LMD konf. darbai, 249–254
1997, Vilnius

Abstraktaus interfeiso specifikavimas produkcijomis

A. Čaplinskas (MII)

Atliekant programų sistemos eskizinį projektavimą, greta kitų tos sistemos komponentų paprastai yra projektuojamas ir jos abstraktusis interfeisas. Projektuojant abstraktųjį interfeisą, apibrėžiamos sistemai pateikiamų pranešimų signatūros ir aprašoma, kokias projektuojamos sistemos funkcijas ir prie kokių sąlygų inicijuos tų signatūrų generuojami pranešimai. Kokiu būdu ir kokiu pavidalu pranešimai bus pateikiami sistemai, projektuojant abstraktųjį interfeisą nenagrinėjama.

Abstrakčiajam programų sistemos interfeisui specifikuoti dažniausiai yra naudojamos vadinamosios perėjimų diagramos [1], t.y. diagramos, vaizduojančio įvykiais valdomus baigtinius automatus.

Apibrėžtis. Įvykiais valdomu baigtiniu automatu vadinamas šešetas

$$M = \langle B, I, P, f, g, b_{\text{prad}} \rangle,$$

kur B – leistinų kuriamos programų sistemos būsenų aibė, I – sistemai iš išorės pateikiamų pranešimų (įvykių) signatūrų aibė, P – sistemos vykdomų funkcijų aibė, f: $B \times \bar{I} \Rightarrow B$ ($\bar{I}$ – aibei I priklausančių signatūrų generuojamų pranešimų aibė) – perėjimo funkcija, $g : B \times \bar{I} \Rightarrow \Xi(P)$ ($\Xi(P)$ – visų galimų aibei P priklausančių funkcijų kompozicijų aibė) – funkcijų kompozicijos parinkimo funkcija, b_{prad} (b_{prad}) $\in B$ – pradinė sistemos būsena.

Naudojant tokį formalizmą, daugelio programų sistemų abstrakčiuosius interfeisus galima specifikuoti pakankamai patogiai. Vienok, projektuojant sudėtingesnes programų sistemas susiduriama su kai kuriomis interfeso specifikavimo problemomis. Pailiustruosime tai konkrečiu pavyzdžiu.

Pavyzdys 1. Tarkime, kad projektams planuoti skirtos programų sistemos funkcinė architektūra yra aprašyta šitokia funkcijų gramatika:

$$u_0 ::= u_1 \mid u_5;$$
$$u_1 ::= [u_2][u_3][u_4];$$
$$u_2 ::= \phi_0\phi_1\phi_2\phi_3;$$
$$u_3 ::= \phi_4 \mid \phi_5;$$
$$u_4 ::= [\phi_6][\phi_7][\phi_8];$$
$$u_5 ::= \phi_9\phi_{10}\phi_2\phi_3u_3;$$

a) Neterminalinių simbolių prasminė interpretacija:

$u_0(x_1, x_2, \ldots)$ – atlikti pirminį nurodytų projektų $x_1, x_2, \ldots$ (jų skaičius nefiksuotas) vykdymo planavimą ir, kai to prireikia, atsižvelgiant į faktinę projektų vykdymo eigą, patikslinti planinius darbų vykdymo terminus; $u_1(x_1, x_2, \ldots)$ – atlikti pirminį nurodytų projektų planavimą; $u_2(x_1, x_2, \ldots)$ – sudaryti nurodytų projektų tinklinius modelius; $u_3(x_1, x_2, \ldots)$ – paskaičiuoti nurodytų projektų darbų ir įvykių laiko charakteristikas; $u_4(x_1, x_2, \ldots)$ – pateikti informaciją apie nurodytų projektų darbų ir įvykių laiko charakteristikas; $u_5(x_1, x_2, \ldots)$ – patikslinti nurodytų projektų darbų planinius vykdymo terminus

b) Terminalinių simbolių prasminė interpretacija:

