ISSN 0239–8044

1
1999

Techniki Komputerowe

BIULETYN INFORMACYJNY

INSTYTUT MASZYN MATEMATYCZNYCH
WARSZAWA 1999

Techniki Komputerowe

BIULETYN INFORMACYJNY

Rok XXXIV, Nr 1, 1999

INSTYTUT MASZYN MATEMATYCZNYCH
WARSZAWA 1999

Wydaje:

INSTYTUT MASZYN MATEMATYCZNYCH
UL. KRZYWICKIEGO 34
02-798 WARSZAWA
TEL. 621.84.41, TLX 81.78.80, FAX 629.92.70
E-MAIL: imasmat@imm.org.pl
INTERNET: http://www.imm.org.pl

Wydanie publikacji dofinansowane
przez Komitet Badań Naukowych

Druk: Zakład Poligrafii Ośrodka Przetwarzania Informacji, al. Niepodległości 186

TECHNIKI KOMPUTEROWE

Rok XXXIV Nr 1 1999

Spis treści

ANDRZEJ ABRAMOWICZ
INSTYTUT MASZYN MATEMATYCZNYCH WARSZAWA

Praktyczne aspekty zastosowania systemów informacyjnych z niepełną informacją
Practical aspects of reasoning in Nondeterministic Knowledge Representation Systems

Streszczenie

Artykuł przedstawia wybrane aspekty zastosowania systemów informacyjnych zawierających niepełną informację w systemach eksperckich. Rozdział 1 stanowi wprowadzenie i zawiera między innymi podstawowe pojęcia i definicje. W kolejnych rozdziałach opisane zostały metody rozwiązywania problemów charakterystycznych dla pewnej klasy systemów eksperckich. Rozwiązaniem jest w każdym przypadku pewien zbiór obiektów systemu informacyjnego oraz przyporządkowane temu zbiorowi prawdopodobieństwo pewności rozwiązania. Szczegółowo przedyskutowane zostały rozwiązania pewne i prawdopodobne.

Abstract

The paper presents some topics of Nondeterministic Knowledge Representation Systems implementation. Chapter 1 contains basic ideas, notions and definitions. The following chapters describe the methods of solving problems characteristic for a certain class of expert systems. Each time some subset of the set of system objects is defined as a solution, as well as probability of solution certainty. Both certain and uncertain solutions are discussed in details.

1. Wstęp

Artykuł jest kontynuacją rozważań dotyczących problemów dedukcji w systemach informacyjnych z niepełną informacją zawartych w [2] i [3]. Tekst ten stanowi jednak spójną całość, a jego lektura nie wymaga sięgania do wcześniejszych publikacji.

W systemie informacyjnym (Systemie Reprezentacji Wiedzy, SRW) będącym obiektem rozważań, własności obiektów są wyrażone poprzez atrybuty (np. KOLOR) i wartości atrybutów, które są zbiorami (np. {BIAŁY, NIEBIESKI, CZERWONY}). Niepełność informacji polega na tym, że nie można podać konkretnej wartości atrybutu, jesteśmy w stanie powiedzieć jedynie tyle, że należy ona do pewnego ustalonego zbioru. Z tego powodu system taki zwany jest w literaturze systemem zawierającym informację niedeterministyczną. Może on – w szczególności – reprezentować wiedzę o zjawiskach na tyle skomplikowanych, że ich opis matematyczny jest niemożliwy,

kłopotliwy lub nieefektywny. W takich przypadkach często pozostaje jedynie rejestrowanie następujących faktów: jeśli wartości $x_1,...,x_n$ parametrów wejściowych $a_1,...,a_n$ spełniają warunki $P_1,...,P_n$, to wartości $y_1,...,y_m$ parametrów wyjściowych $b_1,...,b_m$ spełniają warunki $Q_1,...,Q_m$. Można więc wskazać zbiory potencjalnych wartości parametrów wejściowych i odpowiadające im zbiory wartości parametrów wyjściowych, natomiast nie jest możliwe ustalenie dokładnych wartości parametrów. Sytuacja taka wydaje się być dość powszechnie spotykana (skomplikowane procesy przemysłowe, medycyna i ochrona zdrowia, kryminalistyka, ekonomia, meteorologia). System informacyjny zawierający powyższego typu dane (fakty) może być wykorzystany jako podstawowy składnik systemu eksperckiego przeznaczonego do rozwiązywania pewnej klasy problemów. Ten właśnie aspekt zastosowania systemów informacyjnych będzie przedmiotem naszego zainteresowania.

SRW zawierający niedeterministyczną informację jest czwórką

$$S = (U, AT, (VAL_a)_{a \in AT}, f),$$

gdzie U, AT i VAL_a, dla każdego $a \in AT$, są odpowiednio niepustym zbiorem obiektów, atrybutów i wartości atrybutów,

$$f: U \times AT \to 2^{VAL},$$

gdzie $VAL = \bigcup_{a \in AT} VAL_a$, jest funkcją całkowitą, taką że $f(x,a) \subset VAL_a$ dla każdego $x \in U$, $a \in AT$. Dla ustalonego a, zbiór $f(x,a)$ nazywamy uogólnioną wartością atrybutu a.

Dla każdego $x \in U$ funkcja

$$f_x: AT \to 2^{VAL},$$

jest (niedeterministyczną) informacją o obiekcie x, taką że $f_x(a) = V$ wtedy i tylko wtedy, gdy $f(x,a) = V$, dla $V \subset VAL_a$.

U i AT są zbiorami skończonymi, $AT = C \cup D$, gdzie C, D są niepustymi, rozłącznymi zbiorami atrybutów – odpowiednio – warunków i decyzji.

Niech A będzie podzbiorem AT. Definiujemy relację binarną $in(A) \subset U \times U$ (inkluzji informacyjnej ze względu na zbiór A) w sposób następujący:

$(x,y) \in in(A)$ wtedy i tylko wtedy, gdy $f(x,a) \subset f(y,a)$ dla każdego $a \in A$

oraz relację binarną $sim(A) \subset U \times U$ (podobieństwa informacyjnego ze względu na zbiór A):

$(x,y) \in sim(A)$ wtedy i tylko wtedy, gdy $f(x,a) \cap f(y,a) \neq \emptyset$ dla każdego $a \in A$.

W przypadku opisanych powyżej zjawisk, atrybuty systemu S są odpowiednikami parametrów, a każda informacja f_x w S jest interpretowana w następujący sposób: jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów $f_x(a)$ dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów $f_x(b)$ dla każdego $b \in D$. Tak interpretowane informacje zwane są regułami produkcji.

2. Wersje problemu namiarowania

Mając dany System Reprezentacji Wiedzy utworzony na podstawie doświadczeń, można sformułować następujące oczywiste pytania, będące istotą tzw. problemu namiarowania:

(i) czy jeśli wartości atrybutów warunków spełniają warunki $P_1,...,P_n$, to należy oczekiwać, że odpowiednie wartości atrybutów decyzji będą spełniały warunki $Q_1,...,Q_m$,

(ii) jakie warunki będą spełniały wartości atrybutów decyzji, jeśli atrybuty warunków spełniają $P_1,...,P_n$,

(iii) jakie warunki muszą spełniać atrybuty warunków, aby wartości atrybutów decyzji spełniały $Q_1,...,Q_m$.

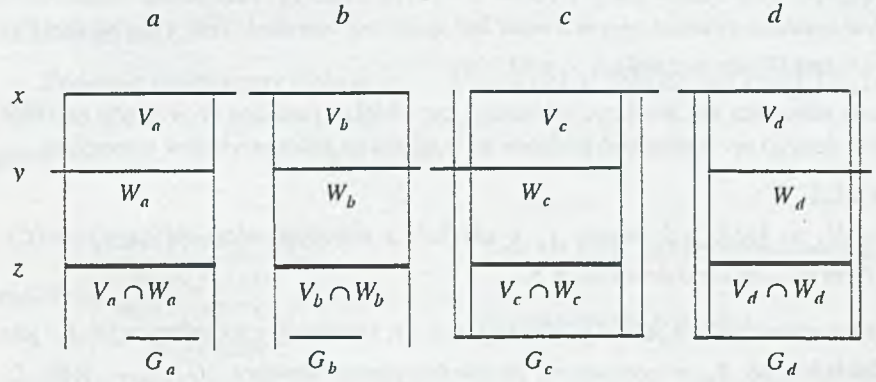
Powyższe pytania będą w dalszym ciągu utożsamiane – odpowiednio – z pierwszą, drugą i trzecią wersją problemu namiarowania.

Dla ustalonego systemu $S=(U,AT,(VAL_a)_{a\in AT},f)$, definiujemy indeksowaną rodzinę $(G_a)_{a\in AT}$ podzbiorów zbioru VAL, taką że $G_a \subset VAL_a$ dla każdego $a\in AT$, zwaną namiarem. Teraz możemy sformułować pierwsze z powyższych pytań w następujący sposób: „czy jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów G_a dla każdego $a\in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów G_b dla każdego $b\in D$". Innymi słowy interesuje nas, czy spełniony jest warunek:

$$\textit{istnieje } x\in U, \textit{ takie że } G_a \subset f(x,a) \textit{ dla każdego } a\in C \textit{ i } f(x,b)\subset G_b \textit{ dla każdego } b\in D. \quad (1)$$

W dalszej części namiarem będziemy nazywali również funkcję $(G_a)_{a\in AT}$ zredukowaną do pewnego podzbioru zbioru atrybutów AT.

System S nie reprezentuje explicite pełnej wiedzy uzyskanej na podstawie przeprowadzonych i zarejestrowanych doświadczeń. Weźmy pod uwagę system informacyjny zawierający tylko dwie informacje: f_x, f_y. Informacje te są przedstawione graficznie na rys. 1.



rys. 1

Na rysunku a i b są atrybutami warunków, c i d są atrybutami decyzji. Uogólnione wartości atrybutów są reprezentowane przez odcinki (przedziały). Taki sposób prezentacji wydaje się być najbardziej czytelny, ponieważ zależności teoriomnogościowe pomiędzy poszczególnymi zbiorami są natychmiast widoczne. Załóżmy, że namiarem jest rodzina $(G_a)_{a\in AT}$, gdzie $AT=\{a,b,c,d\}$, przedstawiona na rysunku w tej samej konwencji. Dla takich danych warunek (1) nie jest spełniony. Nietrudno jednak zauważyć, że skoro wartości atrybutów warunków a i b są elementami przedziałów – odpowiednio – $V_a \cap W_a$ i $V_b \cap W_b$, to wartości atrybutów decyzji c i d muszą być elementami przedziałów – odpowiednio – $V_c \cap W_c$ i $V_d \cap W_d$.

Następującą informacja:

$$f_z(a) = f_x(a) \cap f_y(a) \text{ dla każdego } a \in AT$$

o (hipotetycznym) obiekcie z może być zatem traktowana jako wniosek wyprowadzony (w sposób nieformalny) z przesłanek f_x, f_y. Dla f_z warunek (1) jest oczywiście spełniony. Jeśli C i D są zbiorami jednoelementowymi, poprawny jest również inny wniosek: $f_z(a) = f_x(a) \cup f_y(a)$ dla każdego $a \in AT$, jednak w praktycznych zastosowaniach fakt ten jest mało istotny.

Rozważania na poziomie modelu prowadzą więc do wniosku, że aby odpowiedzieć poprawnie na sformułowane powyżej pytania, należy wziąć pod uwagę nie tylko informacje zawarte explicite w bazie wiedzy, ale i pewne wnioski, które można z nich wyprowadzić. W [2] przedstawiona została formalna metoda rozstrzygania, czy dla danego namiaru, który formalnie może być traktowany jako informacja, warunek (1) jest spełniony. Zdefiniowany został język opisu systemu informacyjnego zawierający operatory teoriomnogościowe, każdej informacji i namiarowi została przyporządkowana pewna formuła oraz określone zostały reguły wnioskowania. W opisanym powyżej przypadku, zarówno formuła reprezentująca informację f_z, jak i formuła reprezentująca namiar mogą być w sposób formalny wyprowadzone ze zbioru formuł odpowiadających zbiorowi informacji $\{f_x, f_y\}$. Udowodniona została pełność aparatu wnioskowania. Pewna metoda automatyzacji procesu wnioskowania została przedstawiona w [3]. Na podstawie rozważań zawartych w [2], [3] można sformułować następujące wnioski:

WNIOSEK 1.1

W każdym systemie informacyjnym S musi być spełniony warunek: jeśli $(x,y) \in sim(C)$, *to* $(x,y) \in sim(D)$ *dla wszystkich* $x, y \in U$.

Implikacja odwrotna nie musi być spełniona, tzn. obiekty podobne ze względu na zbiór atrybutów decyzji nie muszą być podobne ze względu na zbiór atrybutów warunków.

WNIOSEK 1.2

Jeśli $y \in U$, *to każda informacja* f_x *o obiekcie x mającym własność* $(x,y) \in in(C)$ *i* $(y,x) \in in(D)$ *jest wnioskowalna w S.*

Warto zauważyć, że jeśli $(x,y) \in in(C)$ i $(y,x) \in in(D)$, to informacja f_x jest mniej dokładna od f_y w tym sensie, że dla dowolnego namiaru $(G_a)_{a\in AT}$, jeśli f_x spełnia warunek (1), to tym bardziej spełnia ten warunek informacja f_y. Istotnie, każdą

informację f_x o powyższej własności można usunąć z S nie pozbawiając systemu jego „wartości informacyjnej".

Oczywistym jest, że jeśli formuła reprezentująca namiar jest wnioskiem wyprowadzonym ze zbioru formuł reprezentujących zapisane w systemie informacje, czyli jeśli spełniony jest warunek (1), to prawdopodobieństwo zdarzenia reprezentowanego przez namiar wynosi 1. W praktycznych zastosowaniach baz wiedzy mogących znaleźć zastosowanie w systemach eksperckich istotne znaczenie mają przypadki, gdy prawdopodobieństwo to jest mniejsze od jedności. Załóżmy, że spełniony jest warunek:

$$\textit{istnieje } x \in U, \textit{ takie że } f(x,a) \cap G_a \neq \emptyset \textit{ dla każdego } a \in AT.$$

Oznacza to, że jest możliwe, że jeśli wartości atrybutów warunku są – odpowiednio – elementami G_a dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami G_b dla każdego $b \in D$.

Bez zmniejszenia ogólności rozważań możemy przyjąć, że dziedziny atrybutów VAL_a dla $a \in AT$, zbiory $f(x,a)$ będące uogólnionymi wartościami atrybutów oraz zbiory G_a, dla $a \in AT$, są niepustymi zbiorami skończonymi lub przedziałami domkniętymi miary dodatniej.

Dla systemu S i namiaru $(G_a)_{a \in AT}$ definiujemy funkcję $g_I : U \to <0,1>$, taką że $g_I(x) = \prod_{a \in C} g_a(x)$

gdzie

$$g_a(x) = \begin{cases} \dfrac{\text{card}(f(x,a) \cap G_a)}{\text{card}(G_a)} & \text{jeśli } G_a \text{ jest zbiorem skończonym} \\ \dfrac{\text{m}(f(x,a) \cap G_a)}{\text{m}(G_a)} & \text{w przeciwnym wypadku.} \end{cases}$$

$\text{m}(f(x,a) \cap G_a)$, $\text{m}(G_a)$ oznaczają miary odpowiednich zbiorów. Dla ustalonego $x \in U$, $a \in C$, $g_a(x)$ jest prawdopodobieństwem zdarzenia, że element G_a jest jednocześnie elementem $f(x,a)$. $g_I(x) = 1$ wtedy i tylko wtedy, gdy $G_a \subseteq f(x,a)$ dla każdego $a \in C$.

Podobnie definiujemy funkcję $g_O : U \to <0,1>$, taką że $g_O(x) = \prod_{b \in D} g_b(x)$

gdzie

$$g_b(x) = \begin{cases} \dfrac{\text{card}(f(x,b) \cap G_b)}{\text{card}(f(x,b))} & \text{jeśli } f(x,b) \text{ jest zbiorem skończonym} \\ \dfrac{\text{m}(f(x,b) \cap G_b)}{\text{m}(f(x,b))} & \text{w przeciwnym wypadku.} \end{cases}$$

Dla ustalonego $x \in U$, $b \in D$, $g_b(x)$ jest prawdopodobieństwem zdarzenia, że element $f(x,b)$ jest jednocześnie elementem G_b. $g_O(x) = 1$ wtedy i tylko wtedy, gdy $f(x,b) \subseteq G_b$ dla każdego $b \in D$.

Niech $g: U \to <0,1>$ będzie funkcją, taką że $g(x) = g_I(x) g_O(x)$. Warunek (1) odpowiadający pierwszej wersji problemu namiarowania można teraz sformułować w następujący sposób:

$$\textit{istnieje } x \in U, \textit{ takie że } g(x) = 1.$$

Wartość $g(x) = g_I(x) g_O(x) = \prod_{a \in C} g_a(x) \prod_{b \in D} g_b(x)$ stanowi obiektywną miarę „odległości od oczekiwań". Funkcję *g(x)* można więc traktować jako funkcję celu, a za rozwiązanie pierwszej wersji problemu namiarowania przyjąć zbiór obiektów, które maksymalizują tę funkcję. Oczywiście przy rozwiązywaniu problemu, jak już wspomniano, należy traktować system jako całość, tzn. należy brać pod uwagę zarówno fakty zawarte explicite w systemie, jak i informacje generowane przez system. W konsekwencji, elementami rozwiązania mogą być obiekty, które nie są obiektami w sensie podanej definicji systemu z niedeterministyczną informacją. Podany powyżej przykład dobrze ilustruje tę sytuację. W dalszym ciągu będziemy rozważali więc uogólniony system z niedeterministyczną informacją, spełniający warunek:

dla dowolnych obiektów $x, y \in U$, *jeśli* $(x, y) \in sim(C)$, *to istnieje obiekt* $z \in U$, *taki że*

$$f_z(a) = f_x(a) \cap f_y(a) \textit{ dla każdego } a \in AT.$$

Założenie to zapewnia, że system reprezentuje pełną wiedzę (w danym momencie) o opisywanym zjawisku. Nie oznacza to jednak konieczności generowania wszystkich możliwych obiektów w komputerowej implementacji rozwiązania problemu. Wybór strategii generowania obiektów zależy od samego namiaru. Jeśli dla pewnego obiektu x i atrybutu a, $f(x,a) \cap G_a = \emptyset$, to dla dowolnego y, $(f(y,a) \cap f(x,a)) \cap G_a = \emptyset$; nie ma więc w tym przypadku sensu branie pod uwagę jakiegokolwiek obiektu z i informacji $f_z(a) = f_x(a) \cap f_y(a)$ dla każdego $a \in AT$. W takim przypadku $g(x)=0$.

W dalszym ciągu będziemy zakładali, że w systemie nie ma dwóch identycznych informacji. Założenie to ma wyłącznie techniczny charakter i nie wpływa na ogólność rozważań.

2.1. Pierwsza wersja problemu namiarowania

Niech R_1 będzie zbiorem obiektów maksymalizujących funkcję g: $R_1 = \{x \in U : g(x) \geq g(y)$ dla każdego $y \in U\}$. Rozważymy trzy przypadki i dla każdego z nich zdefiniujemy rozwiązanie problemu.

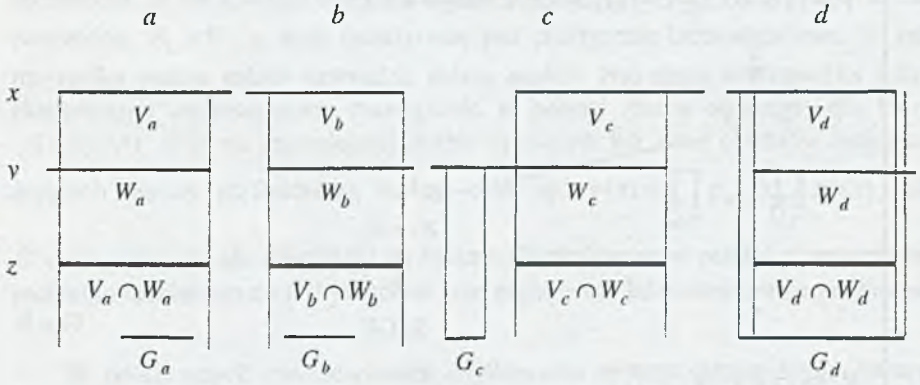
Przypadek 1 $g(x) = 1$

Jeśli $g(x) = 1$, to z pewnością, jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów G_a dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów G_b dla każdego $b \in D$. W takim przypadku formuła odpowiadająca namiarowi może być, jak już wspomniano, formalnie

wywnioskowana ze zbioru formuł odpowiadających informacjom w systemie S. Jako rozwiązanie przyjmujemy zbiór $R = R_1$.

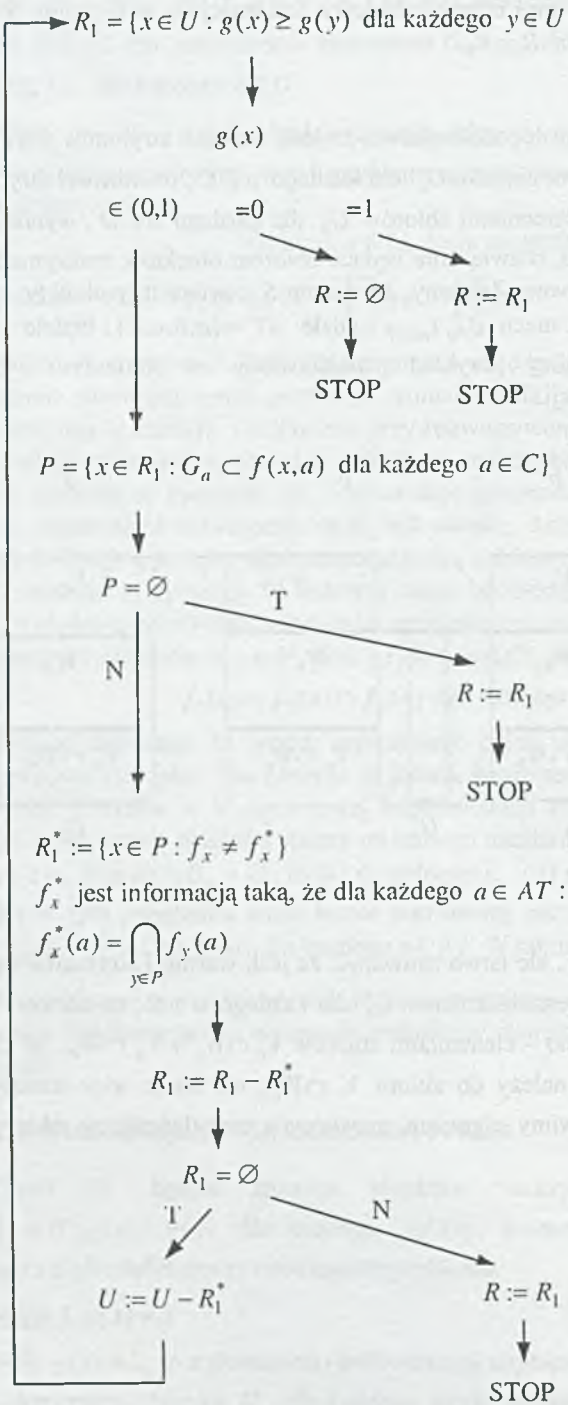
Przypadek 2 $0 < g(x) < 1$

Jeśli $0 < g(x) < 1$, to prawdopodobieństwo, że jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów G_a dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów G_b dla każdego $b \in D$, wynosi $g(x)$. W ogólnym jednak przypadku, rozwiązanie będące zbiorem obiektów maksymalizujących funkcję g nie jest poprawne. Załóżmy, że system S zawiera trzy obiekty x, y, z. Niech $C = \{a,b\}$, $D = \{c,d\}$ i niech $(G_a)_{a \in AT}$, gdzie $AT = \{a,b,c,d\}$, będzie ustalonym namiarem. Przeanalizujmy przykład przedstawiony na poniższym rysunku w przyjętej wcześniej konwencji.



rys. 2

Z danych wynika, że $R_1 = \{y\}$, ale łatwo zauważyć, że jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów G_a dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów $V_c \cap W_c$ i $V_d \cap W_d$. W szczególności, wartość atrybutu c należy do zbioru $V_c \cap W_c$, nie może więc należeć do zbioru G_c. Poniżej przedstawimy algorytm rozwiązania uwzględniający takie przypadki.


