STUDIA INFORMATICA 2008
Volume 29 Number 2A (76)

Przemysław KUDŁACIK, Tomasz WESOŁOWSKI
Politechnika Śląska, Instytut Informatyki

TRWAŁE PRZECHWYTYWANIE ELEMENTÓW GRAFICZNEGO INTERFEJSU UŻYTKOWNIKA

Streszczenie. Artykuł przedstawia mechanizm pobierania danych wskazanych elementów graficznego interfejsu użytkownika oraz metodę ponownego ich zlokalizowania bez względu na instancję aplikacji. Mechanizm nazwano trwałym przechwytywaniem elementów GUI[1]. Podczas realizacji założono wykorzystanie jedynie funkcji API[2] systemu operacyjnego Windows. Działanie przetestowano na interfejsach graficznych aplikacji systemu Windows oraz platform .NET i Java.

Słowa kluczowe: GUI, Windows, interfejs, okno, haki

PERMANENT CAPTURE OF GRAPHICAL USER INTERFACE ELEMENTS

Summary. Paper presents a mechanism of collecting data of graphical user interface elements pointed by user as well as method of searching based on collected data. The mechanism was called a permanent capture of GUI elements because it allows to find pointed element independently on application's instance. Task implementation assumed usage of API functions available in Windows operating system without additional, external libraries. Solution was tested with Windows applications as well as .NET and Java platforms.

Keywords: GUI, Windows, interface, window, hooks

1. Wprowadzenie

Elementy składające się na graficzny interfejs użytkownika (GUI) każdej aplikacji najczęściej mają swe bezpośrednie odpowiedniki w systemie operacyjnym. Wchodzą one w

[1] ang. *Graphical User Interface*
[2] ang. *Application Programming Interface*

skład pewnej struktury systemu operacyjnego zaprojektowanej do tworzenia i zarządzania GUI. Aplikacje użytkowe korzystają właśnie z tych funkcji systemowych do utworzenia i zarządzania pracą własnych interfejsów. Natomiast system operacyjny zapewnia aplikacjom odpowiednie przekazywanie danych z urządzeń wejściowych, takich jak klawiatura czy urządzenie wskazujące (np. popularna mysz).

Znając jednak zasady pracy i strukturę wspomnianej części systemu operacyjnego, możliwe jest komunikowanie się z GUI aplikacji bezpośrednio wykorzystując dostępne funkcje systemowe z pominięciem wprowadzania danych z urządzeń wskazujących lub klawiatury.

W systemach operacyjnych MS Windows, które były przedmiotem badań autorów, każdy element interfejsu graficznego wywodzi się z klasy okna i posiada swój unikalny uchwyt (pewien identyfikator, numer). Uchwyt pozwala na komunikację ze wskazywanym przez niego oknem (elementem GUI) przez wysyłanie komunikatów.

Uchwyty nie są jednak trwałe. Są one przyporządkowane dynamicznie podczas tworzenia interfejsu aplikacji i tracą ważność, gdy aplikacja wraz ze swym interfejsem zakończy pracę. Dlatego opracowanie mechanizmu pozwalającego na pobranie aktualnego uchwytu okna bez względu na instancję aplikacji jest zadaniem niebanalnym. W dalszej części artykułu zostaną przedstawione metody realizacji tak scharakteryzowanego zadania.

2. Mechanizm trwałego przechwytywania okien

Przechwycenie wybranego okna, elementu graficznego interfejsu aplikacji – to pobranie jego uchwytu. Pozwala to na przejęcie pewnej kontroli nad oknem przez możliwość niezależnego przesyłania komunikatów (wprowadzania danych). Natomiast w połączeniu z mechanizmem haków systemowych, opisanych w podpunkcie 2.1, pozwala na przeglądanie komunikatów skierowanych do aplikacji, a nawet ich anulowanie, co daje praktycznie pełną kontrolę pracy okien.

Po zakończeniu działania aplikacji uchwyty przechwyconych okien stają się nieważne, a ponowne uruchomienie programu powoduje wygenerowanie nowych uchwytów, ponieważ obiekty okien tworzone są od nowa. Dlatego trwałym przechwyceniem nazwano mechanizm, pozwalający na uzyskanie aktualnego uchwytu wybranego okna bez względu na instancję aplikacji.