$t_0(x_1, x_2, \ldots)$ – įvesti duomenis apie nurodytų projektų struktūrą; $t_1(x_1, x_2, \ldots)$ – panaudojus duomenis apie nurodytų projektų struktūrą, sudaryti jų tinklinius modelius; $t_2(x_1, x_2, \ldots)$ – patikrinti, ar nurodytų projektų tinkliniai modeliai yra korektiški; $t_3(x_1, x_2, \ldots)$ – įsiminti duomenų bazėje nurodytų projektų tinklinius modelius; $t_4(x_1, x_2, \ldots)$ – atsižvelgiant į nurodytas kontrolinius įvykių terminus, paskaičiuoti nurodytiems projektams jų darbų ir įvykių laiko charakteristikas; $t_5(x_1, x_2, \ldots)$ – ignoruojant įvykių kontrolinius terminus, paskaičiuoti nurodytiems projektams jų darbų ir įvykių laiko charakteristikas; $t_6(x_1, x_2, \ldots)$ – pateikti ekrane informaciją apie nurodytų projektų darbų laiko charakteristikas; $t_7(x_1, x_2, \ldots)$ – pateikti ekrane informaciją apie nurodytų projektų įvykių laiko charakteristikas; $t_8(x_1, x_2, \ldots)$ – pateikti ekrane nurodytų projektų įvykių įvykimo prognozę; $t_9(x_1, x_2, \ldots)$ – įvesti duomenis apie nurodytų projektų faktinę vykdymo eigą; $t_{10}(x_1, x_2, \ldots)$ – iš nurodytų projektų tinklinių modelių pašalinti įvykdytus darbus ir pakoreguoti planinius terminus, atsižvelgiant į darbų baigimo terminų nuokrypius nuo planuotų terminų.

c) Automatas:

$$B = \{b_{\text{prad}}, b_1, b_2, b_3\},$$

$$I = \{i_1, i_2, i_3, i_4, i_5 i_6, i_7\},$$

$$P = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\},$$

$$f\colon \{(b_{\text{prad}}, \overline{i_1}) \to b_1, (b_1, \overline{i_2}) \to b_2, (b_1, \overline{i_3}) \to b_3, (b_3, \overline{i_2}) \to b_3,\ (b_3, \overline{i_2}) \to b_2,$$
$$(b_2, \overline{i_4}) \to b_2, (b_2, \overline{i_5}) \to b_2, (b_2, (\overline{i_6})) \to b_2, (b_3, \overline{i_4}) \to b_3, (b_3, \overline{i_5}) \to b_3,$$
$$(b_3, \overline{i_6}) \to b_3, (b_2, \overline{i_7}) \to b_2, (b_3, \overline{i_7}) \to b_3, (b_1, \overline{i_7}) \to b_1\},$$

$$g\colon \{(b_{\text{prad}}, \overline{i_1}) \to t_3 * t_2 * t_1 * t_0(x), (b_1, \overline{i_2}) \to t_4(x), (b_1, \overline{i_3}) \to t_5(x), (b_2, \overline{i_3}) \to t_5(x),$$
$$(b_3, \overline{i_2}) \to t_4(x), (b_2, \overline{i_4}) \to t_6(x), (b_2, \overline{i_5}) \to t_7(x), (b_2, \overline{i_6}) \to t_8(x),$$
$$(b_3, \overline{i_4}) \to t_6(x), (b_3, \overline{i_5}) \to t_7(x), (b_3, \overline{i_6}) \to t_8(x), (b_2, \overline{i_7}) \to t_4 * t_3 * t_2 * t_{10} * t_9(x),$$
$$(b_3, \overline{i_7}) \to t_5 * t_3 * t_2 * t_{10} * t_9(x), (b_1, \overline{i_7}) \to t_3 * t_2 * t_{10} * t_9(x)\}$$

d) Automato būsenų interpretacija:

b_1 – projekto tinklinis modelis suformuotas ir patikrintas, b_2 – ignoruojant kontrolinius terminus, paskaičiuotos tinklinio modelio darbų ir įvykių laiko charakteristikos. b_3 – atsižvelgiant į kontrolinius terminus, paskaičiuotos tinklinio modelio darbų ir įvykių laiko charakteristikos.