$R_1 = \{x \in U : g(x) \geq g(y)$ dla każdego $y \in U\}$
$g(x)$
$\in (0,1)$
$=0$
$=1$
$R := \emptyset$
$R := R_1$
STOP
STOP
$P = \{x \in R_1 : G_a \subset f(x,a)$ dla każdego $a \in C\}$
$P = \emptyset$
T
N
$R := R_1$
STOP
$R_1^* := \{x \in P : f_x \neq f_x^*\}$
f_x^* jest informacją taką, że dla każdego $a \in AT$:
$f_x^*(a) = \bigcap_{y \in P} f_y(a)$
$R_1 := R_1 - R_1^*$
$R_1 = \emptyset$
T
N
$U := U - R_1^*$
$R := R_1$
STOP

Informacja f_x^* istnieje na mocy definicji uogólnionego systemu z niedeterministyczną informacją. Krok $U := U - R_1^*$ jest realizowany w sytuacji, jak w podanym powyżej przykładzie. Następuje kolejne określenie zbioru R_1, przy czym prawdopodobieństwo $g(x)$ dla każdego $x \in R_1$ będzie tym razem mniejsze niż w przypadku prawdopodobieństwa $g(x)$ obliczonego dla obiektów będących elementami poprzednio wyznaczonego zbioru R_1. Nie jest wykluczone, że tym razem $g(x) = 0$. Algorytm obejmuje wszystkie przypadki wartości $g(x)$ z przedziału $[0,1]$.

Przypadek 3 $g(x) = 0$

Jeśli $g(x) = 0$, to prawdopodobieństwo, że jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów G_a dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów G_b dla każdego $b \in D$, jest zerowe, co oznacza, że dla każdego obiektu x istnieje $a \in AT$, takie że $f(x,a) \cap G_a = \emptyset$. Stąd oczywiście $R_1 = U$, a więc rozwiązanie jest praktycznie bezwartościowe. W takim przypadku można jednak prowadzić dalszą analizę. Być może użytkownika systemu eksperckiego interesowałoby rozwiązanie w postaci zbioru obiektów, dla których $f(x,a) \cap G_a = \emptyset$ na najmniejszej liczbie atrybutów lub zbiór obiektów maksymalizujących inaczej zdefiniowaną funkcję celu, np. $g(x) = \prod_{a \in C'} g_a(x) \prod_{b \in D'} g_b(x)$, gdzie $C' \subset C$, $D' \subset D$, ale odpowiedź na pytanie sformułowane w postaci pierwszej wersji problemu namiarowania jest jednoznacznie negatywna. Jako rozwiązanie przyjmujemy $R = \emptyset$.

W praktycznych zastosowaniach użytkownika systemu eksperckiego interesuje także rozwiązanie drugiej i trzeciej wersji problemu namiarowania.

2.2. Druga wersja problemu namiarowania

W tym przypadku namiarem jest rodzina zbiorów $(G_a)_{a \in C}$. Niech R_1 będzie zbiorem obiektów maksymalizujących funkcję g_I: $R_1 = \{x \in U : g_I(x) \geq g_I(y)$ dla każdego $y \in U\}$. Rozważymy również trzy przypadki i dla każdego z nich zdefiniujemy rozwiązanie.

Przypadek 1 $g_I(x) = 1$

$g_I(x) = 1$ wtedy i tylko wtedy, gdy $G_a \subseteq f(x,a)$ dla każdego $a \in C$. Łatwo zauważyć, że zachodzi następujący

FAKT 2.1.1

Istnieje obiekt $z \in R_1$, *taki że* $f(z,a) = \bigcap_{y \in R_1} f(y,a)$ *dla każdego* $a \in AT$.

DOWÓD wynika wprost z definicji uogólnionego systemu z niedeterministyczną informacją.□

Jako rozwiązanie problemu przyjmujemy zbiór $R_I = \{z\}$, gdzie z jest obiektem o powyższej własności. Istotnie, informacja przyporządkowana obiektowi z najprecyzyjniej ze wszystkich informacji przyporządkowanych obiektom ze zbioru R_1 określa uogólnione wartości atrybutów decyzji. Interpretacja rozwiązania jest oczywista: prawdopodobieństwo, że jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów G_a dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów $f_z(b)$ dla każdego $b \in D$, jest równe 1.

Przypadek 2 $0 < g_I(x) < 1$

Jeśli $0 < g_I(x) < 1$, to $f(x,a) \cap G_a \neq \emptyset$ dla każdego $a \in C$. Niech $x \in R_1$. Prawdopodobieństwo, że jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów G_a dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów $f_x(b)$ dla każdego $b \in D$, wynosi g_I. Ponadto, dla każdego $x \in R_1$, prawdopodobieństwo, że jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów $f(x,a) \cap G_a$ dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów $f_x(b)$ dla każdego $b \in D$, jest równe 1. Konsekwencją tego faktu jest sugestia redukcji namiaru – jeśli to możliwe – do zbiorów $f(x,a) \cap G_a$, gdzie $x \in R_1$. Jako rozwiązanie przyjmujemy $R_I = R_1$.

Przypadek 3 $g_I(x) = 0$

$g_I(x) = 0$ oznacza, że dla każdego $x \in R_1$ istnieje atrybut a, taki że $g_a(x) = 0$, czyli $f(x,a) \cap G_a = \emptyset$. W konsekwencji $R_1 = U$, a więc kryterium maksymalizacji funkcji g_I nie redukuje zbioru obiektów. Na podstawie aktualnego stanu wiedzy reprezentowanej przez system S nie jest więc możliwe określenie własności atrybutów decyzji, jeśli namiarem jest rodzina zbiorów $(G_a)_{a \in C}$. Jako rozwiązanie przyjmujemy $R_I = \emptyset$.

2.3. Trzecia wersja problemu namiarowania

Jako namiar przyjmujemy rodzinę zbiorów $(G_b)_{b \in D}$. Niech R_1 będzie zbiorem obiektów maksymalizujących funkcję g_O: $R_1 = \{x \in U : g_O(x) \geq g_O(y)$ dla każdego $y \in U\}$. Podobnie jak dla poprzednich wersji problemu należy rozważyć trzy przypadki i dla każdego z nich zdefiniować rozwiązanie. W tym przypadku nie można osiągnąć równie precyzyjnych wyników, jak w przypadku poprzedniej wersji. Wynika to z implikacyjnej interpretacji informacji w systemie S.

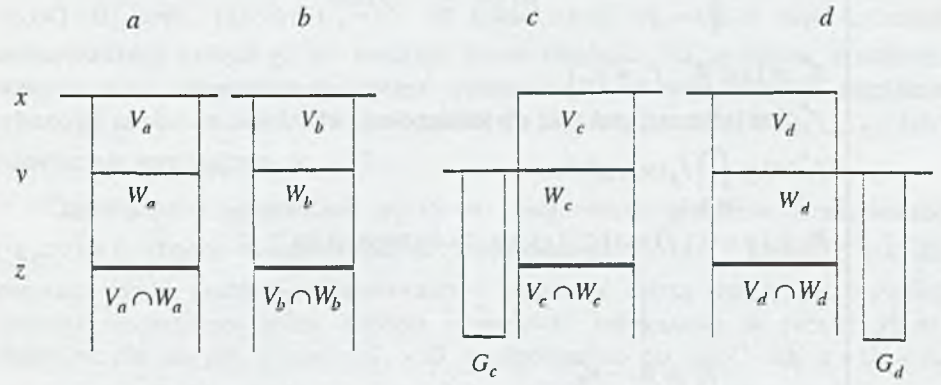
Przypadek 1 $g_O(x) = 1$

$g_O(x) = 1$ wtedy i tylko wtedy, gdy $f(x,b) \subseteq G_b$ dla każdego $b \in D$. W tym przypadku jako rozwiązanie przyjmujemy $R_O = R_1$, a zbiór ten interpretujemy w następujący sposób: dla każdego $x \in R_O$, prawdopodobieństwo, że jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów $f(x,a)$ dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów G_b dla każdego

$b \in D$, jest równe 1. W pewnych sytuacjach, badając zależności pomiędzy uogólnionymi wartościami atrybutów warunków dla obiektów stanowiących rozwiązanie, można pokazać, że dla osiągnięcia sukcesu kryteria dotyczące uogólnionych wartości atrybutów decyzji mogą być słabsze, ale interpretacja rozwiązania pozostaje ta sama.

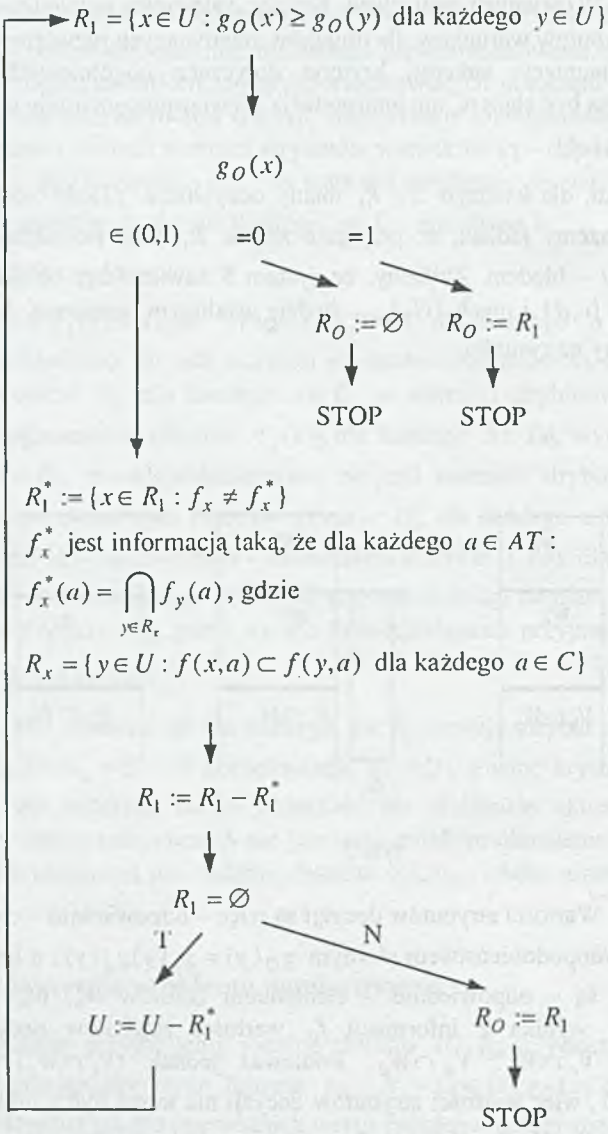
Przypadek 2 $0 < g_O(x) < 1$

W tym przypadku, dla każdego $x \in R_1$ mamy oczywiście $f(x,b) \cap G_b \neq \emptyset$ dla każdego $b \in D$. Pokażemy jednak, że przyjęcie zbioru R_1 jako rozwiązania, jest – w ogólnym przypadku – błędem. Załóżmy, że system S zawiera trzy obiekty x, y, z. Niech $C = \{a,b\}$, $D = \{c,d\}$ i niech $(G_b)_{b \in D}$ będzie ustalonym namiarem. Rozważmy przykład przedstawiony na rysunku.



rys. 3

Oczywiście $R_1 = \{y\}$. Wartości atrybutów decyzji są więc – odpowiednio – elementami zbiorów G_c, G_d z prawdopodobieństwem równym $g_O(y) = g_c(y)g_d(y)$, o ile wartości atrybutów warunków są – odpowiednio – elementami zbiorów W_a, W_b. W takim jednak wypadku, co wynika z informacji f_z, wartości atrybutów decyzji będą elementami zbiorów $V_c \cap W_c$, $V_d \cap W_d$. Ponieważ jednak $(V_c \cap W_c) \cap G_c = \emptyset$ i $(V_d \cap W_d) \cap G_d = \emptyset$, więc wartości atrybutów decyzji nie mogą być – odpowiednio – elementami zbiorów G_c, G_d. Poniżej przedstawimy algorytm rozwiązania generujący poprawne wyniki.



Zbiór $R_x = \{y \in U : f(x,a) \subseteq f(y,a)$ dla każdego $a \in C\}$ jest niepusty, ponieważ dla każdego obiektu x istnieje co najmniej jeden obiekt y, taki że $f(x,a) \subseteq f(y,a)$ dla każdego $a \in C$. Dla każdego $x \in R_1$ istnieje więc niepusty zbiór R_x i – na mocy definicji uogólnionego systemu z niedeterministyczną informacją – dokładnie jedna informacja f_x^*. Krok $U := U - R_1^*$ jest realizowany w sytuacji jak w podanym powyżej przykładzie. Następuje kolejne określenie zbioru R_1, przy czym prawdopodobieństwo $g_O(x)$ dla każdego $x \in R_1$ będzie tym razem mniejsze niż w przypadku

prawdopodobieństwa $g_O(x)$ obliczonego dla obiektów będących elementami poprzednio wyznaczonego zbioru R_1. Nie jest wykluczone, że tym razem $g_O(x) = 0$. Algorytm obejmuje wszystkie przypadki wartości $g_O(x)$ z przedziału [0,1].

Interpretacja rozwiązania R_O dla przypadku $0 < g_O(x) < 1$ jest następująca: dla każdego $x \in R_O$, prawdopodobieństwo, że jeśli wartości atrybutów warunków są – odpowiednio – elementami zbiorów $f(x,a)$ dla każdego $a \in C$, to wartości atrybutów decyzji są – odpowiednio – elementami zbiorów G_b dla każdego $b \in D$, wynosi $g_O(x)$.

Przypadek 3 $g_O(x) = 0$

$g_O(x) = 0$ oznacza, że dla każdego $x \in R_1$ istnieje atrybut $b \in D$, taki że $g_b(x) = 0$, czyli $f(x,b) \cap G_b = \emptyset$. W konsekwencji $R_1 = U$, a więc kryterium maksymalizacji funkcji g_O nie redukuje zbioru obiektów. Na podstawie aktualnego stanu wiedzy reprezentowanej przez system S nie jest więc możliwe określenie własności atrybutów warunków, jeśli namiarem jest rodzina zbiorów $(G_b)_{b \in D}$. Jako rozwiązanie przyjmujemy $R_O = \emptyset$.

Oczywiście w przypadkach $g_I(x) = 0$ (druga wersja problemu namiarowania) i $g_O(x) = 0$ (trzecia wersja problemu namiarowania), można – podobnie jak dla pierwszej wersji problemu namiarowania – prowadzić dalszą analizę. Użytkownika systemu eksperckiego może bowiem interesować rozwiązanie w postaci zbioru obiektów, dla których $f(x,a) \cap G_a = \emptyset$ – odpowiednio dla $a \in C$ lub $a \in D$ – na najmniejszej liczbie atrybutów lub zbiór obiektów maksymalizujących inaczej zdefiniowaną funkcję celu, np. – odpowiednio – $g_I(x) = \prod_{a \in C'} g_a(x)$, $C' \subset C$, lub $g_O(x) = \prod_{a \in D'} g_a(x)$, $D' \subset D$, ale odpowiedź na pytanie sformułowane w postaci – odpowiednio – drugiej lub trzeciej wersji problemu namiarowania jest niemożliwa.

3. Dynamika systemów informacyjnych

W dalszym ciągu zakładamy, że dany jest uogólniony system z niedeterministyczną informacją S oraz namiar $(G_a)_{a \in AT}$. W przypadku konstruowania zbioru R_I i obliczania prawdopodobieństwa g_I jako namiar przyjmujemy funkcję $(G_a)_{a \in AT}$ zredukowaną do zbioru atrybutów warunków C, zaś w przypadku konstruowania zbioru R_O i obliczania prawdopodobieństwa g_O jako namiar przyjmujemy funkcję $(G_a)_{a \in AT}$ zredukowaną do zbioru atrybutów decyzji D.

FAKT 3.1

$g(z)=1$ dla pewnego $z \in U$ wtedy i tylko wtedy, gdy spełnione są warunki:

(a) istnieje obiekt $x \in U$, taki że $g_I(x)=1$,

(b) istnieje obiekt $y \in U$, taki że $g_O(y)=1$,

(c) $R_I \subset R_O$.

DOWÓD:

Załóżmy, że $g(z)=1$ dla pewnego $z \in U$. Stąd dla obiektu z natychmiast otrzymujemy (*a*) i (*c*). Niech $R_I=\{t\}$. Z definicji R_I mamy $f(t,a) \subset f(z,a)$ dla każdego $a \in AT$. Ponieważ $z \in R_O$, więc $t \in R_O$, skąd wynika (*c*).

Załóżmy, że spełnione są (*a*), (*b*), i (*c*). Z (*a*) wynika, że istnieje $t \in U$, takie że $R_I=\{t\}$. Oczywiście $g_I(t)=1$. Z (*c*) wynika, że $t \in R_O$. Ponieważ z założenia (*b*) istnieje obiekt $y \in U$, taki że $g_O(y)=1$, więc dla każdego $u \in R_O$ $g_O(u)=1$, stąd $g_O(t)=1$. Mamy zatem $g_I(t)=g_O(t)=1$, więc $g(t)=g_I(t)g_O(t)=1$, co kończy dowód.□

System S będący jądrem systemu eksperckiego nie jest systemem statycznym. W danej chwili odzwierciedla on pewien stan wiedzy o opisywanym zjawisku. Wiedza ta – w ogólnym przypadku – nie jest wiedzą kompletną. Nawet w przypadku, gdy dziedziny atrybutów są zbiorami skończonymi, liczba wszystkich możliwych informacji może być ogromna, a w wielu zastosowaniach dziedziny atrybutów są przedziałami z dziedziny liczb rzeczywistych. Baza wiedzy jest uaktualniana poprzez sukcesywne dodawanie nowych informacji uzyskiwanych na podstawie kolejnych doświadczeń.

Załóżmy, że dane są dwa uogólnione systemy informacyjne $S=(U,AT,(VAL_a)_{a\in AT},f)$ i $S'=(U',AT,(VAL_a)_{a\in AT},f')$, takie że $U \subset U'$ i $f'(x,a)=f(x,a)$ dla każdego $x \in U$, $a \in AT$. S' jest więc systemem powstałym z S poprzez dodanie nowych informacji.

FAKT 3.2

Jeśli istnieje obiekt $x \in U$, taki że $g_I(x) \neq 0$ i atrybut $b \in D$, taki że $g_b(x)=0$, to $g(y)<1$ dla każdego obiektu $y \in U'$.

DOWÓD:

Załóżmy, że istnieje obiekt $z \in U'$, taki że $g(z)=1$, czyli $G_a \subset f(z,a)$ dla każdego $a \in C$ i $f(z,b) \subset G_b$ dla każdego $b \in$ D. Z założenia mamy $g_I(x) \neq 0$, więc $f(x,a) \cap G_a \neq \emptyset$ dla każdego $a \in C$. Stąd $f(x,a) \cap f(z,a) \neq \emptyset$ dla każdego $a \in C$. Ponieważ zakładamy, że S i S' mają własność wyrażoną we Wniosku 1.1, $f(x,d) \cap f(z,d) \neq \emptyset$ dla każdego $d \in$ D. Z założenia jednak $g_b(x)=0$ dla pewnego $b \in$ D, co oznacza, że $f(x,b) \cap G_b = \emptyset$. Ale $f(z,b) \subset G_b$ i $f(x,b) \cap f(z,b) \neq \emptyset$, skąd dostajemy $f(x,b) \cap G_b \neq \emptyset$, czyli sprzeczność.□

Nawet jeśli warunek namiarowania wyrażony w pierwszej wersji problemu namiarowania nie jest spełniony, nie oznacza to, że nie będzie mógł być spełniony po

dodaniu do systemu nowych informacji. Znaczenie powyższego lematu polega na tym, że pozwala wykryć przypadek, gdy warunek ten nie może być spełniony dla danego namiaru bez względu na stan wiedzy reprezentowany przez system.

4. Wnioski

Z przeprowadzonej analizy wynika kilka istotnych wniosków, które warto wziąć pod uwagę przy tworzeniu konkretnych aplikacji. W praktycznych zastosowaniach uogólnione wartości atrybutów są najczęściej przedziałami domkniętymi ze zbioru liczb rzeczywistych. Konstruując bazę wiedzy należy unikać – jeśli tylko jest to możliwe – podobieństwa informacyjnego obiektów ze względu na zbiór atrybutów warunków. W przeciwnym wypadku istnieje konieczność brania pod uwagę informacji będących wnioskami, chociaż nie oznacza to – jak już wspomniano – konieczności generowania wszystkich możliwych obiektów i przyporządkowanych im informacji. Ponadto warto wstępnie eliminować pewne informacje z bazy wiedzy. Jeśli informacja f_x jest mniej dokładna od f_y w sensie definicji z rozdziału 2, to łatwo zauważyć, że eliminacja f_x nie wpływa na praktyczną wartość rozwiązania w żadnym z analizowanych przypadków.

Wprowadzona miara pewności rozwiązania wydaje się obiektywnie charakteryzować „odległość od oczekiwań", chociaż zawodzi w przypadkach, gdy jako rozwiązanie przyjmujemy zbiór pusty, co oznacza, że prawdopodobieństwo oczekiwanego zdarzenia wynosi zero lub że stan wiedzy reprezentowanej przez system nie jest wystarczający do określenia własności wybranych atrybutów. Oczywiście zarówno w takim wypadku, jak i w sytuacji, gdy zbiór będący rozwiązaniem jest zbyt liczny, można zastosować dodatkowe kryteria. Pewne związane z tym problemem pomysły zostały zasygnalizowane podczas analizy konkretnych przypadków. Można również zastanawiać się nad celowością – w pewnych sytuacjach – niejednakowego traktowania atrybutów poprzez, na przykład, przyporządkowanie im odpowiednich wag. Takie dodatkowe kryteria inicjujące dalszą analizę mogłyby być wprowadzane dynamicznie, w trybie konwersacyjnym. Ważne jest to, że w ogólnym przypadku użytkownik dostaje jako rozwiązanie zbiór obiektów, spośród których może wybrać – automatycznie lub nie – najbardziej mu odpowiadające. Podczas rozwiązywania konkretnego problemu warto zapewnić również możliwość przedstawiania pewnych wyników pośrednich, jak np. wartości typu $\prod_{a \in E} g_a(x)$, gdzie $E \subset AT$, czy zbiory atrybutów, dla których para: uogólniona wartość atrybutu, odpowiedni element namiaru, są rozłączne. Szczególnie istotne jest, że użytkownik systemu może również otrzymać informację o tym, że dla danego namiaru warunek odpowiadający pierwszej wersji problemu namiarowania nie może być spełniony bez względu na stan wiedzy reprezentowanej przez system.

Literatura

[1] Abramowicz A.: Bearings Problem in Nondeterministic Information Systems. Bull. of the PAS, Tech. Sciences, Vol. 36, No. 3-4, 1988, s. 241-250

[2] Abramowicz A.: Dedukcja w systemach informacyjnych zawierających niepełną informację. Techniki Komputerowe Biuletyn Informacyjny, Nr 1, 1996, s. 5-18

[3] Abramowicz A.: Metoda dekompozycji formuł w systemach informacyjnych z niepełną informacją. Techniki Komputerowe Biuletyn Informacyjny, Nr 1, 1997, s. 23-34

[4] Alchourron C., Gardenfors P., Makinson D.: On the logic of theory change: Partial meet contraction and revision function. Journal of Symbolic Logic 50, 1985, s. 510-530

[5] Gardenfors P., Makinson D.: Revision of knowledge systems and epistemic entrenchment. (w rękopisie), 1988

[6] Mrózek A.: Rough sets and some aspects of expert system realization. (w rękopisie), 1988

[7] Orlowska E.: Logic of nondeterministic information. Studia Logica XLIV, 1985, s. 93-102

[8] Orlowska E., Pawlak Z.: Logical foundations of knowledge representation. ICS PAS Reports 537, 1984

[9] Pawlak Z.: Systemy informacyjne. Podstawy teoretyczne. WNT, Warszawa, 1983

ROMAN CZAJKOWSKI, WOJCIECH NOWAKOWSKI
INSTYTUT MASZYN MATEMATYCZNYCH WARSZAWA
MARCIN PAŚNIKOWSKI
POLITECHNIKA WARSZAWSKA

Analiza możliwości automatycznej generacji polskich znaków w fontach postscriptowych Opis programu – część druga

An analysis of the possibilities of the automatic generation of Polish fonts in the Postscript format A description of the program – Part Two

Streszczenie

W artykule omówiono strukturę i zadania poszczególnych modułów kolejnych bloków programu automatycznej generacji polskich znaków w fontach postscriptowych.

Abstract

The article discusses the structure and tasks of every module within the following blocks of a program that automatically generates Polish fonts in the Postscript format.