Na proces trwałego przechwycenia okna składają się trzy etapy. Pierwszym jest zwykłe przechwycenie. Użytkownik wskazuje wybrane okno w dowolnej aplikacji i następuje pobranie jego uchwytu. Kolejnym etapem jest wykorzystanie uzyskanego uchwytu do pobrania pewnych niezmiennych informacji, na podstawie których okno będzie mogło zostać zlokali-

zowane niezależnie od instancji aplikacji. Ostatni etap – to metoda wyszukania trwale przechwyconego okna wykonywana zawsze w celu pobrania jego aktualnego uchwytu.

2.1. Przechwytywanie okien dowolnej aplikacji systemowej

Wyboru przechwytywanego okna dokonuje użytkownik, zaznaczając je za pomocą urządzenia wskazującego lub klawiatury. Jednak pobranie uchwytu tak wskazanego okna dowolnej aplikacji jest zadaniem skomplikowanym, ponieważ komunikaty wygenerowane przez urządzenie wskazujące (np. mysz) kierowane są przez system operacyjny wyłącznie do tej aplikacji, do której wybrany przez użytkownika element GUI należy. Innymi słowy, każda aplikacja otrzymuje tylko te komunikaty, które są do niej zaadresowane. Dlatego proces chcący dokonać przechwycenia wskazanego okna nie otrzyma informacji o jego wskazaniu.

Problem można rozwiązać stosując mechanizm haków systemowych [1]. Mechanizm ten umożliwia zdefiniowanej przez programistę funkcji filtrującej przetwarzanie komunikatów przeznaczonych dla wybranej aplikacji lub wszystkich aplikacji aktualnie pracujących w systemie. Na potrzeby opisanego problemu należy użyć haka działającego w zakresie systemowym, ponieważ funkcja filtrująca musi otrzymywać komunikaty przeznaczone dla wszystkich aplikacji. Najważniejszym elementem konfiguracji mechanizmu jest dobór rodzaju haka, ponieważ każdy różni się grupą przeglądanych komunikatów oraz zakresem działania.

W rozwiązaniu zadania użyto haka WH_MOUSE stworzonego dla komunikatów pochodzących z urządzenia wskazującego oraz posiadającego możliwość anulowania komunikatu, blokując jego przesłanie do aplikacji. Systemowy zakres pracy wybranego haka wymaga umieszczenia funkcji filtrującej w dynamicznie dołączanej bibliotece DLL. Pamięć danych dzielonych pomiędzy biblioteką a aplikacją przechwytującą okno zrealizowano wykorzystując mechanizm MMF[3] [1].

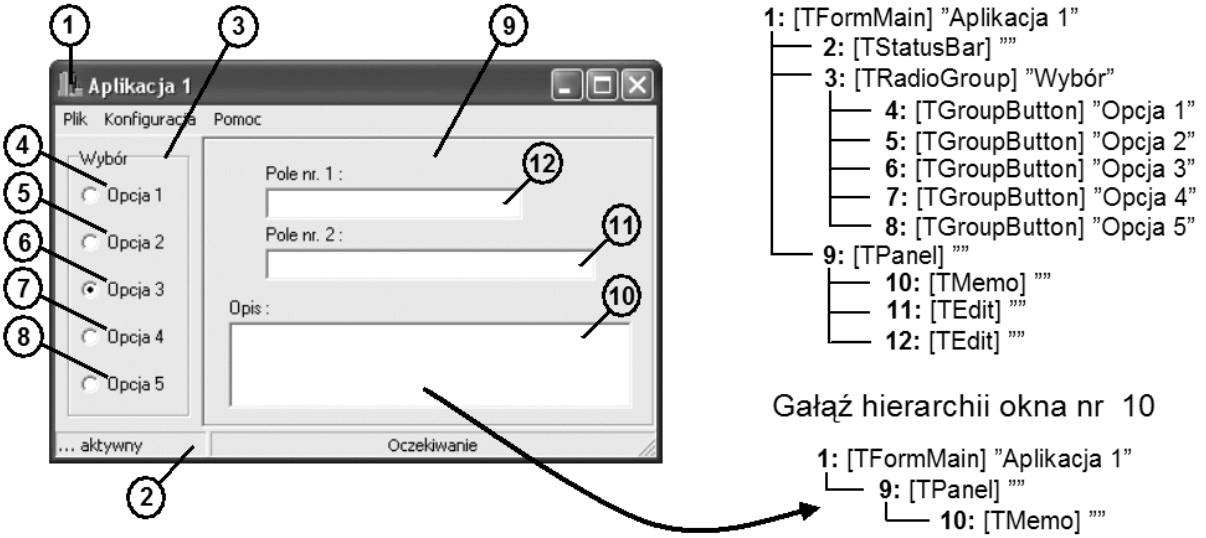
Do założenia i zakończenia pracy haka systemowego służą funkcje SetWindowsHookEx oraz UnhookWindowsHookEx. Należy również pamiętać o wywołaniu funkcji CallNextHookEx na końcu procedury filtrującej, aby umożliwić pracę kilku hakom tego samego typu założonym w tym samym czasie. Dokładny opis wspomnianych funkcji znajduje się w dokumentacji [1] oraz pracy [2].

