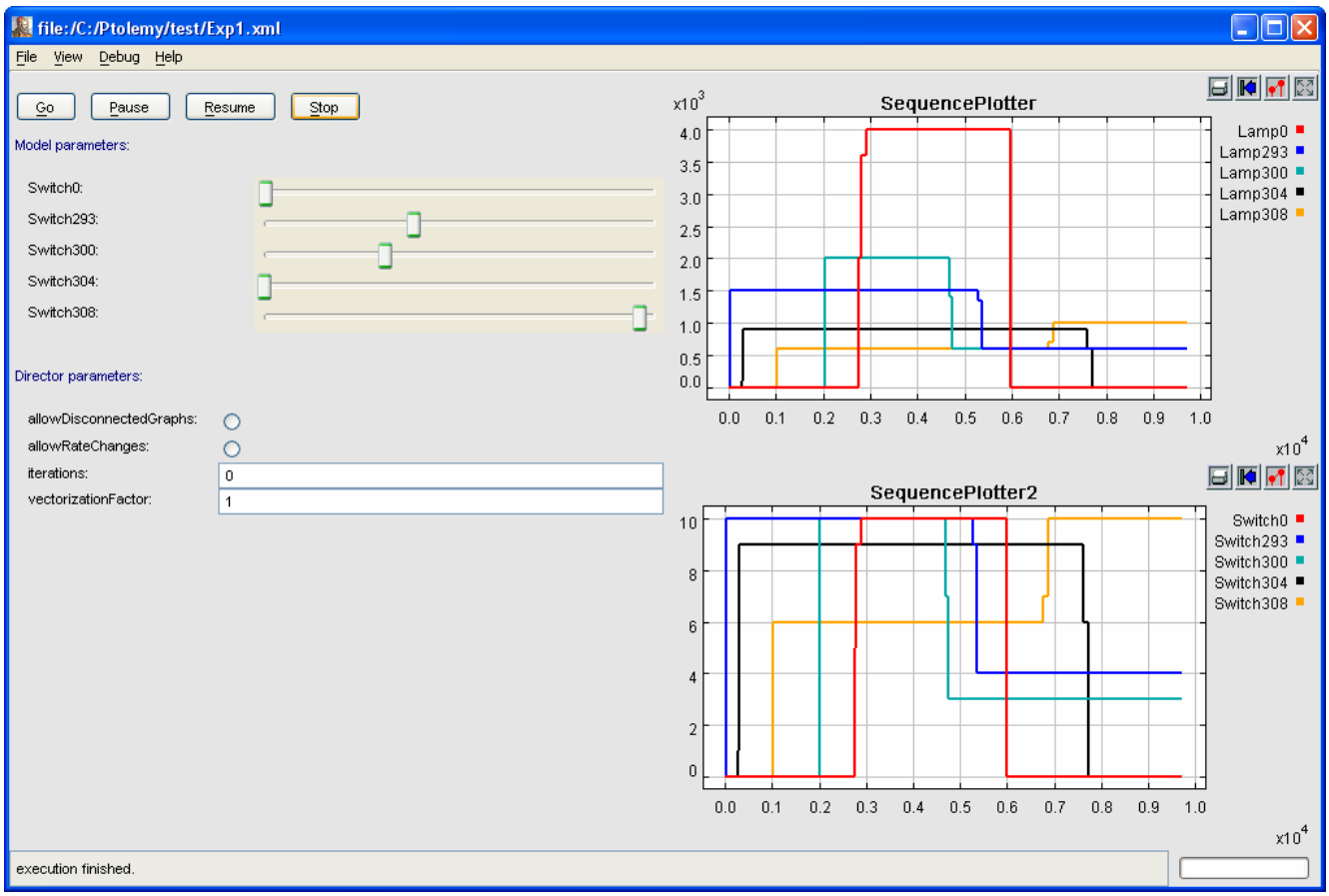
Duotame plane panaudotos 5 apšvietimo lempos su nurodytais skirtingais galingumo nustatymais. Įvykužius pervedimo procedūrą (*ConvertToPtolemy*), suformuojamas XML failas,

kuriame aprašytas Ptolemy II sistemos modelis. Minėtą failą atsidarius su Ptolemy II sistema gauname modelį, pavaizduotą 3.2 paveiksle.



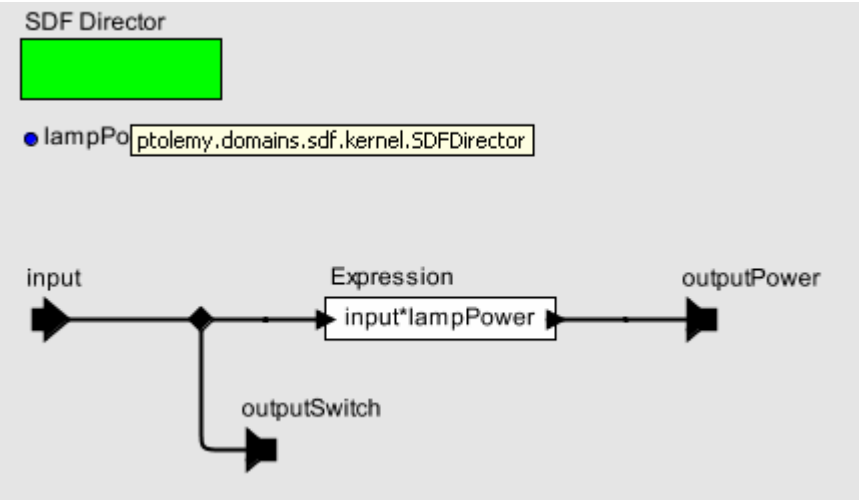
3.2 pav. Sugeneruotas Ptolemy II modelis

Gautą modelį modeliuojame šliaužiklių pagalba priskirdami skirtingas reikšmes kiekvienai apšvietimo lempai. Gauti rezultatai pateikti 3.3 paveiksle.



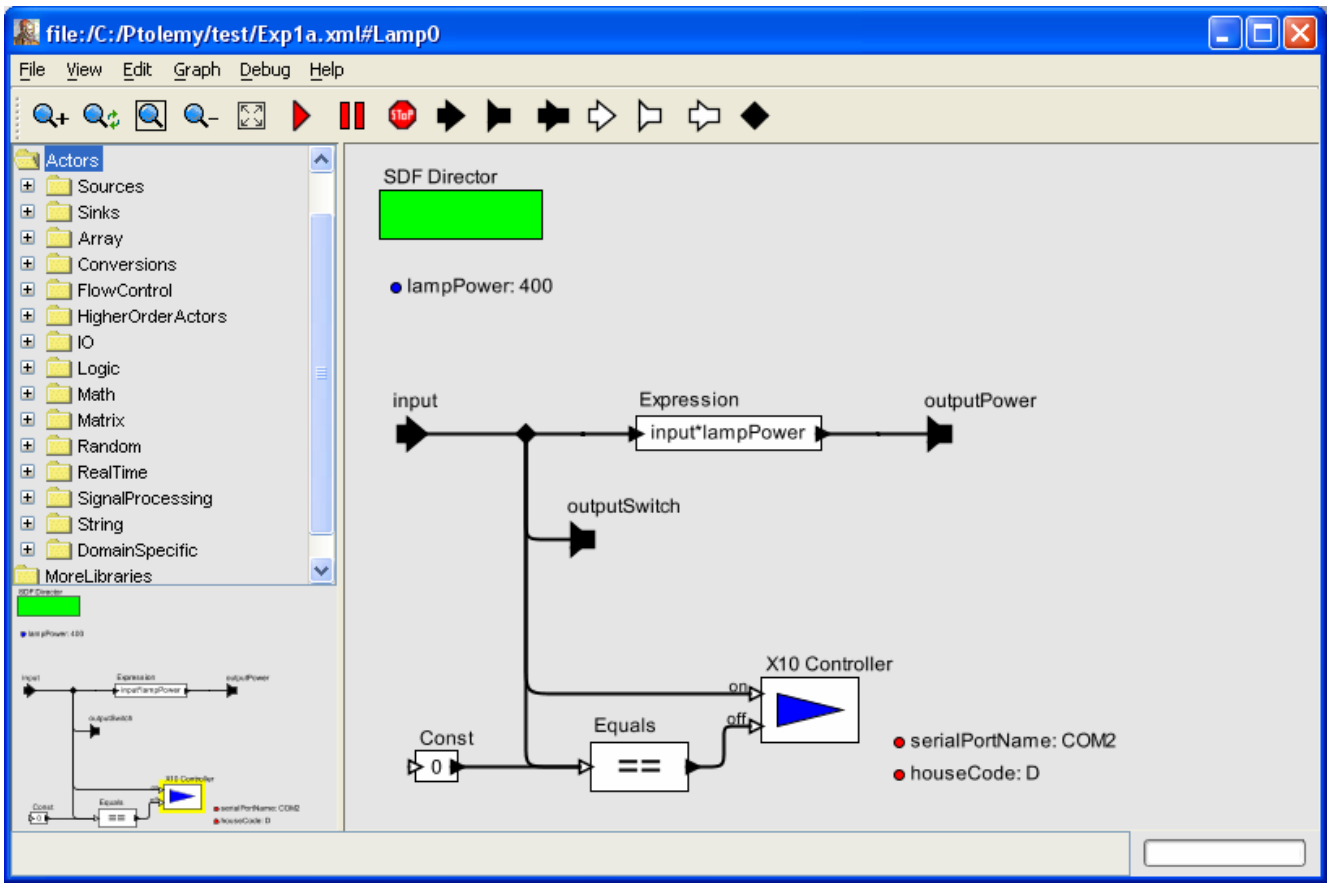
3.3 pav. Apšvietimo lempų modeliavimo rezultatai