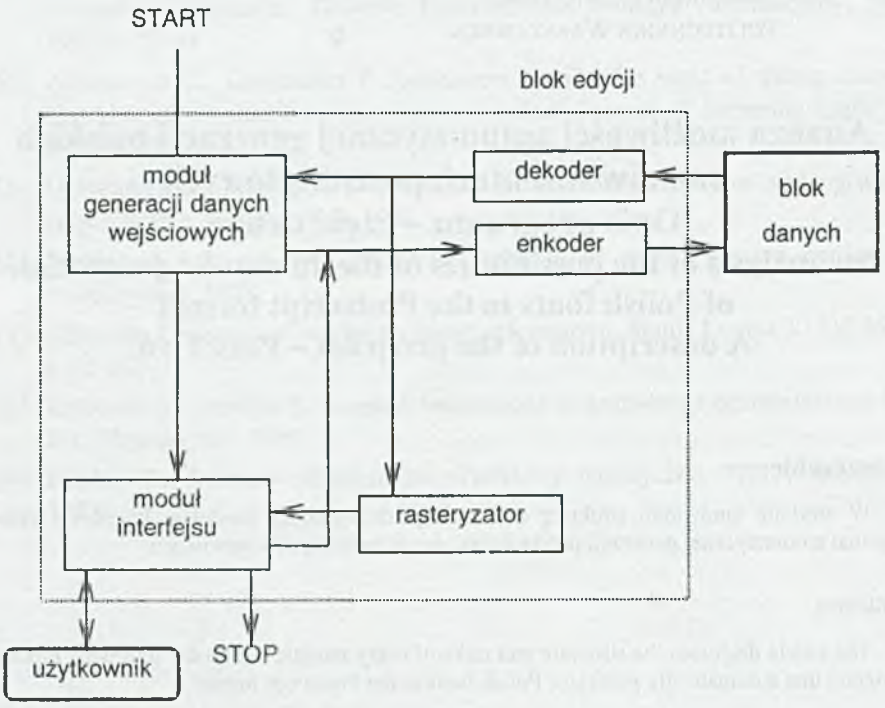
1.3. Blok danych

Przeznaczeniem bloku danych jest magazynowanie informacji wykorzystywanych w procesie obróbki fontu. Pełni on rolę bazy danych gromadzącej i udostępniającej informacje na żądanie pozostałych części programu. Blok danych został zorganizowany w postaci klasy (rozumianej w sensie techniki programowania zorientowanej obiektowo) zawierającej dane oraz operujące na nich funkcje. Dane zorganizowane są w postaci wzajemnie powiązanych struktur o hierarchii odpowiadającej budowie słowników fontowych opisanych uprzednio. Zadaniem funkcji jest wykonywanie zadań związanych z inicjacją struktur, alokacją i usuwaniem danych z pamięci oraz dbanie o zachowanie spójności referencyjnej pomiędzy danymi.

Blok danych jest widziany przez program jako zamknięty obiekt gromadzący i udostępniający dane przez odpowiednie funkcje interfejsu. Większość czynności związanych z administracją danych jest wykonywana przez blok w sposób automatyczny, odciążając tym samym pozostałe modułu programu.

1.4. Blok edycji

Zadaniem bloku edycji jest stworzenie mechanizmów, które dałyby użytkownikowi możliwość edycji polskich znaków. Pod pojęciem tym rozumiany jest proces ustalający położenie znaków diakrytycznych względem liter podstawowych, jak również edycja innych parametrów opisujących czcionkę. Schemat bloku edycji z podziałem na poszczególne moduły przedstawiony jest na rys. 11.



Rys. 11 Schemat bloku edycji

Zadaniem modułu generacji danych wejściowych jest przygotowanie danych do procesu edycji. Oznacza to wygenerowanie polskich liter z domyślnie przyjętą orientacją znaku diakrytycznego względem litery podstawowej. Dane potrzebne do przeprowadzenia tej operacji pobierane są z bloku danych poprzez moduł dekodera. Jest on niezbędny, ponieważ informacje przechowywane są w bloku danych w postaci zakodowanej. Rezultat działalności modułu generacji danych wejściowych jest zapisywany w bloku danych. W procesie tym pośredniczy moduł enkodera wykonujący funkcje odwrotne względem funkcji realizowanych przez dekoder.

Po tych czynnościach wstępnych można uruchomić moduł interfejsu. Jest to ta część oprogramowania, z którą użytkownik ma bezpośredni kontakt. Jej celem jest stworzenie warunków dla interaktywnej edycji czcionki. Zależnie od typu, informacje przeznaczone do wizualizacji dostarczane są do modułu interfejsu na dwa różne

sposoby. W przypadku danych o charakterze tekstowym, np. nazwa czcionki, czy rodziny do której należy, dane są pobierane bezpośrednio z bloku danych. W przypadku, gdy są to informacje opisujące kształty poszczególnych znaków, należy pomiędzy blokiem danych a modułem interfejsu zastosować moduł rasteryzatora. Jego zadanie polegać będzie na konwertowaniu danych z postaci poleceń języka *BuildChar Postscript* na bezpośrednio zrozumiałe przez interfejs ciągi rozkazów graficznych, umieszczających punkty na ekranie monitora. Nowe dane, powstałe w wyniku interakcji zachodzącej pomiędzy użytkownikiem a modułem interfejsu, zapisywane będą ponownie w bloku danych.

1.4.1. Moduł dekodera

Dane opisujące kształty znaków zlokalizowane są w dwóch miejscach. Pierwszym z nich jest słownik *CharStrings*, drugim – element słownika *Private* o nazwie *Subrs*. Dane te przechowywane są w postaci zaszyfrowanej i zakodowanej. Przyjęcie takiego rozwiązania jest tłumaczone przez firmę Adobe troską o to, by żaden przypadkowy czynnik nie mógł ingerować w ich zawartość. Zadaniem modułu dekodera będzie złamanie tego zabezpieczenia.

Pierwszym koniecznym krokiem jest zaimplementowanie algorytmu deszyfrującego, co nie nastręcza większych trudności, ponieważ jest to ten sam algorytm, jaki został użyty do deszyfrowania zbioru ***.pfb**. Inna jest tylko wartość inicjująca klucz kodujący **R**. W tym przypadku musi ona wynosić 4330.

Po rozszyfrowaniu *charstring'u* i odrzuceniu czterech pierwszych bajtów otrzymujemy strumień bajtów w postaci zakodowanej. Operacja dekodowania polega na zinterpretowaniu i przetłumaczeniu poszczególnych zakodowanych bajtów na elementy języka *BuildChar Postscript*.

Do kodowania poleceń przeznaczone są bajty o numerach zawartych w przedziale od 0 do 31. Wszystkie polecenia kodowane są za pomocą jednego lub dwóch bajtów. Poniżej przedstawiono listę poleceń i odpowiadających im kodów (rys. 12).

Oprócz poleceń, w języku *BuildChar Postscript* występują jeszcze liczby całkowite. Są one kodowane za pomocą jednego, dwóch lub pięciu bajtów o kodach z zakresu od 32 do 255. Przyjęto następujące zasady kodowania:

1. Bajt (dla ustalenia uwagi nazwijmy go v) o kodzie z zakresu od 32 do 246 oznacza liczbę całkowitą o wartości:
   $v - 139$
   W rezultacie daje to przedział liczb całkowitych z zakresu od -107 do 107.

2. W przypadku bajtu v o kodzie zawierającym się w przedziale od 247 do 250 należy wziąć kolejny bajt w i wyznaczyć odpowiadającą im liczbę całkowitą na podstawie wzoru:
   $[(v - 247) * 256] + w + 108$
   Oznacza to, że do zakodowania liczby z przedziału <108, 1131> potrzeba dwóch bajtów.

Kod bajtu	komenda
1	*hstem*
3	*vstem*
4	*vmoveto*
5	*rlineto*
6	*hlineto*
7	*vlineto*
8	*rrcurveto*
9	*closepath*
10	*callsubr*
11	*return*
12,0	*dotsection*
12,1	*vstem3*
12,2	*hstem3*
12,6	*seac*
12,7	*sbw*
12,12	*div*
12,16	*callothersubr*
12,17	*pop*
12,33	*setcurrentpoint*
13	*hsbw*
14	*endchar*
21	*rmovet*
30	*vhcurveto*
31	*hvcurveto*

Rys. 12 Lista poleceń i odpowiadających im kodów

3. Bajty z zakresu od 251 do 254 odpowiadają liczbie wyrażonej wzorem:
   - [(*v* - 251) * 256] - *w* - 108
   gdzie *w* jest kolejnym bajtem występującym po bajcie *v*. Daje to liczby z zakresu <-1131, -108>
4. Ostatni przypadek ma miejsce, gdy bajt *v* posiada wartość 255. Wtedy cztery następne bajty należy interpretować jako czterobajtową liczbę całkowitą ze znakiem, bajt pierwszy jest bajtem najbardziej znaczącym. Oznacza to, że największy zakres liczb, jakie można zakodować, jest równy ±32000.

Algorytm dekodujący ma postać pętli programowej skanującej kolejne bajty i tłumaczącej je na elementy języka. W przypadku napotkania niedozwolonych kodów, procedura jest przerywana i sygnalizowane jest o wystąpienie błędu.

1.4.2. Moduł enkodera

Moduł enkodera jest to ta część bloku edycji, która realizuje proces transmisji danych pomiędzy wewnętrznymi strukturami bloku edycji a blokiem danych. Zasada budowy i funkcjonowania jest analogiczna jak w module dekodera. Jedyną cechą odróżniającą te dwa moduły jest to, że moduł enkodera wykonuje czynności odwrotne

niż moduł dekodera. Oznacza to, że enkoder realizuje najpierw operację kodowania poleceń języka *BuildChar Postscript*, a następnie poddaje je procesowi szyfrowania.

1.4.3. Moduł rasteryzatora

Rezultatem działania modułu dekodera jest ciąg rozkazów opisujących kontury poszczególnych znaków. Zadaniem modułu rasteryzatora jest przetworzenie ich do formatu bezpośrednio zrozumiałego przez plecenia graficzne, sterujące procesem umieszczania punktów na ekranie monitora. Językiem użytym do definiowania konturów jest wydzielony fragment języka *Postscript* określany mianem *Type 1 BuildChar Postscript*. Ponieważ dalsze etapy omawiania bloku odczytu wymagają wiedzy o tym języku, konieczne jest krótkie wprowadzenie w jego budowę.

BuildChar Postscript jest to zbiór dwudziestu sześciu poleceń, które ze względu na charakter i przeznaczenie można podzielić na pięć grup tematycznych. Są to polecenia:

- rozpoczynające i kończące definicję znaku,
- konstruujące kontury znaku,
- odpowiedzialne za proces *hintingu*,
- arytmetyczne,
- *subroutine commands*, czyli polecenia odpowiedzialne za dynamiczną zmianę hintingu w ramach pojedynczego znaku.

Większość z tych 26 poleceń pobiera na wejściu pewną liczbę argumentów. Oczywiście, jak zwykle języku *Postscript*, są one pobierane ze stosu. Dla potrzeb języka *BuildChar Postscript* wprowadzono, specjalnie do tego celu przeznaczony, stos o nazwie *Type 1 BuildChar Stack*. Sposób jego organizacji jest nieco odmienny od sposobu budowy i funkcjonowania pozostałych stosów, używanych w postscripcie. Inny jest mianowicie porządek zdejmowania elementów znajdujących się na stosie. Elementy pobierane są nie z wierzchołka, ale ze spodu stosu. Przyczyną takiej odmiennej obsługi stosu jest to, że polecenia oczekują parametrów wejściowych w takiej kolejności, w jakiej zostały na stos położone.

Ograniczono również pojemność stosu do 24 elementów. Wprowadzono także ograniczenie co do typu elementów mogących spoczywać na stosie. Mogą to być tylko i wyłącznie dane numeryczne w formacie całkowitym.

Kolejnym ograniczeniem wprowadzonym do techniki *BuildChar* jest warunek, że na stos mogą być odkładane argumenty używane wyłącznie przez następny rozkaz. Niedopuszczalne gromadzenie na stosie argumentów przeznaczonych dla później uruchamianych poleceń.

Poniżej przedstawiono listę 25 poleceń wchodzących w skład języka *BuildChar Postscript* z podziałem na grupy tematyczne. Znak ├ znajdujący się przed nazwą polecenia oznacza, że polecenie pobiera ze stosu argumenty wejściowe. Są one wyspecyfikowane tuż po tym znaku. Analogicznie, znak ┤ następujący po poleceniu wskazuje, że polecenie umieszcza argumenty wyjściowe na stosie. W pozostałych przypadkach, w których nie ma odwołania do zawartości stosu, występuje znak –.

Polecenia rozpoczynające i kończące definicję znaku:

sbw – sbx sby wx wy **hsbw** ├

Polecenie *sbw* przypisuje punktowi *left sidebearing point* współrzędne o wartości (*sbx, sby*), a wektorowi *character width vector* wartość (*wx, wy*). Przesuwa również punkt bieżący, tj. punkt, w którym rozpoczyna się kreślenie nowego elementu ścieżki, do punktu o współrzędnych (*sbx, sby*). Należy podkreślić, że punkt ten nie należy jeszcze do znaku. Do określenia pierwszego punktu rozpoczynającego definicję konturu znaku należy użyć rozkazu *rmoveto*. Rozkaz *sbw* lub jego uproszczona postać *hsbw* musi rozpoczynać definicję każdego znaku i może zostać użyta tylko raz.

hsbw ├ sbx wx **hsbw** ├

Jest to szczególny przypadek polecenia *sbw*. Stosuje się go w przypadku, gdy parametry *sby* i *wy* przyjmują wartości zerowe. Jest to najczęściej spotykany przypadek. Rezultatem wykonania tego polecenia jest przypisanie *left sigebearing point* wartości (*sbx,* 0), a *character width vector* – (*wx,* 0). Podobnie jak *sbw*, *hsbw* przesuwa punkt bieżący, nie włączając go jednocześnie do ścieżki definiującej znak. W przypadkach znaków nie zawierających konturów, np. spacji, *left sidebearing point* powinien być równy (0, 0).

seac ├ asd adx ady bchar achar **seac** ├

Jest to bardzo ważne polecenie z punktu widzenia celu omawianego tematu. Przy generowaniu polskich liter będą wykorzystywane mechanizmy, jakich dostarcza polecenie *seac*. Służy ono do tworzenia liter złożonych z dwóch innych znaków zdefiniowanych w foncie. W większości przypadków jest to znak akcentu i litery podstawowej. Parametr *asb* to współrzędna *x*-owa *left sidebearing point* tworzonego znaku i musi być równa *x*-wej współrzędnej *left sidebearing point* samego akcentu. Punkt *origin* akcentu jest przesunięty względem punktu *origin* znaku podstawowego o wektor (*adx, ady*). O tym z jakich komponentów składa się tworzona litera decydują dwa pozostałe argumenty: *bchar* i *achar*. *bchar* jest numerem znaku podstawowego, a *achar* – znaku akcentowego. Numery te są określane jako pozycje, pod którymi znaki występują w wektorze kodowym. Oznacza to, że nie można za pomocą polecenia *seac* zbudować znaku, którego składniki nie występują w wektorze kodowym.

endchar – **endchar** ├

Polecenie to musi zawsze wystąpić jako ostatni element definicji każdego znaku (wyjątkiem są tu znaki, do których definicji użyto rozkazu *seac*). Rezultatem jego wykonania jest podjęcie przez *Type 1 BuildChar* szeregu działań, w wyniku których wcześniej zdefiniowane kontury są wypełniane i umieszczane na stronie dokumentu. Oczywiście, nie zawsze kontury muszą być wypełniane, zależy to od wartości zmiennej *PaintType*.

Polecenia konstruujące kontury znaku:

closepath – **closepath** ├

closepath zamyka daną podścieżkę tworząc zamknięty kontur. Operacja ta jest wykonywana przez dodanie do podścieżki odcinka zaczynającego się w punkcie bieżącym i kończącym na pierwszym punkcie danej podścieżki (ustalonym poleceniem *rmoveto* lub jednym z jego uproszczonych wariantów). *closepath* nie modyfikuje położenia punktu bieżącego. Jego zmianą trzeba wymusić jawnie użyte polecenie *rmoveto*, dopiero wtedy można przystąpić do tworzenia kolejnych ścieżek. Stosując polecenie *closepath* należy upewnić się, czy w wyniku jego działania nie stworzono ścieżki o długości zerowej lub ścieżki złożonej z pokrywających się odcinków. Twórcy Type 1 zdecydowanie zalecają używanie tego polecenia wszędzie tam, gdzie zachodzi potrzeba dotknięcia ścieżek. Eliminuje to ewentualne błędy i znacznie upraszcza opis konturów.

rlineto ├ dx dy **rlineto** ├

Wszystkie polecenia uczestniczące w procesach tworzenia konturów znaków operują na współrzędnych względnych. Oznacza to, że nie posługują się współrzędnymi określanymi względem początku układu współrzędnych (*character space*), ale względnymi przesunięciami względem punktu bieżącego. Właśnie dlatego większość z tych poleceń rozpoczyna się od litery *r* (ang. *relative*). Zadaniem polecenia *rlineto* jest wykreślenie odcinka linii prostej rozpoczynającego się w punkcie bieżącym i kończącym się w punkcie $(cp_x + dx, cp_y + dy)$, gdzie cp_x i cp_y odpowiadają współrzędnym punktu bieżącego. W przypadku tego polecenia, jak również dwóch jego szczególnych przypadków: *vmoveto*, *hmoveto*, punkt bieżący jest przesuwany o wektor [*dx, dy*].

hlineto ├ dx **hlineto** ├

Jest to szczególny przypadek polecenia *rmoveto* przeznaczony do kreślenia linii poziomych. Jest równoważny poleceniu *dx 0 rlineto*.

vlineto ├ dy **vlineto** ├

Szczególny przypadek polecenia *rlineto*, równoważny poleceniu 0 *dy rlineto*. Jego celem jest kreślenie linii pionowych.

rmoveto ├ dx dy **rmoveto** ├

Przesuwa punkt bieżący o wektor [*dx, dy*].

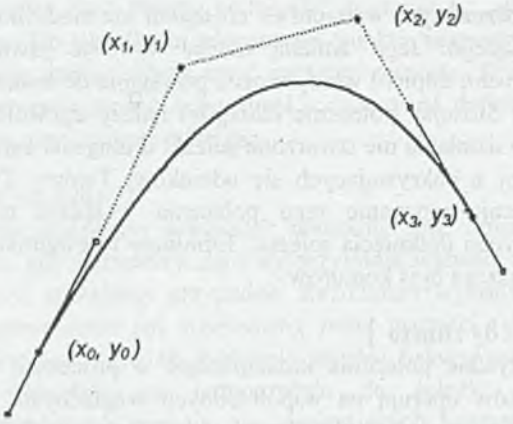
hmoveto ├ dx **hmoveto** ├

Wykonuje przesunięcia punktu bieżącego w poziomie o wartość *dx*, równoważne poleceniu *dx* 0 *rmoveto*.

vmoveto ├ dy **vmoveto** ├

Wykonuje przesuniecie punktu bieżącego w pionie o wartość *dy*; jest równoważne poleceniu 0 *dy rmoveto*.

rrcurveto ├ dx1 dy1 dx2 dy2 dx3 dy3 **rrcurveto** ┤
Polecenie to powoduje wykreślenie krzywej Béziera. Jest to krzywa trzeciego stopnia rozpoczynająca się w punkcie o współrzędnych oznaczonych na rys. 13 jako (x_0, y_0) i kończąca się w punkcie (x_3, y_3). Punkty (x_1, y_1) i (x_2, y_2) są punktami sterującymi.



Rys. 13 Krzywa Béziera

Poniżej opisano sposób konstruowania krzywej Béziera. W momencie rozpoczynania kreślenia krzywa jest styczna w punkcie startowym (x_0, y_0) do odcinka $(x_0, y_0) - (x_1, y_1)$. Im dłuższy jest ten odcinek, tym bardziej krzywa do niego przylega. Analogicznie jest w punkcie końcowym, tj. (x_3, y_3). Analityczna formuła opisująca krzywą Béziera przedstawia się następująco:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + x_0$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + y_0$$

gdzie parametr t jest liczbą rzeczywistą z przedziału <0, 1>, a parametry a_x, b_x, c_x i a_y, b_y, c_y są funkcjami punktów sterujących:

$$a_x = x_3 - 3x_2 + 3x_1 - x_0$$
$$b_x = 3x_2 - 6x_1 + 3x_0$$
$$c_x = 3x_1 - 3x_0$$

$$a_y = y_3 - 3y_2 + 3y_1 - y_0$$
$$b_y = 3y_2 - 6y_1 + 3y_0$$
$$c_y = 3y_1 - 3y_0$$

Jest to jedyny opis linii krzywych stosowanych w *BuildChar Type 1*. Oznacza to, że wszystkie łuki wchodzące w skład znaku kreślone są krzywymi Béziera.

Punkty sterujące (x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3) dla polecenia *rrcurveto* wyznacza się następująco:

(x_0, y_0) = *[współrzędne punktu bieżącego]*
$(x_1, y_1) = (x_0 + dx_1, y_0 + dy_1)$
$(x_2, y_2) = (x_0 + dx_1 + dx_2, y_0 + dy_1 + dy_2)$
$(x_3, y_3) = (x_0 + dx_1 + dx_2 + dx_3, y_0 + dy_1 + dy_2 + dy_3)$

Polecenie *rrcurveto* przesuwa punkt bieżący do punktu (x_3, y_3).

hvcurveto ├ dx1 dx2 dy2 dy3 **hvcurveto** ┤
Jest to szczególny przypadek polecenia *rrcurveto*, w którym odcinki wyznaczone przez pary punktów sterujących $(x_0, y_0) - (x_1, y_1)$ i $(x_2, y_2) - (x_3, y_3)$ ułożone są w kierunkach pionowym i poziomym. Odpowiada to wywołaniu polecenia *rrcurveto* w następującej postaci:
*dx*1 0 *dx*2 *dy*2 0 *dy*3 *rrcurveto.*

vhcurveto ├ dy1 dx2 dy2 dx3 **vhcurveto** ┤
Kolejna odmiana polecenia *rrcurveto*, równoważna z postacią:
0 *dy*1 *dx*2 *dy*2 *dx*3 0 *rrcurveto.*

Polecenia odpowiedzialne za proces *hintingu*

hstem ├ y dy **hstem** ┤
Wyznacza *horizontal stem zone* – poziomy pas płaszczyzny zawarty pomiędzy rzędnymi o wartościach równych *y* i *y* + *dy*, gdzie współrzędna y jest wyrażona względem punktu *origin*.

vstem ├ x dx **vstem** ┤
Wyznacza *vertical stem zone*, jako pionowy pas płaszczyzny zwarty pomiędzy odciętymi o wartościach *x* i *x*+*dx*, gdzie współrzędna *x* jest wyrażona względem punktu *origin*.

hstem3 ├ y0 dy0 y1 dy1 y2 dy2 **hstem3** ┤
Wyznacza trzy *horizontal stem zones*, jako trzy pasy płaszczyzny ograniczone odpowiednio przez rzędne o wartościach *y*0 i *y*0+*dy*0, *y*1 i *y*1+*dy*1 oraz *y*2 i *y*2+*dy*2. Polecenie to została specjalnie wprowadzona do definiowania *hintingu* w znakach składających się z trzech poziomych elementów o jednakowych grubościach, przy zachowaniu jednakowych odległości pomiędzy nimi. Przykładem takich znaków może być symbol równoważności lub operator dzielenia.

vstem3 ├ x0 dx0 x1 dx1 x2 dx2 **vstem3** ┤
Wyznacza trzy *vertical stem zones* jako trzy pasy płaszczyzny pomiędzy odciętymi o wartościach *x*0 *i* *x*0+*dx*0, *x*1 i *x*1+*dx*1 oraz *x*2 i *x*2+*dx*2. Przykładem znaku, w którym należy użyć polecenia *vstem*3 zamiast trzech poleceń *vstem* jest litera *m*.

dotsection – **dotsection** ├
Polecenie to rozpoczyna i kończy sekcję definiującą symbol kropki w znakach *i, j* oraz *!*. Mówi ona, iż sekcję tę należy traktować jako oddzielny element, a nie jako część znaku głównego, i co za tym idzie, nie należy stosować w stosunku do niej *hintingu* zdefiniowanego dla znaku podstawowego. W chwili obecnej *BuildChar Postscript* dysponuje znacznie lepszym mechanizmem zmiany *hintingu* wewnątrz znaku *vstem, hstem, vstem*3 i *hstem*3 niż omawiane polecenie *dotsection*. Polecenie *dotsection* jest nadal stosowane jedynie w celu zachowania kompatybilności ze starszymi wersjami interpreterów.

Polecenie arytmetyczne

div num1 num2 **div** quotient
Przeznaczona jest do realizowania operacji dzielenia. Argumenty wejściowe *num1* i *num2* to dwie liczby całkowite a *quotient* to liczba rzeczywista równa ich ilorazowi *num1/num2*. W języku *BuildChar Postscript* przewidziano jedynie liczby całkowite. Jednak w sytuacjach, gdy taka precyzja okazuje się niewystarczającą, stosuje się polecenie *div*. W sytuacji gdy zachodzi np. potrzeba zapisania liczby 1.5, stosuje się notację w postaci 2 3 div.