2.2. Pobieranie informacji o oknach

Drugim etapem procesu trwałego przechwycenia jest zebranie informacji, które pozwolą zlokalizować wybrane okno i pobrać jego aktualny uchwyt systemowy. System operacyjny Windows udostępnia funkcje, pozwalające pobrać dane okna na podstawie jego uchwytu.

[3] ang. *Memory Mapped File*

Oczywiście, do tego celu zostanie użyty uchwyt przechwycony metodą opisaną w poprzednim podpunkcie. Należy jednak zastanowić się, jakie dane będą wystarczające do odnalezienia okna. W tym celu warto przyjrzeć się hierarchicznej strukturze okien przykładowej aplikacji.



Rys. 1. Hierarchia okien w aplikacji
Fig. 1. Hierarchy of windows in application

Rysunek 1 pokazuje, jak cenną informacją jest położenie okna w hierarchii okien aplikacji. Okna kontrolne, najbardziej interesujące od strony wprowadzania danych, są dziećmi (wchodzą w obszar) okien grupujących położonych wyżej. Pierwszym oknem w hierarchii jest główne okno aplikacji, będące bezpośrednim potomkiem pulpitu.

Drzewo hierarchii w prawej części rys. 1 zawiera nazwy klas okien oraz nazwy okien każdego elementu GUI. Nazwy klas są w ogromnej większości stałe i nie zależą od instancji aplikacji. Natomiast nazwy okien, w przypadku wielu okien kontrolnych i nie tylko, są zmienne. Przykładowo, nazwa okna edycyjnego klasy TEdit jest tożsama z zawartością wprowadzoną przez użytkownika. Z drugiej strony, nazwy głównych okien są najczęściej stałe lub zmienna jest najwyżej część nazwy.

Inne dane okna mogące posłużyć w jego lokalizacji to rozmiary oraz położenie względem rodzica. Oczywiście, te informacje mają dużo większe znaczenie w przypadku okien kontrolnych położonych najniżej w hierarchii, ponieważ położenie i rozmiary głównych okien aplikacji z reguły można dowolnie zmieniać.

W rozwiązaniu użyto funkcji GetClassName i GetWindowText do pobrania nazw klas okien i nazw okien oraz GetWindowPlacement do pobrania położenia i rozmiarów. Dane te zostały pobrane dla każdego elementu gałęzi hierarchii przechwyconego okna. Poruszanie się w drzewie w kierunku głównego okna aplikacji umożliwia funkcja GetParent. Dokładny opis wspomnianych funkcji znajduje się w dokumentacji [1].

Wynikiem tego etapu są dane okien reprezentujących całą gałąź hierarchii okna przechwytywanego. Przykład gałęzi wybranego okna pokazano również na rys. 1.

2.3. Wyszukanie okna w systemie operacyjnym

Rozwiązanie opisane do tej pory byłoby niepełne bez opracowania metody wyszukania trwale przechwyconego okna. Dlatego w tym podpunkcie zostanie przedstawiony algorytm wyszukiwania okien i pobierania ich aktualnych uchwytów.

Po pierwsze, potrzebny jest mechanizm umożliwiający przeglądanie okien aktualnie istniejących w systemie. Realizują to funkcje EnumWindows oraz EnumChildWindows [1]. Pierwsza z nich pozwala na przejrzenie uchwytów kolejnych głównych okien aplikacji, czyli bezpośrednich potomków pulpitu. Natomiast druga, zgodnie ze swą nazwą, pozwala przejrzeć wszystkich potomków określonego okna. Wykonanie każdej funkcji powoduje uruchomienie pętli, w której wywoływana jest funkcja przetwarzająca przygotowana przez programistę. Otrzymuje ona kolejne uchwyty okien aż do wyczerpania dostępnej puli. Proces może zostać przerwany wcześniej, gdy funkcja przetwarzająca zwróci wartość różną od zera.