Kaip matosi iš 3.3 paveikslo lempų sunaudojamas galingumas tiesiogiai proporcingas jungiklio padėčiai. Pats lempos modelis (*Lamp0*, *Lamp293* ir kt.) sudarytas iš aktoriaus „Expression“ (3.4 pav.).



3.4 pav. Apšvietimo lempos modelis

Šį lempos modelį galima patobulinti pridedant aktorių „X10 Controller“. Šio komponento dėka galima valdyti gyvenamojo būsto įrenginį X10 protokolu, kuriam suteiktas kodas (angl. *houseCode*). Komunikacija tarp kompiuterio ir X10 protokolu valdomo įtaiso vykdoma nuoseklaus prievado (COM) pagalba (3.5 pav.).



3.5 pav. Patobulintas apšvietimo lempos modelis

Aktorius „X10 Controler“ turi dvi įėjimo jungtis: *on* ir *off*. Kai jungiklio pozicija nelygi „0“ paduodamas signalas į jungtį „on“ (įjungiamas įtaisas). Kai jungiklio pozicija lygi „0“, suformuojamas signalas į jungtį „off“ (įtaisas išjungiamas).

3.2. Naujo inžinerinio mazgo kūrimas

Apšvietimo lempos modelis sugeneruoja sunaudojamą lemputės galią pagal mygtuko poziciją. Mums yra įdomu, kiek sunaudoja galios visos lemputės bendrai. Tam reikia sukurti elektros skaitiklio modelį. Elektros skaitikliui sudaryti reikia atlikti šiuos žingsnius:

- Ptolemy II sistemos aktorių pagalba sudaryti modelį;
- aprašyti įėjimo bei išėjimo jungtis;
- modelio testavimas.

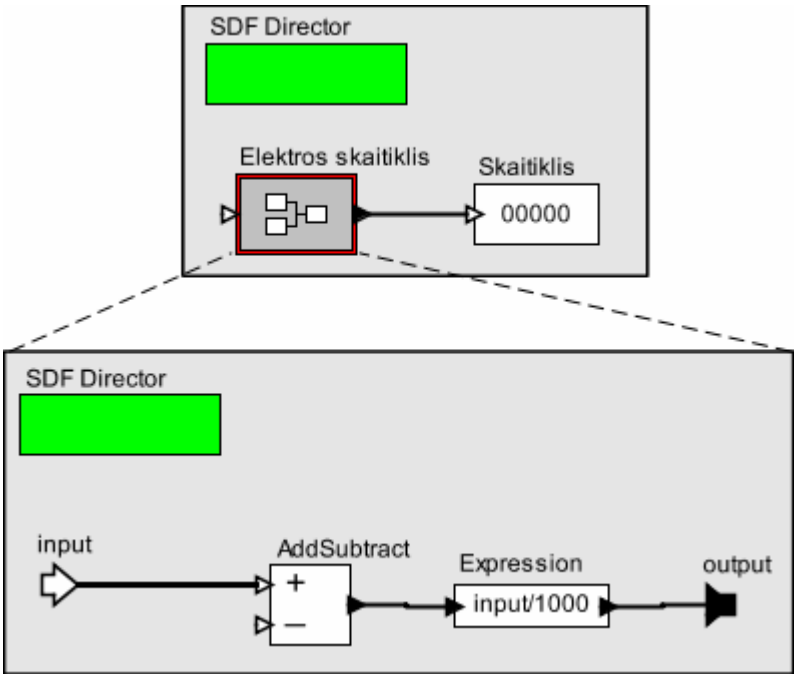
Elektros skaitiklis tiesiog sumuoja visų elektros įtaisų suvartojamą galią. Taigi, modelyje bus naudojamas „AddSubtract“ aktorius. Kadangi apšvietimo lemputės suvartojama galia išreikšta vatais (W), tai minėto aktoriaus (sumatorius) gautą reikšme reikia padalinti iš 1000. Modelio jungtys pateiktos 3.1 lentelėje.

3.1 lentelė

Elektros skaitiklio modelio jungtys

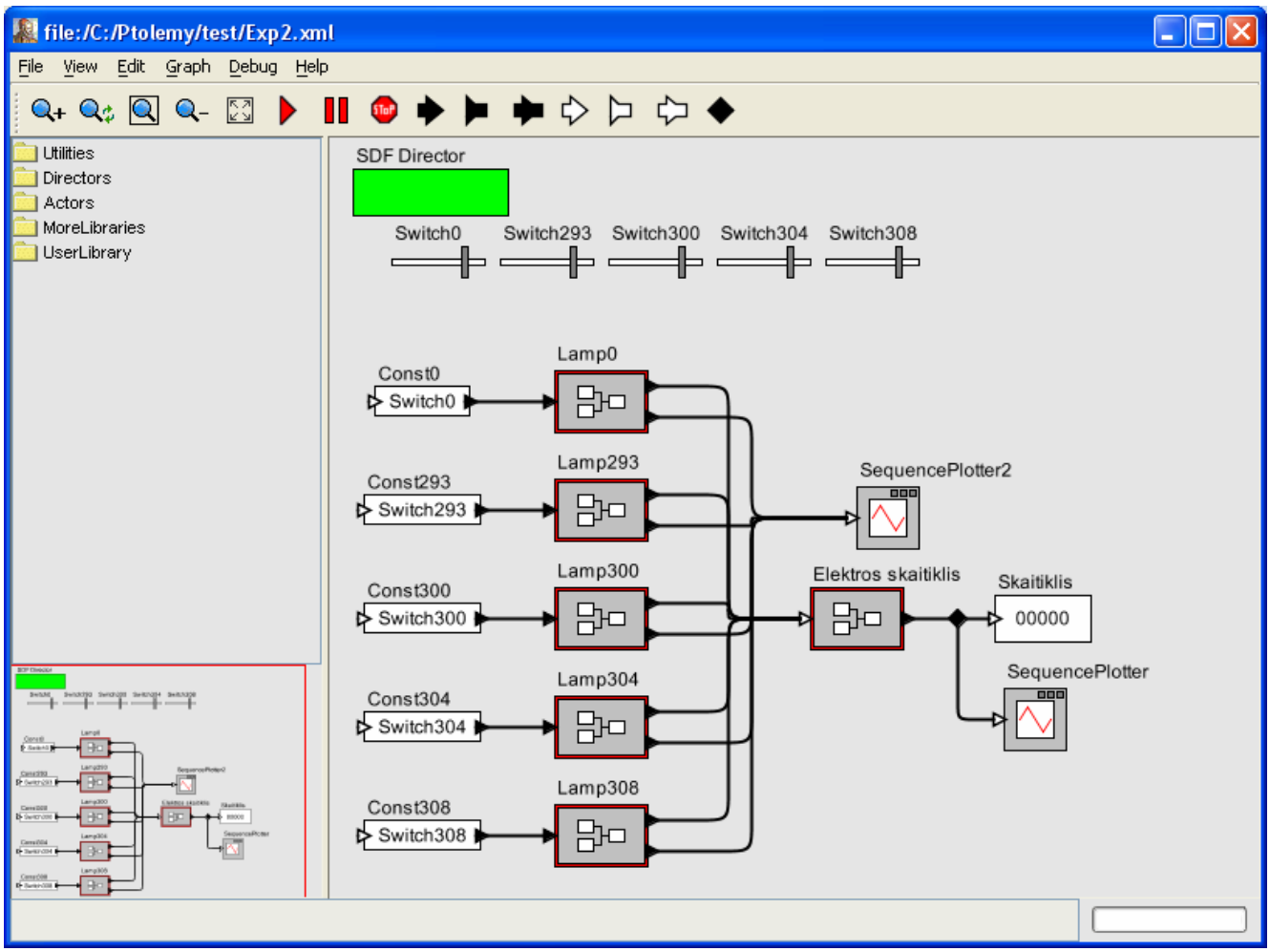
Pavadinimas	Tipas	Aprašymas
input	įvedimo	Jungiklio padėties įvedimo jungtis
output	išvedimo	Sunaudojamo galingumo išvedimo jungtis