Subroutine commands:

callsubr subr# **callsubr** –
Polecenie *callsubr* służy do uruchomienia wybranej procedury o numerze *subr#* zdefiniowanej w słowniku *Private* pod pozycją *Subrs*. W przypadku gdy procedura oczekuje argumentów wejściowych, są one umieszczane na stosie przed argumentem *subr#*.

return – **return** ├
return realizuje operację powrotu z procedury *Subrs* uruchomianej poleceniem *callsubr* i przekazuje sterowanie do programu głównego.

pop – **pop** numer
Przenosi liczbę ze stosu operandów na stos *Type 1 BuildChar*. Używana jest w połączeniu z poleceniem *callothersubr*.

callothersubr arg1 ... argn n othersubr# **callothersubrs** –
Przeznaczeniem *callothersubr* jest dynamiczna zmiana *hintingu* wewnątrz pojedynczego znaku.
Polecenia *hstem* i *vstem* definiują strefy *hintingu*, obowiązujące wewnątrz całego znaku. Może się jednak zdarzyć, że dla pewnego fragmentu ścieżki opisującej znak trzeba zastosować odrębne strefy *hintingu*. Definiuje się je wtedy wewnątrz procedury *Subrs* o dowolnie wybranym numerze lecz zawsze większym od trzech. Dla ustalenia uwagi załóżmy, że ma ona numer *subr#*. Następnie w miejscu ścieżki,

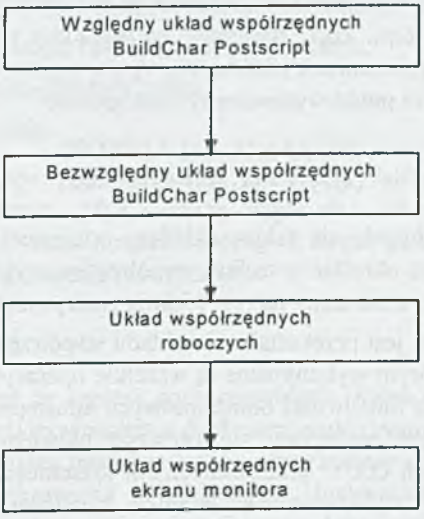
w którym należy dokonać zmiany *hinting*, uruchamia się ją. Sprawa byłaby prosta, gdyby nie to, że starsze wersje interpreterów nie potrafią realizować operacji dynamicznej zmiany *hintingu*. Aby zachować kompatybilność z poprzednimi wersjami wprowadzono polecenie *callothersubrs*. Jego zadanie polega na stwierdzeniu czy interpreter jest w stanie zrealizować omawiany mechanizm czy nie. Przykładowy sposób wywołania polecenia *callothersubrs* wygląda następująco:

Subr# 1 3 **callothersubr** pop callsubr

Uruchamia ona procedurę *OtherSubrs* o numerze 3. Sprawdza, czy interpreter jest w stanie wykonać operację zmiany *hintingu*. Jeśli tak, to kładzie na stos operandów liczbę *subr#*, jeśli nie – liczbę 3. Następnie polecenie *pop* przenosi tę liczbę na stos *Type 1 BuildChar* i polecenie *callsubr* wywołuje procedurę *Subrs* o zadanym numerze. W przypadku, gdy omawiany mechanizm nie jest implementowany, uruchamiana jest procedura o numerze 3, zawierająca jedynie kod powrotu *return*.

Po zapoznaniu się z poleceniami języka *BuildChar Postscript* można przystąpić do realizacji algorytmu opisującego działanie rasteryzatora. Pierwszym napotkanym problemem jest wybór odpowiednich układów współrzędnych. Jak wiadomo, język *BuildChar* posługuje się względnym układem współrzędnych, a ekran monitora, będący urządzeniem docelowym wykorzystuje bezwzględny układ odniesienia, w którym wielkość jednostki podstawowej zależy od rozdzielczości karty graficznej, a położenie początku układu współrzędnych znajduje się w górnym lewym rogu ekranu monitora. Należy więc opracować zbiór przekształceń, które bez względu na parametry środowiska pracy dawałyby zamierzony rezultat.

Po przeanalizowaniu zagadnienia okazało się, że najlepszym rozwiązaniem jest zastosowanie modelu przekształceń przedstawionego na rys. 14.

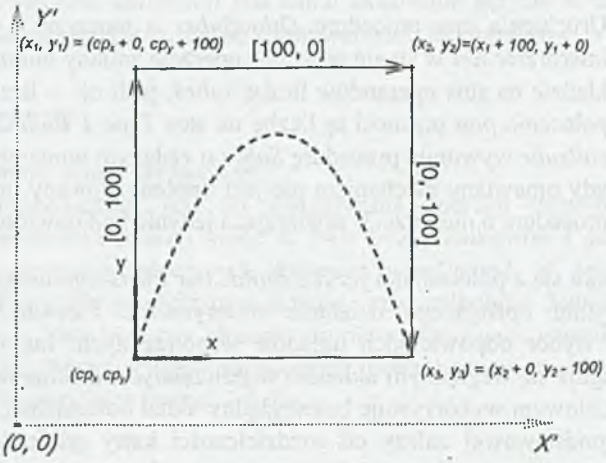


Rys. 14 Model przekształceń układów współrzędnych

Punktem wyjścia jest względny układ odniesienia *BuildChar Postscript*. Względność tego układu oznacza, że istnieje on tylko w ramach pojedynczego punktu, a współrzędne kolejnych punktów określane są w postaci przesunięć względem punktu poprzedniego. Dla lepszego zobrazowania tej sytuacji rozważmy jako przykład rozkaz

0 100 100 0 0 – 100 **rrcurveto**

generujący krzywą Béziera jak na rys. 15.



Rys. 15 Względny układ współrzędnych *BuildChar PostScript*

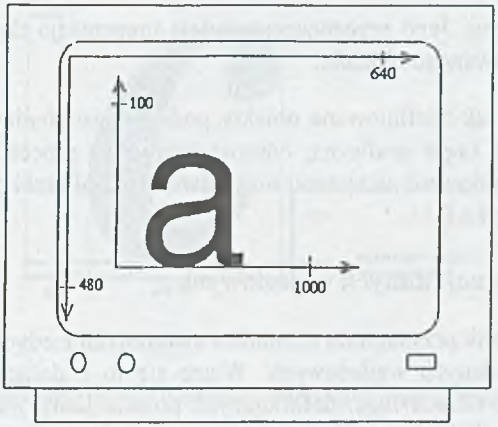
Początek krzywej znajduje się w pewnym nieznanym punkcie płaszczyzny, a jego położenie jest rezultatem wykonania wcześniejszych poleceń. Przyjmijmy, że punkt ten jest punktem bieżącym (cp_x, cp_y). Następnie wyznaczamy pierwszy punkt sterujący krzywej, powstały jako przesunięcie punktu (cp_x, cp_y) o wektor [0, 100] i oznaczamy go jako (x_1, y_1). Drugi i trzeci punkt wyznaczamy analogicznie:

$$(x_2, y_2) = (x_1, y_1) + [100, 0]$$
$$(x_3, y_3) = (x_1, y_1) + [0, -100]$$

Jak widać, posługiwanie się takim układem odniesienia jest dość kłopotliwe. Znacznie wygodniej jest określać wszelkie współrzędne wykorzystując bezwzględny układ odniesienia OX'Y', oznaczony na rys. 15 linią przerywaną.

Następnym krokiem jest przekształcenie układu współrzędnych OX'Y' na roboczy układ odniesienia, w którym wykonywane są wszelkie operacje obliczeniowe związane z blokiem edycji. Wobec możliwości obliczeniowych udostępnianych przez kompilator języka C++ wygodnie jest posługiwać się roboczym układem odniesienia związanym z układem współrzędnych OX'Y' przekształceniem tożsamościowym. W tak zdefiniowanym układzie OX"Y" wykonywane są wszelkie transformacje graficzne typu przesuwanie akcentu względem litery podstawowej itp.

Ostatni krok to przeniesienie układu odniesienia OX″Y″ na współrzędne ekranu monitora OX‴Y‴. Zważywszy, że standardowym trybem graficznym stosowanym w systemie DOS jest tryb SVGA, założono, że interfejs będzie pracował w rozdzielczości 640 na 480 punktów. Transformacja układu OX″Y″ do OX‴Y‴ wymaga odpowiedniego przeskalowania i przesunięcia, tak aby początek układu współrzędnych znalazł się w pobliżu dolnego lewego rogu ekranu (patrz rys. 16).



Rys. 16 Układ odniesienia na ekranie monitora

Dysponując modelem układu współrzędnych i biblioteką procedur graficznych języka C++, zaimplementowano funkcje modułu rasteryzatora, interpretujące polecenia *BuildChar Postscript*. Wśród nich znalazły się polecenia należące do trzech spośród pięciu grup zdefiniowanych na początku rozdziału, to znaczy:

- polecenia rozpoczynające i kończące definicję znaku,
- polecenia konstruujące kontury znaku,
- polecenia arytmetyczne.

Pozostałe dwie grupy odpowiedzialne za proces *hintingu* nie biorą udziału w procesie tworzenia obrazu, gdyż medium wyjściowe, jakim jest ekran monitora, dysponuje wystarczająco dużą rozdzielczością. Z tego powodu nie ma potrzeby stosowania tych poleceń w module rasteryzatora.

1.4.4. Moduł interfejsu

Moduł interfejsu jest tą częścią oprogramowania, która uczestniczy w procesie komunikacji pomiędzy użytkownikiem a funkcjami realizowanymi przez aplikację. Od sposobu jego realizacji zależy funkcjonalność i przejrzystość obsługi całego programu. Pierwszym etapem projektowania interfejsu jest zbudowanie zbioru obiektów, na których bazować będzie proces dialogu pomiędzy oprogramowaniem a obsługującą go osobą. Zdefiniowano następujące obiekty podstawowe:

- obiekt typu *menu poziome*. Jest to pozioma belka, usytuowana w dowolnie wybranym miejscu ekranu, umożliwiająca dokonania wyboru jednej spośród paru zdefiniowanych opcji;
- obiekt typu *menu pionowe*. Został zrealizowany w postaci listy o dowolnej długości, przedstawianej na ekranie w postaci okienka o przewijanej zawartości;
- obiekt typu *desktop*. Jest to tło, na którym umieszczane są pozostałe obiekty. Pełni rolę dekoracyjną;
- obiekt typu *okno*. Jego przeznaczeniem jest prezentacja danych i komunikatów w postaci stosowanego okienka.

Wykorzystując tak zdefiniowane obiekty podstawowe zbudowano tekstową część interfejsu. Natomiast część graficzną, odpowiedzialna za proces wizualizacji i edycji znaków polskich, zbudowano w oparciu o standardowe biblioteki graficzne dostarczane przez kompilator języka C++.

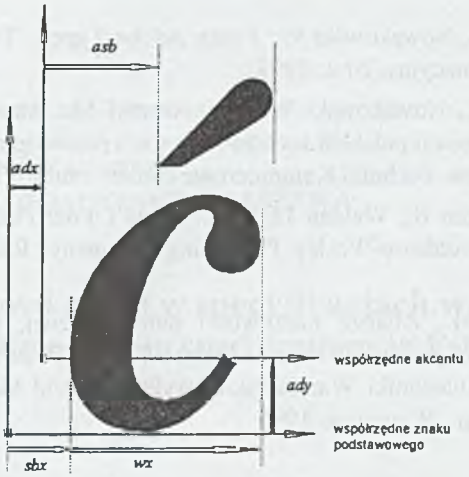
1.4.5. Moduł generacji danych wejściowych

Zanim użytkownik przystąpi do czynności związanych z edycją znaków, konieczne jest przygotowanie danych wejściowych. Wiąże się to z dołączeniem dodatkowych pozycji do słownika *CharStrings*, definiujących polskie litery z domyślnie przyjętym położeniem znaków diakrytycznych względem liter podstawowych. Koniecznym jest automatyczne wygenerowanie wszystkich polskich liter za wyjątkiem małych i wielkich liter *ó* i *ł*, gdyż te są już zdefiniowane w czcionkach.

Do tworzenia polskich znaków wykorzystany zostanie, opisany w rozdziale 1.4.3. rozkaz *seac*. Jest on przeznaczony do generowania znaków złożonych z więcej niż jednego elementu składowego. Nadaje się więc znakomicie do realizacji zamierzonego celu. Ciąg rozkazów definiujący *charstring* złożony z dwóch znaków prostych przedstawia się następująco:

sbx wx **hsbw**
asb adx ady bchar achar **seac**

Znaczenie poszczególnych parametrów zostało objaśnione na poniższym rysunku.



Rys. 17. Konstrukcja znaku akcentowego za pomocą polecenia *seac*

Parametry *achar* i *bchar* określają, z jakich znaków składać się będzie tworzona litera. Mówiąc bardziej precyzyjnie są to numery pozycji, pod którym znaki te występują w wektorze kodowym. Na powyższym rysunku, *bchar* odpowiada literze *a*, *achar* – znakowi akcentu. Parametry *adx* i *ady* określają położenie akcentu względem litery podstawowej. Jest ono definiowane jako przesunięcie początku układu współrzędnych akcentu względem początku układu współrzędnych litery podstawowej i jest wyrażane w jednostkach równych 1/1200 cala.

Każdy znak musi posiadać dwie miary: *left sidebearing bpoint* i długość znaku. Pierwsza z tych miar jest definiowana przez parametr *sbx*, druga przez *wx*. Ostatni z parametrów *asb* określa położenie *left sidebearing point* akcentu.

W przypadku jak na rys. 17 parametry te przyjmują następujące wartości:

41 516 **hsbw**
189 240 10 99 194 **seac**

Po zakończeniu procesu edycji blok danych zawiera komplet informacji na temat polskich znaków, które umożliwiają wygenerowanie spolonizowanej wersji czcionki. Zadanie to realizuje ostatni blok funkcjonalny zwany blokiem zapisu.

Opis kolejnych bloków będzie przedstawiony w trzeciej części artykułu.

Literatura:

[1] Czajkowski R., Nowakowski W.: Fonty Adobe Type 1. Techniki Komputerowe Biuletyn Informacyjny, Nr 1, 1997

[2] Czajkowski R., Nowakowski W., Paśnikowski M.: Analiza możliwości automatycznej generacji polskich znaków w fontach postscriptowych. Opis programu – część pierwsza. Techniki Komputerowe Biuletyn Informacyjny, Nr 1, 1998

[3] Brotz D., Paxton B., Walden J.: Adobe Type 1 Font Format / Adobe Systems Incorporated. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1993

[4] Paśnikowski M.: Analiza możliwości automatycznej, programowej syntezy znaków polskich w fontach formatu Adobe Type 1. Praca dyplomowa na Wydz. Elektroniki Politechniki Warszawskiej wykonana pod kier. dr inż. Wojciecha Nowakowskiego, Warszawa, 1996

MAREK KACPRZAK
INSTYTUT MASZYN MATEMATYCZNYCH WARSZAWA

Symulacja upływu czasu w specyfikacjach w języku Estelle
Simulation of time flow in specifications in Estelle language

Streszczenie

W pracy przedstawiono zasady symulacji działania systemów rozproszonych specyfikowanych w języku Estelle. Opisano struktury danych reprezentujące drzewa systemowe specyfikacji w modelu symulacyjnym. Przedstawiono algorytm symulacji sekwencyjnej, w której wszystkie drzewa systemowe są zaalokowane w komputerze jednoprocesorowym.

Abstract

In the paper principles of simulation of distributed systems specified in Estelle language are presented. Data structures representing in simulation model the system trees of specifications are described. Algorithm of sequential simulation with allocation of all system trees onto computer with one processor is also described.

1. Język specyfikacji Estelle

Język Estelle (skrót od Extended State Transition Model – model wykorzystujący rozszerzone systemy przejść) jest narzędziem opisu systemów rozproszonych [1], [2]. Głównymi pojęciami w Estelle są: moduł, specyfikacja i kanał. Definicja modułu określa niedeterministyczny sekwencyjny system przejść (automat) ze skończoną liczbą stanów sterowania. Specyfikacja jest wyróżnionym, głównym modułem. Specyfikacja opisuje hierarchicznie uporządkowany zbiór składników – instancji modułów. Instancje mogą komunikować się między sobą w dwojaki sposób:

- przez przesyłanie interakcji przez dwukierunkowe łącza między punktami interakcji instancji,
- przez zmienne dzielone (w ograniczonym zakresie).

Definicja kanału określa jakie interakcje mogą być przesyłane przez łącze w obu kierunkach. Kanały umożliwiają także deklarację punktów interakcji. Odbierane interakcje są gromadzone w nieskończonych kolejkach wejściowych z regulaminem FIFO. Każdy punkt interakcji jest powiązany z kolejką wejściową.

Instancje modułów mogą tworzyć i usuwać instancje modułów-dzieci, łączyć i rozłączać punkty interakcji, wysyłać interakcje. Specjalny rodzaj instancji modułów – instancje modułów systemowych (zwane dalej krótko systemami) mogą być utworzone, ale nie mogą być usunięte. Specyfikacja określa drzewo, którego korzeniem jest instancja modułu specyfikacji, a gałęziami i liśćmi (jeżeli zostaną powołane) – instancje-potomkowie. W chwili początkowej, po inicjacji, instancja specyfikacji jest korzeniem drzewa zawierającego I systemów S_i, gdzie $i \geq 1$, $i = 1,...,I$. Liczba systemów zależy od inicjacji specyfikacji (niedeterminizm). Drzewo z korzeniem w systemie S_i (stanowiące poddrzewo specyfikacji), nazywane i-tym drzewem systemowym, tworzą instancje składowe: system S_i oraz (jeżeli zostały powołane) instancje-potomkowie M_{ij}, gdzie $j = 1,...,J_i$ (wartość J_i może się zmieniać w „czasie życia" specyfikacji).

Stan instancji modułu tworzą: stan sterowania, wartości pewnych stałych i zmiennych, zawartość kolejek wejściowych, łącza z innymi instancjami modułów. Instancja modułu działa w dwóch fazach: wyboru przejść i wykonania przejść. W fazie wyboru przejść, zależnie od pewnych warunków dotyczących stanu sterowania, interakcji wejściowych i warunków logicznych, jest wyznaczany zbiór przejść do wykonania. W fazie wykonania przejść wykonywane są wybrane przejścia. Przejścia są opisywane jako procedury w języku Pascal. Wykonanie przejścia jest niepodzielne. Możliwe jest opóźnienie wykonania przejścia o wartość czasu określoną rozkładem prawdopodobieństwa; przejścia takie są nazywane opóźnionymi.

Systemy działają wzajemnie równolegle i niezależnie. Każdy system S_i zarządza wszystkimi instancjami-potomkami (jeżeli powołał takie instancje). Zarządzanie przez system S_i polega na synchronizacji rozpoczynania i kończenia faz wyboru przejść i wykonania przejść we wszystkich instancjach-potomkach (działanie synchroniczne składowych drzewa systemowego). Instancje modułów, które należą do różnych drzew systemowych mogą się komunikować wyłącznie przez przesyłanie interakcji (komunikacja przez zmienne dzielone nie jest możliwa).

Pojęcie czasu w Estelle jest wprowadzone w sposób ograniczony. Nie są uwzględniane czasy wyboru przejść, ani czasy wykonania przejść, gdyż zależą od implementacji specyfikowanego systemu rozproszonego, a zatem są nieznane. Pojęcie czasu jest wprowadzone jedynie w odniesieniu do przejść opóźnionych. Przyjęto, że niezależnie od specyfikacji istnieje zewnętrzny „proces upływu czasu", który powoduje zmianę wartości opóźnień przejść opóźnionych. Założono, że upływ czasu jest jednakowy dla wszystkich przejść opóźnionych. Każde inne założenia odnośnie czasu są dopuszczalne.

2. Język Estelle jako język symulacyjny

Bezpośrednio ze specyfikacji systemu rozproszonego w języku Estelle może być utworzony model symulacyjny tego systemu rozproszonego. Symulacja upływu czasu umożliwia symulowanie działania systemu rozproszonego. W niniejszej pracy jest rozpatrywany sekwencyjny model symulacyjny, tzn. symulator jest zrealizowany na komputerze jednoprocesorowym.

W dalszej części pracy specyfikacja systemu rozproszonego w języku Estelle będzie nazywana krótko specyfikacją. Dla celów symulacji konieczne jest wprowadzenie do specyfikacji wartości liczbowych czasów wyboru i wykonania przejść, a także innych uzupełnień, określających szczegóły prowadzenia symulacji. W pracy [3]

zaproponowano sposób wprowadzenia czasów, które w ogólnym przypadku mogą być zmiennymi losowymi o rozkładach zależnych od wartości pewnych zmiennych. Wyliczenie wartości czasu wyboru przejść wymaga wykonania tego wyboru. Wyliczenie czasu wykonania przejścia wymaga wykonania tego przejścia. W specyfikacji dla dowolnego modułu może być wprowadzona definicja wyróżnionego stanu końca symulacji; przejście instancji modułu do tego stanu wyznacza chwilę zakończenia symulacji [3]. W pracy [4] opisano sposób wprowadzenia do specyfikacji wymaganych miar wydajności (takich jak: przepustowość i stopień wykorzystania instancji modułów, długość kolejek interakcji, czasy oczekiwania interakcji na obsłużenie ich w instancjach modułów), które mają być wyliczone jako wynik liczbowy symulacji (standardowymi statystykami są: wartość minimalna, wartość maksymalna, średnia i wariancja). W pracy [5] przedstawiono sposób wprowadzenia do specyfikacji warunków wizualizacji przebiegu symulacji, zapisywania danych do plików i zakończenia symulacji (zakończenie symulacji mogą określać warunki logiczne na zmienne w modelu symulacyjnym, wymagana liczba kroków symulacji, wymagana wartość czasu symulacyjnego, wymagany rzeczywisty czas trwania symulacji). Uzupełnienie specyfikacji w powyższy sposób umożliwia utworzenie kompletnego modelu symulacyjnego bezpośrednio ze specyfikacji.

Idea przejścia od specyfikacji do modelu symulacyjnego jest następująca:

- sekwencyjny model symulacyjny składa się z dwóch części: programu planowania symulacji SPP (ang. Simulation Planning Program) i z list połączonych LL (ang. Linked List), z których każda reprezentuje drzewo systemowe specyfikacji,
- program tłumaczący generuje listy LL bezpośrednio ze specyfikacji,
- program SPP jest niezależny od specyfikacji.

Działanie modelu symulacyjnego polega na:

- zgłaszaniu do programu SPP przez działające równolegle i niezależnie systemy reprezentowane przez listy LL zachodzących zdarzeń i aktualnego zapotrzebowania na przyrost czasu symulacyjnego,
- zwiększaniu przez program SPP wielkości czasu symulacyjnego o dopuszczalne przyrosty i zapewnianiu równoległej pracy systemów,
- dokonywaniu przez systemy, w miarę upływu czasu, wyboru przejść i wykonywaniu wybranych przejść.

W pracy [3] opisano szczegółowo algorytm programu tłumaczącego specyfikacje na listy LL. W niniejszej pracy opisano tylko te struktury danych, które są niezbędne do symulacji specyfikacji.

3. Zasady symulacji specyfikacji

Symulacja upływu czasu w specyfikacjach wymaga wprowadzenia do specyfikacji parametrów czasowych i ustalenia zasad symulacji, niesprzecznych z semantyką języka Estelle.

3.1. Symulacja faz wyboru i wykonania przejść

Działanie systemu S_i rozpoczyna się od fazy wyboru przejść. Instancje składowe S_i wybierają (jeżeli jest to możliwe) autonomicznie jedno przejście oferowane. System S_i pozostaje w fazie wyboru do chwili wybrania do wykonania co najmniej jednego przejścia spośród oferowanych przez instancje składowe S_i. W fazie wykonania przejść są wykonywane równolegle wybrane przejścia we wszystkich wybranych instancjach składowych S_i. Faza wykonania kończy się w chwili zakończenia wszystkich wykonywanych przejść S_i, a następnie zaczyna się nowa faza wyboru.

3.1.1. Faza wyboru przejść w modelu symulacyjnym

Istotne jest ustalenie chwili dokonywania wyboru przejść oferowanych. Mając na uwadze uzasadnienia implementacyjne, w zaproponowanym tutaj modelu symulacyjnym przyjęto, że **wybór przejść oferowanych następuje w chwili rozpoczęcia fazy wyboru**. Przyjęcie każdej późniejszej chwili może już prowadzić do wyboru innych przejść, gdyż wskutek upływu czasu mogą napłynąć do kolejek instancji składowych S_i nowe interakcje lub mogą upłynąć czasy opóźnienia przejść opóźnionych, które automatycznie stają się przejściami gotowymi. Przyjęcie chwili rozpoczęcia fazy wyboru (oznaczanej dalej jako T_{bm} – begin of management phase) jako chwili wyboru przejść oferowanych oznacza, że w każdej instancji składowej S_i:

- jeżeli w chwili T_{bm} są przejścia gotowe, to z nich zostaje wybrane przejście oferowane; do czasu zakończenia fazy wyboru żadna zmiana stanu instancji (następująca na skutek przyjścia nowych interakcji) ani żadne nowe gotowe przejścia opóźnione (powstałe na skutek upływu czasu) nie zmienią dokonanego wyboru,
- jeżeli w chwili T_{bm} nie ma przejść gotowych, to nie zostanie zaoferowane żadne przejście; do czasu zakończenia fazy wyboru żadna zmiana stanu instancji (następująca na skutek przyjścia nowych interakcji) ani żadne nowe gotowe przejścia opóźnione (powstałe na skutek upływu czasu) nie spowodują już zaoferowania jakiegokolwiek przejścia.

