STUDIA INFORMATICA 2001
Volume 22 Number 2 (44)

Stanisław WIDEŁ, Halina RYCHLICKA
Politechnika Śląska, Instytut Informatyki

WYBRANE PROBLEMY ARCHITEKTURY SERWERÓW FONICZNYCH

Streszczenie. W artykule autorzy dyskutują nad wybranymi zagadnieniami architektury serwerów fonicznych. Do założeń konstrukcyjnych należy zaliczyć: sposób implementacji fonicznego interfejsu z użytkownikiem oraz jego opis, metodę realizacji aplikacji wielowątkowej, budowę biblioteki oprogramowania odpowiedzialnej za komunikację ze specjalizowanym sprzętem.

VOICE RESPONSE SYSTEM DESIGN

Summary. In this paper we discussed same implementation assumption for human-machine interaction provided by voice response systems. Main architectural goal is to break the application to parts. One parts is a low-level programming and resource management. Another parts is a voice service is viewed as a sequence of interaction dialogs between a user and an implementation platform.

1. Wstęp

Mowa jako środek komunikacji może być również wykorzystywana do komunikacji ludzi z systemami komputerowymi.

Wykorzystując interfejs foniczny użytkownik może pracować na końcówce multimedialnej – wtedy będzie używał lokalnych interfejsów i zasobów swojej końcówki. Możliwa jest również sytuacja, w której użytkownik komunikuje się za pomocą mowy z odległymi systemami komputerowymi, niekoniecznie wykorzystując komputer, a na przykład jedynie telefon - wówczas możemy mówić o pracy z serwerami fonicznymi.

Przez serwery foniczne będziemy rozumieli rozwiązania sprzętowo-programowe, w których:

- komunikacja człowieka z systemami komputerowymi odbywa się za pomocą interfejsu fonicznego,
- istnieje możliwość obsługi wielu użytkowników w tym samym czasie,
- użytkownik jest połączony z serwerem poprzez sieć telefoniczną bądź sieć komputerową.

Serwery foniczne mogą być oddzielnymi stanowiskami, bądź też mogą stanowić część większego systemu. Dla przykładu: serwer foniczny realizujący interakcyjny dostęp do bazy danych poprzez sieć telefoniczną będzie częścią systemu realizującą interfejs pomiędzy użytkownikiem a bazą danych. Innym przykładem wykorzystywania serwerów fonicznych jako jednego z elementów większego systemu są centra obsługi klienta (tzw. Call Center). Również środowiska integrujące w sobie wszystkie usługi przesyłania wiadomości (ang. *unified messages*) muszą zawierać jako element swojego podsystemu serwer foniczny.

Konwergencja sieci komputerowych powodująca coraz większą integrację usług pozwala na wykorzystanie i dostęp poprzez sieci komputerowe rozwiązań serwerów fonicznych opartych na sprzęcie obsługującym jedynie linie sieci telefonicznej. Dzięki opracowaniu w firmie INTEL rodziny protokołów i upowszechnieniu interfejsu fonicznego w licznych implementacjach dla różnych systemów operacyjnych (dla przykładu w systemie Windows w programie NetMeeting) możliwe jest połączenie telekonferencyjne między użytkownikami sieci komputerowej. Zastosowanie bram H.323 umożliwia transport kodów DTMF i głosu pomiędzy sieciami komputerowymi a sieciami telefonicznymi. Połączenie foniczne może zostać nawiązane z sieci telefonicznej do sieci komputerowej, jak również odwrotnie.

Oprogramowanie realizujące funkcje narzucone przez zastosowanie serwera fonicznego będziemy nazywali aplikacją. Możemy wyróżnić odpowiednie klasy lub typy zastosowań serwerów fonicznych:

- audiotekst,
- automatyczny recepcjonista,
- system rejestracji rozmów,
- poczta głosowa [1],
- interakcyjna odpowiedź głosowa.

Do implementacji serwerów fonicznych konieczne jest rozwiązanie:

- obsługi wielu użytkowników w czasie rzeczywistym, spełniające wymagania interfejsu fonicznego,
- sposobu opisu aplikacji z danej klasy zastosowań oraz możliwość modyfikacji tego opisu,
- sposobu budowy i testowania interfejsu fonicznego.