Sudarytas Ptolemy II elektros skaitiklio modelis pavaizduotas 3.6 paveiksle.



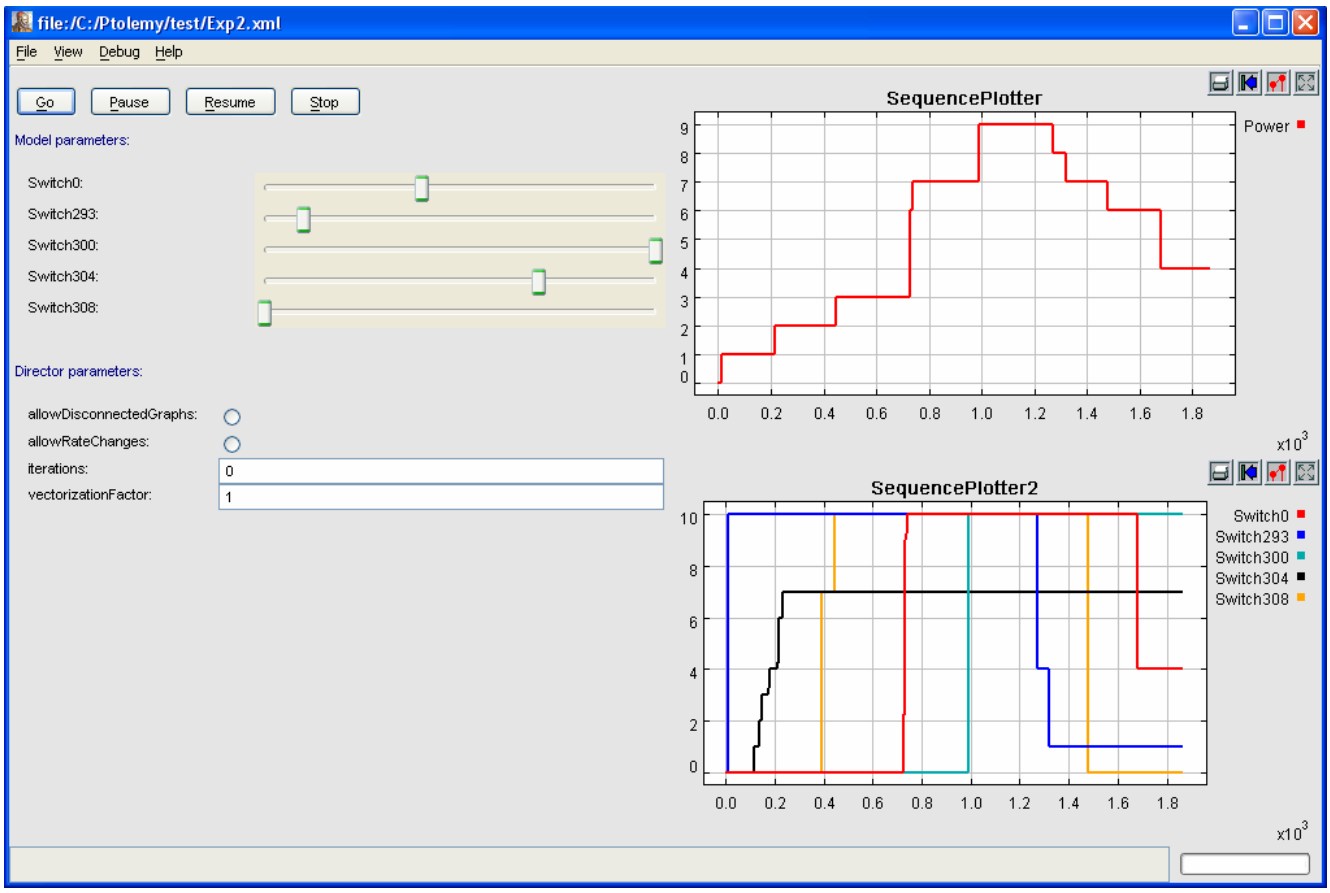
3.6 pav. Elektros skaitiklio modelis

Sudaryto modelio XML išraiška pateikta 2 priede. Apjungus apšvietimo lempų ir elektros skaitiklio modelius MS Visio programoje, gaunamas pilnas apšvietimo lempų modelis Ptolemy II sistemoje (3.7 pav.).



3.7 pav. Apšvietimo lempų ir elektros skaitiklio modelis

Modeliavimo rezultatai pateikti 3.8 paveiksle. Jungiklių pagalba keičiamas apšvietimo lempų suvartojamas galingumas pavaizduotas aktoriaus „SequencePlotter2“ pagalba. Visų apšvietimo lempų suvartojamas bendras galingumas pavaizduotas aktoriaus „SequencePlotter“ pagalba.