Czas trwania fazy wyboru, oznaczany dalej przez *mt* (ang. management time), jest wyliczany w chwili T_{bm}. Czas *mt* jest odliczany w liczniku *manTimer* w systemie S_i. Każda instancja składowa może mieć w fazie wyboru niepusty zbiór przejść opóźnionych *delayedTransSet*. Każdy element tego zbioru zawiera licznik czasu opóźnienia. Upływ czasu powoduje zmniejszanie zawartości licznika *manTimer* i liczników czasów opóźnienia przejść w *delayedTransSet*.

Po upływie czasu *mt*, jeżeli co najmniej jedna instancja składowa S_i oferuje przejście, to rozpoczyna się faza wykonania przejścia dla S_i, w przeciwnym przypadku rozpoczyna się nowa faza wyboru (jeżeli którakolwiek instancja składowa S_i ma niepusty zbiór przejść opóźnionych) lub następuje zakleszczenie S_i (jeżeli dla każdej instancji składowej S_i zbiór przejść opóźnionych jest pusty).

3.1.2. Faza wykonania przejść w modelu symulacyjnym

W opisywanym modelu symulacyjnym przyjęto, że **na początku fazy wykonania** (w chwili $T_{bm} + mt$), **odbywa się wykonanie wybranych przejść**, a w trakcie wykonywania bloku przejścia instancji składowej jest wyliczany łączny czas wykonania przejścia, oznaczany dalej przez *et* (ang. execution time).

Dla zachowania zasad Estelle, konieczne jest spełnienie warunku, że zewnętrzne oddziaływania każdej wykonującej przejście instancji modułu muszą nastąpić po zakończeniu przejścia, czyli w chwili $T_{bm} + mt + et$. Te zewnętrzne oddziaływania to:

- wysłanie interakcji przez zewnętrzne punkty interakcji,
- zmiana wartości zmiennych współdzielonych (eksportowanych) instancji-dzieci,
- tworzenie i usuwanie instancji-dzieci,
- łączenie i rozłączanie punktów interakcji.

Z punktu widzenia symulacji, jedynym istotnym wymaganiem jest **wysłanie interakcji przez zewnętrzne punkty interakcji w chwili kończenia wykonywania przejścia**. W trakcie wykonywania bloku przejścia przez instancje, interakcje te są gromadzone w kolejce *outputInteractionQueue* instancji. Pozostałe oddziaływania mogą być wykonane w dowolnej chwili fazy wykonania – na przykład w chwili $T_{mt} + mt$– gdyż, zgodnie z zasadami Estelle, wykonywanie przejścia przez określoną instancję wyklucza równoczesne wykonywanie przejścia przez którąkolwiek instancję-przodka lub instancję-potomka.

Czas wykonania przejścia jest odliczany w liczniku *execTimer* w instancjach składowych S_i. Każda instancja składowa może mieć w fazie wykonania niepusty zbiór przejść opóźnionych *delayedTransStack*. Upływ czasu powoduje zmniejszanie zawartości liczników *execTimer* i liczników czasów opóźnienia przejść w *delayedTransStack*. Po upływie czasu wykonania przejścia w instancji składowej kończone jest wykonanie przejścia:

- wysyłane są interakcje z kolejki *outputInteractionQueue*,
- w przypadku, gdy wykonywane było przejście dotyczące interakcji przysłanej do określonego punktu interakcji, interakcja ta jest usuwana z kolejki, związanej z tym punktem,
- instancja składowa przechodzi do nowego stanu sterowania.

Faza wykonania kończy się w chwili zakończenia ostatniego przejścia ze zbioru przejść wybranych. Następnie rozpoczyna się kolejna faza wyboru przejścia lub występuje koniec symulacji, jeżeli którykolwiek system osiągnął stan końca symulacji.

3.2. Symulacja równoległej pracy systemów

Wyniki symulacji równoległej pracy systemów nie powinny zależeć od algorytmów programu SPP. Ponieważ w ogólnym przypadku czasy wyboru i wykonania przejść w instancjach mogą być zerowe, więc, jak wykazano w [3], konieczne jest **rozdzielenie w modelu symulacyjnym faz wyboru przejść od faz wykonania przejść w każdym systemie**.

W celu ustalenia jednoznacznych zasad symulacji, przyjęto, że:

- **kolejność obsługi systemów przez program SPP jest losowa** (wybór z jednakowym prawdopodobieństwem),
- **kolejność obsługi instancji-dzieci systemu S_i przez instancje-ojców jest losowa** (wybór z jednakowym prawdopodobieństwem).

4. Lista LL

Uwzględniając specyfikę języka Estelle, jest rzeczą naturalną użycie do budowy modelu symulacyjnego techniki programowania obiektowego. Konkretny rodzaj języka programowania obiektowego nie jest dla dalszych rozważań istotny; zakłada się jedynie, że:

- obiekt jest zmienną typu złożonego zwanego klasą,
- definicja klasy zawiera pola (zmienne) i metody (procedury) działające na polach,
- na postawie definicji jednej klasy (bazowej) można definiować inne klasy (pochodne).

Podstawowe pojęcie Estelle – moduł jest wygodnie reprezentować w modelu symulacyjnym jako klasę pochodną pewnej klasy bazowej EstelleModule. Drzewo specyfikacji jest reprezentowane w modelu symulacyjnym przez dynamiczną strukturę listową złożoną z list LL. Każda lista LL składa się z obiektów – zmiennych określonych klas. Klasa EstelleModule zawiera pewne pola i metody, wynikające bezpośrednio z reguł języka Estelle, oraz dodatkowe pola i metody, niezbędne do celów symulacji. Klasa EstelleModule zawiera, między innymi, następujące pola:

- *treeStatus* – bieżący stan sterowania drzewa, którego korzeniem jest system S_i,
- *tickRequired* – bieżące zapotrzebowanie na upływ czasu symulacyjnego składowych systemu S_i,
- *offTrans, delTrans, execCompleted, endOfSimulation* – 4 zmienne logiczne, zwane dalej flagami,
- *manTimer* – licznik czasu wyboru przejść systemu S_i,
- *execTimer* – licznik czasu wykonania przejścia instancji składowej systemu S_i,
- *delayedTransSet* – zbiór przejść opóźnionych instancji składowej systemu S_i,
- *outputInteractionQueue* – zbiór interakcji do wysłania po zakończeniu przejścia instancji składowej systemu S_i.

Zbiór wartości zmiennej *treeStatus*:

- *readyToTransChoice* - drzewo systemowe gotowe do rozpoczęcia fazy wyboru przejść,
- *transChoice* - drzewo systemowe w fazie wyboru przejść,
- *readyToTransExec* - drzewo systemowe gotowe do rozpoczęcia fazy wykonania przejść,
- *transExec* - drzewo systemowe w fazie wykonywania przejść,

- *localDeadlock* - drzewo systemowe w stanie lokalnego zakleszczenia,
- *endOfSimul* - co najmniej jedna składowa drzewa systemowego osiągnęła stan końca symulacji.

Pola *treeStatus* i *tickRequired* oraz 4 flagi są wykorzystywane tylko w obiektach, reprezentujących systemy S_i.

Klasa EstelleModule zawiera, między innymi, następujące metody:

- *startTransChoice, startTransExec, timeProcess* – metody zapewniające współdziałanie programu SPP z systemami i obsługę zdarzeń w instancjach składowych systemów,
- *downBroadcastStartTransChoice, downBroadcastStartTransExec, downBroadcastTimeProcess* – metody zapewniające współdziałanie systemu z instancjami-potomkami,
- *upBroadcastOffTransTrue, upBroadcastDelTransTrue, upBroadcastExecCompletedFalse, upBroadcastEndOfSimulationTrue, upBroadcastTickRequiredValue* – metody zapewniające współdziałanie instancji-potomków systemu z systemem.

Każde drzewo systemowe jest reprezentowane przez listę LL_i, złożoną z obiektów Ob_{ij}, gdzie $i = 1,...,I$; $j \geq 0$. System S_i jest reprezentowany przez obiekt Ob_{i0}. W strukturze listowej odzwierciedlona jest relacja starszeństwa instancji modułów typu ojciec – dzieci i przodkowie – potomkowie. W strukturze listowej jest również określona struktura komunikacyjna – połączeń punktów interakcji. Struktura listowa ma charakter dynamiczny – obiekty mogą tworzyć i usuwać inne obiekty-dzieci, zmieniać połączenia między punktami interakcji, wysyłać i odbierać interakcje. Działanie zarządzające każdego systemu w stosunku do instancji modułów potomnych w modelu symulacyjnym polega na:

- synchronizacji rozpoczynania i kończenia faz wyboru przejść i wykonania przejść,
- uzgadnianiu wielkości przyrostu czasu, o jaki może być zwiększony czas symulacyjny.

Obiekty reprezentujące instancje składowe drzewa z korzeniem w S_i współdziałają przez podział wspólnych zmiennych, należących do obiektu Ob_{i0}, reprezentującego system S_i: *tickRequired* i 4 flagi. Wykorzystanie flag opisano w tabeli 1.

Tabela 1

Wykorzystanie flag

Nazwa flagi	Wartość początkowa	Zmiana stanu flagi następuje, jeżeli co najmniej jedna instancja składowa:
offTrans	FALSE	oferuje przejście
delTrans	FALSE	ma niepusty zbiór przejść opóźnionych
execCompleted	TRUE	nie zakończyła wykonania przejścia
endOfSimulation	FALSE	osiągnęła stan końca symulacji

Wartość początkową flag ustala system S_i. Każda instancja-potomek S_i może zmienić stan każdej z flag Ob_{i0} metodami: *upBroadcastOffTransTrue*, *upBroadcastDelTransTrue*, *upBroadcastExecCompletedFalse*, *upBroadcastEndOfSimulationTrue*.

Podobnie może być zmieniona (metodą *upBroadcastTickRequiredValue*) wartość pola *tickRequired* w Ob_{i0}, jeżeli zapotrzebowanie na wielkość przyrostu czasu instancji jest mniejsze od wartości aktualnej *tickRequired*.

Wywołanie przez SPP w Ob_{i0} metody *startTransChoice* powoduje wywołanie (metodą *downBroadcastStartTransChoice*) metod *startTransChoice* we wszystkich obiektach Ob_{ij}, $j > 0$.Wywołanie przez SPP w Ob_{i0} metody *startTransExec* powoduje wywołanie (metodą *downBroadcastStartTransExec*) metod *startTransExec* we wszystkich obiektach Ob_{ij}, $j > 0$. Wywołanie przez SPP w Ob_{i0} metody *timeProcess* z aktualnym przyrostem czasu powoduje wywołanie (metodą *downBroadcastTimeProcess*) metod *timeProcess* z tym przyrostem we wszystkich obiektach Ob_{ij}, $j > 0$. Po zakończeniu fazy wyboru przejść w S_i, nowy stan drzewa S_i jest ustalany na podstawie flag *offTrans* i *delTrans*. Chwila zakończenia fazy wykonania przejść jest ustalana na podstawie flagi *execCompleted*, a nowy stan drzewa S_i na podstawie flagi *endOfSimulation*.

5. Program SPP

W symulatorze sekwencyjnym listy LL reprezentujące wszystkie drzewa systemowe danej specyfikacji są zaalokowane w komputerze jednoprocesorowym.

Ze względu na specyfikę języka Estelle:

- konieczność wykonania faz wyboru przejść w celu wyliczenia czasu *mt*,
- konieczność wykonania fazy wykonania przejść w celu wyliczenia czasów *et*,
- działania zarządzające systemów w stosunku do instancji-potomków,

dokonano modyfikacji powszechnie stosowanego algorytmu planowania zdarzeń (ang. event scheduling) [6], [7]. Program SPP współpracuje z każdym obiektem Ob_{i0}:

- śledzi stan sterowania wszystkich I drzew systemowych – pola *treeStatus*,
- śledzi zapotrzebowanie na przyrost czasu we wszystkich I drzewach systemowych – pola *tickRequired*,
- w zależności od pól *treeStatus* i *tickRequired*, wywołuje odpowiednie metody obsługi zdarzeń w obiektach Ob_{i0}: *startTransChoice*, *startTransExec*, *timeProcess*.

Zmienna *clock* reprezentuje w modelu symulacyjnym czas symulacyjny.

Algorytm programu SPP jest następujący:

utwórz strukturę listową reprezentującą drzewo specyfikacji;
ustal warunki zakończenia symulacji;
clock :=0;
L2: **if** (jest spełniony warunek zakleszczenia specyfikacji – wszystkie listy LL są w stanie *localDeadlock*) **then goto** L1;

wywołaj metody obsługi zdarzeń wszystkich systemów, które są w stanie *readyToTransChoice* lub *localDeadlock* – są gotowe do rozpoczęcia fazy wyboru przejść (kolejność wyboru systemów do obsługi jest losowa);
wywołaj metody obsługi zdarzeń wszystkich systemów, które są w stanie *readyToTransExec* – są gotowe do rozpoczęcia fazy wykonania przejść (kolejność wyboru systemów do obsługi jest losowa);
wyznacz wielkość przyrostu czasu *tickGranted*, o jaki może być przesunięty czas symulacyjny (jest to najmniejsza wartość spośród zgłoszonych przez systemy wartości *tickRequired*);
wywołaj metody upływu czasu symulacyjnego o wartość *tickGranted* we wszystkich systemach (kolejność wyboru systemów do obsługi jest losowa);
clock := *clock* + *tickGranted*;
if (nie jest spełniony warunek końca symulacji) **then goto** L2; (* wykonanie kolejnego kroku symulacji *)
L1: podaj wyniki symulacji i zakończ symulację.

Algorytm obsługi listy LL_i jest następujący:

case *treeStatus* **of**
readyToTransChoice **or** *localDeadlock*: wywołanie metody *startTransChoice()* w Ob_{i0};
(* powoduje: w Ob_{i0}: wyliczenie czasu mt; *manTimer* := *mt*;
we wszystkich składowych LL_i:
– usunięcie ze zbioru *delayedTransSet* tych przejść, które nie są już możliwe,
– wyliczenie wartości czasu opóźnienia dla nowych przejść opóźnionych i zapisanie tych przejść do zbioru *delayedTransSet*;
– wybór przejść do wykonania w składowych LL_i;
– uzgodnienie przez składowe LL_i wartości minTick – najmniejszej spośród wartości *manTimer* i czasów opóźnień przejść w zbiorach *delayedTransSet*;
tickRequired := minTick;
treeStatus := *transChoice* *)
transChoice: wywołanie metody *timeProcess(tickGranted)* w Ob_{i0};
(* powoduje: w Ob_{i0} : *manTimer* := *manTimer* – *tickGranted*;
zmniejszenie o wartość *tickGranted* czasów opóźnień przejść w zbiorach *delayedTransSet* we wszystkich składowych LL_i.
if *manTimer* = 0 **then**
{*tickRequired* := 0;
if not *offTrans* (*zbiór przejść do wykonania pusty*) **then**
if not (*zbiór przejść opóźnionych pusty*) **then** *treeStatus* := *localDeadlock*
else *treeStatus* := *readyToTransChoice*;
else treeStatus := *readyToTransExec*};
else {uzgodnienie przez składowe LL_i wartości minTick – najmniejszej spośród wartości *manTimer* i czasów opóźnienia przejść w zbiorach *delayedTransSet* we wszystkich składowych LL_i;
tickRequired := minTick} *)
readyToTransExec: wywołanie metody *startTransExec()* w Ob_{i0};
(* powoduje: wykonanie w wybranych składowych przejść, z wyjątkiem wysłania interakcji przez zewnętrzne punkty interakcji – interakcje takie są zapisywane do kolejek wyjściowych *outputInteractionQueue* w składowych listy LL_i;
wyliczenie czasów *et* dla wszystkich składowych wykonujących przejście;

uzgodnienie przez składowe LL_i wartości minTick - najmniejszej wartości spośród wszystkich *et* i wartości czasów opóźnienia przejść w zbiorach *delayedTransSet* we wszystkich składowych LL_i;
tickRequired := minTick;
treeStatus := *transExec* *)
transExec: wywołanie metody *timeProcess(tickGranted)* w Ob_{i0};
(* powoduje: zmniejszenie o wartość *tickGranted* wartości czasów opóźnienia przejść w zbiorach *delayedTransSet* we wszystkich składowych LL_i;
zmniejszenie o wartość *tickGranted* wartości liczników czasu wykonania przejść w składowych LL_i wykonujących przejście;
zakończenie wykonania przejść przez składowe LL_i, których liczniki czasu wykonania przejścia osiągnęły wartość zero:
– przepisanie interakcji z kolejek wyjściowych *outputInteractionQueue* do kolejek wejściowych w odpowiednich składowych list LL;
– nadanie nowej wartości stanowi sterowania;
if *execCompelted* (*zbiór przejść wykonywanych pusty*) **then**
{wyliczenie miar wydajności; wizualizacja, zapisy do pliku
if *endOfSimulation* (*co najmniej jedna składowa listy LL_i osiągnęła stan końca symulacji*)
then *treeStatus* := *endOfSimul*
else *treeStatus* := *readyToTransChoice*;
tickRequired := 0}
else {uzgodnienie przez składowe LL_i wartości minTick - najmniejszej wartości spośród liczników czasów wykonania przejść i wartości czasów opóźnienia w zbiorach *delayedTransSet* we wszystkich składowych LL_i;
tickRequired := minTick;
treeStatus := *transExec*} *)
end.

6. Wnioski, kierunki dalszych prac

Opisane struktury danych reprezentujące drzewa systemowe specyfikacji w języku Estelle dla potrzeb symulacji sekwencyjnej mogą być również użyte w symulacji równoległej. Celem stosowania symulacji równoległej jest skrócenie czasów wykonywania eksperymentów symulacyjnych w porównaniu z symulacją sekwencyjną. Możliwość skrócenia tych czasów jest ważna przy rozwiązywaniu praktycznych zadań symulacji, np. symulacji komputerowych systemów komunikacyjnych.

Prowadzone obecnie prace dotyczą symulacji równoległej specyfikacji w środowisku rozproszonym, złożonym z co najmniej dwóch komputerów, połączonych medium transmisyjnym. W pracach [8], [9] opisano zasady dekompozycji specyfikacji do poziomu drzew systemowych i alokację w każdym węźle dokładnie jednego drzewa systemowego. Dalsze prace będą dotyczyły symulatora rozproszonego z alokacją dowolnej liczby drzew systemowych w węźle.

Literatura:

[1] ISO 9074: Estelle: A formal description technique based on an extended transition model, 1997

[2] Budkowski S., Dembiński P.: An introduction to Estelle: a specification language for distributed systems. Computer Networks and ISDN Systems, 14 (1), 1987, s. 3 – 23

[3] Kacprzak M.: Metoda oceny charakterystyk czasowych projektów adapterów komunikacyjnych do sieci komputerowych. Rozprawa doktorska, Politechnika Śląska w Gliwicach, 1995

[4] Dembiński P., Kacprzak M., Siatkowski J.: Using the Edb simulator as performance evaluation tool for distributed systems, COPERNICUS Project #62: High Speed Communication Systems Supporting Multimedia Applications, November 1996

[5] Kacprzak M.: Symulacja rozproszona protokołów komunikacyjnych i ich implementacji. Raport 97/1. Instytut Maszyn Matematycznych, Warszawa, 1997

[6] Ed Lavenberg S. S.: Computer performance modelling handbook, Academic Press, 1983

[7] Tyszer J.: Symulacja cyfrowa, WNT, Warszawa, 1990

[8] Kacprzak M.: Symulacja komputerowa sieci rozszerzonych automatów asynchronicznych w środowisku rozproszonym, Techniki Komputerowe Biuletyn Informacyjny, Nr 1, 1996, s. 19-30

[9] Kacprzak M.: Distributed simulation of sets of asynchronous automata. In Proceedings of 13th European Simulation Multiconference (Warsaw, June 1999), vol. I, s. 215-219

MAREK KACPRZAK
INSTYTUT MASZYN MATEMATYCZNYCH WARSZAWA

Symulatory rozproszone – przegląd rozwiązań
Distributed simulators – a review of implementations

Streszczenie

Przedstawiono przegląd implementacji symulatorów zdarzeń dyskretnych, działających równolegle w środowisku rozproszonym.

Abstract

A review of implementations of parallel simulators for discrete events systems is presented; the simulators are implemented in distributed environment.

1. Wstęp

Symulacja komputerowa jest powszechnie stosowanym narzędziem projektowania i analizy systemów. Metodyka korzystania z symulacji polega na:

- opracowaniu modelu symulowanego obiektu,
- wykonywaniu modelu przy symulacji, zazwyczaj, upływu czasu,
- analizie i interpretacji uzyskanych wyników.

Modele opisują systemy przy użyciu pojęć zbioru stanów i zbioru zdarzeń. Zdarzenia powodują zmiany stanów. W niniejszej pracy omawiane są wyłącznie modele zdarzeń dyskretnych (ang. discrete events models), tzn. takie, w których zmiany stanu następują skokowo w ustalonych punktach czasu symulacyjnego. Symulacja komputerowa może być wykonywana na jednym procesorze (symulacja sekwencyjna) lub na więcej niż jednym procesorze (symulacja równoległa).

Cechą charakterystyczną rozwiązywania rzeczywistych, praktycznych problemów przy użyciu symulacji sekwencyjnej (np. projektowania protokołów komunikacyjnych do sieci komputerowych) są długie czasy trwania eksperymentów symulacyjnych. Czasy takie wynikają zazwyczaj:

- ze złożoności symulowanego modelu (wielowymiarowa przestrzeń parametrów i wynikający z tego obszerny program badań),

- ze względów metodycznych – wyniki symulacji mają charakter losowy, konieczne jest wyliczanie wielu statystyk (np. średnia, wariancja i momenty wyższych rzędów) z zawężonym przedziałem ufności (eksperyment musi być powtórzony wielokrotnie lub prowadzony wystarczająco długo).

Istnieją przypadki, gdy czas symulacji musi być skrócony:

- w systemach decyzyjnych czasu rzeczywistego, w których wyniki symulacji muszą być uzyskane możliwie szybko (sterowanie procesami szybkozmiennymi w sytuacjach awaryjnych, symulacja pola walki, kierowanie ruchem lotniczym itp.),
- w sytuacjach, gdy dany eksperyment musi być wielokrotnie powtórzony, a wyniki jednego eksperymentu są podstawą do ustalenia parametrów modelu w następnym eksperymencie.

W wielu przypadkach (np. projektowania sieci przenośnych systemów telefonii bezprzewodowej, reorganizacji procesów biznesowych) zbudowany model wymaga reprezentacji w postaci danych nie mieszczących się w pamięciach operacyjnych dostępnych komputerów.

2. Symulacja równoległa i rozproszona

Najbardziej obiecującym sposobem rozwiązania żądania minimalizacji czasów symulacji i wymaganej pamięci jest **symulacja równoległa**, w której jeden eksperyment symulacyjny jest wykonywany równocześnie na więcej niż jednym procesorze. Symulacja równoległa systemów dyskretnych (PDES – ang. Parallel Discrete Events Simulation) jest stabilną dyscypliną na pograniczu badań systemowych i informatyki. Pierwsze artykuły naukowe zostały opublikowane w latach 60-tych. W ciągu ostatnich 30 lat nastąpił rozwój i ugruntowanie tematyki. Przegląd algorytmów podano w [1] i [2].

Źródła równoległości w symulacji równoległej dotyczą procedur lub składowych ([3]). Istnieje wiele algorytmów symulacji równoległej, ale nie można jednoznacznie określić, który z nich jest najlepszy w sensie minimalizacji czasu symulacji i minimalizacji wymaganej pamięci – wszystko zależy od konkretnego symulowanego modelu, od sposobu jego dekompozycji na procedury lub składowe i sposobu alokacji reprezentacji procedur lub składowych w procesorach. Najlepiej zbadany i najczęściej stosowany jest algorytm oparty na pojęciu symulacji procesów logicznych [3]. Istnieją różne warianty tego algorytmu:

- z centralnym lub rozproszonym zegarem czasu symulacyjnego,
- ze zcentralizowaną lub zdecentralizowaną listą zdarzeń,
- z symulacją synchroniczną lub asynchroniczną,
- w przypadku symulacji asynchronicznej – z różnymi protokołami synchronizacji: zachowawczym, optymistycznym, mieszanymi,
- z alokacją statyczną lub dynamiczną.

Prowadzone prace nad symulacją równoległą dotyczą podstaw naukowych oraz praktycznego ich wykorzystania w implementacjach – symulatorach. Przegląd kierunków prac naukowych podano w [4].

