ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: INFORMATYKA z. 35 Nr kol. 1402

Krzysztof DOBOSZ
Politechnika Śląska, Instytut Informatyki

ANALIZA OBIEKTOWA W WIZUALIZACJI ALGORYTMÓW

Streszczenie. W publikacji przedstawiono obiektowe podejście do problemu wizualizacji algorytmów. Starano się zamodelować w postaci obiektowej warstwę wyświetlania, będącą częścią składową systemu wizualizacji dynamicznych struktur danych. Artykuł obejmuje zarówno analizę problemu, jak i przykładową implementację podstawowego obiektu graficznego.

OBJECT ANALYSIS IN VISUALIZATION OF ALGORITHMS

Summary. In the paper, the object approach to the problem of algorithm visualization is presented. I try to model the display part of the dynamic visualization system of data structures. The paper comprises both the analysis of the problem, and a sample implementation of a basic graphic object.

1. Wstęp

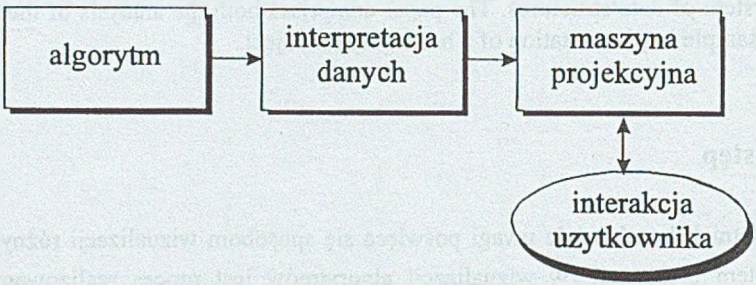
W ostatnich latach wiele uwagi poświęca się sposobom wizualizacji różnych procesów. Przedmiotem prezentacji w wizualizacji algorytmów jest proces realizowany najczęściej przez maszynę cyfrową. Proces taki charakteryzuje się określonym przepływem sterowania oraz dokonywaniem zmian zawartości w wykorzystywanych strukturach danych. Demonstracja tych aspektów procesu jest celem wizualizacji algorytmów. Systemy wizualizacji wykorzystują do tego statyczne i dynamiczne metody prezentacji. W metodach statycznych zmianę stanu obserwowanego obiektu prezentuje się wyświetlając odpowiedni element graficzny w ustalonym miejscu ekranu. Natomiast dynamiczna wizualizacja zawiera elementy animacji, które umożliwiają zilustrowanie wielu interesujących aspektów algorytmów, trudnych do przedstawienia za pomocą metod statycznych.

Celem tego artykułu jest rozwiązanie problemu realizacji podstawowego elementu graficznego wchodzącego w skład bloku wyświetlania systemu wizualizacji algorytmów, przy użyciu techniki obiektowej. Problem doczekał się już wielu poważnych rozwiązań [1, 2, 3, 4], jednak brak w nich spojrzenia ściśle obiektowego, chociaż temat ten nie pozostaje niezauważony [5]. Dotychczas istniejące systemy wizualizacji nie podejmują się również w ogóle bądź podejmują się tylko w niewielkim stopniu wizualizacji dynamicznych struktur danych. Niniejszy artykuł wzbogacony jest przykładami z wizualizacji prostych operacji na strukturze listowej. Szerzej omówiona zostanie wymiana komunikatów pomiędzy obiektami, zachodząca podczas dołączania nowego elementu w środek listy.

Analiza obiektowa odwzorowuje zarówno dziedzinę problemu, jak i zakres usług systemu na obiektowo zorientowany model [6]. Modelujemy informację o problemie, czyli staramy się za pomocą obiektów i komunikatów zaprezentować określone elementy ze świata rzeczywistego. Uzyskany model pozwala na szczegółowe przeanalizowanie badanego problemu. Podejście obiektowe znajduje również zastosowanie w wizualizacji algorytmów, co zostanie przedstawione w dalszej części artykułu.