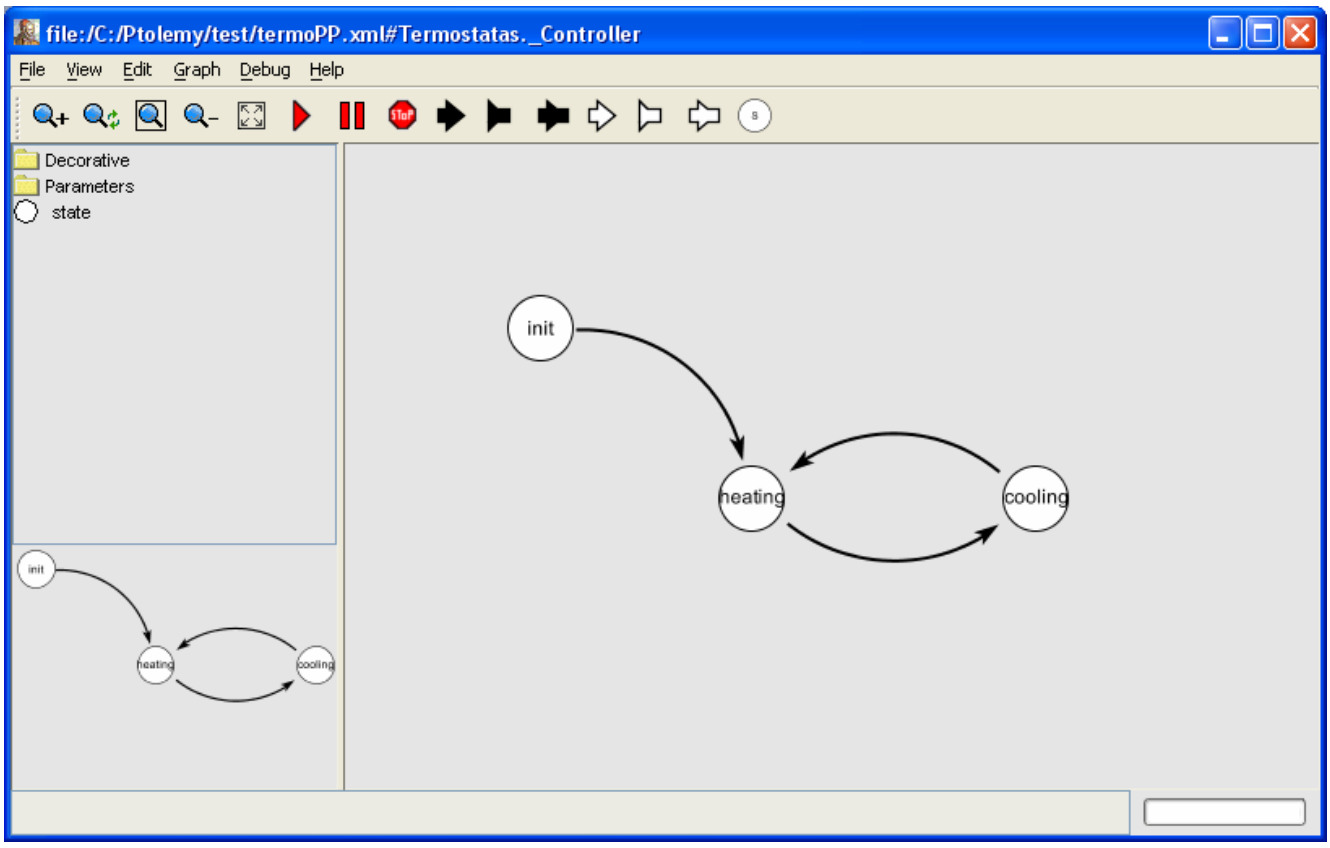
3.8 pav. Apšvietimo lempų ir elektros skaitiklio modeliavimo rezultatai

Sudarytas modelis veikia korektiškai, t.y. didinant vienos apšvietimo lemputės galią didėja bendras galios suvartojimas, mažinant vienos lemputės galią – mažėja.

3.3. Termostato modeliavimas

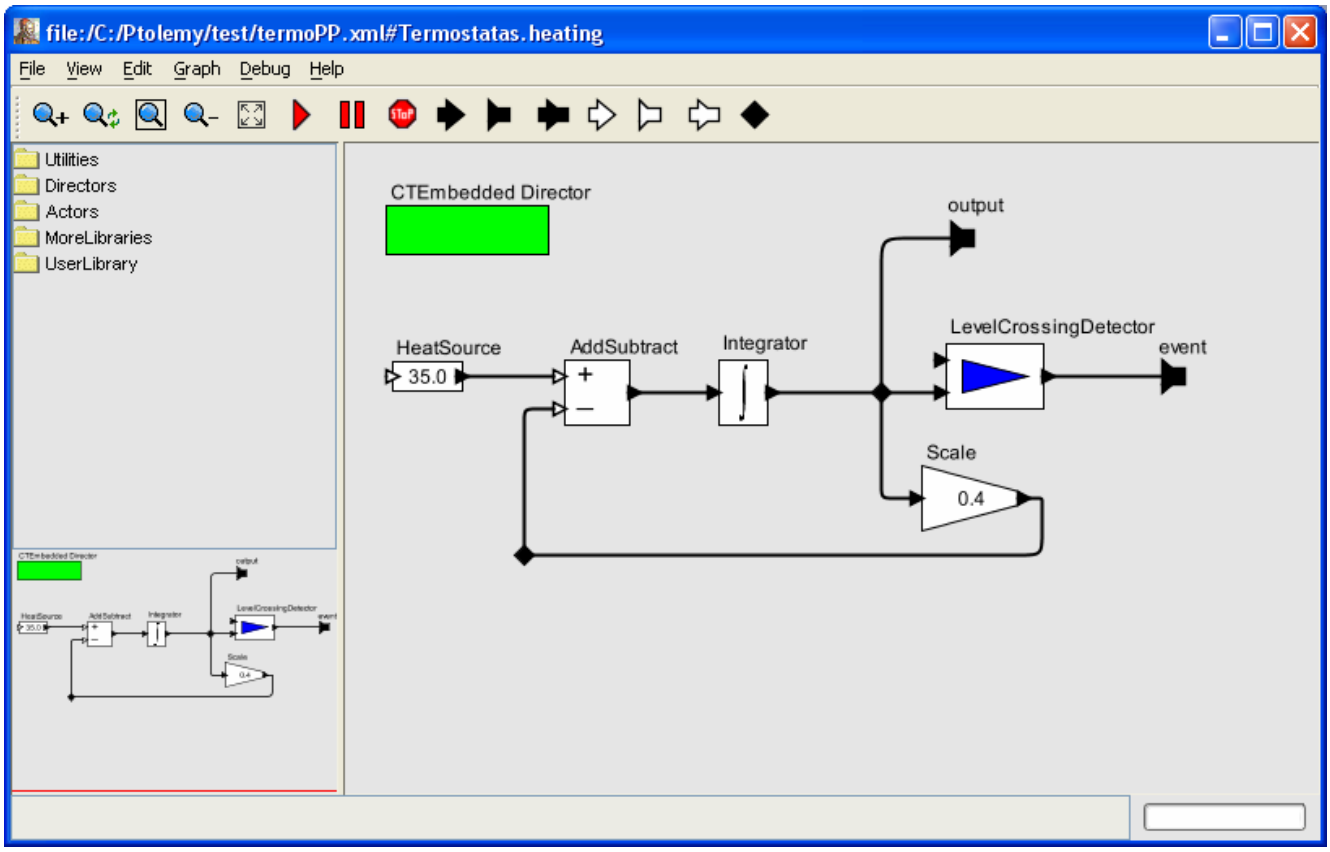
Anksčiau aprašyti modeliai naudojo formalias išraiškas. Dabar sudarysime termostato modelį, kurio veikimas pagrįstas baigtinių būsenų automatu. Termostato modelis susideda iš dviejų ciklų: kaitinimo ir vėsimo. Apibrėžiamos šios būsenos (3.9 pav.):

- pradinė (*init*);
- kaitinimo (*heating*);
- vėsimo (*cooling*).



3.9 pav. Termostato būsenos

Kaitinimo ir vėsimo būsenas atitinka modelis, pateiktas 3.10 paveiksle.