Symulacja równoległa może być wykonywana na komputerach wieloprocesorowych lub w środowisku rozproszonym – na procesorach połączonych ośrodkiem przesyłania danych. Większość opracowanych symulatorów równoległych została zaimplementowana na komputerach wieloprocesorowych – uniwersalnych lub zaprojektowanych specjalnie do zadań symulacji. Komputery wieloprocesorowe mają różne architektury: pamięć wspólną lub nie, różne zasady przesyłania danych między procesorami. Algorytmy symulacji równoległej są dostosowane do architektury konkretnego komputera wieloprocesorowego. Z natury rzeczy symulatory takie mogą być wykorzystywane do zadań zlecanych do ośrodków (najczęściej – uniwersyteckich), w których są zainstalowane komputery wieloprocesorowe lub udostępniane w tych ośrodkach użytkownikom.

Alternatywą dla symulacji równoległej na komputerach wieloprocesorowych jest symulacja równoległa w środowisku rozproszonym, nazywana **symulacją rozproszoną**. Środowiskiem rozproszonym mogą być komputery (najczęściej jednoprocesorowe) połączone siecią komputerową (z transmisją szeregową) lub tworzące system wielokomputerowy (z transmisją równoległą). Komputery wchodzące w skład symulatora rozproszonego są nazywane węzłami.

W latach 90-tych nastąpił gwałtowny wzrost zainteresowania symulacją rozproszoną. Powodem były nowe potrzeby i nowe możliwości. Symulacja, określana jako „narzędzie następnego milenium" ([5]), jest powszechnie stosowana praktycznie w każdej dziedzinie projektowania i analizy systemów. Szeroko dostępne i tanie komputery osobiste i stacje robocze, wielki postęp w urządzeniach transmisji danych, służących do łączenia komputerów (duża niezawodność, szybkości transmisji rzędu 1 Gb/s) – czynią symulację rozproszoną wykonalną i umożliwiają stosowanie jej bezpośrednio u użytkownika. Węzłami symulatora rozproszonego mogą być komputery o różnej mocy obliczeniowej, a nawet różnych linii i działające pod kontrolą różnych systemów operacyjnych (np. symulator z węzłami Sun/Solaris i Intel PC/Linux). Liczba węzłów jest dowolna, choć ze względów praktycznych rzadko przekracza 10. Koszt zestawienia symulatora rozproszonego jest nieporównywalnie niższy od kosztów zakupu komputera wieloprocesorowego. Aby jednak osiągnąć zasadnicze cele – skrócenie czasów symulacji i zmniejszenie zapotrzebowania na pamięć, konieczny jest właściwy dobór środowiska rozproszonego i dostosowanie algorytmów symulacji równoległej do tego środowiska. W algorytmach symulacji rozproszonej, w odróżnieniu od symulacji równoległej na komputerach wieloprocesorowych, muszą być uwzględnione zagadnienia przesyłania danych przez nieidealny ośrodek (tzn. taki, w którym dane przesyłane od źródła do ujścia mogą być gubione, zniekształcane, powtarzane, mogą docierać do ujścia w innej kolejności, niż były nadane).

3. Ankieta

W Instytucie Maszyn Matematycznych od trzech lat są prowadzone prace nad symulacją rozproszoną specyfikacji systemów rozproszonych w języku Estelle ISO 9074 ([6]). Język Estelle jest narzędziem opisu systemów rozproszonych, głównie sieci

komputerowych. Prowadzone prace dotyczą zarówno opracowania algorytmów symulacji rozproszonej, specyficznych dla języka Estelle, jak i opracowania prototypu symulatora rozproszonego DSE (Distributed Simulator for Estelle) ([7]). Głównym założeniem projektowym dla DSE jest skrócenie czasów symulacji w porównaniu z istniejącymi symulatorami sekwencyjnymi do języka Estelle.

Dla porównania opracowywanego symulatora DSE z innymi rozwiązaniami światowymi, przeprowadzono w sierpniu i wrześniu 1999 r. przy użyciu Internetu ankietę wśród 27 głównych ośrodków zajmujących się przetwarzaniem równoległym. Odpowiedź udzieliło 7, z których 5 prowadzi prace z zakresu implementacji symulatorów rozproszonych. Wyniki ankiety przedstawiono w tabeli. W tabeli umieszczono także dane o symulatorze DSE.

Symulatory rozproszone

	Ośrodek					
	Georgia Institute of Technology, Atlanata, USA	Dept. of Electrical and Computer Engineering, Univ. of Waterloo, Toronto, Kanada	Experimental Computing Laboratory, University of Cincinnati, USA	Simulation group, CISE Department, University of Florida, Gainesville, USA	University of Vienna, Dept. of Applied Computer Science, Austria	Instytut Maszyn Matematycznyc h Warszawa, Polska
	1	2	3	4	5	6
Nazwa symulatora lub projektu	FDK = Federated Simulation Kit– praca dotyczy połączeń symulatorów	Projekt Parsimony	WARPED	DSX = Distributed Simulation Executive	N-MAP	DSE = Distributed Simulator for Estelle
Stan prac	prototyp, bezpłatny do celów naukowych	prace w toku / prototyp	produkt dostępny od 4 lat, bezpłatny	prace w toku	produkt	prace w toku
Język lub technika opisu modelu symulacyjnego	program wykonywalny, można stosować dowolne modele i abstrakcje	obiektowe procesy logiczne	modele opisywane w C++, dostępny interfejs API	model bloków funkcyjnych FBM	czasowe sieci Petri	język Estelle ISO 9074
Algorytm symulacji:						
• sterowanie czasem lub zdarzeniami	sterowanie czasem lub zdarzeniami	sterowanie zdarzeniami	sterowanie zdarzeniami	sterowanie zdarzeniami	sterowanie zdarzeniami	sterowanie zdarzeniami
• poziom równoległości (procedury lub składowe)	nie dotyczy	składowe	składowe	procedury	składowe	składowe

	1	2	3	4	5	6
• typ zegara (centralny lub rozproszony)	rozproszony	możliwy zarówno centralny, jak i rozproszony	rozproszony	rozproszony	rozproszony	rozproszony
• typ listy zdarzeń (zcentralizowana lub zdecentralizowana)	zdecentralizowana	możliwa zarówno zcentralizowana, jak i zdecentralizowana	zdecentralizowana	zdecentralizowana	zdecentralizowana	zdecentralizowana
• dla symulacji procesów logicznych – typ symulacji (synchroniczna lub asynchroniczna)	asynchroniczna lub synchroniczna	możliwa zarówno synchroniczna, jak i asynchroniczna	asynchroniczna	synchroniczna	asynchroniczna	asynchroniczna
• dla asynchronicznej symulacji procesów logicznych – protokół synchronizacji	zachowawczy, optymistyczny w opracowaniu	zachowawczy, optymistyczny, globalnie synchronizowany	optymistyczny	nie dotyczy	zachowawczy, optymistyczny, adaptacje	zachowawczy
Alokacja procesów	statyczna	statyczna	statyczna (dynamiczna w opracowaniu)	statyczna	statyczna	statyczna
Środowisko rozproszone:						
• typ węzłów	stacje robocze (Sun, Intel PC) i/lub komputery wieloprocesorowe SGI	Sun	Sun, Intel PC, komputery wieloprocesorowe Intel Paragon, IBM SP1/SP2, SGI Origin 2000	Sun Ultra, Intel PC	komputery wieloprocesorowe CM-5, Meiko CS-2, Intel Paragon	Intel PC
• stosowany system operacyjny	Solaris	Solaris + JVM	SunOS and Linux	Solaris, Windows NT/98/95	specjalizowane	Linux
• sposób połączenia węzłów	szybka sieć lokalna Myrinet*, dowolna sieć z protokołem TCP/IP)	sieć lokalna	sieci lokalne: Ethernet, przełączany szybki Ethernet, szybka sieć lokalna Myrinet*	Internet	sieci lokalne	sieć lokalna Ethernet
• typowa liczba węzłów	do 8	3	4 do 8	do 10	128	4

* sieć lokalna firmy Myricom

	1	2	3	4	5	6
Użyte specjalne programy do przetwarzania rozproszonego	Illinois-FM	Java RMI	MPI	nie używano	CMMD, PVM, MPI	nie używano
Język użyty do oprogramowania symulatora	dowolny, interfejs do C lub C++	Java	C++	C++	C	C++
Dziedzina zastosowań	telekomunikacja, obronność	ogólne zastosowania PDES, rozproszona symulacja interakcyjna	jądro symulacji ogólnego przeznaczenia, stosowane do symulacji: sieci kolejkowych, sieci komputerowych, tradycyjnych układów logicznych i układów VHDL	symulacja ekosystemów	badania naukowe w dziedzinie PDES	protokoły komunikacyjne sieci komputerowych i ich implementacje, systemy produkcyjne integrowane komputerowo

4. Wnioski, kierunki dalszych prac

Jak wynika z otrzymanych odpowiedzi, można określić typowy stosowany w symulatorach rozproszonych algorytm symulacji:

- sterowanie zdarzeniami,
- równoległość składowych,
- zegar rozproszony,
- zdecentralizowana lista zdarzeń,
- asynchroniczna symulacja procesów logicznych z zachowawczym lub optymistycznym protokołem synchronizacji.

Powszechnie stosowana jest alokacja statyczna. Rozwiązania różnią się językami opisu modelu symulacyjnego i rodzajami węzłów. Podstawowym sposobem połączenia węzłów są sieci lokalne.

Przyjęte rozwiązania dla symulatora DSE są zgodne z wynikającymi z ankiety. W DSE przyjęto zachowawczy protokół synchronizacji, który mimo potencjalnie gorszej od optymistycznego protokołu synchronizacji efektywności (w sensie czasów trwania symulacji) jest prostszy w implementacji – ma prostsze struktury danych i algorytmy. Środowisko rozproszone DSE stanowią węzły Intel PC połączone siecią lokalną – obecnie klasycznym Ethernetem 10 Mb/s, a w przyszłości siecią ATM lub Ethernetem 1 Gb/s. Rozważana jest także możliwość połączenia węzłów w system wielomaszynowy. Zaletą takiego rozwiązania byłoby uzyskanie dużych szybkości przesyłania danych między węzłami, wadą – silne ograniczenie na odległości między węzłami

i konieczność napisania własnych programów sterujących do urządzeń transmisji równoległej.

Spełnienie wymogu skrócenia czasów symulacji w DSE w stosunku do innych symulatorów sekwencyjnych do języka Estelle wymaga dokładnego zbadania wpływu protokołów komunikacyjnych na czas transmisji danych w sieciach lokalnych. Prowadzone badania dotyczą standardowych mechanizmów komunikacyjnych systemu Linux, a także wpływu na efektywność transmisji użycia narzędzi przetwarzania rozproszonego, takich jak MPI lub PVM.

Z punktu widzenia użytkowego, zaletą symulatora DSE jest zgodność modeli symulacyjnych ze stabilną normą języka Estelle – ISO 9074. Formalizm ten jest powszechnie stosowany do specyfikacji protokołów sieci komputerowych, ale może być używany także do specyfikacji systemów produkcyjnych integrowanych komputerowo [8], czy architektury urządzeń komputerowych.

Tematyka symulacji rozproszonej jest dynamicznie rozwijana i aktualna, co potwierdza uwzględnienie jej w Piątym Ramowym Programie Badań i Rozwoju Technologicznego Unii Europejskiej – temat IV.4.1 „Real-time simulation and visualisation technologies" programu „Information Society Technologies", akcja kluczowa IV – „Essential Technologies and Infrastructures".

Literatura:

[1] Fujimoto R. M.: Parallel discrete event simulation. Communications of the ACM, 33 (10), 1990, s. 30-53

[2] Zomaya A. Y. Ed.: Parallel and distributed computing handbook. McGraw – Hill, 1996

[3] Kacprzak M.: Symulacja komputerowa sieci rozszerzonych automatów asynchronicznych w środowisku rozproszonym. Techniki Komputerowe Biuletyn Informacyjny, Nr 1, 1996, s. 19-30

[4] Nicol D., Fujimoto R.: Parallel simulation today. Annals of Operations Research, December 1994

[5] Proceedings of 13th European Simulation Multiconference (Warsaw, June 1999)

[6] ISO 9074: Estelle: A formal description technique based on an extended transition model, 1997

[7] Kacprzak M. Distributed simulation of sets of asynchronous automata. In Proceedings of 13th European Simulation Multiconference (Warsaw, June 1999), vol. I, 215-219

[8] Kacprzak M., Kaczmarczyk A., Slesicki M.: Application of Estelle language for SME modelling a model for examination of a creamery tolerance from temporal water shortage. In Proceedings of 9th IFAC Symp. On Information Control in Manufacturing INCOM'98 (Nancy – Metz, France, June 24–26 1998), vol. III, s. 293-297

ANDRZEJ KACZMARCZYK
INSTYTUT MASZYN MATEMATYCZNYCH WARSZAWA

Prace Ośrodka Badawczo-Szkoleniowego Technik Komputerowych IMM na rzecz małych i średnich przedsiębiorstw (MSP) w dziedzinie komputerowego wspomagania biznesu

Activities of the Computer Technology Research and Training Center of the IMM on Behalf of SMEs in the Area of Business Computer Aiding

Streszczenie

Przedstawiono potrzeby MSP oraz prowadzone obecnie i zamierzone prace na ich rzecz w zakresie szkolenia oraz modelowania komputerowego przedsiębiorstw.

Abstract

Presentation of needs of SMEs, as well as present and future activities in the area of training and enterprises computer modelling.

1. Potrzeby MSP

Według danych Ministerstwa Gospodarki, przedstawionych przez ministra Janusza Steinhoffa dn. 13.05.1999 r. na konferencji *Kierunki działań rządu wobec małych i średnich przedsiębiorstw do 2002 roku*, jest obecnie w Polsce ponad 2,5 miliona MSP, co stanowi 99,8% wszystkich zarejestrowanych przedsiębiorstw. MSP wytwarzają połowę PKB i zatrudniają 67% wszystkich zatrudnionych w gospodarce. Jednym z głównych problemów rozwoju MSP jest podniesienie ich konkurencyjności. Potrzebne w tym celu są zmiany legislacyjne i wsparcie finansowe przedsiębiorstw oraz transfer myśli technicznej i wspomaganie przedsięwzięć innowacyjnych.

W nadchodzących latach MSP będą działały w warunkach szybko postępującej globalizacji gospodarki oraz w warunkach społeczeństwa informacyjnego – jedno i drugie wymaga zmian w dziedzinie informatyki i komunikacji, wchodzenia MSP do cyberprzestrzeni, która w coraz większej mierze staje się przestrzenią biznesu.

Mijają już czasy, gdy niezależne małe przedsiębiorstwo mogło znaleźć swoją niszę gospodarczą, oferując przez długi czas stałym odbiorcom jakiś „dobrze trafiony" swój produkt czy usługę, na które był niesłabnący popyt. Rynek jest kapryśny i szybkozmienny, a globalizacja gospodarki wyraża się w tym, że przedsiębiorstwa nie są już niezależnymi jednostkami, lecz ogniwami długich łańcuchów ściśle powiązanych dostawców i podwykonawców, przy czym łańcuchy te ulegają ciągłym zmianom. Pojawia się w gospodarce twór nazywany przedsiębiorstwem wirtualnym, skomponowany z funkcji „dostarczanych" przez inne przedsiębiorstwa, zdolny do szybkiego powstania, reorganizacji i rozwiązania. Dziś szczególnie ważne są przedsięwzięcia innowacyjne ułatwiające współdziałanie MSP z wielkimi korporacjami. Zagadnienia te omówione są szerzej w referacie autorów z Instytutu Maszyn Matematycznych, Akademii Górniczo-Hutniczej i Uniwersytetu w Glasgow przedstawionym na 13. Konferencji Międzynarodowej ISPE/IEEE [1].

W celu polepszenia i utrzymania konkurencyjności MSP powinny stosować elektroniczną wymianę dokumentów oraz umieć integrować się – za pośrednictwem sieci komputerowej – z systemami zarządzania partnerów, w szczególności wielkich korporacji. Już dziś partnerzy udostępniają sobie nawzajem komputerowe bazy danych, co staje się warunkiem koniecznym efektywnej współpracy. Bezbłędny przebieg skomplikowanych transakcji (procesów biznesowych), przebiegających wzdłuż długiego łańcucha przedsiębiorstw, wymaga ich bieżącej kontroli środkami komputerowymi. Nieodzownym warunkiem powodzenia współpracy z innymi jest zrobienie „elektronicznego" porządku u siebie i zainstalowanie wiedzy o przedsiębiorstwie, nawet małym, w jego komputerach oraz przyjęcie standardowego języka opisu tej wiedzy.

2. Obecne działania OBSTK IMM

2.1. Szkolenie użytkowników komputerów

IMM od 9 lat prowadzi szkolenia użytkowników komputerów. Kursy komputerowe ukończyło w Instytucie ponad 7000 osób, w większości z MSP.

Oferujemy kursy podstawowe i zaawansowane w zakresie wiedzy bazowej w użytkowaniu komputerów – oprogramowania biurowego dla pracowników administracji, sekretariatów i indywidualnych użytkowników komputerów osobistych. Obecnie programy tych kursów są dostosowane do wymagań *Europejskiego Komputerowego Prawa Jazdy (ECDL – European Computer Driving Licence)* – międzynarodowego certyfikatu zaświadczającego, że jego posiadacz ma podstawowe umiejętności obsługi mikrokomputerów i potrafi je wykorzystywać w codziennej pracy. W ramach kursów podstawowych uczymy obsługiwania poczty elektronicznej i nawigowania po Internecie.

Kursy ukierunkowane na zaspokojenie specyficznych potrzeb MSP obejmują:

- komputerowo wspomagane projektowanie,
- grafikę komputerową,
- obróbkę zdjęć fotograficznych,
- projektowanie stron WWW.

Program kursów można znaleźć na stronie: http://www.imm.org.pl/kursy.htm.

Najnowszą ofertą jest *Szkoła Intranetu* przeznaczona dla informatyków i menedżerów odpowiedzialnych za rozwój informatyczny w przedsiębiorstwie, mająca ich wprowadzić w nowe technologie tworzenia sieci i oprogramowania intranetowego (bliższe informacje: http://www.imm.org.pl/szkola/index.html).

2.2. Modelowanie przedsiębiorstw

Stworzenie modelu przedsiębiorstwa, tzn. uproszczonej, użytecznej reprezentacji rzeczywistości przedsiębiorstwa, daje wspólny język porozumiewania się wszystkich uczestników działalności przedsiębiorstwa, umożliwia gromadzenie i kapitalizację wiedzy o przedsiębiorstwie, wspomaganie decyzji operacyjnych i dotyczących rozwoju. W obecnych warunkach działania przedsiębiorstw zasadnicze znaczenie mają modele komputerowe. IMM oferuje opracowanie takich modeli na zlecenie przedsiębiorstw. Opracowano dotychczas model mleczarni, przeznaczony do wykorzystania w przeciwdziałaniu negatywnym skutkom okresowego niedoboru wody, przedstawiony w referacie autorów z IMM i Instytutu Meteorologii i Gospodarki Wodnej na 9. Sympozjum IFAC [2].

3. Podjęte i planowane prace

W 1998 r. w IMM rozpoczęto realizację projektu badawczego pn. *Model komputerowy małego przedsiębiorstwa na przykładzie ośrodka szkoleniowego*, który ma się zakończyć w 2000 r. próbną eksploatacją opracowanego modelu. Zaawansowanie prac nad projektem przedstawiono w referacie autorów z IMM [3]. Wyniki z rozpoznania metod inżynierii biznesu przydatnych dla MSP przedstawiono w publikacjach autora w miesięczniku *Informatyka* [4], [5].

W 1999 r. opracowano i uzgodniono z partnerem zagranicznym założenia innego projektu badawczego dotyczącego MSP, który ma być realizowany we współpracy polsko–meksykańskiej (umowa międzyrządowa o współpracy naukowo-technicznej) w latach 2000-2001. Jednostkami współpracującymi są IMM oraz Wydział Inżynierii Mechanicznej, Elektrycznej i Elektronicznej Uniwersytetu Guanajuato. Projekt nosi tytuł *Metody i narzędzia dla małych i średnich przedsiębiorstw do telepracy on-line.* Celem projektu (z uwzględnieniem sytuacji w Polsce i w Meksyku, a w szczególności w Stanie Guanajuato) jest rozpoznanie potrzeb i możliwości ich zaspokojenia oraz wybranie przedmiotów dalszych badań i rozwoju w następujących obszarach: obiektów technicznych do użytkowania w MSP, infrastruktury (w której MSP będą działały) i szkolenia. Szczegółowe dane projektu przedstawiono w dokumencie [6] przygotowanym zgodnie z wymaganiami KBN.

Zamierzeniem, nad którym obecnie pracujemy, jest uruchomienie szkolenia adresowanego do menedżerów – w tym z MSP – w zakresie komputerowego projektowania i rekonstrukcji biznesu, poświęconego metodom oraz narzędziom BPR *(Business Process Reengineering)* i BPI *(Business Process Improvement).*

Wykaz publikacji i dokumentów IMM przywołanych w tekście:

[1] Kaczmarczyk A., Szpytko J., Harrison D. K.: Globalisation Aspects of the Boundary Processes and Mutual Relations of The Enterprises – A SME Perspective. 13th ISPE/IEEE International Conference on CAD/CAM, Robotics & Factories of the Future, Pereira, 15-17 December 1997

[2] Kacprzak M., Kaczmarczyk A., Ślesicki M.: Application of Estelle Language for SME Modelling: A Model for Examination of a Creamery Tolerance from Temporal Water Shortage. IFAC 9th Symposium on Information Control in Manufacturing INCOM'98, Nancy–Metz, 24-26 June 1998

[3] Kaczmarczyk A., Kacprzak M., Jaworska E.: Enterprise Computer Model for a Training Center. Communication about Work in Progress. 2nd International Conference Quality Assurance within Engineering Higher Education EQAS'98, Zakopane, 13-16 September 1998

[4] Kaczmarczyk A.: IDEF – metody modelowania i projektowania do komputerowo wspomaganej inżynierii biznesu. Część I: fundamenty. Informatyka 11/1998, s.15-23

[5] Kaczmarczyk A.: IDEF – metody modelowania i projektowania do komputerowo wspomaganej inżynierii biznesu. Część II: integracja. Informatyka 12/1998, s.16-20

[6] Scientific and Technological Cooperation Joint Project for Years: 2000-2001 Methods and tools for SMEs on-line telework

WOJCIECH PRZYŁUSKI
INSTYTUT MASZYN MATEMATYCZNYCH WARSZAWA

Moduły egzaminujące w publikacjach edukacyjnych
Computerised examination systems

Streszczenie

Niniejsze opracowanie stanowi trzecią część cyklu poświęconego w naszym biuletynie egzaminom komputerowym. W części pierwszej ([3]) zaprezentowano system Egzaminator przeznaczony do budowania testów komputerowych. W części drugiej ([5]), adresowanej szczególnie do nauczycieli szkolnych i akademickich, zwrócono uwagę na przydatność tego oprogramowania w ocenianiu uczniów i studentów. W części trzeciej porównamy system Egzaminator z innymi wybranymi programami egzaminującymi omawiając jednocześnie ich wady i zalety.

Abstract

The paper is the consecutive part of author's considerations on computerised examination problems. In the first part ([3]), the original software called Egzaminator designed for computer test building was presented. In the second one ([5]) the usefulness of that software in pupils' evaluation was described. This paper contains comparison between the Egzaminator system and other programs of similar application area.

1. Wstęp

Na polskim książkowym rynku wydawniczym pojawia się coraz więcej publikacji edukacyjnych z dołączonymi na dyskietkach lub płytach CD kwizami, sprawdzianami lub egzaminami. Również w bardzo wielu edukacyjnych publikacjach elektronicznych można znaleźć mniej lub więcej rozbudowane moduły, które umożliwiają zadawanie użytkownikowi pytań, proponujące zadania do rozwiązania czy też przeprowadzające z nim grę.

Te wszystkie kwizy, gry i testy służą uatrakcyjnieniu korzystania z danej publikacji książkowej czy elektronicznej, ale oprócz funkcji czysto zabawowych, umożliwiają użytkownikowi sprawdzenie stopnia opanowania danej dziedziny wiedzy. Poświęcimy uwagę tym modułom, które przeprowadzają w sposób profesjonalny egzaminy (testowe) i mogą być przydatne w szkole czy na uczelni.

2. Moduły egzaminujące

W występujących na rynku modułach egzaminujących dostrzec można wiele podobnych elementów. Z każdym modułem związana jest na ogół pewna baza pytań, wykorzystywana do tworzenia egzaminu, dany moduł steruje przebiegiem egzaminu, poszczególne odpowiedzi użytkownika mogą być skomentowane, czasem pojawiają się dodatkowe informacje w postaci ilustracji lub różnych pomocy naukowych, a egzamin najczęściej kończy się komunikatem o uzyskanym wyniku. Tak więc baza pytań, tworzenie testu, przebieg egzaminu i system oceniania – to charakterystyczne elementy, które można wykorzystać do porównania poszczególnych modułów.

Wybraliśmy do tego porównania trzy produkty: *Przyjazne testy z komputera*, *Test* i *Egzaminator*. Różnią się one znacznie między sobą, ale w sposób fachowy realizują sprawdzanie wiedzy i mogą być wykorzystane w procesie nauczania przez nauczycieli szkolnych czy nawet, jak w przypadku *Egzaminatora*, nauczycieli akademickich.