e) Pranešimų interpretacija:

i_1 – įvesti duomenis, i_2 -ignoruojant kontrolinius terminus, paskaičiuoti tinklinio modelio darbų ir įvykių laiko charakteristikas, i_3 – atsižvelgiant į kontrolinius terminus, paskaičiuoti tinklinio modelio darbų ir įvykių laiko charakteristikas, i_4 – pateikti ekrane informaciją apie projekto darbų laiko charakteristikas, i_5 – pateikti ekrane informaciją apie projekto įvykių laiko charakteristikas, i_6 – pateikti ekrane projekto įvykių įvykimo prognozę, i_7 – pakoreguoti projekto tinklinį modelį, atsižvelgiant į to projekto faktinę vykdymo eigą. □

Kaip matome iš pateikto pavyzdžio, specifikuojant interfeisą pritrūksta priemonių specifikuoti duomenimis valdomą funkcijų kvietimą. To pasekoje tenka reikalauti, kad naudotojas žinotų, kuriuose projektuose kontroliniai terminai yra nurodyti, o kuriuose ne. Pageidautina, kad, priklausomai nuo to, ar tinkliniame modelyje yra nurodyti kontroliniai terminai, programų sistema pati automatiškai parinktų atitinkamą skaičiavimo būdą (funkciją t_4 arba funkciją t_5). Be to, būtų būtų patogu, kad, paprašius pateikti ekrane projekto įvykių ar darbų laiko charakteristikas, nereikėtų žinoti, ar jos jau yra išskaičiuotos ir, jei to reikia, būtų skaičiuojamos automatiškai. Naudojant įvykiais valdomą automatą, to specifikuoti irgi nepavyko. Dar viena problema – projekto ir sistemos būsenų sutapatinamas. Automatas neturi informacijos apie tai, kada jis atsidūrė esamoje būsenoje, einamojo seanso ar anksčiau atliktų skaičiavimų metu. To pasekoje negalima įgyvendinti funkcinėje architektūroje numatyto draudimo to paties seanso metu formuoti projekto tinklinį modelį ir jį koreguoti.

Nurodytus interfeiso specifikavimo trūkumus galima pašalinti, specifikuojant jį gramatinėmis produkcijomis. Tuo tikslu pranešimus, projekto būsenas ir sistemos būsenas, kuriamas einamojo seanso metu, reikia traktuoti kaip faktus ir darbui su jais panaudoti tokio pavidalo produkcijas

$$i, e, u \Rightarrow s,$$

kur i – pranešimo tipas, e – projekto būsena, u – sistemos būsena einamojo seanso metu, s – funkcijų kompizicijų seka. Leidžiamos faktų apie sistemos ar projekto būseną keitimo funkcijos bei sistemos vykdomų funkcijų kompozicijos.

Kaip specifikuoti interfeisą tokiu formalizmu, pailiustruosime konkrečiu pavyzdžiu.

Pavyzdys 2. Panagrinėsime, kaip specifikuoti gramatinėmis produkcijomis 1 pavyzdyje nagrinėtą funkcinę architektūrą. Visų pirma, pranešimus i_2 ir i_3 sujunksime į vieną pranešimą ir gausime tokį pranešimų rinkinį:

i_1 – įvesti duomenis, i_2 – paskaičiuoti tinklinio modelio darbų ir įvykių laiko charakteristikas, i_3 – pateikti ekrane informaciją apie projekto darbų laiko charakteristikas, i_4 – pateikti ekrane informaciją apie projekto įvykių laiko charakteristikas, i_5 – pateikti ekrane projekto įvykių įvykimo prognozę, i_6 – pakoreguoti projekto tinklinį modelį, atsižvelgiant į to projekto faktinę vykdymo eigą.

Interfeiso specifikavimą pradėsime faktų bazės korektiškumo analize:

$t_1 : \neg e_1 \& \neg e_2 \& \neg e_3, \quad u_{\text{prad}} \Rightarrow$ šalinti(u_{prad}); u_0;

$t_2 : e_1 \& \neg e_2 \& \neg e_3, \quad u_{\text{prad}} \Rightarrow$ šalinti(u_{prad}); u_0;

$t_3 : \neg e_1 \& (e_2 \vee e_3), \quad u_{\text{prad}} \Rightarrow$ spausdinti(„Klaida:neleistina projekto būsena“); u_{gal};

Jei programų sistema jau galutinai baigta, tokia analizė yra nebūtina. Ji naudinga programų sistemos derinimo metu. Naudojant įvykiais valdomą automatą, galima aprašyti tik pranešimų analizę. Pačiam automatui derinti, formalizme priemonių nėra. Mūsų atveju, bazę gadinti gali ne tik specialiosios bazės koregavimo programos, bet ir dalykinės programos, rašančios informaciją apie kontrolinius įvykius (faktas e_3 – projekte nurodyti kontroliniai terminai). Analizei atlikti, mes įvedame papildomą būseną u_{prad}.