2. Zastosowanie techniki obiektowej w systemie wizualizacji

2.1. Struktura systemu wizualizacji algorytmów



Rys. 1. Ogólna struktura systemu wizualizacji
Fig. 1. General structure of visualization system

W omawianym systemie wizualizacji przedmiotem badań jest algorytm. Aby było możliwe przeanalizowanie jego działania, musi on zostać zapisany w konkretnym języku opisu, zgodnie z ustaloną gramatyką. System wizualizacji wykonuje tak zapisany algorytm, kontrolując przy tym przepływ sterowania w algorytmie oraz śledząc zmiany, jakich dokonuje on w strukturach danych.

Kolejną częścią systemu jest blok interpretacji przechwytywanych danych. W tej części wszystkie zaobserwowane skutki działania algorytmu zostają odpowiednio zidentyfikowane i przeanalizowane. Na podstawie rodzaju zmian wywołanych przez uruchomienie badanego algorytmu zostaje wygenerowana określona informacja dla kolejnej części systemu wizualizacji - maszyny projekcyjnej.

W maszynie projekcyjnej informacja wejściowa inicjuje wszystkie cząstkowe procesy wizualizacji. Najpierw następuje dobór odpowiednich operacji graficznych w celu zobrazowania poszczególnych zmian wykonanych przez algorytm. Następnie operacje te zostają wykonane z uwzględnieniem wcześniej zadeklarowanych przez użytkownika parametrów charakteryzujących koordynację wizualizacji. Do parametrów tych możemy zaliczyć: szybkość i płynność animacji czy też szczegółowość wizualizacji danych. Maszyna projekcyjna zajmuje się też tym, aby sposób ułożenia reprezentacji graficznych wizualizowanych danych odzwierciedlał obrazowaną strukturę danych.

Proces wizualizacji musi również uwzględniać interakcję użytkownika, który w czasie projekcji powinien mieć możliwość ingerencji w sposób prezentacji działania algorytmu.

2.2. Obiektowe podejście do zarządzania danymi w systemie wizualizacji

Jeśli program wymaga tylko niewielkiej ilości danych, to bez utraty bezpieczeństwa dane te mogą być udostępniane wszystkim podprogramom wchodzącym w jego skład. Taki sposób jest bardzo wygodny dla programistów, ponieważ dostarcza im współdzielony obszar danych, w którym różne podprogramy mogą wymieniać informacje, jeśli tylko tego potrzebują. Jeśli jednak obszar współdzielonych danych jest zbyt duży, a liczba komunikujących się poprzez niego części programu zbyt wielka, dochodzi wtedy często do nieprzewidywalnego zachowania się programu.

Rozwiązaniem problemu jest podział danych na moduły odpowiadające procedurom. Realizowane jest to przez przypisanie każdemu podprogramowi własnego, lokalnego miejsca na dane, które tylko przez niego mogą być modyfikowane. Taka metoda minimalizuje niepożądane interferencje pomiędzy podprogramami i pozwala na bardziej niezależne projektowanie. Jednakże w programowaniu modułowym trudno jest stworzyć aplikacje o poprawnej strukturze. Sztywny podział na moduły sprawia wiele problemów przy wielu próbach modyfikacji programu.

Bardziej spójnym stylem programowania jest programowanie strukturalne, polegające na dekompozycji funkcjonalnej, czyli rozbijaniu problemu na składniki i podskładniki, a następnie realizacji tych części przez niezależne zespoły programistów. Trudno jednak przewidzieć postać poszczególnych podzadań, zanim system zostanie rzeczywiście zaimplementowany. Podział pracy ustalony w fazie projektowania może prowadzić do

trudnego do poprawienia rozmieszczenia problemów w poszczególnych podzadaniach. A poprawki w systemie, który jest zaprojektowany z myślą o przyszłej rozbudowie, są nieuniknione.