Użycie mechanizmu dostarczonego przez funkcje EnumWindows oraz EnumChildWindows pozwala na przeprowadzenie wyszukania przechwyconego okna, wykorzystując dane o całej jego hierarchii w zapamiętanej gałęzi[4]. Proces rozpoczyna wyszukanie okna głównego aplikacji funkcją EnumWindows, a następnie porusza się w dół przeszukując kolejne poziomy okien dzieci wybierając właściwe okno na danym poziomie. Proces jest kończony odnalezieniem i zapisaniem uchwytu okna docelowego, znajdującego się na ostatnim miejscu w zapamiętanej gałęzi. Przebieg przedstawia poniższy algorytm.

```
1:pobierz dane głównego okna aplikacji z zapamiętanej gałęzi
2:wyszukaj i zapamiętaj główne okno aplikacji (funkcja EnumWindows)
3:jeśli główne okno znalezione, to
początek
  4:wykonuj dla wszystkich okien w zapamiętanej gałęzi
  początek
    5:pobierz dane kolejnego okna z zapamiętanej gałęzi
    6:wyszukaj kolejne okno z gałęzi (funkcja EnumChildWindows)
    7:jeśli nie znaleziono okna, to przerwij
  koniec
koniec
```

Algorytm przerywa swe działanie w wypadku nieznalezienia okna na dowolnym etapie lokalnego wyszukiwania w gałęzi i cały proces kończy się niepowodzeniem.

Najważniejszym elementem etapu wyszukiwania jest zdefiniowana przez programistę funkcja przetwarzająca kolejne uchwyty, wywoływana dla każdego okna przez EnumWindows i EnumChildWindows. W funkcji zaimplementowano porównanie danych przeglądanych okien z danymi okien podawanych z wyszukiwanej gałęzi. Wynikiem każdego porów-

[4] kryteria wyszukiwania scharakteryzowano w opisie funkcji przetwarzającej

nania jest ocena zgodności okien. Gdy wszystkie okna na danym poziomie zostaną przetworzone (funkcja EnumWindows lub EnumChildWindows zakończy działanie), wybierane jest okno o najwyższej ocenie, czyli największej zgodności. Dokładny algorytm funkcji przetwarzającej przedstawiono poniżej. Wszystkie porównania dotyczą okien przeglądanych i wyszukiwanych.

```
1:pobierz dane aktualnie przeglądanego okna
 (nazwa klasy, nazwa, rozmiary, położenie względem rodzica)
2:porównaj pierwszą połowę nazwy klasy
3:jeśli połowa nazwy klasy zgodna, to
początek
  4:porównaj całą nazwę klasy
  5:jeśli nazwa klasy zgodna, to zwiększ ocenę o 5
  6:porównaj nazwę okna
  7:jeśli nazwa okna zgodna, to zwiększ ocenę o 5
  8:porównaj położenie okna względem rodzica
  9:jeśli położenie zgodne, to zwiększ ocenę o 2
  10:porównaj rozmiary okna względem rodzica
  11:jeśli rozmiary zgodne, to zwiększ ocenę o 2
koniec
```

Założono, iż minimalnym progiem podobieństwa jest zgodność przynajmniej pewnej części nazwy klasy porównywanych okien. W tym wypadku jest to pierwsza połowa nazwy klasy. Założenie to jest umotywowane zmiennością przyrostków nazw klas w przypadku niektórych okien (np. okien w aplikacjach MDI). W tym wypadku założenie zgodności całej nazwy klasy byłoby niewystarczające i uniemożliwiłoby proces wyszukiwania we wspomnianych sytuacjach.

W następnym etapie algorytm przypisuje różne wagi każdej kolejno występującej zgodności częściowej. Sumarycznie zwiększają one ocenę, a tym samym końcowy wynik porównania. Za najważniejsze uznano nazwę okna i jego klasy. Oczywiście, jak wspomniano w podpunkcie 2.2, te wartości mają większe znaczenie w przypadku głównych okien aplikacji. Na niższych poziomach rozstrzygające będą wymiary oraz położenie w przypadku wystąpienia większej ilości podobnych okien kontrolnych (jak np. okna edycyjne).

Można oczywiście modyfikować algorytm, aby po znalezieniu głównego okna zmienił zasady, oceny lub parametry w procesie porównania w zależności od potrzeb, czy też zastosowania. Jednak przeprowadzone testy wykazały, iż algorytm w przedstawionej wersji wystarczająco dobrze spełnia swoje zadanie w różnych sytuacjach.