$t_4 : i_6, \neg e_1, u_0 \Rightarrow$spausdinti(„Klaida:projektui nesuformuotas tinklinis modelis ir todėl jo koreguoti negalima."); u_{gal};

$t_5 : i_6\&(i_1 \vee i_2 \vee i_3 \vee i_4 \vee i_5), u_0 \Rightarrow$spausdinti(„Klaida:koreguoti projektą ir tuo pat metu vykdyti kitus skaičiavimus draudžiama"); u_{gal};

Kaip matome, technologinį draudimą vieno seanso metu vykdyti modelio koregavimą ir skaičiavimus šiuo atveju pavyksta specifikuoti labai paprastai. Tai padaroma taisykle t_5.

Pereisime prie skaičiavimų specifikavimo:

$t_6 : \neg i_6\&i_1, \neg e_1, u_0 \Rightarrow t_3 * t_2 * t_1 * t_0(x)$; šalinti($u_0$); u_1; e_1; $\neg e_2$; $e_3(x)$;

$i_7 : \neg i_6\&\neg i_1\&(i_2 \vee i_3 \vee i_4 \vee i_5), u_0 \Rightarrow$ šalinti(u_0); u_1;

$t_8 : \neg i_2\&\neg i_3\&\neg i_4\&\neg i_5, u_1 \Rightarrow$ šalinti(u_1); u_{gal};

$t_9 : \neg i_6\&(i_2 \vee i_3 \vee i_4 \vee i_5), \neg e_1, u_1 \Rightarrow$ Spausdinti(„Klaida:neįvesti projekto duomenys."); u_{gal};

$t_{10} : i_2, e_1\&e_3, u_1 \Rightarrow t_4(x)$; šalinti($u_1$); u_2; e_2;

$t_{11} : i_2, e_1\&\neg e_3, u_1 \Rightarrow t_5(x)$šalinti($u_1$); u_2, e_2;

$t_{12} : \neg i_2\&(i_3 \vee i_4 \vee i_5), e_2, u_1 \Rightarrow$ šalinti(u_1); u_2;

$t_{13} : \neg i_3\&\neg i_4\&\neg i_5, u_2 \Rightarrow$ šalinti(u_2); u_{gal};

$t_{14} : \neg i_2\&(i_3 \vee i_4 \vee i_5), \neg e_2\&\neg e_3, u_2 \Rightarrow t_4(x)$; e_2;

$t_{15} : \neg i_2\&(i_3 \vee i_4 \vee i_5), \neg e_2\&e_3, u_2 \Rightarrow t_5(x)$; e_2;

$t_{16} : i_3, e_2, u_2 \Rightarrow t_6(x)$; šalinti($u_2$); u_3;

$t_{17} : \neg i_3\&(i_4 \vee i_5), u_2 \Rightarrow$ šalinti(u_2); u_3;

$t_{18} : \neg i_3\&\neg i_4\&\neg i_5, u_3 \Rightarrow$ šalinti(u_3); u_{gal};

$t_{19} : i_4, e_2, u_3 \Rightarrow t_7(x)$; šalinti($u_3$); u_4;

$t_{20} : \neg i_4\&i_5, u_3 \Rightarrow$ šalinti(u_3); u_4;

$t_{21} : \neg i_5, u_4 \Rightarrow$ šalinti(u_4); u_{gal};

$t_{22} : i_5, u_4 \Rightarrow t_8(x)$; šalinti($u_4$); u_{gal};

$t_{23} : i_6\&\neg i_1\&\neg i_2\&\neg i_3\&\neg i_4\&\neg i_5, e_1, u_0 \Rightarrow$ šalinti(u_0); u_5; $\neg e_2$;

$t_{24} : e_3, u_5 \Rightarrow t_4 * t_3 * t_3 * t_3 * t_3(x)$; šalinti($u_5$); e_2; u_{gal};