2. Metody opisu aplikacji

Ze względu na charakter interfejsu z użytkownikiem tworzenie aplikacji dla serwerów fonicznych jest procesem złożonym. Dla aplikacji tego typu należy zaprojektować i przetestować foniczny interfejs z użytkownikiem. W typowym systemie komputerowym programista projektuje moduł komunikacji z użytkownikiem poprzez interfejs alfanumeryczny bądź graficzny (ang. GUI *Graphic User Interface*). W tym celu wykorzystuje dobrze znane i opracowane standardy. Tworzenie i testowanie fonicznego interfejsu z użytkownikiem, wymaga zastosowania innych narzędzi w stosunku do środowiska tworzenia programu komputerowego [10]. W czasie tworzenia programu komputerowego informacja zawarta w komunikatach głosowych nie jest bezpośrednio dostępna dla programisty. Tworzenie aplikacji dla serwera fonicznego jest zadaniem stosunkowo złożonym, gdyż w procesie testowania wymaga wielu iteracji i zmian pomiędzy środowiskiem multimedialnym dla testowania informacji słownych a środowiskiem alfanumerycznym dla testowania oprogramowania.

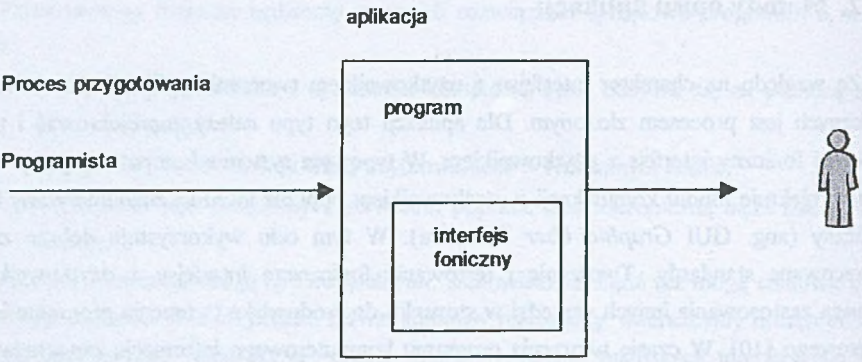
2.1. Opis fonicznego interfejsu z użytkownikiem zaszyty w kodzie programu

Pisząc program w typowym języku programowania nie można określić, jak będzie wyglądał interfejs foniczny pomiędzy serwerem a użytkownikiem bez uruchomienia tej aplikacji i wypełnienia jej odpowiednimi komunikatami. Można zatem powiedzieć, że mamy do czynienia z dwoma procesami: pierwszym – związanym z pisaniem oprogramowania i tworzeniem odpowiednich algorytmów dla aplikacji fonicznej, drugim – z przygotowaniem, testowaniem i uruchomieniem interfejsu fonicznego dla użytkownika systemu.

Na rysunku 1 przedstawiono, zdaniem autorów, najgorszy możliwy model tworzenia aplikacji dla serwera fonicznego, w którym w kodzie programu zawarta jest również definicja interfejsu fonicznego.

Rozwiązaniem docelowym, jakie należy przyjąć przy tworzeniu tych aplikacji dla serwerów fonicznych, jest wprowadzenie podziału na odpowiednie moduły funkcjonalne. Zakładamy, że definicja interfejsu fonicznego w danej klasie aplikacji będzie zewnętrzna w stosunku do kodu programu. Tworzeniem programu będzie się zajmować programista, natomiast tworzeniem interfejsu fonicznego – administrator systemu wyposażony w odpowiednie środki do nagrywania i edycji komunikatów fonicznych. W ten sposób rodzi się naturalny rozdział na specjalizację w czasie tworzenia aplikacji.

Sposób opisu interfejsu fonicznego jest podstawowym zagadnieniem, wpływającym na architekturę systemu.



Rys.1. Przykład architektury serwera, w którym foniczny interfejs z użytkownikiem jest zaszyty w kodzie programu
Fig. 1. Voice prompt in source code design

Opis interfejsu fonicznego jest to struktura danych zawierająca informacje sterujące oraz zapis dźwiękowy. Najprostszą formą takiego opisu jest tekst w postaci zapisu alfanumerycznego. Aby zamienić tekst alfanumeryczny na postać foniczną, należy użyć konwertera tekstu na mowę. Z punktu widzenia jakości dźwięku systemy syntezujące dźwięk bezpośrednio z tekstu ustępują systemom, które wykorzystują nagrania składające się z ciągłych wypowiedzi [2]. Wynika to również z faktu, że poza treścią alfanumeryczną interfejs foniczny przekazuje informacje poprzez intonację, szybkość wypowiedzi, dobór głosu lektora itp.

Opis interfejsu fonicznego poza tekstem powinien również zawierać możliwość składania wypowiedzi z predefiniowanych, nagranych wcześniej, komunikatów.