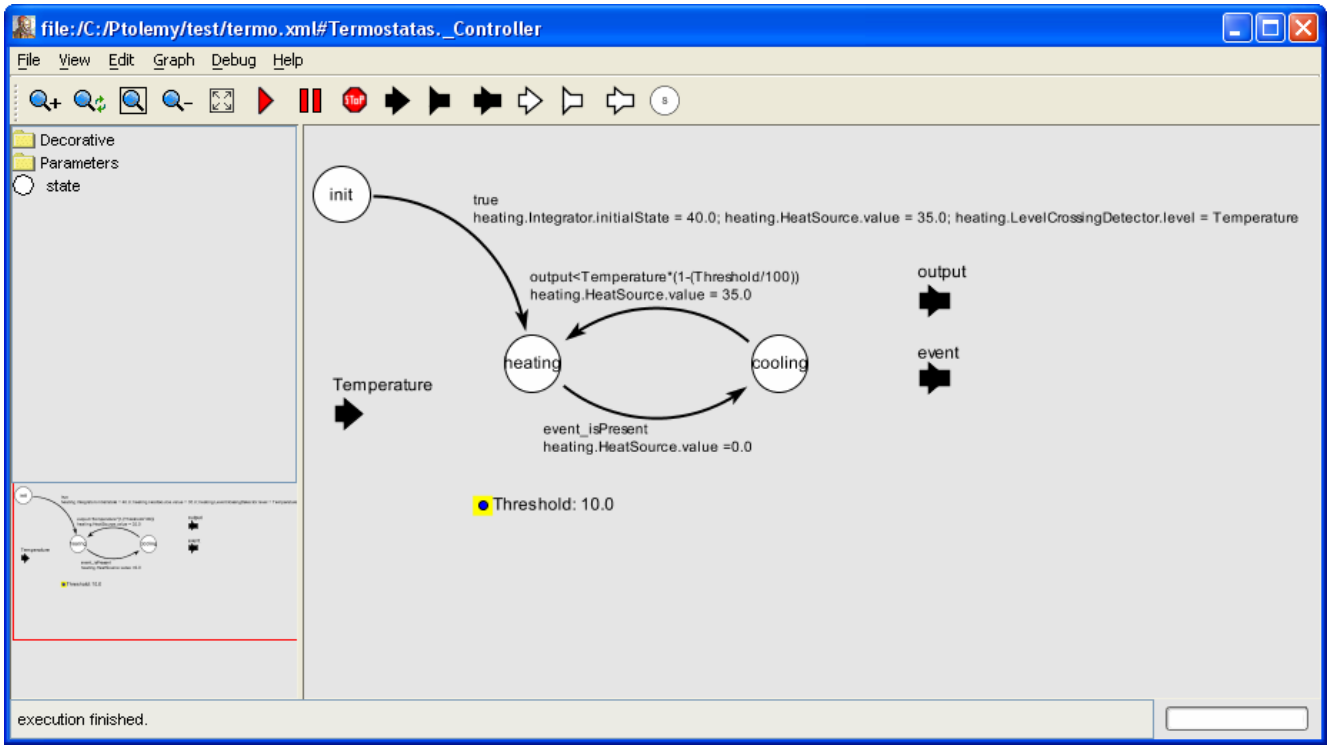


3.10 pav. Kaitinimo ir vėsimo būsenų modelis

Iš pradinės būsenos pereinama prie kaitinimo priskiriant aktoriui „HeatSource“ (kaitinimo šaltinis) reikšmę 40, aktoriui „Integrator“ (integratorius) reikšmę 35 bei aktoriui „LevelCrossingDetector“ (lygių pasikeitimo indikatorius) reikšmę 80. Pradinei būsenai priskiriama perėjimo sąlyga reikšmė True, t.y. ši būsena pereina į kitą būseną, jeigu tenkinama sąlyga.

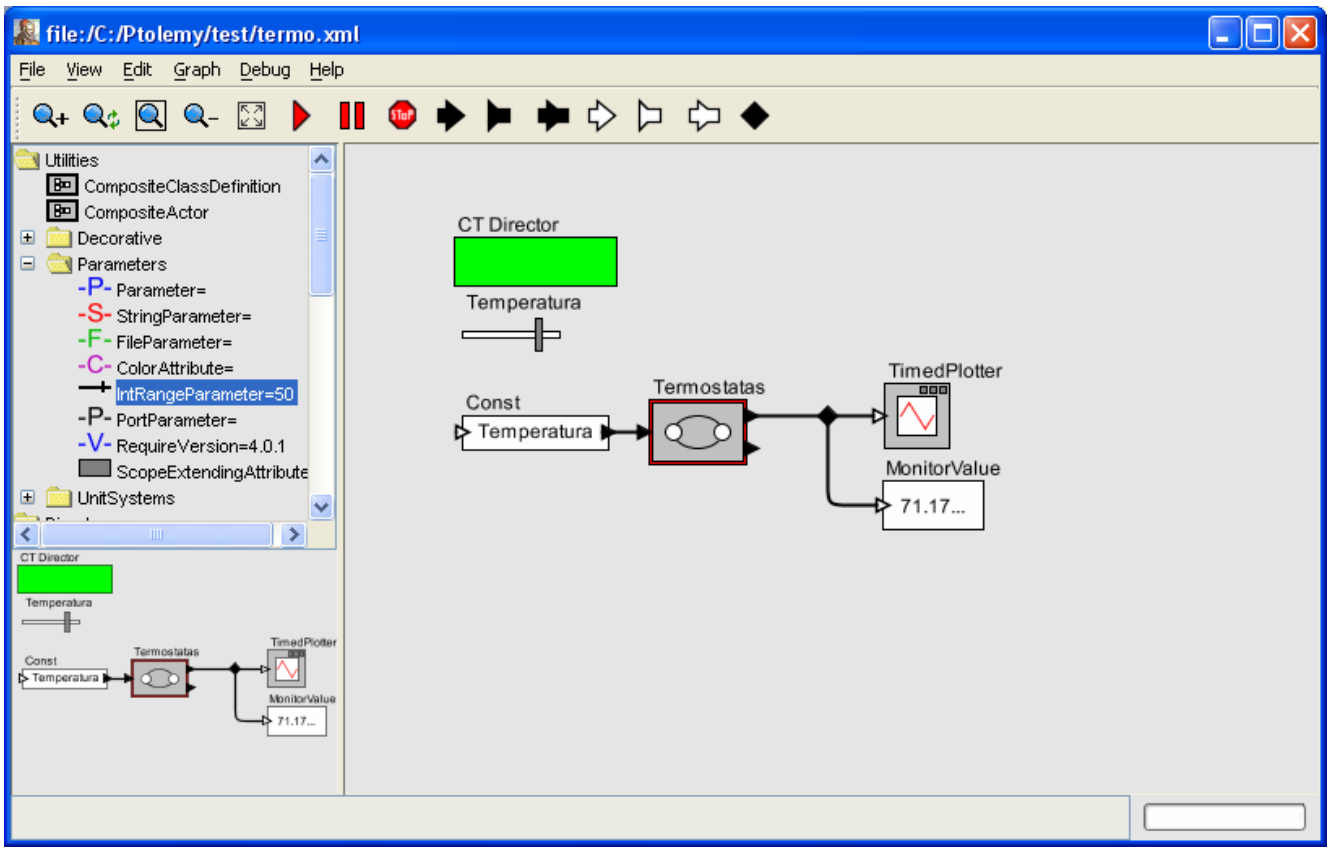
Iš kaitinimo būsenos pereinama prie vėsimo būsenos, kai temperatūra pasiekia nurodytą ribą pvz., 40° C ir aktoriui „HeatSource“ perduodama reikšmė „0“.

Iš vėsimo būsenos pereinama prie kaitinimo būsenos, kai temperatūra pasiekia nurodyta slenkstį, pvz., 35° C ir aktoriui „HeatSource“ perduodama reikšmė „35“ (3.11 pav.).