3. Wnioski

W testach przeprowadzonych na różnych aplikacjach stworzonych dla systemu Windows zaobserwowano prawidłowe identyfikowanie okien. Wszędzie tam, gdzie z poziomu systemu można przejrzeć hierarchię elementów GUI aplikacji, opisane podejście pozwala na bardzo sku-

teczne wyszukiwanie trwale przechwyconych komponentów. Dotyczy to nie tylko aplikacji działających bezpośrednio w systemie, ale również tych pracujących na platformie .Net oraz Java.

Ograniczenia rozwiązania objawiają się jedynie w przypadku aplikacjinie korzystających z elementów GUI systemu operacyjnego. Najlepszym przykładem są tu aplikacje platformy Java, wykorzystujące graficzne komponenty biblioteki Swing[5]. W tym wypadku od strony systemu operacyjnego widziane są jedynie główne okna aplikacji i okna dialogowe, a hierarchia komponentów GUI jest zamknięta wewnątrz struktury środowiska Javy. Dlatego trwałe przechwycenie zapisze dane jedynie okna głównego. Oczywiście, wraz z zapamiętaniem współrzędnych miejsca wskazania elementu przez użytkownika pozwoli to na późniejsze przeprowadzenie komunikacji z aplikacją. Jednak rozwiązanie to będzie wrażliwe na zmianę rozmiarów okna głównego i współrzędne wskazania mogą się stać nieaktualne. Częściowym złagodzeniem efektu w niektórych zastosowaniach może być przeskalowanie miejsca wskazania, zgodnie z rozmiarami okna głównego.

Biorąc pod uwagę fakt, iż ogromna większość aplikacji tworzonych dla systemów Windows opiera swe interfejsy na dostępnych komponentach systemowych, rozwiązanie może znaleźć szerokie zastosowanie, począwszy od aplikacji śledzących stan pracy użytkownika, przez aplikacje testujące oprogramowanie na automatycznym wprowadzaniu danych kończąc.

W przyszłości planowane jest rozbudowanie przedstawionego mechanizmu o elementy rozpoznawania obrazów, co zapewne zmniejszyłoby scharakteryzowane ograniczenie i zwiększyło precyzję wyszukiwania komponentów graficznych w rzadziej spotykanych podejściach.

LITERATURA

1. Microsoft® Corporation: Microsoft® Win32® Programmer's Reference, 1996.
2. Kudłacik P.: Aplikacja wspomagająca pracę laboratorium analitycznego w zakresie raportowania i komunikacji z urządzeniem laboratoryjnym. Praca dyplomowa magisterska (prowadzący D. R. Augustyn), Politechnika Śląska, Gliwice 2004.

Recenzent: Dr inż. Maciej J. Bargielski

Wpłynęło do Redakcji 1 grudnia 2007 r.

[5] Przy wykorzystaniu komponentów biblioteki AWT problem nie występuje

Abstract

In most cases GUI components of every application have direct equivalents in operating system. They are a part of operating system's structure designed to create and manage GUI. Programs use this particular functionality to build and manage work of their graphical interfaces. Moreover, operating system provides a proper data transfer from input devices such keyboard or popular mouse.

Nevertheless, knowing the rules of work and structure of mentioned operating system's part, it is possible to communicate with application's GUI directly using available system functions which omits input devices.

In Windows operating systems, which were a subject o authors' interests, every element of graphical user interface is qualified as a window. Moreover, every window owns a unique handle (identification, number) which allows to communicate with pointed GUI element. Of course handles are not permanent. They are assigned dynamically during interface creation and loose their validity when application is destroyed.

Authors developed a mechanism of collecting data of GUI elements indicated by user and a method of search based on gathered information. The mechanism was called a permanent capture of GUI elements because it allows to find chosen element independently on application's instance.

Task was divided into three subsequent parts. Method of acquiring a handle of a window indicated by user was developed first. System's functionalities as hooks and memory mapped files were used in solution. Next part concerned process of collecting information needed to find window. Class name, window name, dimensions and position were picked as sufficient. The last part included development of search process using information gathered previously to get an actual handle of chosen window.

Developed method of permanent capture of GUI elements can be applied in many solutions. Worker's monitoring system, programs testing other applications or automatic data input could be presented as an example.

Adresy

Przemysław KUDŁACIK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, przemyslaw.kudlacik@polsl.pl

Tomasz WESOŁOWSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, tomasz.wesolowski@polsl.pl