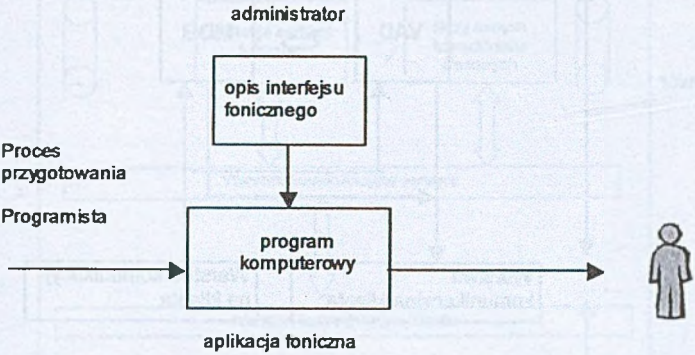
2.2. Zewnętrzny statyczny opis interfejsu fonicznego

Opisanie interfejsu fonicznego będzie stanowiło strukturę danych w postaci języka opisu, pliku konfiguracyjnego dla aplikacji danego typu.

Implementacja polega na zdefiniowaniu wykonywalnego programu, który będzie sterowany poprzez zewnętrzną strukturę danych, definiującą interakcję z użytkownikiem za pomocą interfejsu fonicznego.

Autorzy implementując tego rodzaju system pierwotnie oparli się na pliku konfiguracyjnym implementującym strukturę grafową opisaną w pracy [3]. W strukturze tej zdefiniowano stany, warunki tranzycji, tranzycje oraz procedury wykonywane w danym stanie.

Po dokonanej implementacji okazało się, że opis tego rodzaju stanowi rozwiązanie wystarczające. Zmiana typu aplikacji wiązałaby się z definiowaniem odpowiednich nowych stanów, procedur i tranzycji.



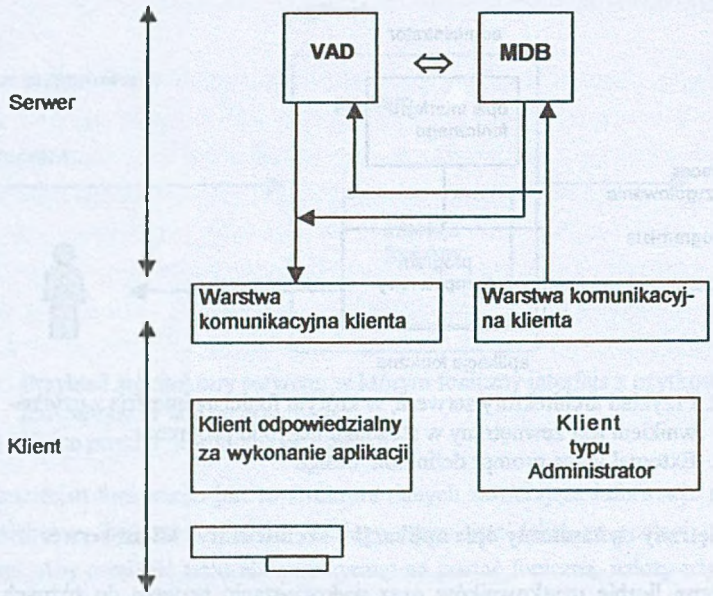
Rys. 2. Przykład architektury serwera, w którym foniczny interfejs z użytkownikiem jest zewnętrzny w stosunku do kodu programu
Fig. 2. External voice prompt definition design

2.3. Zewnętrzny dynamiczny opis aplikacji o architekturze klient-serwer

Przy większej liczbie użytkowników oraz wykorzystaniu serwera do różnych aplikacji może się okazać, że zmiany opisu intrefejsu fonicznego muszą być dokonane dynamicznie. W tym przypadku aplikacja musi sama zmodyfikować opis interfejsu fonicznego. Sytuacja taka występuje, gdy użytkownik bez wyłączania aplikacji chce dokonać zmian w systemie (np. poprzez interfejs foniczny). W takim wypadku musimy mieć dostęp i możliwość zmodyfikowania opisu aplikacji. Prowadzi to do podziału pomiędzy system typu *runtime*, czyli odpowiedzialny za bezpośrednie wykonywanie aplikacji na serwerze fonicznym, oraz system typu administrator, który umożliwia zmianę opisu aplikacji. Możemy powiedzieć, że w tym przypadku mamy do czynienia z architekturą typu klient-serwer, gdzie na serwerze jest przechowywany opis aplikacji fonicznej.

W aplikacji klient-serwer przedstawionej na rys. 3a uwzględniono dwóch klientów: jeden jest bezpośrednio odpowiedzialny za wykonanie aplikacji na kanałach fonicznych, drugi – to klient typu administrator odpowiedzialny za dokonywanie zmian i testowanie opisu aplikacji fonicznej.

Istotne jest również to, że aplikacja foniczna może być zmienna ze względu na zdarzenia lub warunki zewnętrzne (np. zdarzenia wywołane przez zegar w związku ze zmianą daty, godziny lub dnia tygodnia). Dla przykładu, system informacyjny urzędu miasta może zmieniać algorytm, w zależności od tego, czy telefonujemy w godzinach pracy czy po godzinach urzędowania.