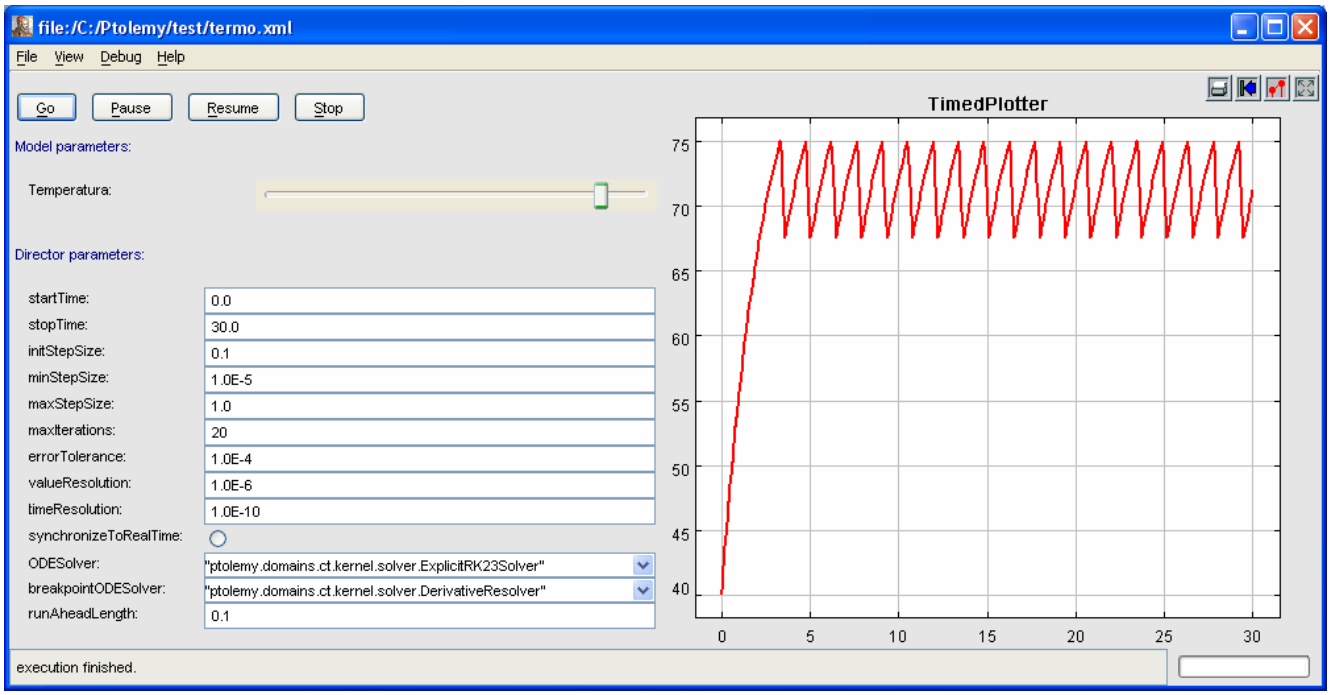
3.11 pav. Termostato būsenų perėjimo sąlygos

Termostato modelis pateiktas 3.12 paveiksle.



3.12 pav. Termostato modelis

Termostato modeliavimo rezultatai pateikti 3.13 paveiksle.



3.13 pav. Termostato modeliavimo rezultatai

Eksperimentams atlikti reikalinga MS Visio programa. Ant namo plano sudėtus norimus komponentus procedūros *ConvertToPtolemy* pagalba suformuojamas XML failas, kuriame aprašytas Ptolemy II sistemos modelis. Atsidarius minėtą failą Ptolemy II sistemoje, galima modeliuoti situacijas keičiant modelyje esančių aktorių parametrus.

Norint sukurti naują inžinerinį mazgą, reikia Ptolemy II sistemoje aktorių pagalba sudaryti modelį ir jį įtraukti į *ConvertToPtolemy* procedūrą.

Ptolemy II sistemoje modeliai gali būti aprašomi formaliomis specifikacijomis ir baigtinių būsenų automatu. Iš pateiktų eksperimentų matyti, kad visi modeliai veikia korektiškai.

IŠVADOS

Išsprendus uždavinius padarytos tokios išvados:

- atlikus modernaus būsto architektūrinių projektavimo metodų analizę nustatyta, kad pradinėje fazėje reikia nustatyti modernaus būsto komponento charakteristikas bei elgseną. Testavimų metu išaiškinami konfliktai ar neapibrėžta elgsena bendroje sistemoje;
- išanalizavus moderniame būste naudojamus įrenginius nustatyta, kad modeliavimui tinka šie inžineriniai mazgai: apšvietimo, šildymo sistemos;
- sudaryta namo plane esančių būsto įrenginių transformacijos į Ptolemy II modeliavimo ir kūrimo metodika. Šiam uždaviniui spręsti buvo naudojamas MS Visio paketas, kuriame yra integruotas gyvenamųjų patalpų planavimo posistemis. Minėtas posistemis buvo papildytas naujais komponentais, leidžiančiais atlikti transformacijas;
- modelio elementui sudaryti naudojami aktoriai, kurie leidžia aprašyti elemento elgseną. Jungiant aktorius į hierarchinę struktūrą gaunamas norimas būsto inžinerinis mazgas;
- atlikus modelio tyrimą Ptolemy II sistemoje nustatyta, kad modelis veikia korektiškai. Modeliavimas buvo pritaikytas Kauno technologijos universiteto Informatikos fakulteto Kompiuterių katedroje vykdomame projekte „Ateities būsto aukštosios technologijos ir įranga“.

LITERATŪROS SĄRAŠAS

[1] AGHA, Gul; THATI, Prasanna. *An Algebraic Theory of Actors and its Application to a Simple Object-Based Language*. 2004. p. 26-57.

[2] BROOKS, Christopher; LEE, Edward A.; LIU, Xiaojun; NEUENDORFFER, Steve; AHAO, Yang; ZHENG, Haiyang. *Ptolemy II heterogeneous concurrent modeling and design in java. Volume 1: introduction to Ptolemy II*. University of California. 2004. p. 250.

[3] BROOKS, Christopher; LEE, Edward A.; LIU, Xiaojun; NEUENDORFFER, Steve; AHAO, Yang; ZHENG, Haiyang. *Ptolemy II heterogeneous concurrent modeling and design in java. Volume 2: Ptolemy II software architecture*. University of California. 2004. p. 178.

[4] BROOKS, Christopher; LEE, Edward A.; LIU, Xiaojun; NEUENDORFFER, Steve; AHAO, Yang; ZHENG, Haiyang. *Ptolemy II heterogeneous concurrent modeling and design in java. Volume 3: Ptolemy II domains*. University of California. 2004. p. 132.

[5] BOWERS, S.; LUDÄSCHER, B. *Actor-Oriented Design of Scientific Workflows*. In 24th International Conference on Conceptual Modeling (ER). 2005.

[6] BULL, P.; LIMB, R.; PAYNE, R. *Pervasive home environments* // BT Technology Journal. ISSN: 1358-3948. 2004. Nr. 22. p. 65-72.

[7] DURRETT, John R.; BURNELL, Lisa J.; PRIEST, John W. *A Hybrid analysis and architectural design method for development of smart home components* // IEEE Wireless Communications. ISSN: 1536-1284. 2002. p. 85-91.

[8] EKER, Johan; JANNECK, Jörn W.; LEE, Edward A.; LIU, Jie; LIU, Xiaojun; LUDVIGY, Jozsef; NEUENDORFFER, Stephen; SACHS, Sonia; XIONG, Yuhong. *Taming Heterogeneity – the Ptolemy Approach*. // Proceedings of the IEEE. 2002. p. 14.

[9] HALL, Robert; HARVEY, Steve; SANBURN, Alan; SLACK, Daniel. *ModX10: A Security Layer Modification to the X10 Home Automation Protocol*. University of Manitoba. 2003. p. 50.

[10] HELAL, Sumi; MANN, Williams; EL-ZABADANI, Hicham; KING, Jeffrey; KADDOURA, Youssef; JANSSEN, Erwin. *The Gator Tech Smart House: A Programmable Pervasive Space* // Computer. ISSN: 0018-9162. 2005. p. 50-60.

[11] JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *The Unified Software Development Process*. Boston: Addison-Wesley, 1999. p. 512.

[12] LIU, Jie; EKER, Johan; JANNECK, Jörn W.; LIU, Xiaojun; LEE, Edward A. *Actor-Oriented Control System Design: A Responsible Framework Perspective* // Control Systems Technology, IEEE Transactions on. ISSN: 1063-6536. 2004. p. 250-262.