Do tworzenia systemu wizualizacji, który ma być wciąż rozwijany, dobrze nadaje się więc programowanie zorientowane obiektowo. Podejście obiektowe do systemu wizualizacji niesie z sobą dodatkowy nacisk na zrozumienie problemu uniwersalnego projektowania. Owocuje hierarchią klas zawierającą systematyczną organizację atrybutów i usług. Klasy, oczywiście, muszą zostać tak zdefiniowane, aby posiadały uniwersalne zastosowanie w systemie wizualizacji i mogły być wielokrotnie używane. Jeśli jakaś klasa zostanie raz zdefiniowana, to można zagwarantować, że wszystkie obiekty tej klasy będą charakteryzowały się określonymi składnikami. Oczywiście, zadanie to nie jest proste. O ile natychmiastowe zastosowanie danej klasy do opisania wybranej części systemu wizualizacji jest oczywiste, to jednak późniejsze jej zastosowania mogą być trudne do przewidzenia. Dlatego też istotą problemu jest projektowanie klas na tyle kompletnych, aby dobrze opisywały wybrany fragment systemu wizualizacji. Otrzymujemy wtedy oprogramowanie niezawodne i przede wszystkim elastyczne, co czyni je podatnym na modyfikacje, dostosowując je do zmieniających się potrzeb rozwijającego się systemu wizualizacji.

3. Identyfikacja cech środowiska obiektowego

Celem tego rozdziału jest wskazanie i przedstawienie cech środowiska obiektowego występujących w warstwie wyświetlania, będącej integralną częścią bloku maszyny projekcyjnej systemu wizualizacji algorytmów. Zidentyfikowanie klas, obiektów, atrybutów, występujących relacji oraz przepływu komunikatów pozwala na ujęcie problemu w ramy systemu obiektowego. Operacja ta jest niezbędna do zgłębienia istotnych szczegółów problemu wizualizacji. Jest ona również podstawą do implementacji systemu wizaualizacji w obiektowo zorientowanym języku programowania.

3.1. Abstrakcja

Każdy z elementów, które chcemy obserwować, może mieć dużą liczbę aspektów. Z powodu ograniczeń przestrzeni na ekranie monitora, który jest dla nas sceną projekcji, nie jest możliwe przedstawienie wszystkich. Musimy zatem zignorować te aspekty, które z punktu widzenia aktualnego celu wizualizacji są nieistotne, a skoncentrować się na tych najbardziej interesujących, które będą reprezentowane przez graficzne reprezentacje obiektu: grele (GRaphic ELement). One właśnie muszą zostać pokazane na ekranie w relacji

z elementem, który opisują. Jednocześnie pociąga to za sobą ograniczenie liczby wizualizowanych właściwości elementu tylko do tych, które operują na wyodrębnionych atrybutach.

Grele skojarzone są nie tylko z elementami, ale również z niektórymi ich atrybutami, Najczęściej za ich pomocą pokazuje się na ekranie logiczne połączenia pomiędzy obiektami. Obrazowanie wzajemnych zależności pomiędzy poszczególnymi elementami dynamicznej struktury danych jest jednym z głównych celów, jakie stoją przed twórcami systemu wizualizacji.

3.2. Organizacja elementów graficznych

W celu uporządkowania greli, pozwalającego na ich późniejszą skuteczniejszą analizę, musimy dokonać ich klasyfikacji. Podziału możemy dokonać ze względu na istniejące podobieństwa między elementami graficznymi. Brane jest tu pod uwagę podobieństwo wyglądu, będące odzwierciedleniem fizycznych podobieństw obiektów (rys. 2). Wygląd grela w znacznym stopniu uzależniony jest od stopnia abstrakcji zastosowanego podczas obrazowania elementu rzeczywistego w postaci obiektu graficznego [7, 8].

Rys. 2. Graficzne podobieństwa obiektów
Fig. 2. Graphic likeness of objects

Można również dokonać podziału greli ze względu na podobieństwo ich zachowań na scenie projekcji algorytmu:

- statyczne i posiadające zdolność przemieszczania,
- interaktywne i nie reagujące na działanie użytkownika,
- zmieniające swój rozmiar i stałorozmiarowe.

3.3. Hermetyzacja

Każdy z greli ma pewną ilość atrybutów. Część z nich może być niedostępna dla innych greli, część zaś musi zostać udostępniona za pośrednictwem odpowiednich usług, jakie jeden grel może świadczyć innemu.