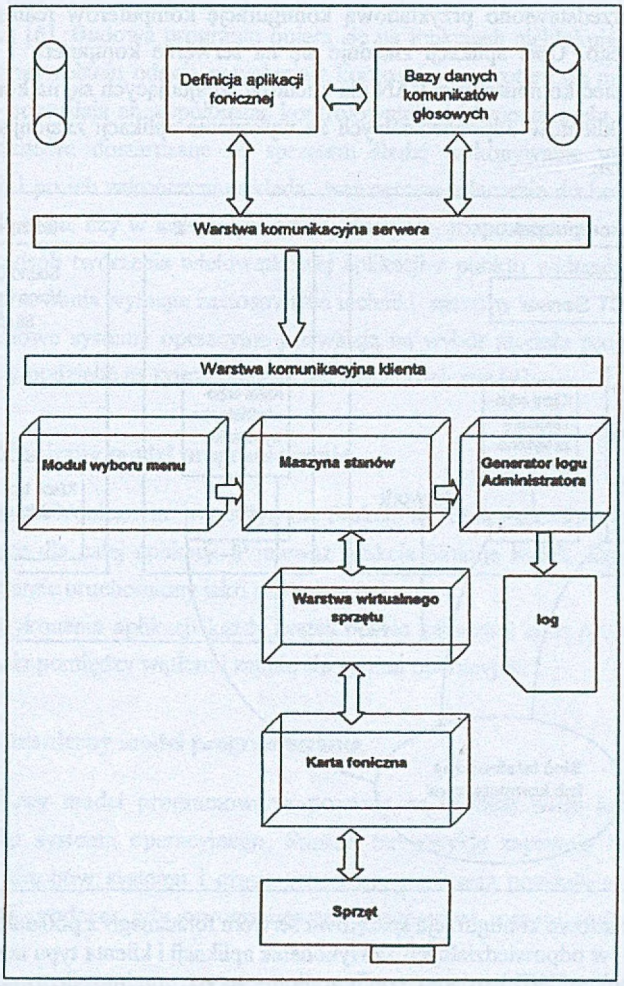


Rys. 3a. Serwer foniczny o architekturze klient serwer
Fig. 3a. Client-Server Interactive Voice Response architecture design

Przykładowy system o architekturze klient-serwer pokazano na rysunku 3b. Część serwerowa składa się z dwóch modułów: pierwszy to definicja opisu aplikacji fonicznej tzw. VAD (ang. Voice Application Definition); drugi to baza danych komunikatów głosowych, tzw. MDB (ang. Message Data Base). Oba te moduły komunikują się ze sobą oraz z warstwą klienta poprzez zdefiniowany protokół komunikacyjny. W module klienta zasadniczym elementem jest maszyna stanów, w której zaimplementowane są wszystkie procedury zdefiniowane w opisie VAD. Maszyna stanów jest modułem wielowątkowym odpowiedzialnym za współpracę pomiędzy opisem aplikacji znajdującym się na serwerze a wirtualną warstwą sprzętową oraz jest modułem wykonującym algorytm aplikacji fonicznej. Maszyna stanów również generuje tzw. log administratora, który pozwala zbierać informacje o tym, co się dzieje w systemie.

Biblioteki API do sprzętu tzw. VH (ang. Virtual Hardware) pozwalają na implementację serwera fonicznego na sprzęcie różnych producentów.

W przedstawionym modelu klient-serwer można dokonywać zmian aplikacji w trakcie jej działania. Z tego punktu widzenia należy podkreślić, że rozwiązanie oparte na dynamicznym opisie jest stosunkowo niebezpieczne. Zmiany wprowadzane on-line mogą prowadzić do błędnego działania aplikacji.