$t_{25} : \neg e_3, u_5 \Rightarrow t_5 * t_3 * t_3 * t_3 * t_3(x)$; šalinti($u_5$); e_2; u_{gal};

$t_{26} : u_{\text{gal}} \Rightarrow$ baigti_vykdyti();

Taigi, reikalavimai po modelio koregavimo automatiškai perskaičiuoti darbų ir įvykių laiko charakteristikas (taisyklės t_{24} ir t_{25}) ir, reikalui esant, automatiškai paskaičiuoti jas, kai naudotojas prašo pateikti informaciją apie projektą (taisyklės t_{14} ir t_{15}), šiuo atveju irgi specifikuojamos visiškai paprastai.

Rašant taisykles, padaryta prielaida, kad bazėje negali būti prieštaringų faktų. Tai reiškia, kad rašant į bazę faktą e, jo neiginys iš bazės yra automatiškai pašalinamas. □

Trumpai aptarsime gramatinėmis taisyklėmis specifikuotų interfeisų įgyvendinimo ypatumus. Tam gerai tinka vadinamoji "nusileidimo žemyn" strategija. Šis metodas plačiai naudojamas sintaksinės analizės teorijoje [2] ir yra pakankamai efektyvus. Taikant jį mūsų pasiūlyto pavidalo taisyklėms, neterminaliniai simboliai nagrinėjamajame žodyje turi būti skleidžiami iš kairės dešinėn, o sintaksinis medis apeinamas "gilyn". Darbas baigiamas, išskleidus visus neterminalinius simbolius, t.y. pasiekus galinę būseną. Pakeliui yra atliekami skaičiavimai. Pailiustruosime skaičiavimų planavimą ir vykdymą konkrečiu pavyzdžiu.

Pavyzdys 3. Tegul faktų bazė yra sudaryta iš tokių faktų:

$$u_{\text{prad}}, \neg e_1, \neg e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$$

Tada išvedimas sistemoje vyksta šitaip:

Nr.	Faktų bazė ir funkcijų vykdymas	Taikoma taisyklė	Skleidžiamasis neterminalinis simbolis
1	$u_{\text{prad}}, \neg e_1, \neg e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$	t_0	u_{prad}
2	$u_0, \neg e_1, \neg e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$ $t_3 * t_2 * t_1 * t_0(x);$	t_6	u_0
3	$u_1, e_1 \neg e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$	t_{12}	u_1
4	$u_2, e_1, \neg e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$	t_{15}	u_2
5	$u_2, e_1, e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$ $t_4(x)$	t_{17}	u_2
6	$u_3, e_1, e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$	t_{19}	u_3
7	$u_4, e_1, e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$ $t_7(x);$	t_{22}	u_4
8	$u_{\text{gal}}, e_1, e_2, \neg e_3, i_1, \neg i_2, \neg i_3, i_4, i_5, \neg i_6;$ $t_8(x);$	t_{26}	u_{gal}
9	$u_{\text{prad}}, e_1, e_2, \neg e_3, \neg i_1, \neg i_2, \neg i_3, \neg i_4, \neg i_5, \neg i_6;$ baigti_vykdyti();		

□

Kaip matome iš pateiktųjų pavyzdžių, specifikuojant programų sistemų abstraktųjį interfeisą gramatinėmis produkcijomis, galima padaryti jį lankstesnį ir intelektualesnį. Kita vertus, taip specifikuotą interfeisą įgyvendinti yra šiek tiek sudėtingiau, negu interfeisą specifikuotą įvykiais valdomu automatu.

LITERATŪRA

[1] B. Shneiderman, *Design the User Interface. Strategies for Effective Human-Computer Interaction. 2nd ed.*, Addison-Wesley Publishing Company, 1992.

[2] P. M. Lewis II, D. J. Rosenkrantz, R. E. Stearns, *Compiler Design Theory*, General Electric Company, Addison-Wesley Publishing Company, 1976.

Grammar production rules as a formalism to specify user interface

A. Čaplinskas

Event-driven automaton is an wide-accepted formalism used to specify abstract user interface. However, this formalism has some disadvantages. For example, if a data base is used, it is difficult to distinguish what events occur in current and what in previous computations. Grammar production rules as a formalism to specify abstract user interface are proposed in this paper. Some advantages of this formalism are demonstrated.