Do atrybutów chronionych należą te, które nie mają wpływu na inne obiekty ze sceny projekcji. Należą do nich przede wszystkim atrybuty określające sposób i rodzaj wyświetlanej, w obszarze grela, informacji.

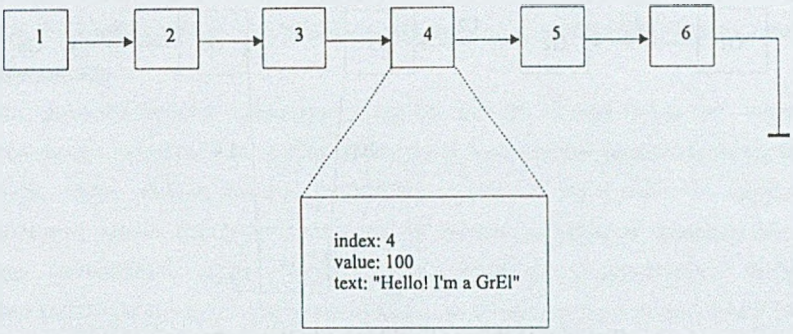
Pozostałe, dostępne dla innych obiektów, zawierają informacje na temat jego graficznych właściwości. Należą do nich:

- współrzędne charakteryzujące obszar zajęty przez grela;
- wygląd obiektu: prosta figura geometryczna (prostokąt, okrąg, trójkąt lub inne), bądź też bardziej złożony symbol;
- kolor grela;
- współrzędne punktów, do których mogą dochodzić linie symbolizujące logiczne połączenia między grelami;
- informacja o widoczności/ukrytości na scenie projekcji;
- informacja o aktualnym stanie obiektu (faza ruchu, faza nieruchoma);
- sposób reagowania na komunikaty;
- inne.

3.4. Dziedziczenie

Dziedziczenie obrazuje nam podobieństwo pomiędzy klasami różnych elementów graficznych. Dzięki niemu upraszczamy definiowanie nowych klas greli, bazując na już istniejących. Stosując dziedziczenie określamy atrybuty i usługi zawarte w klasie bazowej jako wspólne dla wszystkich klas pochodnych. Można również powiedzieć, że poprzez dodawanie własnych metod i atrybutów, klasy pochodne stają się klasami bardziej specjalizowanymi niż bazowe.

Dziedziczenie jest doskonale widoczne w sytuacji, gdy użytkownik dokonuje powiększenia rozmiaru jednego z greli, chcąc dokładniej przyjrzeć się elementowi przez niego reprezentowanemu (rys. 3).



Rys. 3. Przykład procesu dziedziczenia
Fig. 3. Sample inheritance process

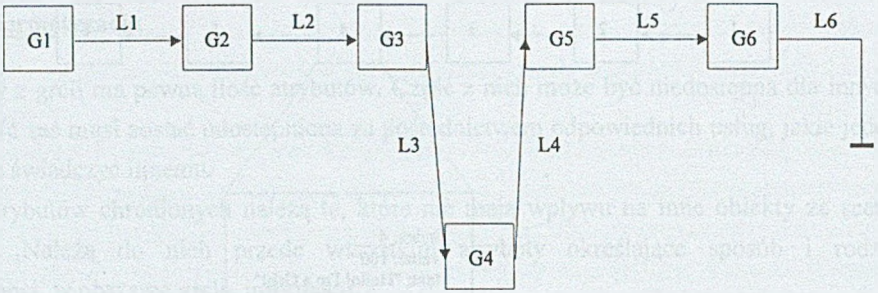
Tworzymy wtedy obiekt graficzny, należący do nowej klasy opierającej się na istniejącej już klasie obiektów, do której należy grel bazowy. Nowy grel wraz z atrybutami dziedziczy również wszystkie metody obiektu bazowego. Niektóre zapewne zostaną zmodyfikowane (metody realizujące mechanizm rysowania obiektu i wyświetlania informacji, którą on udostępnia), lecz większość może zostać w dalszym ciągu wykorzystywana bez konieczności zmian.

Powyższy sposób jest alternatywnym wyjściem w stosunku do rozwiązania polegającego na zastosowaniu obiektów należących tylko do jednej klasy, posiadających jednak możliwość płynnej lub skokowej zmiany swoich rozmiarów.