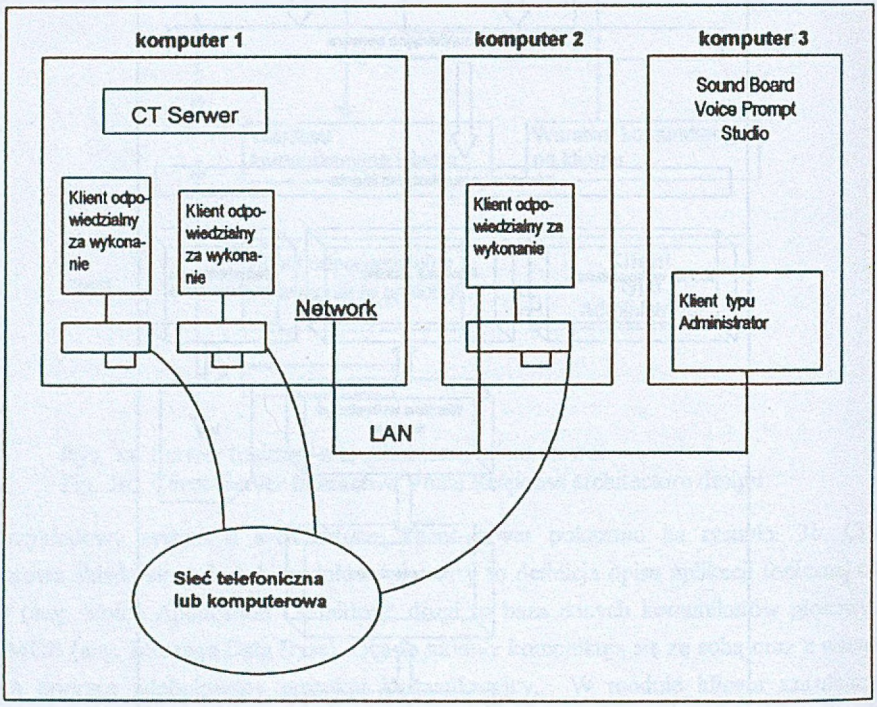


Rys. 3b. Przykładowa architektura serwera fonicznego o organizacji klient-serwer
Fig. 3b. Client Server Design

Szczególną uwagę należy zwrócić na projekt standaryzacji opisu aplikacji fonicznej o nazwie VoiceXML. Jest to standard, który rozszerza funkcjonalność serwera www, wprowadzając do niej możliwość wykorzystania interfejsu fonicznego.

Projekt ten w warstwie komunikacyjnej opiera się na dobrze zdefiniowanym i przetestowanym standardzie komunikacji klient-serwer, bazującym na protokole TCP/IP wprowadzając jednocześnie dodatkowe rozszerzenie do języka http o foniczny interfejs z użytkownikiem. Być może to rozwiązanie stanie się w przyszłości standardem opisu interfejsu dla serwerów fonicznych [11].

Na rys. 4 przedstawiono przykładową konfigurację komputerów realizujących zadania serwera fonicznego. Opis aplikacji znajduje się na serwerze komputera 1 i jest dostępny poprzez lokalną sieć komputerową LAN dla klientów znajdujących się na komputerach 2 i 3, jak również dla klientów odpowiedzialnych za wykonanie aplikacji znajdujących się na tym samym komputerze.



Rys. 4. Przykładowa konfiguracja sprzętowa serwera fonicznego z podziałem na serwer klientów odpowiedzialnych za wykonanie aplikacji i klienta typu administrator

Fig. 4. Example of IVR hardware configuration – server, runtime clients and administrator client with Voice Prompt Studio

3. Wybór modelu programowania dla aplikacji serwera fonicznego

Wybór modelu programowania w największym stopniu zależy od systemu operacyjnego. Wielozadaniowe systemy z wydziedziczaniem dają programiście możliwość wyboru modelu programowania. W przeciwieństwie do systemów jednozadaniowych, takich jak DOS, gdzie tworzenie aplikacji dla serwerów fonicznych można oprzeć tylko na modelu asynchronicznym z przeglądaniem zdarzeń. W wypadku systemu DOS ze względu na jednozadaniowy charakter systemu operacyjnego programista nie ma możliwości wyboru modelu

programowania [6]. Budowa programu opiera się na funkcjach nieblokujących, to znaczy takich, które po wywołaniu oddają natychmiast kontrolę z powrotem do programu głównego. W ten sposób pozwalają na współbieżne kontrolowanie aktywności wielu kanałów. Oprogramowanie systemowe dostarczane ze sprzętem śledzi wykonywanie wszystkich zainicjowanych funkcji i po ich zakończeniu układa wewnętrzne zdarzenia do kolejki zdarzeń. Aplikacja przegląda stale, czy w kolejce są nowe zdarzenia, rozpoznaje je i odpowiednio je obsługuje. Ten sposób tworzenia wielowątkowej aplikacji z punktu widzenia implementacji języków oprogramowania wymaga zastosowania techniki *maszyny stanu*.

Wielozadaniowe systemy operacyjne pozwalają na wybór modelu programowania, które ogólnie możemy podzielić na synchroniczne lub asynchroniczne [9].

3.1. Synchroniczny model programowania

W modelu synchronicznym wywoływane funkcje są blokujące dla wątku, ale jednocześnie nieblokujące dla całej aplikacji. Ponieważ funkcja blokuje wątek, każdy kanał foniczny musi być w systemie uruchomiony jako osobny wątek.

W czasie wykonania aplikacji każdy wątek będzie zajmował czas procesora, a sterowaniem przełączania pomiędzy wątkami zajmie się system operacyjny.