[13] LIU, Jie; LEE, Edward A. *A Component-Based Approach to Modeling and Simulating Mixed-Signal and Hybrid Systems*. // ACM Transactions on Modeling and Computer Simulation. ISSN: 1049-3301. 2002. p. 343-368.

[14] LIU, Jie; LIU, Xiaojun; LEE, Edward A. *Modeling Distributed Hybrid Systems in Ptolemy II*. 2001 American Control Conference. ISSN: 0743-1619. 2001, p. 4984-4985.

[15] NEUENDORFFER, Stephen, *A. Actor-oriented Metaprogramming*. University of California, Berkeley. 2004. p. 134.

[16] NEUENDORFFER, Stephen. *Automatic Specialization of Actor-oriented Models in Ptolemy II*. 2002. p. 47.

[17] NEUENDORFFER, Stephen. *Modeling real-world control systems: beyond hybrid systems*. Proceedings of the 2004 Winter Simulation Conference. 2004.

[18] POSTREL, V. *Rising Heat: New Thermostat Designed by Brilliant Morons*, Forbes ASAP, 1999.

[19] PRIEST, J. W.; SANCHEZ, J. *Product Development and Design for Manufacturing, A Collaborative Approach to Producibility and Reliability*. New York: Marcel Dekker, 2001.

[20] SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Pub. 1998. p. 383.

[21] WACKS, K. *Home Systems Standards: Achievements and Challenges* // IEEE Communications Magazine. ISSN: 0163-6804. 2002. p. 152-159.

[22] WEST, G.; GREENHILL, S.; VENKATESH, S. *A Probabilistic Approach to the Anxious Home for Activity Monitoring*. Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05). 2005.

SANTRAUKA

RAZMUS, Stasys. (2006) *Simulation methodology of home devices using Ptolemy II system*. MBA Graduation Paper. Kaunas: Faculty of Informatics, Kaunas University of Technology. 54 p.

SUMMARY

The appliance area of the computer technologies involves more and more living spheres because of its rapid development. Nowadays smart devices are integrated in the domestic electric appliances. For decades, technologists have been promising the ‘intelligent house’. The vision is usually portrayed as a house filled with technology which will do the dweller’s bidding and take all domestic drudgery out of their lives. Availability of faster, smaller and ever cheaper computing equipment and a variety of wired and wireless network technologies are enabling technologies that bring this vision closer to reality. These technology trends lead to the concept that computing and other ‘smart’ devices will become pervasive, fully networked and ‘disappear’ into the infrastructure of the home.

There is no unified standard methodology to describe interface of interconnection between other devices in creation of new home devices. In such cases only one device is evaluated independent of entire system, therefore compatibility problem occurs. Seeking to avoid these mentioned problems, it is necessary to have simulation of home devices methodology. This methodology enables to simplify developers work and to evaluate interconnection between devices.

SANTRUMPŲ IR TERMINŲ ŽODYNAS

CT – (Contingency Theory) tikimybių teorija.

FIFO – (First In First Out) Eilės tipas, kai pirmasis į struktūrą padėtas elementas ir paimamas pats pirmas, o paskutinis padėtas – paskutinis.

Framework – rėminė konstrukcija.

HOQ – (House Of Quality) Modernaus būsto kokybės matas.

IPT – (Information Processing Theory) Informacijos apdorojimo teorija.

Middleware – tarpinė komunikavimo aplinka.

MOC – (Model Of Computation) Skaičiavimo modelis.

X10 – tai komunikacinis protokolas, leidžiantis suderinamiems įrenginiams bendrauti tarpusavyje naudojant 220 V elektros tinklą.

XML – (eXtensible Markup Language) yra „žymėjimo“ kalba, kurioje saugoma struktūrizuota tekstinė informacija.

1 PRIEDAS

Pervedimo iš MS Visio į Ptolemy II sistemą paprogramio kodas

```
Dim intTermostatai As Integer
Dim intLempos As Integer
Dim sLemposRel As String
Dim Path As String
Dim HasError As Boolean
Dim HasElektrosSkaitiklis As Boolean


'---------------------------------------------------------------------
'Pagrindinė procedūra
'---------------------------------------------------------------------

Public Sub ConvertToPtolemy()
    Dim intCounter As Integer
    Dim intShapeCount As Integer
    Dim vsoShapes As Visio.Shapes
    Dim sTemp As String
    Dim sLine As String
    Dim DataFile As String
    Dim ResultFile As String

    Path = "C:\Ptolemy\"
    ResultFile = Path & "ptolemy.xml"
    HasError = False
    HasElektrosSkaitiklis = False

    Set vsoShapes = Application.ActiveWindow.Page.Shapes

    intShapeCount = vsoShapes.Count
    intTermostatai = 0
    intLempos = 0

    If intShapeCount > 0 Then
        Open ResultFile For Output As #1

        sTemp = ""
        DataFile = Path & "header.dat"
        On Error GoTo Klaida
        Open DataFile For Input As #2
        Do While Not EOF(2)
            Line Input #2, sLine
            sTemp = sTemp + sLine
        Loop
        Close #2
        Print #1, sTemp

        sLemposRel = ""
        For intCounter = 1 To intShapeCount
            Call AddComponent(vsoShapes.Item(intCounter).Name,
                              vsoShapes.Item(intCounter).Text)
        Next intCounter

        sTemp = ""
        DataFile = Path & "footer.dat"
        On Error GoTo Klaida
        Open DataFile For Input As #2
        Do While Not EOF(2)
            Line Input #2, sLine
```

```
            sTemp = sTemp + sLine
        Loop
        Close #2
        sTemp = Replace(sTemp, "%RELATIONS", sLemposRel)
        Print #1, sTemp
        Close #1
        If Not HasError Then
            MsgBox "Konvertavimas sėkmingai atliktas!" & vbCrLf &
                   "Suformuotas rezultatų failas: " & ResultFile,
                   vbOKOnly + vbInformation, "Pranešimas"
        Else
            MsgBox "Perspėjimas! Ptolemy II." & vbCrLf &
                   "Konvertavimas neatliktas nes buvo klaidų!",
                   vbOKOnly + vbExclamation, "Perspėjimas"
            Kill ResultFile
        End If
    Else
        MsgBox "Klaida! Ptolemy II." & vbCrLf & "Nerasta elementų!",
               vbOKOnly + vbCritical, "Klaida"
    End If
    Exit Sub
Klaida:
    HasError = True
    MsgBox "Klaida! Ptolemy II." & vbCrLf & "Nerastas failas " &
           DataFile & "!", vbOKOnly + vbCritical, "Klaida"
    Kill ResultFile
End Sub


'---------------------------------------------------------------------
'Procedūra atskiria komponentus ir iškviečia konkretaus komponento
'įterpimo funkciją
'  Name - komponento unikalus pavadinimas
'  Value - komponento perduodama reikšmė
'---------------------------------------------------------------------

Sub AddComponent(Name As String, Value As String)
    Dim Component As String
    Dim sID As String
    Dim sTemp As String

    i = InStr(1, Name, ".")
    If i = 0 Then
        Component = Name
        sID = 0
    Else
        Component = Mid(Name, 1, i - 1)
        sID = Mid(Name, i + 1, Len(Name))
    End If

    sTemp = ""
    If Component = "Lempa" Then
        sTemp = AddLempa(sID, Value)
    ElseIf Component = "Termostatas" Then
        sTemp = AddTermostatas(sID)
    ElseIf Component = "Elektros skaitiklis" Then
        sTemp = AddElektrosSkaitiklis(Component)
    End If

    Print #1, sTemp
End Sub
```

```
'---------------------------------------------------------------------
'Funkcija įdeda komponento "Lempa" aprašymą į modelį
'  sID - komponento unikalus numeris
'  sValue - komponentui perduodama reikšmė
'---------------------------------------------------------------------

Function AddLempa(sID As String, sValue As String) As String
    Dim sTemp As String
    Dim sLine As String
    Dim DataFile As String
    DataFile = Path & "Lempa.dat"

    sTemp = ""
    On Error GoTo Klaida
    Open DataFile For Input As #2
    Do While Not EOF(2)
          Line Input #2, sLine
          sTemp = sTemp + sLine
    Loop
    Close #2

    sTemp = Replace(sTemp, "%ID", sID)
    sTemp = Replace(sTemp, "%POWER", sValue)
    intLempos = intLempos + 1
    sTemp = Replace(sTemp, "%LEFT", intLempos * 70)
    sTemp = Replace(sTemp, "%TOP", (intLempos * 70) + 100)

    sLemposRel = sLemposRel + "<relation name=""relationLamp" & sID
          & """ class=""ptolemy.actor.TypedIORelation""></relation>"
    sLemposRel = sLemposRel + "<link port=""Const" & sID &
          ".output"" relation=""relationLamp" & sID & """/>"
    sLemposRel = sLemposRel + "<link port=""Lamp" & sID & ".input""
          relation=""relationLamp" & sID & """/>"

    AddLempa = sTemp
    Exit Function
Klaida:
    AddLempa = ""
    HasError = True
    MsgBox "Klaida! Ptolemy II." & vbCrLf & "Nerastas failas " &
           DataFile & "!", vbOKOnly + vbCritical, "Klaida"
End Function


'---------------------------------------------------------------------
'Funkcija įdeda komponento "Termostatas" aprašymą į modelį
'  sID - komponento unikalus numeris
'---------------------------------------------------------------------

Function AddTermostatas(sID As String) As String
    Dim sTemp As String
    Dim sLine As String
    Dim DataFile As String
    DataFile = Path & "Termostatas.dat"

    sTemp = ""
    On Error GoTo Klaida
    Open DataFile For Input As #2
    Do While Not EOF(2)
        Line Input #2, sLine
        sTemp = sTemp + sLine
    Loop
    Close #2
```

```
    AddTermostatas = sTemp
    Exit Function
Klaida:
    AddTermostatas = ""
    HasError = True
    MsgBox "Klaida! Ptolemy II." & vbCrLf & "Nerastas failas " &
           DataFile & "!", vbOKOnly + vbCritical, "Klaida"
End Function


'---------------------------------------------------------------------
'Funkcija įdeda komponento "Elektros skaitiklis" aprašymą į modelį
'  sID - komponento unikalus numeris
'---------------------------------------------------------------------

Function AddElektrosSkaitiklis(sID As String) As String
    Dim sTemp As String
    Dim sLine As String
    Dim DataFile As String
    DataFile = Path & "ElektrosSkaitiklis.dat"

    If HasElektrosSkaitiklis Then Exit Function
    sTemp = ""
    On Error GoTo Klaida
    Open DataFile For Input As #2
    Do While Not EOF(2)
        Line Input #2, sLine
        sTemp = sTemp + sLine
    Loop
    Close #2
    sTemp = Replace(sTemp, "%ID", sID)

    AddElektrosSkaitiklis = sTemp
    HasElektrosSkaitiklis = True
    Exit Function
Klaida:
    AddElektrosSkaitiklis = ""
    HasError = True
    MsgBox "Klaida! Ptolemy II." & vbCrLf & "Nerastas failas " &
           DataFile & "!", vbOKOnly + vbCritical, "Klaida"
End Function
```