2.1. Przyjazne testy z komputera

Na jednej płycie CD umieszczone są trzy programy komputerowe ([4]):

SuperMemo Nauka – służący do przyswajania i utrwalania wiedzy.

Tester – przeznaczony do sprawdzania i samooceny wiedzy.

Generator testów – umożliwiający tworzenie testów i drukowanie ich wraz z kartami odpowiedzi i szablonami.

W rozważaniach dotyczących modułów egzaminujących interesować nas będzie oczywiście tylko drugi z wymienionych programów – *Tester*. Środowiskiem wykonawczym programu jest Windows 95.

Baza i tworzenie egzaminu

Program *Tester* zawiera zbiór 600 zadań z zakresu klasy VI-tej. Zadania są uporządkowane tematycznie oraz mają przyporządkowania do odpowiednich poziomów wymagań programowych. Wyodrębniono trzy grupy zadań: łatwe, średnie i trudne. Określono również kategorię zadań podstawowych dla przedmiotu nauczania. Aby utworzyć test należy wybrać jedynie odpowiednie działy tematyczne, po czym program automatycznie wybiera zadania do sprawdzianu (30-zadaniowy test standardowy) stosując schemat:

- 4 zadania z wiadomości podstawowych,
- 8 zadań łatwych,
- 10 zadań średnio trudnych,
- 8 zadań trudnych.

Możliwe jest również przygotowanie 60-zadaniowego testu powtórzeniowego.

Przebieg egzaminu i system oceniania

Pasek *menu* umożliwia zdającemu obsługę programu. Cztery przyciski A-D pozwalają na wybór jednej z odpowiedzi. Przyciskami „>" i „<" można odpowiednio przechodzić do następnych lub wracać do poprzednich zadań zmieniając nawet wielokrotnie swoje odpowiedzi. Na pasku *menu* wyświetlany jest również czas, jaki pozostał do zakończenia egzaminu. Dla testów 30-zadaniowych ograniczono czas pracy do 30 minut, a do 60 minut dla testów 60-zadaniowych. Egzamin można zakończyć w każdej chwili przyciskiem *Zakończ*. Po zakończeniu egzaminu pojawia się tablica podsumowująca, zawierająca krótką informację o wyniku testu. Zdający może wpisać do niej swoje dane, a następnie wydrukować „świadectwo" egzaminu. Na zakończenie można też sprawdzić, które zadania rozwiązano poprawnie, a które błędnie.

Oceną testu jest ocena „szkolna" z zakresu niedostatecznie – bardzo dobrze, dobrana w zależności od liczby odpowiedzi poprawnych, ściśle według specjalnej tabeli.

2.2. Test

Program *Test* dołączony jest na dyskietce do książki Waldemara Ufnalskiego „Chemia w szkole średniej. Powtórzenie i testy egzaminacyjne" ([7]). Środowiskiem wykonawczym programu jest MS-DOS.

Baza i tworzenie egzaminu

W programie *Test*, wykorzystując bazę 1681 pytań testowych podzielonych na 8 tematycznych działów, możliwe jest tworzenie różnych egzaminów. Główne menu programu ułatwia szybkie przygotowanie egzaminu. Aby sprawdzić swoją wiedzę, wystarczy wybrać zakres materiału (dział lub „cały materiał"), określić liczbę pytań oraz czas przeznaczony średnio na jedno pytanie, oraz ustalić metodę pracy („trening" lub „egzamin"). W metodzie „egzamin" pytania są losowane z podanego zakresu, natomiast w przypadku „treningu" można również wybrać do testu konkretne, ale tylko kolejne, pytania z danego działu. Specjalna opcja programu *Edytor zestawów pytań* umożliwia, korzystając z listy wszystkich pytań, tworzenie dowolnych zestawów pytań oraz dowolne określenie parametrów pracy. Tak przygotowane testy mogą już być zapisywane w plikach dyskowych i wielokrotnie wykorzystywane do różnych sprawdzianów czy egzaminów.

Przebieg egzaminu i system oceniania

Podczas egzaminu prezentowane są na ekranie komputera dwa paski: *status* i *menu*.

Pasek *status* zawiera opis postępów pracy zdającego. Wyświetlana jest informacja: o liczbie pytań, na które nie udzielono odpowiedzi, o łącznej liczbie pytań oraz analogiczne informacje dotyczące czasu. Porównywany jest stopień zaawansowania pracy z wykorzystaniem czasu. Na tej podstawie określa się *stan* informujący zdającego, czy pracuje zgodnie z założoną szybkością (*stan*: 0), szybciej (*stan*: dodatni), czy też wolniej (*stan*: ujemny). Przekroczenie limitu czasu w przypadku metody pracy „egzamin" powoduje przerwanie testu i wyświetlenie wyników.

Pasek *menu* umożliwia zdającemu obsługę programu. Cztery przyciski A-D umożliwiają wybór jednej z odpowiedzi. Przycisk *Pomiń* powoduje chwilowe pominięcie pytania, pominięte pytania zadane zostaną na końcu testu. Przycisk *Esc* pozwala na zakończenie egzaminu w dowolnej chwili. Jeśli wybraną metodą pracy jest „trening", wtedy błędne odpowiedzi są zaznaczane sygnałem dźwiękowym oraz na pasku *menu* dostępne będą przyciski *Wyjaśnij*, *Oblicz*, i *Narzędzia*. Przycisk *Wyjaśnij* umożliwia wyświetlenie poprawnej odpowiedzi oraz ewentualnie dodatkowych wyjaśnień. Przycisk *Oblicz* udostępnia zdającemu kalkulator umożliwiający wykonanie szybkich obliczeń, np. mas molowych czy składu elementarnego. Przycisk *Narzędzia* pozwala skorzystać z wielu programów narzędziowych pomocnych przy rozwiązywaniu testu. Jednym z nich jest *Układ okresowy* z podstawowymi danymi pierwiastków i substancji prostych. Inne narzędzia umożliwiają szybkie wykonanie wielu rutynowych zadań, jak np. obliczanie składu mieszaniny, przeliczanie stężeń roztworów. Czas korzystania z narzędzi jest traktowany jako czas pracy nad testem.

Oceną ogólną testu jest ocena „szkolna" z zakresu niedostatecznie – bardzo dobrze, dobrana w zależności od procentu odpowiedzi poprawnych w ogólnej liczbie pytań. Ocenie towarzyszy komentarz o osiągniętych wynikach i ewentualne zalecenia jak dalej sprawdzać opanowanie materiału i przygotowywać się do egzaminów.

Po zakończeniu egzaminu istnieje możliwość wyświetlenia poprawnych odpowiedzi na wszystkie pytania egzaminacyjne lub tylko na pytania, na które udzielono złych odpowiedzi lub które pominięto.

2.3. Egzaminator

Egzaminator umożliwia pisanie i realizację egzaminów komputerowych z dowolnych dziedzin wiedzy. Przygotowanie egzaminu polega na opracowaniu specjalnego pliku tekstowego (źródła testu) zawierającego pytania z zestawami odpowiedzi oraz wszystkimi parametrami w pełni definiujących przebieg egzaminu. *Egzaminator* składa się z pięciu programów:

1. *Edytora Testów* – ułatwiającego (szczególnie początkującym) przygotowanie źródła testu.
2. *Edytora Ilustracji* – pomagającego w ułożeniu ilustracji wykorzystywanych podczas egzaminu.
3. *Analizatora* – służącego do analizy poprawności źródła testu i zbioru ilustracji. Wynikiem analizy, która nie wykazała żadnych błędów, jest zakodowane źródło testu oraz zbiór adresów ilustracji.
4. *Interpretatora* – wykorzystującego zakodowane źródło testu i zbiór adresów ilustracji do przeprowadzania egzaminu.
5. *Monitora* – monitorującego egzamin, opracowującego protokół i pokazującego rozkłady odpowiedzi.

W rozważaniach porównawczych interesować nas będzie jedynie opracowana przy pomocy ww. programów konkretna aplikacja – moduł, jaką jest np. *Egzamin z dziejów Polski* – program egzaminujący przeznaczony dla uczniów przygotowujących się z historii do matury oraz do egzaminów wstępnych na wyższe uczelnie. Egzamin ten

opracowany został przez historyka oraz informatyków z Instytutu Maszyn Matematycznych w Warszawie i dołączony jest na dyskietce do książki Alicji Dybkowskiej, Jana Żaryna oraz Małgorzaty Żaryn pt. „Polskie dzieje od czasów najdawniejszych do współczesności" ([2]). Środowiskiem wykonawczym programu jest MS-DOS.

Baza i tworzenie egzaminu

Wspomniane wyżej źródło testu to właściwie tekst programu napisany w specjalnym języku do tworzenia egzaminów komputerowych. Opracowujący egzamin może w tym języku łatwo zapisać pytania z zestawami odpowiedzi, przypisać każdemu pytaniu wagę, która wyznacza jego merytoryczną ważność w stosunku do pozostałych pytań, odpowiednio wprowadzić ich paragrafowy podział, ustalić koncepcję ich sekwencjonowania, związać odpowiednie ilustracje z pytaniami lub odpowiedziami, określić system ocen, zdecydować o czasie trwania i zakończeniu egzaminu, przygotować formę dialogu z egzaminowanym (komentarze, pytania niemerytoryczne), jednym słowem zrealizować swoją wizję egzaminu komputerowego. Tak właśnie opracowany został *Egzamin z dziejów Polski* zawierający 350 pytań i kilkanaście ilustracji.

Dokładniejszy opis tworzenia egzaminów w środowisku *Egzaminator* znajdzie czytelnik w pracach [5] i [6].

Przebieg egzaminu i system oceniania

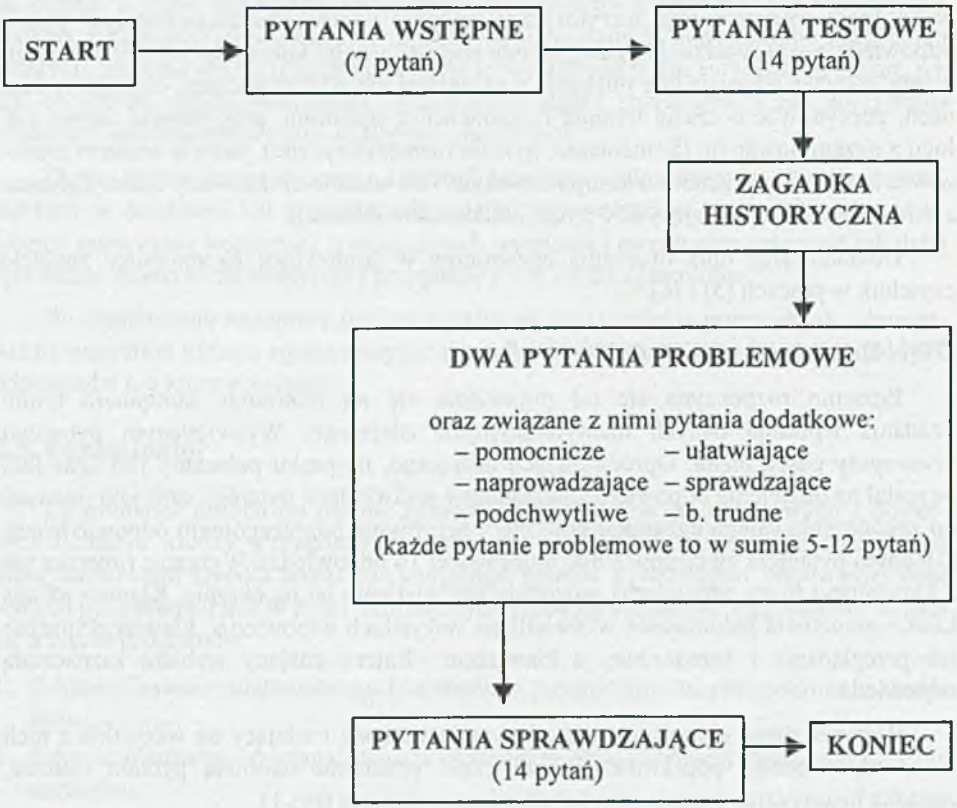
Egzamin rozpoczyna się od pojawienia się na monitorze komputera tytułu i żądania wpisania danych identyfikacyjnych zdającego. Wyświetlanym pytaniom towarzyszy pasek *menu*. Oprócz danych zdającego, na pasku pokazany jest czas jaki pozostał na udzielenie odpowiedzi na aktualnie wyświetlane pytanie i czas jaki pozostał do zakończenia całego egzaminu oraz litery przypisane poszczególnym odpowiedziom. Z jednym pytaniem związanych być może nawet 16 odpowiedzi. Wybranie (myszką lub z klawiatury) litery odpowiedzi powoduje wyświetlenie jej na ekranie. Klawisz <Caps Lock> umożliwia jednoczesne wyświetlenie wszystkich odpowiedzi, klawisz <Spacja> ich przeglądanie i zaznaczanie, a klawiszem <Enter> zdający wybiera zaznaczoną odpowiedź.

Pierwsza część egzaminu to łatwe pytania wstępne i zdający na wszystkie z nich musi odpowiedzieć poprawnie. Dalszą część egzaminu stanowią pytania testowe, zagadka historyczna, pytania problemowe i sprawdzające (rys.1).

Pytania problemowe stanowią najważniejszą część egzaminu. Odpowiedzi na te pytania są przez program szczegółowo analizowane i w wyniku tej analizy zdający poddawany jest odpowiedniej serii pytań dodatkowych: pomocniczych, ułatwiających, naprowadzających, sprawdzających, podchwytliwych i bardzo trudnych. W konsekwencji zwiększa to znacznie rzetelność końcowej oceny. Jak już wspomniano czas na udzielanie niektórych odpowiedzi jest ograniczony. Natomiast czas całkowity egzaminu ograniczono do 45 minut.

Po zakończeniu egzaminu wyświetlone są wyniki lub komunikat o przerwaniu egzaminu. Wynik egzaminu jest przedstawiony w końcowym oknie wraz z porównaniem poprzednich egzaminów zdającego oraz wyników innych zdających. Jeśli egzamin zakończono prawidłowo, dane identyfikacyjne oraz wyniki będą zapamiętane w pliku wyników, a ponadto osobno pamiętana jest również lista wszystkich udzielonych

odpowiedzi. Przebieg egzaminu zależy od elementów losowych (losowanie pytań), a każdy następny krok egzaminu zależy od kroków poprzednich, bowiem treść pytań, stopień ich trudności, ich liczba, kolejność oraz pojawiające się komentarze zależą od odpowiedzi zdającego. Dlatego też każde nowe uruchomienie programu to inny niepowtarzalny przebieg egzaminu.



Rys. 1 *Egzamin z dziejów Polski* – schemat przebiegu

Oceną ogólną egzaminu jest ocena „szkolna" z zakresu od 1 do 6. Poszczególne etapy egzaminu wpływają na tę ocenę w sposób następujący: część testowa oraz zagadka historyczna 15% oceny, część problemowa 75% oceny oraz część sprawdzająca 10% oceny.

Egzamin jest trudny i wykracza poza ramy dołączonego podręcznika. Kilkakrotne zdanie egzaminu z oceną powyżej 3,5 świadczy, że uczeń jest dobrze przygotowany do matury oraz do egzaminów wstępnych na uczelnie.

3. Porównanie modułów

Trudno jest porównać ze sobą trzy wyżej opisane moduły. Bowiem twórcy programów *Test* i *Przyjazne testy z komputera* postawili sobie trochę inny cel niż twórcy *Egzaminatora*. Dwa pierwsze programy to narzędzia do szybkiego i wygodnego tworzenia sprawdzianów testowych z wykorzystaniem dużej bazy pytań. Moduły egzaminujące tych programów to komputerowa realizacja klasycznych szkolnych testów kartkowych. *Przyjazne testy z komputera* zawierają wręcz specjalny *Generator testów* umożliwiający tworzenie testów i drukowanie ich wraz z kartami odpowiedzi i szablonami. Natomiast ostatni moduł to próba wykorzystania komputera do przeprowadzenia egzaminu indywidualnego zbliżonego bardziej do egzaminu ustnego niż do kartkowego testu.

W tabeli poniżej zestawiono wybrane cechy dwóch pierwszych modułów egzaminujących.

Przyjazne testy z komputera to cykl (nie tylko matematycznych) programów, których twórcy położyli nacisk na pełną standaryzację testów. Przestrzegają precyzyjnie określonych warunków testowania, to znaczy liczby i doboru pytań, normy czasowej i zasady przeliczania punktów na ocenę szkolną. Program ten uniemożliwia tworzenie testów niezgodnych z przyjętymi standardami. Takie podejście pozwala na międzyszkolne, a nawet interdyscyplinarne porównywanie wyników nauczania. Natomiast *Test* to pojedyncza aplikacja zorientowana wyłącznie na tematykę chemiczną, a tworzone w niej różnorodne testy i ich wyniki mają służyć tylko uczniowi i nauczycielowi do powtarzania materiału i sprawdzania wiadomości w toku zajęć szkolnych

Jeśli nawet w cyklu *Przyjazne testy* powstanie program „chemiczny" według opisanych wyżej reguł, to uwzględniając w programie *Tester* znakomite wyposażenie w narzędzia pomocnicze oraz rozbudowane komentarze, wydaje się – z punktu widzenia pojedynczego użytkownika, że wygodniejsze i bardziej kształcące jest korzystanie z tego właśnie programu. Jednak pewne cechy *Przyjaznych testów,* takie jak: swobodny wybór tematów do zestawu, możliwość zmiany udzielonych już odpowiedzi czy bieżące odświeżanie czasu, są godne naśladowania i stanowią niewątpliwe atuty tego programu.

Zauważmy, że w czasie każdego egzaminu komputerowego zdający wybiera **konkretne** odpowiedzi na kolejno zadawane pytania i ta jakże istotna informacja choć zapisana w pamięci komputera jest w większości modułów egzaminujących wykorzystana tylko (ilościowo lub procentowo) na końcu egzaminu do wystawienia oceny i zaprezentowania osiągniętych wyników. Natomiast informacja ta mogłaby zostać wykorzystana po pierwsze w trakcie egzaminu, a po drugie – ocena mogłaby uwzględniać to, **jakich** odpowiedzi udzielono na **jakie** pytania. Niestety w większości modułów egzaminujących komputer spełnia rolę „sekretarki szkolnej" – przechowuje pytania i zestawy egzaminacyjne, tworzy nowe zestawy pytań, drukuje testy i szablony, nadzoruje rozwiązywanie testu, sprawdza mechanicznie rozwiązane testy i informuje o uzyskanych wynikach. A przecież można pokusić się o to, by komputer spełniał również rolę „nauczyciela" – dobierał pytania indywidualnie, zależnie od dotychczasowych odpowiedzi zdającego, wyjaśniał trudniejsze problemy w czasie lub po egzaminie, naprowadzał jeśli trzeba na właściwą odpowiedź, oceniał pełniej (w wielu aspektach) stopień opanowania wiedzy i komentując uzyskany wynik dawał wskazówki dotyczące dalszej nauki. W pewnym, niewielkim co prawda, zakresie funkcje te realizuje program.

CECHY MODUŁÓW	*Test*	*Przyjazne testy z komputera*
Środowisko wykonawcze	MS-DOS	Windows 95
Kategoryzacja bazy pytań	brak	czterostopniowa
Wybór tematyki zestawu	całość materiału i 8 grup tematycznych	dowolny wybór tematów
Możliwość tworzenia różnych rodzajów zestawów pytań	zestawy z różną liczbą pytań oraz możliwość wybierania konkretnych pytań do zestawu	zestawy 30 lub 60-pytaniowe generowane przez system
Standaryzacja zestawów pytań	brak	wysoka
Czas trwania egzaminu	nieograniczony lub uzależniony od liczby pytań w zestawie i średniego czasu przeznaczonego na jedno pytanie (od 1 do 5 minut)	30 lub 60 minut
Możliwość zmiany udzielonych odpowiedzi	nie	tak
Podstawa wyliczania oceny	procent poprawnych odpowiedzi	liczba poprawnych odpowiedzi
Bieżące odświeżanie czasu	nie	tak
Dodatkowe narzędzia i pomoce naukowe	bardzo rozbudowane	brak
Opis wyników, instrukcje i zalecenia	obszerne	bardzo ograniczone

Test. Natomiast w pełni rolę „nauczyciela-egzaminatora" realizują moduły egzaminujące opracowane w programie *Egzaminator*. Ponadto warto zauważyć, że każdy test (z wyłączeniem specjalnych narzędzi pomocniczych programu *Test*) zrealizowany przy użyciu programów *Test* lub *Przyjazne testy z* komputera można również zrealizować za pomocą *Egzaminatora*. Uniwersalność *Egzaminatora* ma jednak dla potencjalnych użytkowników swoją cenę, a jest nią konieczność opanowania specjalnego języka programowania umożliwiającego jednak tworzenie nawet bardzo wyrafinowanych egzaminów komputerowych.

Literatura

[1] Brzostek J., Janowski J., Przyłuski W.: Egzaminator i Edytor testów. XI Konferencja Informatyka w Szkole, Kielce, 1995

[2] Dybkowska A., Żaryn J., Żaryn M.: Polskie dzieje od czasów najdawniejszych do współczesności. (z dyskietką zawierającą: egzamin z dziejów Polski), PWN, Warszawa,1999

[3] Janowski J., Przyłuski W.: Jądro komputerowego systemu oceny poziomu wiedzy. Techniki Komputerowe Biuletyn Informacyjny, Nr 1, 1993

[4] Przyjazne testy z komputera. Matematyka dla szkoły podstawowej (kl. VI). Wydawnictwo Szkolne PWN Sp. z o. o. i Vulcan Media, Warszawa-Wrocław, 1998

[5] Przyłuski W.: Egzaminator – system do tworzenia egzaminów komputerowych. Techniki Komputerowe Biuletyn Informacyjny, Nr 1, 1998

[6] Przyłuski W.: Egzaminy przyszłości. Pckurier, Nr 20, 1998

[7] Ufnalski W.: Chemia w szkole średniej. Powtórzenia i testy egzaminacyjne. (z dyskietką zawierającą program Test), Wydawnictwa Naukowo-Techniczne, Warszawa, 1998

LEON ROZBICKI, JAN RYŻKO, JERZY SŁAWIŃSKI
INSTYTUT MASZYN MATEMATYCZNYCH WARSZAWA

Charakterystyki systemów kontroli dostępu i kierunki ich rozwoju
Characteristics of access control systems and directions of their development

Streszczenie

Artykuł przedstawia podstawowe zadania i podzespoły systemów kontroli dostępu oraz pokazuje przykładowe rozwiązania, również w wersji sieciowej. Nakreślone są kierunki rozwoju tych systemów, a także problemy związane z ich integracją z innymi systemami.

Abstract

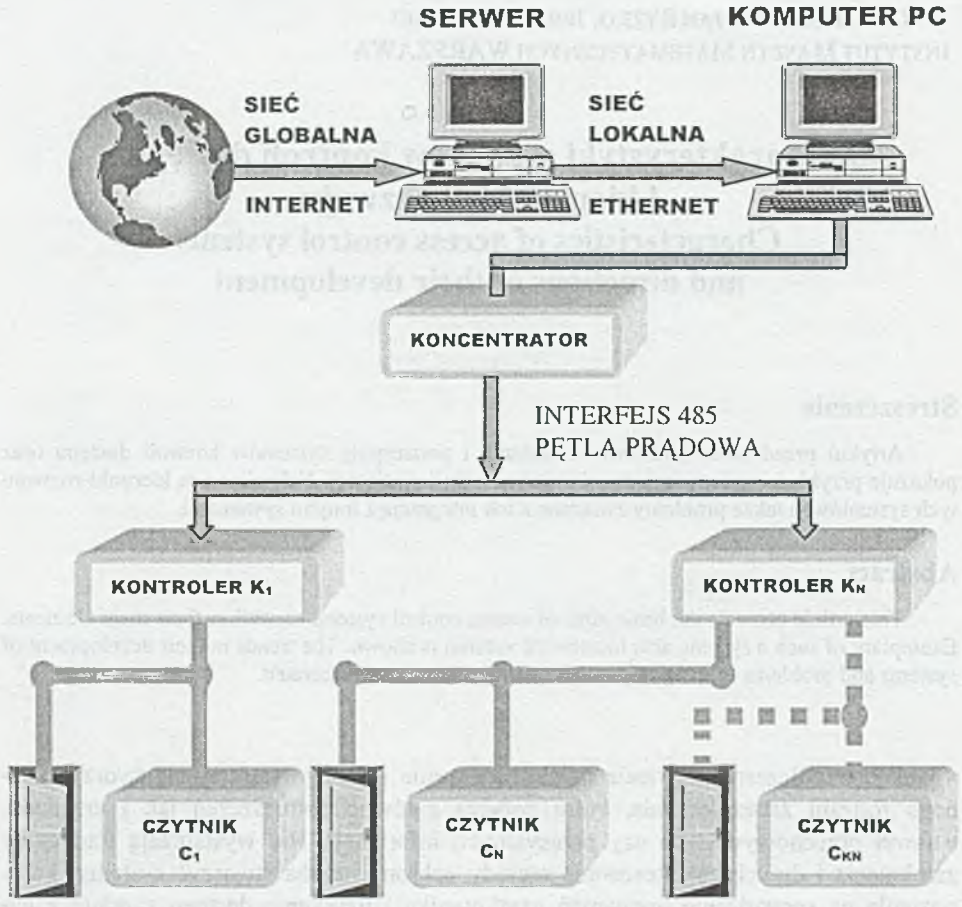
This article presents the basic aims of access control systems as well as their main elements. Exemplary of such a system, also in network version is shown. The trends in their development of systems and problems of integration with other systems are enumerated.