3.2. Asynchroniczny model programowania

Asynchroniczny model programowania pozwala na obsługę wielu kanałów fonicznych w jednym wątku systemu operacyjnego. Sposób ten zwykle zapewnia bardziej efektywne wykorzystanie zasobów systemu i czasu procesora, ponieważ pozwala na kontynuowanie obsługi aplikacji, podczas gdy oprogramowanie systemowe sprzętu realizuje odpowiednie funkcje.

Model ten wydaje się być również odpowiedniejszy do synchronizowania zdarzeń pochodzących z różnych kanałów.

Model asynchroniczny możemy podzielić na następujące przypadki.

3.2.1. Model asynchroniczny z przeglądaniem

Jest to model, który jest zgodny z modelem programowania w systemach jednozadaniowych, takich jak DOS. Ostatnim elementem wykonywania funkcji jest wygenerowanie tego zdarzenia, które oznacza, że funkcja została zakończona; aplikacja w nieskończonej pętli przegląda, jakie zdarzenia zostały wygenerowane, pobiera je, rozpoznaje przyczynę zakończenia funkcji i podejmuje odpowiednie decyzje, co robić dalej.

3.2.2. Model asynchroniczny z wywołaniem

Model ten używa tych samych zdarzeń jak model asynchroniczny z przeglądaniem. Różnica polega na tym, że przychodzące zdarzenie powoduje wywołanie podprogramu systemowego (ang. *handler*). Wywoływany program jest skokiem do odpowiedniego miejsca w aplikacji, które obsługuje odpowiednie zdarzenie. Po zakończeniu obsługi wywołanego podprogramu sterowanie wraca do pętli głównej aplikacji.

3.2.3. Model asynchroniczny z wywołaniem dla systemu WINDOWS

W systemie WINDOWS zdefiniowano specjalną pętlę do przekazywania komunikatów, tzw. „Windows Messaging Loop". W modelu tym wykorzystuje się mechanizm komunikatów wbudowany w system WINDOWS, aby sterować aplikacją serwera fonicznego. Jeżeli pojawi się jakieś zdarzenie pochodzące ze sprzętu, odpowiedni komunikat zdefiniowany przez użytkownika systemu (programistę) jest umieszczany w kolejce wiadomości systemu WINDOWS („Windows Message Queue"). Przekazany komunikat, poprzez mechanizmy systemu operacyjnego Windows, powoduje, że aplikacja wie, że ma pobrać zdarzenie pochodzące od sprzętu i obsłużyć je w taki sam sposób jak w modelu asynchronicznym z przeglądaniem. Zdarzenia pochodzące ze sprzętu nie wywołują bezpośrednio programów obsługi aplikacji, a wykorzystują do tego mechanizm wymiany komunikatów systemu Windows. W tym przypadku w przeciwieństwie np. do modelu asynchronicznego aplikacja nie wymaga zaimplementowania nieskończonej pętli oczekującej na zdarzenia pochodzące od sprzętu.

3.2.4. Rozszerzony model asynchroniczny

W modelu tym programista określa urządzenie, którego zdarzenia będzie obsługiwał. To pozwala na dowolne przyporządkowanie różnych kanałów sprzętu do wątków systemu operacyjnego. Obsługa otrzymanego zdarzenia może być wykonana w dowolny, opisany wcześniej, sposób.

4. Proponowany model programowania

Autorzy w wyniku prac wybrali asynchroniczny model programowania, jako że w najmniejszym stopniu zależy on od systemu operacyjnego oraz ze względu na możliwość zaimplementowania zarówno dla systemów wielozadaniowych, jak i dla systemów jednozadaniowych, takich jak DOS. Niestety, ze względu na inną obsługę zdarzeń pochodzących ze sprzętu, oprogramowanie aplikacyjne serwera fonicznego nie jest bezpośrednio przenośne między systemami operacyjnymi.

4.1. Model programowy obsługi specjalizowanej karty realizującej interfejs foniczny

Serwery foniczne muszą być wyposażone w specjalizowany sprzęt pozwalający na równoczesną obsługę wielu kanałów fonicznych w czasie rzeczywistym. Z punktu widzenia interfejsu fonicznego możemy powiedzieć, że użytkownik może się łączyć z serwerem fonicznym poprzez linię telefoniczną (analogową lub cyfrową) lub poprzez sieć komputerową.

Istnieje wiele różnych standardów sprzętu, jakiego można użyć do budowy serwera fonicznego. Sprzęt taki jest wyposażony przez producenta w odpowiednie oprogramowanie systemowe.

Oczywiście, oprogramowanie systemowe nie jest bezpośrednio przenośne pomiędzy różnymi producentami. Istnieje zatem pytanie, czy można zwirtualizować tak model sprzętu dla serwera fonicznego, aby można było przenosić oprogramowanie w danej klasie rozwiązań sprzętowych różnych producentów.