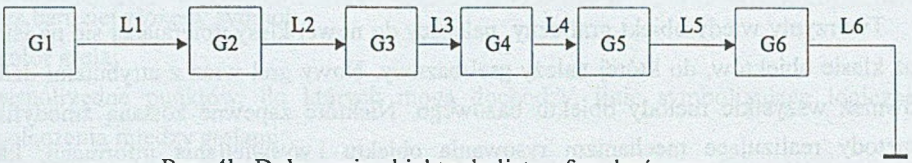
3.5. Komunikacja pomiędzy obiektami graficznymi

Komunikacja między obiektami jest podstawową ideą w modelu obiektowym. Jedyna droga do atrybutu obiektu prowadzi przez usługę. Grele komunikują się między sobą, świadcząc sobie nawzajem usługi polegające na udzielaniu o sobie pewnych informacji bądź też na wykonaniu odpowiedniej operacji na scenie projekcji.

Doskonałym przykładem jest sytuacja, gdy jeden z greli ma zostać umieszczony w jednej linii z pozostałymi należącymi do tej samej struktury listowej, a chcemy uniknąć odświeżania całej sceny projekcji i związanego z tym odrysowywania każdego z obiektów graficznych. Poza przykładową strukturą listową, na scenie projekcji mogą bowiem znajdować się jeszcze inne obiekty, o bardziej skomplikowanej graficznej reprezentacji. Wystarczy więc, że grel wyśle do sąsiadów komunikaty żądania rozsunięcia się. Sąsiedzi (jeśli przestrzeń na to pozwoli) przesuwają się w obie strony zmniejszając odległości między sobą (rys. 4a, 4b).



Rys. 4a. Dołączanie obiektu do listy - faza początkowa
Fig. 4a. Joinning an object to the list - an early stage



Rys. 4b. Dołączanie obiektu do listy - faza końcowa
Fig. 4b. Joinning an object to the list - a final stage

W powyższym, prostym przykładzie, niektóre z greli zmuszone są zmienić swoje położenie i rozmiar. Dotyczy to przede wszystkim graficznych reprezentacji logicznych połączeń pomiędzy wizualizowanymi elementami struktury danych.

Sposób postępowania:

1. Obiekt G4 żąda od obiektów G3 i G5 informacji o ich położeniu.
2. G4 ustala miejsce, w którym powinien się znaleźć.
3. G4 wysyła do G3 i G5 komunikaty z prośbą o rozsunięcie się.
4. G3 komunikuje się z L2 dowiadując się o jego długość, a G5 komunikuje się w tym samym celu z L5.
5. G3 i G5 odpowiadają na komunikat G4 z punktu 3. W przypadku gdy G3 lub G5 nie mają się gdzie przesunąć, wysyłają komunikat do swoich sąsiadów z prośbą o przesunięcie się bądź do systemu z prośbą o odświeżenie wyglądu całej istniejącej struktury.
6. G3 przesuwa się w stronę G2 wysyłając do L2 i L3 komunikaty z żądaniem odświeżenia ich wyglądu, z powodu zmiany swojego położenia.
7. G5 przesuwa się w stronę G6 wysyłając do L4 i L5 komunikaty z żądaniem odświeżenia ich wyglądu, z powodu zmiany swojego położenia.
8. G4 przemieszcza się na miejsce docelowe, informując obiekty L3 i L4 o zmianie swojego położenia.

Operacje 6, 7 i 8 mogą zostać wykonane sekwencyjnie lub równolegle. Jest to zależne od sposobu implementacji.

Być może prościej byłoby odświeżyć wygląd całego ekranu wraz ze wszystkimi obiektami, lecz dzięki powyższej metodzie, polegającej na płynnym przesuwaniu się greli po scenie projekcji, akcja zyskuje więcej dynamizmu i bardziej przemawia do użytkownika chcącego zrozumieć zasadę działania algorytmu. W realnie pracującym systemie nie da się jednak uniknąć odświeżania całości wyglądu sceny projekcji. „Samodzielne" ruchy greli mogłyby doprowadzić po pewnym czasie do sytuacji, w której pomimo intensywnej wymiany komunikatów, żaden z nich nie mógłby już zmienić swego położenia.