We współczesnym świecie rośnie zagrożenie i dlatego staramy się tworzyć różnego rodzaju zabezpieczenia, które dotyczą zarówno pomieszczeń jak i urządzeń, a nawet przechowywanych czy przesyłanych informacji. Nie wystarczają tradycyjne zamknięcia i dotychczas stosowane metody ochrony, trzeba stworzyć systemy, które pozwolą na sprawdzenie uprawnień użytkownika i uzyskanie dostępu zgodnie z ich zakresem. Są to zazwyczaj systemy informatyczne, obejmujące szeroki zakres od prostych urządzeń wbudowanych np. w klawiaturę komputera, umożliwiających korzystanie z niego uprawnionym osobom, po zintegrowane systemy tzw. inteligentnego budynku łączące w sobie zabezpieczenia, telewizję przemysłową, sygnalizację włamania czy pożaru, nadzór itp. Do zadań tych systemów można zaliczyć:

- kontrolę ruchu użytkowników i ich obecności w obszarach chronionych,
- zabezpieczenie majątku i chronionych danych,
- bezpieczeństwo pracowników i klientów,
- dostarczanie danych do dalszego przetwarzania,
- możliwość integracji z innymi systemami.

1. Przykładowe systemy kontroli dostępu

Przyjrzyjmy się systemowi kontroli dostępu pokazanemu na rys. 1. System ten sterowany jest z komputera PC, który może być dołączony do serwera lokalnej sieci komputerowej (np. typu Ethernet), a ten z kolei do sieci rozległej (może to być Internet).

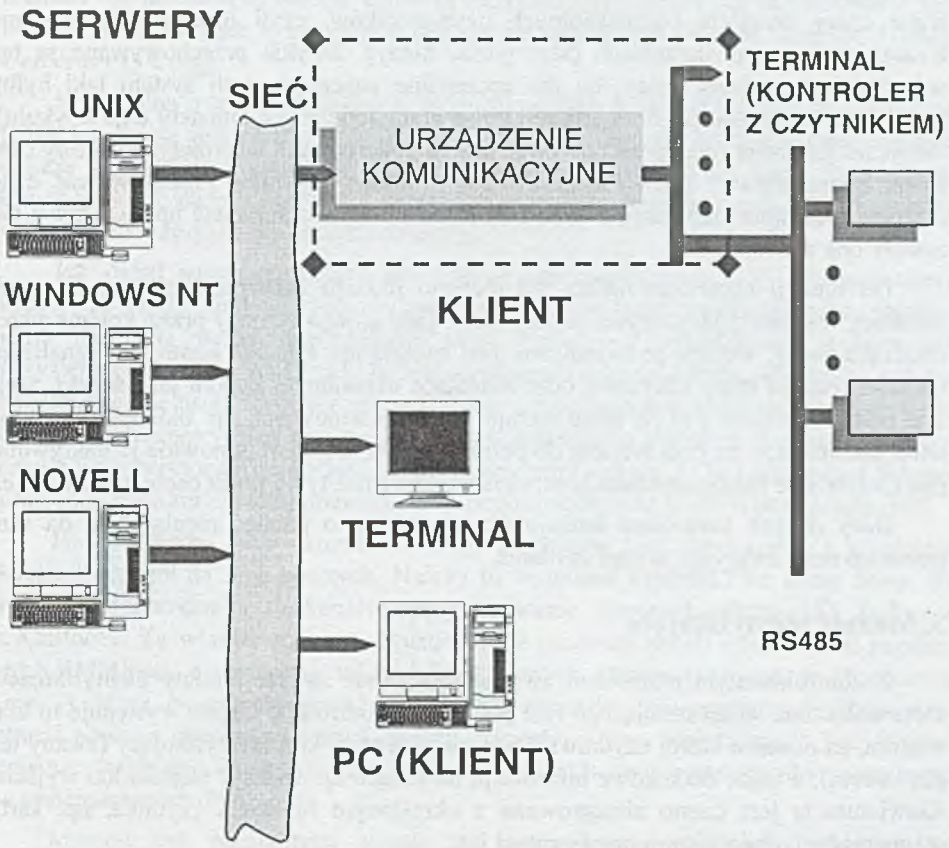


Rys 1. Przykład struktury systemu kontroli dostępu

Jeśli system jest bardziej rozbudowany i wykorzystuje wiele kontrolerów, wybór aktywnego kontrolera odbywa się poprzez koncentrator, do którego można dołączyć określoną ilość kontrolerów. Połączenie z koncentratora do kontrolerów odbywa się poprzez interfejs RS485 lub pętlę prądową. Kontroler jest podstawowym elementem systemu i do niego może być dołączona określona ilość modułów dostępu złożonych z czytnika i elementu wykonawczego sterowania. Chronione mogą być zarówno pojedyncze obiekty jak i ich zespoły. Praca może być wykonywana w systemie off-line

i wówczas zastosowanie inteligencji rozproszonej pozwala na szybkie podejmowanie przez kontroler decyzji o udostępnieniu przejścia uprawnionemu użytkownikowi. Natomiast praca w systemie on-line daje znacznie większe możliwości działania systemu łącznie z jednoczesną obsługą odległych filii przedsiębiorstwa, dostępem do dużej bazy danych itd.

Zanim przejdziemy do szczegółowego omówienia funkcji poszczególnych podzespołów systemu, przypatrzmy się przykładowemu rozwiązaniu sieciowemu, pokazanemu na rys. 2. Działa on w systemie klient – serwer, przy czym serwery mogą pracować pod nadzorem systemu operacyjnego UNIX, Windows NT lub Novell. Poprzez sieć komputerową docierają one do klientów, którzy mogą być reprezentowani przez swe komputery osobiste (PC), terminale z monitorem ekranowym, bądź bezpośrednio przez urządzenia komunikacyjne, które tak jak poprzednio sterują szereg terminali, którymi są zazwyczaj kontrolery z czytnikami.



Rys. 2. Przykładowe rozwiązanie sieciowe kontroli dostępu

2. Podstawowe funkcje kontrolera

Kontroler steruje obiektami i nadzoruje ich stany. Najczęściej chodzi tu o tzw. przejścia, a więc przede wszystkim różnego rodzaju drzwi, a także bramy, szlabany, kraty czy inne urządzenia chroniące przed dostępem osób niepowołanych. Przejście ma zwykle jednoznaczną nazwę związaną z jego funkcją lub lokalizacją. Jeśli dany punkt wymaga kontroli uprawnień w obu kierunkach z określaniem kierunku ruchu, to określone tu muszą być dwa przejścia – oddzielnie dla wejścia i wyjścia.

Główną funkcją kontrolera jest więc otwarcie (odblokowanie) przejścia dla osób uprawnionych. Ponadto powinien on przechowywać dane o przechodzących osobach i zakresie ich uprawnień, a także o sygnalizacji sforsowania przejścia. Dane te odczytywane są zwykle przez moduły identyfikujące, które będą omówione w następnym rozdziale.

Kontroler przechowuje też zwykle kalendarz zawierający kolejne dni tygodnia, z uwzględnieniem zmian uprawnień w poszczególne dni (np. świąteczne) i zarejestrowane czasy przejścia poszczególnych użytkowników, czyli inaczej mówiąc dane o zarejestrowanych zdarzeniach (kto, gdzie, kiedy). Zwykle przechowywane są też w kontrolerze godziny pracy, co ma szczególne znaczenie, jeśli system taki byłby wykorzystywany również do rejestracji czasu pracy (co nie jest tematem tego artykułu). Wówczas kontroler musi przechowywać szereg dodatkowych informacji jak strefy czasowe, harmonogramy itd. Natomiast każdy kontroler powinien przechowywać dane o zarządzaniu uprawnieniami, a więc o tym kto ma prawo zmieniać uprawnienia i jak zostały one zmienione.

Do funkcji kontrolera należą też różnego rodzaju zabezpieczenia jak np. antipassback, zapobiegający użyciu jednej karty (lub innego żetonu) przez kolejne przechodzące osoby, wejście ze świadkiem, pod eskortą itp. Ponadto kontroler sygnalizuje różnego rodzaju stany alarmowe odpowiadające określonym kodom jak sabotaż, wejście pod przymusem i in., a także steruje liniami alarmowymi, np. blokując linie systemu alarmowego na czas wejścia do pomieszczenia, a potem ponownie je uaktywniając. Oczywiście konfigurowanie kontrolera możliwe jest tylko przez osoby uprawnione.

Bazy danych kontrolera funkcjonują w oparciu o pamięć nieulotną, a on sam powinien mieć awaryjny system zasilania.

3. Moduły identyfikujące

Z kontrolowanym przejściem związane są pawie zawsze moduły identyfikujące, które dołączone są do sterującego tym przejściem kontrolera. Często występuje tu klawiatura, za pomocą której użytkownik wprowadza swój kod identyfikujący (zwany też pin-kodem), a także dodatkowe informacje dotyczące np. rodzaju wejścia lub wyjścia. Klawiatura ta jest często zintegrowana z określonym rodzajem czytnika, np. karty magnetycznej, zbliżeniowej, inteligentnej itp.

Dotychczas najbardziej popularne były karty magnetyczne, zawierające pasek materiału magnetycznego, których przesunięcie przez szczelinę czytnika powodowało odczyt danych zapisanych na tym pasku. Pozwala to sprawdzić zgodność kodu wprowadzonego z klawiatury. Inne rodzaje stosowanych tu czytników to czytniki kodu

kreskowego jaki jest stosowany do oznaczania towarów w sklepach, lub czytniki kart Wieganda ([1]), w których zatopione są druciki magnetyczne o określonej konfiguracji, indukujące przy bezdotykowym przesunięciu przez szczelinę czytnika – odpowiedni sygnał w jego spirali.

Ostatnio stają się bardziej popularne karty elektroniczne, zwane też chipowymi lub inteligentnymi (SmartCards), które dzielą się na mikroprocesorowe i pamięciowe, na których można zapisać więcej informacji niż na innych rodzajach kart, a ponadto informacja ta może być modyfikowana w trakcie odczytu. Stąd zastosowania takich kart do bankomatów, gdzie przy operacji uaktualniany jest stan konta użytkownika, lub do zabezpieczenia dostępu do komputera (np. program Siemensa SmartGuard). Również rola tych kart i ich czytników w systemach kontroli dostępu będzie niewątpliwie rosła.

Ostatnim rodzajem omawianych tu kart (lub tego rodzaju nośników o innym kształcie, jak breloczki, „pastylki" itp.) są elementy zbliżeniowe ([2]). Zaletą ich, jak łatwo się domyślić, jest możliwość kontroli dostępu bez konieczności wyjmowania karty, czy w pewnych okolicznościach opuszczania pojazdu, o ile czytnik takich elementów ma dostateczny zasięg. Czytniki elementów zbliżeniowych są stosowane jako moduły identyfikujące w systemach kontroli dostępu takich znanych firm jak Aritech, Cerberus czy Cotag.

Szczególną rolę wśród modułów identyfikujących spełniają czytniki biometryczne ([3]). Ich zaletą jest wyeliminowanie możliwości posługiwania się cudzymi kartami, a także konieczności zapamiętywania kodów identyfikacyjnych i haseł. Początkowo były one dosyć drogie, ale wobec nawet kilkunastokrotnego obniżenia ceny ([4]) stały się one konkurencyjne dla innych rozwiązań.

Jak dotąd najpopularniejszy był czytnik linii papilarnych ([5]). Przypomnijmy krótko, że dzięki niepowtarzalności odcisków palców poszczególnych ludzi, uzyskujemy skuteczną metodę identyfikacji, którą możemy z powodzeniem wykorzystać w systemach dostępu. Wadą są tu pewne wymagania odnośnie stanu odbijanych palców i występowanie osób, które nie mogą być tak identyfikowane. Powszechne stosowanie tej metody biometrycznej stwarza dodatkowe wymagania odnośnie dobrego zabezpieczeniu przechowywanych odcisków, gdyż popularny system AFIS (Automated Fingerprint Identification System) pozwala łatwo przyporządkować je ich właścicielom ([6]).

Istnieje szereg technicznych realizacji czytników linii papilarnych i systemów kontroli dostępu na nich opartych. Należy tu wymienić czytniki FIU firmy Sony, do których różnorodne i doskonałe oprogramowanie stworzyła firma I/O Software z Kalifornii. Te właśnie elementy wykorzystuje pierwszy polski czytnik linii papilarnych IMMSkan opracowany w Instytucie Maszyn Matematycznych w Warszawie (rys. 3). Może on przechowywać ponad 1000 wzorców linii papilarnych. Czas rejestracji nowego użytkownika trwa poniżej sekundy, a czas weryfikacji 0,3 s. Współczynnik fałszywej akceptacji jest mniejszy od 0,1%, co wystarcza do zastosowań w systemach kontroli dostępu.

Metoda linii papilarnych została też wykorzystana dla kontroli dostępu do komputerów, np. w układzie stosowany przez firmę Compaq i urządzenie BioMouse, opracowane przez firmę American Biometric Company.

Inny rodzaj czytników biometrycznych, stosowanych powszechnie na największych lotniskach amerykańskich ([6]), to czytniki geometrii dłoni ([7]). Pierwsze

symbolem ID3D, a już obecnie ma wiele wdrożeń różnych innych firm. Czytniki tego rodzaju działają poprawnie mimo postępujących zmian geometrii ciała.

Trzecią pod względem popularności metodą biometryczną jest rozpoznawanie głosu. Czytnik taki jest szczególnym mikrofonem odpowiednio podłączonym do sytemu. W systemach o dużych wymaganiach bezpieczeństwa stosuje się często kombinację kilku metod i rozpoznawanie głosu jest zwykle wtedy stosowane.

Inna grupa czytników biometrycznych związana jest z okiem użytkownika. Aczkolwiek wykorzystanie tych czytników jest o rząd wielkości mniejsze niż omówionych poprzednio, są one coraz częściej stosowane, np. przy bankomatach. Rejestrowany jest zwykle obraz siatkówki użytkownika, choć w przypadku popularnego systemu IriScan, stosowanego zarówno do kontroli dostępu do komputerów, jak i przez wiele banków (NCR, Siemens i Dresdner Bank), dotyczy to tęczówki. Wystarczy spojrzeć w specjalną kamerę i w ciągu około 2 sekund dokonuje się realizacja zadania (uzyskanie dostępu, wypłata pieniędzy).

Rys. 3. Czytnik linii papilarnych IMMSkan

Należy wspomnieć jeszcze o dwóch metodach dotychczas nie omawianych. Są to: weryfikacja podpisu i rozpoznawanie twarzy. Obie wiążą się z tradycyjnymi metodami identyfikowania podpisów i zdjęć na dokumentach, jednakże tutaj nabierają one nowych, „biometrycznych" cech. Porównywany jest nie tylko kształt podpisu, ale i takie elementy jak kąt nachylenia długopisu, czy prędkość z jaką powstają poszczególne fragmenty podpisu. Obraz twarzy zaś uzyskiwany jest natychmiast z kamery cyfrowej i automatycznie porównywany z wzorcami w bazie danych, a w szczególnych przypadkach mamy do dyspozycji rozpoznawanie termiczne, które wykryje próby charakteryzacji lub różnice niewidoczne w wyglądzie zewnętrznym.

Śledząc rozwój układów biometrycznych w systemach kontroli dostępu ([8]) widzimy, że rozwijają się one najbardziej w Ameryce Płn., aczkolwiek ostatnio Europa stara się odrobić zaległości na tym polu. Jeśli chodzi natomiast o udział poszczególnych metod, to jeszcze w 1998 roku prawie połowa zastosowań dotyczyła geometrii ręki. Obecnie udział ten nieco zmalał, podobnie jak rozpoznawanie głosu i cech siatkówki oka. Wzrosła natomiast, w stosunku do roku ubiegłego, popularność rozpoznawania twarzy, linii papilarnych i tęczówki. Przede wszystkim jednak, trzeba podkreślić przechodzenie od eksperymentów do wdrożeń, a nawet seryjnej produkcji w tej dziedzinie, co wiąże się ze znaczną obniżką cen.

Z modułami identyfikującymi łączą się elementy wykonawcze sterowania, którymi są najczęściej blokady elektromagnetyczne drzwi i rygle elektromagnetyczne.

4. Oprogramowanie

Wspominając o poszczególnych systemach kontroli dostępu, wskazywaliśmy, że producenci sprzętu zatrudniają wyspecjalizowane firmy programistyczne do opracowania odpowiedniego oprogramowania, gdyż nie jest to łatwe zadanie. Spróbujmy teraz scharakteryzować oprogramowanie tych systemów.

Tak jak w większości różnych systemów możemy tu rozróżnić oprogramowanie systemowe i użytkowe. Pierwszy rodzaj określony jest przyjętą platformą sprzętowo-programową, na której przewidziane jest użytkowanie systemu. Najczęściej, w przypadku mniejszych i średnich użytkowników, będzie to Windows 95/98 (wkrótce 2000), a w przypadku dużych przedsiębiorstw UNIX lub Windows NT. Rzadziej już napotkamy rozwiązania systemu DOS.

Oprogramowanie systemowe powinno zapewnić możliwość pracy w sieci komputerowej lokalnej lub globalnej; przez tę ostatnią rozumiemy zwykle Internet. Ponadto ten rodzaj oprogramowania zapewnia zwykle systemowi możliwość autokonfiguracji i związany jest z topologią systemu.

Natomiast oprogramowanie użytkowe, mające postać programów związanych z konkretnym zastosowaniem danego systemu, określa przede wszystkim interfejs użytkownika (graficzny lub multimedialny) i stanowi pomost pomiędzy sprzętem systemu (kontrolery, czytniki itd.), a administratorem, operatorami i zwykłymi użytkownikami. Oprogramowanie to charakteryzuje szereg istotnych parametrów systemu takich jak czas reakcji, ilość obsługiwanych użytkowników i obiektów, zakresy uprawnień obsługi, rodzaje obsługiwanych alarmów itp. Ponadto umożliwia ono uprawnionym osobom wykonywanie wielu ważnych dla pełnego wykorzystania systemu czynności jak konfigurowanie systemu, zarządzanie uprawnieniami, tworzenie raportów, autotestowanie i serwisowanie, obsługę zdarzeń alarmowych, a także zapewnia bezpieczeństwo systemu.

Wśród programów wchodzących w skład zintegrowanego pakietu oprogramowania użytkowego systemu kontroli dostępu możemy zwykle spotkać moduły programowe, takie jak menadżer systemu, oprogramowanie operatora, alarmy graficzne, monitor zdarzeń i wiele innych, w zależności od rodzaju systemu.

5. Kierunki rozwoju

Znajdujemy się w okresie burzliwego rozwoju systemów kontroli dostępu. Zwiększa się ilość stosowanych i opracowywanych systemów, ale przede wszystkim poprawia się ich jakość i możliwości, jakie oferują użytkownikom.

Wśród głównych kierunków rozwoju tych systemów należałoby na pierwszym miejscu wymienić udoskonalanie przyjaznego interfejsu użytkownika, który umożliwiałby łatwe i sprawne sterowanie systemem, a także wykorzystanie jego możliwości w zakresie uzyskiwanych informacji, raportów itp. Może to być interfejs zarówno graficzny, pokazujący np. rozmieszczenie czytników w poszczególnych pomieszczeniach i budynkach, jak i multimedialny, z wykorzystaniem dźwiękowego porozumiewania się, telewizji przemysłowej itp.

Istotne jest również rozszerzanie funkcji systemu, poprzez wprowadzanie takich możliwości jak np. monitoring, i skalowalność. Wspominano tu już o integracji systemu kontroli dostępu z innymi systemami, poczynając od ogólnych systemów bezpieczeństwa i zarządzania po specjalizowane systemy takie jak wentylacyjno-klimatyczne i grzewcze, zarządzania energią, sterowania oświetleniem, sterowania i monitorowania urządzeń technicznych, sygnalizacji i wykrywania pożarów i inne.

Obiecującym kierunkiem jest wprowadzenie mechanizmu autokonfiguracji, prowadzące do automatycznego dostosowania się do istniejącej, aktywnej postaci systemu i zmniejszenia ingerencji obsługi. Cenną właściwością jest też rozproszona inteligencja, czyli inaczej mówiąc decentralizacja, pozwalająca na poprawną pracę systemu przy lokalnych awariach, a nawet przy uszkodzeniu centralnego sterowania.

Po tym, co powiedziano w punkcie 3 o biometrycznych metodach identyfikacji, jest oczywiste, że rozwój tych metod jest jak najbardziej wskazany.

Istotną sprawą dla systemów kontroli dostępu jest ich bezpieczeństwo, co niejako wynika z ich natury. Zwiększanie odporności systemu na celowe zakłócanie jego pracy z zewnątrz oraz na podsłuch to jeden ze sposobów poprawiania bezpieczeństwa. Wiąże się z tym utajnianie danych zarówno przy ich przesyłaniu jak i w programach aplikacyjnych.

Wreszcie powinna zwiększać się niezawodność systemów kontroli dostępu, by działały one bardziej pewnie przy możliwie uproszczonej obsłudze.

6. Problemy integracji

Integracja systemów kontroli dostępu z innymi systemami natrafia na szereg problemów, wśród których na czoło wysuwa się brak standaryzacji interfejsów sprzętowych i programowych dla tych systemów, a także funkcji poszczególnych modułów systemu. Brak jest również jednolitych protokołów komunikacyjnych. W istniejących systemach kontroli dostępu stosowane są różne ich rodzaje: RS485, Ethernet, protokół Wieganda, pętla prądowa, Lonworks i szereg innych. Dodatkowym problemem jest niechęć do ujawniania danych, niezbędnych do celów integracji. Utrudnia to wykorzystanie najlepszych rozwiązań poszczególnych firm.

Ostatnim z występujących tu problemów jest brak wystarczającej kadry specjalistów, co wiąże się z tym, że dziedzina jest stosunkowo nowa.

7. Podsumowanie

Mimo wymienionych problemów, integracja systemów kontroli dostępu z innymi systemami inteligentnego budynku przynosi różne, wymierne korzyści. Przede wszystkim dotyczy to projektowania, gdyż można zrobić jeden projekt obejmujący wszystkie systemy.

Również koszt sprzętu i oprogramowania jest w sumie niższy w przypadku integracji, ponieważ te same zasoby mogą być wykorzystane w różnych systemach.

Z reguły wyższa jest niezawodność systemu zintegrowanego, gdyż może on być pod tym kątem optymalizowany. Łatwiejsza jest obsługa takiego systemu – nie potrzeba obsługiwać poszczególnych systemów z osobna.

Z braku miejsca nie będziemy omawiać konkretnych rozwiązań systemów kontroli dostępu, odsyłając Czytelników do dokumentacji technicznej takich rozwiązań. Chcemy wymienić tylko niektóre z nich. System *Granta* firm Cerberus ([9]) i Cotag ([10]), System Kontroli Dostępu ARITECH ([11]) a z systemów zintegrowanych *Infinity* firmy AndoverControls ([12]) i szereg systemów firmy Honeywell ([13]).

Literatura:

[1] Technologia kart Wieganda. Twierdza, 2/99, s. 30

[2] Czytniki kart zbliżeniowych, karty wyrobu. Systemy alarmowe, 5/99, s. 44-45

[3] Skup R.: Biometryczne systemy zabezpieczeń. Twierdza, 2/99, s. 31

[4] Ashbourn J.: Testing Biometrics. http://www.biometric.freeserve.co.uk/testing

[5] Rozbicki L., Ryżko J.: Linie papilarne – najpopularniejsza metoda biometryczna rozwiązywania problemów identyfikacji i weryfikacji. Informatyka, 12/98, s. 25

[6] Ciało jako klucz dostępu, oprac. Nowak M., Chip, 7/99, s. 38

[7] HANDKEY II Czytnik geometrii dłoni. Systemy alarmowe, 5/99, s. 43

[8] Survey: Physical access control. Biometric Technology Today, vol. 7, nr 5, s. 8

[9] Cerberus Granta, System kontroli dostępności do pomieszczeń – opis systemu, Cerberus Poland Sp. z o. o.

[10] Systemy kontroli dostępu Cotag. ID Electronics Sp. z o. o.

[11] http://www.aritech.com.pl

[12] http://www.andovercontrols.com.pl

[13] http://www.honeywell.com.pl