Autorzy postawili sobie cel badawczy porównania oprogramowania systemowego dostarczonego wraz z kartami różnych producentów w celu stworzenia odpowiedniej biblioteki wywołań umożliwiającej budowę aplikacji, które byłyby przenośne. Do badań wybrano, jako najpopularniejszy, sprzęt firmy Dialogic [5] (obecnie własność firmy INTEL) oraz kartę firmy Rhetorex [8] (obecnie własność firmy Brooktrouth).

Rezultatem prac jest biblioteka pozwalająca na uruchomienie zaprojektowanych aplikacji fonicznych na kartach wymienionych producentów.

W celu wymiany informacji pomiędzy sprzętem a oprogramowaniem aplikacyjnym zaprojektowano odpowiednie struktury parametrów. Pierwszą strukturą są parametry globalne dotyczące całej karty sprzętowej, takie jak:

- przerwanie,
- liczba dostępnych kanałów fonicznych,
- kolejka zdarzeń generowanych przez kartę zawierającą następujące informacje: numer kanału, kod zdarzenia i dane.

Następną zaprojektowaną strukturą są dane dla każdego kanału fonicznego przechowujące następujące informacje:

- procedura aplikacji fonicznej (stan), która obsługuje dany kanał,
- status kanału (wolny lub zajęty),
- plik – jeżeli został otwarty dla obsługi danego kanału,
- kody przekazane od użytkownika do serwera poprzez interfejs foniczny – w przypadku analogowej linii telefonicznej są to kody DTMF,
- maksymalna liczba kodów, na jakie oczekuje aplikacja serwera fonicznego,
- kody przerywające,

- maksymalny czas oczekiwania pomiędzy rozpoczęciem a zakończeniem operacji,
- maksymalny okres ciszy w kanale,
- parametry połączenia, np. numer telefonu,
- komunikat błędu,
- warunki wywołania i rozpoczęcia działania aplikacji na danym kanale.

Struktury danych służą do przekazywania informacji pomiędzy wywoływanymi funkcjami. Aplikacja komunikuje się ze sprzętem, poprzez odpowiednie oprogramowanie systemowe producenta. Dla kart fonicznych zaprojektowano następujące funkcje:

- inicjacja karty fonicznej,
- zatrzymanie pracy karty fonicznej,
- zatrzymanie pracy kanału,
- pobranie zdarzenia,
- odebranie i rozłączenie,
- wybranie numeru zewnętrznego,
- wybranie numeru zewnętrznego z analizą przebiegu połączenia,
- odebranie pojedynczego kodu od użytkownika,
- odebranie ciągu kodów od użytkownika,
- odtwarzanie komunikatu fonicznego,
- nagrywanie komunikatu fonicznego.

Analizie poddano również zdarzenie pochodzące od karty fonicznej. Zdarzenia te zostały zaklasyfikowane do jednego z poniżej wymienionych kodów [7]:

- zdarzenie oznaczające zgłoszenie połączenia fonicznego w celu jego odebrania,
- zdarzenie oznaczające otwarcie kanału fonicznego,
- zdarzenie oznaczające zamknięcie kanału fonicznego,
- zdarzenie oznaczające ciszę w kanale fonicznym,
- zdarzenie oznaczające przerwanie ciszy,
- zdarzenie oznaczające otrzymanie od użytkownika maksymalnej liczby kodu,
- zdarzenie oznaczające otrzymanie od użytkownika kodu zdefiniowanego jako przerywający,
- zdarzenie oznaczające zatrzymanie kanału,
- zdarzenie oznaczające wybranie numeru,
- zdarzenie oznaczające odebranie tonu ciągłego,
- zdarzenie oznaczające zakończenie analizy przebiegu połączenia.

Wyżej wymienione elementy, czyli parametry globalne dotyczące całej karty sprzętowej, parametry kanału odnoszące się do poszczególnych kanałów fonicznych, funkcje służące do inicjowania określonych zadań oraz związane z nimi zdarzenia informujące o sposobach za-

kończenia zainicjowanych funkcji, pozwoliły na zbudowanie biblioteki API, pozwalającej na przenośność oprogramowania serwerów fonicznych pomiędzy kartami różnych producentów.

W tym miejscu należy również wymienić standard TAPI, który jest biblioteką wirtualizującą sprzęt telefoniczny w systemie Windows. Biblioteka ta pozwala osiągnąć ten sam cel, jaki postawili sobie autorzy, jednak opierając się na standardzie systemu operacyjnego Windows.