4. Implementacja

Przedmiotem implementacji był zespół klas związanych z organizacją systemu dla wizualizacji algorytmów operujących na dynamicznych strukturach danych. Zaimplementowano m.in. klasy opisujące graficzne reprezentacje poszczególnych elementów struktur dynamicznych [13]. Stworzony system pozwolił na wizualizację podstawowych operacji na strukturze listowej. Poniżej skoncentrowano się na omówieniu podstawowej klasy grela.

4.1. Język programowania

Jako narzędzie do implementacji wybrany został język Smalltalk [9, 10]. Smalltalk jako język i środowisko w pełni obiektowe doskonale nadaje się do tworzenia w pełni obiektowych aplikacji [11, 12].

Obiekt w Smalltalku jest elementem zawierającym informacje i opis sposobu manipulowania tą informacją. Posiada zmienne prywatne i odpowiednie metody, które operują na tych zmiennych. Innymi słowy: obiekt zawiera dane i wie co z nimi zrobić. Komunikat Smalltalka mówi obiektowi, jaką operację powinien wykonać. Komunikat zawiera nazwę operacji, ale faktycznie wykonywana czynność jest zamknięta w obiekcie, który otrzymuje komunikat. Komunikat sprowadza się do wyboru jednej z czynności, o której obiekt wie, jak ją wykonać.

Klasy w Smalltalku wyrażają statyczny opis danych i metod. Natomiast obiekty klas są egzemplarzami klas powstałymi w trakcie działania programu.

4.2. Obiekty graficzne

Bazowa klasa grela została zdefiniowana następująco:

Model subclass: #GrEl

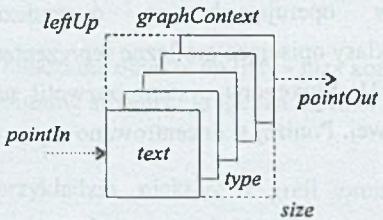
instanceVariableNames: 'leftUp size type pointIn pointOut graphContext text'

classVariableNames: ''

poolDictionaries: ''

category: 'Visualization'

Na rys. 5 przedstawiono podstawową strukturę grela wraz z jego charakterystycznymi atrybutami. Należy zwrócić uwagę na to, że grel bez względu na to, jaki jest jego typ (na rysunku widzimy obiekt *#middleGroup*), zawsze opisywany jest jako pole o kształcie prostokąta. Sposób ten znacznie upraszcza implementację wszystkich operacji związanych z przeliczaniem współrzędnych greli.



Rys. 5. Struktura obiektu graficznego
Fig. 5. A graphic object structure

Charakterystyka poszczególnych atrybutów klasy GrEl:

- *leftUp* - współrzędne lewego górnego narożnika prostokątnego obszaru, w którym znajduje się grel;
- *size* - rozmiar prostokątnego obszaru zawierającego obiekt graficzny, odległość pomiędzy prawym dolnym a lewym górnym narożnikiem tego obszaru;
- *type* - określa typ obiektu graficznego ze względu na jego wygląd graficzny, np.: *#smallSingle, #mediumSingle, #bigSingle, #smallGroup,* itd.;
- *pointIn* - współrzędne punktu, do którego dochodzą groty strzałek symbolizujących logiczne połączenia pomiędzy grelami;
- *pointOut* - współrzędne punktu, z którego wychodzą strzałki obrazujące istnienie logicznego połączenia z innym grelem;
- *graphContext* - atrybut kojarzący dany obiekt graficzny z określoną platformą wyświetlania;

• *text* - atrybut zawierający te informacje o obiekcie reprezentowanym przez grela, które mogą zostać wyświetlone w jego obszarze.

Zaimplementowano też podstawowe operacje na grelach [13]. Pozwalają one na wyświetlanie obiektów graficznych różnych typów oraz ich przemieszczanie się po obszarze projekcji.