2 PRIEDAS

Elektros skaitiklio modelio XML struktūra

```
<entity name="%ID" class="ptolemy.actor.TypedCompositeActor">
    <property name="_location" class="ptolemy.kernel.util.Location"
              value="[340.0, 310.0]"></property>
    <property name="SDF Director"
              class="ptolemy.domains.sdf.kernel.SDFDirector">
        <property name="_location" class="ptolemy.kernel.util.Location"
                  value="{50.0, 30.0}"></property>
    </property>
    <port name="input" class="ptolemy.actor.TypedIOPort">
        <property name="input"/>
        <property name="multiport"/>
        <property name="_location" class="ptolemy.kernel.util.Location"
                  value="[20.0, 110.0]"></property>
    </port>
    <port name="output" class="ptolemy.actor.TypedIOPort">
        <property name="output"/>
        <property name="_location" class="ptolemy.kernel.util.Location"
                  value="[350.0, 120.0]"></property>
    </port>
    <entity name="AddSubtract" class="ptolemy.actor.lib.AddSubtract">
        <property name="_location" class="ptolemy.kernel.util.Location"
                  value="[145.0, 120.0]"></property>
    </entity>
    <entity name="Expression" class="ptolemy.actor.lib.Expression">
        <property name="expression" class="ptolemy.kernel.util.StringAttribute"
                  value="input/1000"></property>
        <property name="_icon" class="ptolemy.vergil.icon.BoxedValueIcon">
            <property name="attributeName"
                      class="ptolemy.kernel.util.StringAttribute"
                      value="expression"></property>
            <property name="displayWidth" class="ptolemy.data.expr.Parameter"
                      value="60"></property>
        </property>
        <property name="_location" class="ptolemy.kernel.util.Location"
                  value="[245.0, 120.0]"></property>
        <port name="input" class="ptolemy.actor.TypedIOPort">
            <property name="input"/>
            <property name="_type" class="ptolemy.actor.TypeAttribute"
                      value="unknown"></property>
        </port>
    </entity>
    <relation name="relation_1" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation_2" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation1" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation2" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation3" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation4" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation5" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation6" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation7" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation8" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation9" class="ptolemy.actor.TypedIORelation"/>
    <relation name="relation10" class="ptolemy.actor.TypedIORelation"/>
    <link port="input" relation="relation1"/>
    <link port="input" relation="relation2"/>
    <link port="input" relation="relation3"/>
    <link port="input" relation="relation4"/>
    <link port="input" relation="relation5"/>
```

```
    <link port="input" relation="relation6"/>
    <link port="input" relation="relation7"/>
    <link port="input" relation="relation8"/>
    <link port="input" relation="relation9"/>
    <link port="input" relation="relation10"/>
    <link port="output" relation="relation_1"/>
    <link port="AddSubtract.plus" relation="relation1"/>
    <link port="AddSubtract.plus" relation="relation2"/>
    <link port="AddSubtract.plus" relation="relation3"/>
    <link port="AddSubtract.plus" relation="relation4"/>
    <link port="AddSubtract.plus" relation="relation5"/>
    <link port="AddSubtract.plus" relation="relation6"/>
    <link port="AddSubtract.plus" relation="relation7"/>
    <link port="AddSubtract.plus" relation="relation8"/>
    <link port="AddSubtract.plus" relation="relation9"/>
    <link port="AddSubtract.plus" relation="relation10"/>
    <link port="AddSubtract.output" relation="relation_2"/>
    <link port="Expression.input" relation="relation_2"/>
    <link port="Expression.output" relation="relation_1"/>
</entity>
<entity name="Skaitiklis" class="ptolemy.actor.lib.MonitorValue">
    <property name="value" class="ptolemy.data.expr.Parameter" value="00000">
    </property>
    <doc>Monitor and display values</doc>
    <property name="displayWidth" class="ptolemy.data.expr.Parameter"
              value="5"></property>
    <property name="_icon" class="ptolemy.vergil.icon.UpdatedValueIcon">
        <property name="attributeName" class="ptolemy.kernel.util.StringAttribute"
                  value="value"></property>
        <property name="displayWidth" class="ptolemy.data.expr.Parameter"
                  value="displayWidth"></property>
    </property>
    <property name="_location" class="ptolemy.kernel.util.Location"
              value="[460.0, 310.0]"></property>
</entity>
<relation name="relationElektrosSkaitiklis" class="ptolemy.actor.TypedIORelation">
</relation>
<link port="Elektros skaitiklis.output" relation="relationElektrosSkaitiklis"/>
<link port="Skaitiklis.input" relation="relationElektrosSkaitiklis"/>
```