Z punktu widzenia zastosowań fonicznych system Windows, zdaniem autorów, wprowadza duży narzut i zabiera moc obliczeniową oraz zasoby komputera na realizację zadań (np. graficznego interfejsu z użytkownikiem), które w przypadku serwerów fonicznych mogą być w ogóle nie wykorzystane. Na pewno system Windows nie jest jedyną możliwą platformą na osadzenie serwera fonicznego. Dlatego ograniczenie się do biblioteki TAPI, której implementacja jest dostępna tylko na jednej platformie systemowej, zdaniem autorów, byłoby niewystarczające.

5. Wnioski

Z punku widzenia architektury serwerów fonicznych istotne dla konstruktora systemu jest podjęcie decyzji co do:

- sposobu rozwiązania wielowątkowości aplikacji fonicznej i wyboru systemu operacyjnego, na którym zaimplementowany zostanie projekt,
- opisu fonicznego interfejsu z użytkownikiem i odpowiedniego rozwiązania architektonicznego pozwalającego na wydzielenie go w stosunku do kodu samej aplikacji,
- wyboru odpowiedniego sprzętu.

Jako architekturę najbardziej zaawansowaną przedstawiono model klient-serwer opisu aplikacji fonicznej, w której opis ten znajduje się na zewnętrznym, w stosunku do aplikacji, serwerze. W tym przypadku trzeba podkreślić, że autorzy mają na myśli opis aplikacji, a nie dane, z których aplikacja korzysta. Dla przykładu, aplikacja typu interakcyjna odpowiedź głosowa, która pobiera do swojego działania dane z zewnętrznego serwera bazy danych, może być oparta na interfejsie fonicznym zaszytym bezpośrednio w kodzie aplikacji. W rozumieniu autorów może stanowić najgorszy typ implementacji dla serwera fonicznego.

Aktualnie trwają prace nad językiem VoiceXML, który może się stać standardem w dziedzinie opisu dla serwerów fonicznych o architekturze klient-serwer.

LITERATURA

1. Rychlicka H., Suszczańska N., Wideł S.: Generacja wybranych komunikatów poczty głosowej w postaci tekstów fonicznych w języku polskim, Studia Informatica (Formerly: Zeszyty Naukowe Politechniki Śląskiej, seria „Informatyka"), vol. 21, No1 (39), Gliwice 2000.
2. Wideł S., Rychlicka H.: Algorytmy translacji podstawowych formatów danych na postać głosową dla interakcyjnych systemów telefonicznych, ZN Pol. Śl. seria Informatyka, z. 36, Gliwice 1999.
3. Wideł S., Stera P.: Zdalny dostęp do danych za pomocą interakcyjnych systemów telefonicznych w jednostkach administracji publicznej, na przykładzie Urzędu Miasta Pszów, ZN Pol. Śląskiej, seria Informatyka, z. 34, Gliwice 1998.
4. Grzywak A., Wideł S.: Multimedialny rozproszony system zarządzania Urzędem Miasta Pszów, Komputerowe Systemy Wielodostępne – Seminarium KBN, Bydgoszcz 1998.
5. Dialogic, "Voice Software Reference for MS-DOS" Volume 1, 2 Dialogic Corporation 05-004-006 1992-1993.
6. Dialogic, "Voice Programmer's Guide for MS-DOS" Dialogic Corporation 05-004-006 1994.
7. Dialogic, "Voice Features Guide for MS-DOS" Dialogic Corporation 05-004-006 1994.
8. Rhetorex, "RealCT User Guide for Windows NT and Windows 95", rev 1, 00 00-00 3528, October 1996.
9. Rhetorex, "API Reference for Windows 95 and Windows NT", rev 1.01, 00-00-3037, March 1996.
10. Bob Edgar: PC-Based Voice Processing" Second Edition Flatiron Publishing Inc..ISBN 0-936648-45-7 1994r.
11. http://www.voicexml.org

Recenzent: Prof. dr inż. Stefan Węgrzyn

Wpłynęło do Redakcji 25 kwietnia 2001 r.

Abstract

In this paper we discussed same implementation assumption for human-machine interaction provided by voice response systems. Main architectural goal is to break the application to parts. One parts is a low-level programming and resource management. Another parts is a

voice service is viewed as a sequence of interaction dialogs between a user and an implementation platform.

The implementation platform generates events in response to user actions, system events, produce dialogs, and generate the system log. In this paper we discussed, several programming models for the implementing multithread applications. The pre-emptive multitasking nature of the operating systems has allowed to provide an API that offers the developer several programming models from which to choose. Just as it is important to understand the differences between the operating systems themselves, which must be understood before attempting an application developing.

Voice services enables the integration with data services using the familiar client-server paradigm. The dialogs are described by servers, which may be external to the implementation platform. Servers maintain overall service logic, perform database and legacy system operations, and interaction dialog. User input affects dialog interpretation and is collected into requests submitted server.