5. Uwagi końcowe

W ramach opisanej powyżej pracy przeanalizowano słuszność zastosowania obiektowego punktu widzenia do rozwiązania problemu graficznej reprezentacji dynamicznych struktur danych oraz wykonywanych na nich operacji. Podstawą do prac była częściowa implementacja, będąca efektem przeprowadzonych wcześniej badań [13,14]. Natomiast wynikiem jest stworzenie teoretycznych założeń do dalszej implementacji systemu wizualizacji algorytmów w obiektowo zorientowanym języku programowania Smalltalk. Kontynuowane prace będą ukierunkowane na wizualizowanie operacji na dynamicznych strukturach danych o bardziej złożonym systemie wewnętrznych, logicznych powiązań niż istniejący w strukturze listowej.

Równolegle prowadzone są prace nad optymalizacją sposobu przechwytywania komunikatów na użytek systemu wizualizacji. Całość badań zmierza w kierunku realizacji spójnego, w pełni graficznego systemu inspekcji dynamicznych struktur danych, pozwalającego na wygodne i obrazowe śledzenie przepływu informacji oraz analizowanie działania algorytmów operujących na wyżej wspomnianych strukturach.

LITERATURA

1. Brown M.H., Sedgewick R.: A System for Algorithm Animation, Computer Graphics; SIGGRAPH'84 Conference Proceedings, Minneapolis 1984, s. 177-186.
2. Brown M.H.: Exploring Algorithms Using Balsa-II. IEEE Computer, 1988, t. 21, s. 14-36.
3. Brown M.H.: Zeus: A System for Algorithm Animation and Multi-View Editing. IEEE Computer Society Press, Kobe 1991, s. 4-9.
4. Stasko J.: Animating Algorithms with XTANGO, SIGTAC News, Spring 1992, t. 23, s. 67-71.
5. Naps T., Swander B.: An object-oriented approach to algorithm visualization - easy, extensible, and dynamic. SIGSCE, Phoenix 1994.

6. Coad. P., Yourdon E.: Analiza obiektowa. Oficyna Wydawnicza READ ME, Warszawa 1994.
7. Roman G.C., Cox K.C.: Abstraction in Algorithm Animation, Proceedings of 1992 IEEE Workshop on Visual Languages. Seattle 1992, s. 18-24.
8. Roman G.C., Cox K.C.: Program Visualization: The Art of Mapping Programs to Pictures. International Conference on Software Engineering, New York 1992, s. 412-420.
9. Goldberg A., Robson D.: Smalltalk-80. The language and implementation. Addison Wesley 1983.
10. Czarnożyński A.: Wprowadzenie do języka Smalltalk. Politechnika Śląska, Skrypt uczelniany nr 1559, Gliwice 1994.
11. Coad. P., Nicola J.: Programowanie obiektowe. Oficyna Wydawnicza READ ME, Warszawa 1993.
12. Visual Works: object reference. ParcPlace Systems Inc., Palo Alto 1994.
13. Szmal P., Dobosz K., Francik J., Migas A., Gonera P.: Metody wizualizacji sterowanej danymi w środowisku Smalltalk-80. Raport końcowy z badań statutowych (BK 228/RAu2/95), Gliwice 1995.
14. Szmal P., Dobosz K., Francik J., Migas A., Gonera P.: Wizualizacja algorytmów i procesów. Raport końcowy z badań statutowych (BK 228/RAu2/96), Gliwice 1996.

Recenzent: Doc. dr hab. inż. Adam Mrózek

Wpłynęło do Redakcji 30 kwietnia 1997 r.

Abstract

The paper presents an object approach to the display part of the algorithm visualization system. Its major part is identification of the qualities of object environments found in the display part of the visualisation system, described in chapter 3. Figure 2 presents one of the possible kinds of classification of graphic objects being part of the display part. Figure 3 shows the application of the inheritance mechanism for changing the size of a graphic object. Exchange of information between the elements of a sample line structure is presented in figures 4a and 4b. Then in chapter 4, Smalltalk is proposed as one of the possible tools for implementation of the graphic elements of the display in the visualization system. In figure 5,

the basic structure of a graphic element is shown, and then its most important properties are discussed.