

Mediu pentru dezvoltarea sistemelor de dialog prin metoda Vrăjitorului din Oz

Cosmin Munteanu

Conducător proiect: ș.l. ing. **Marian Boldea**

Rezumat

Scopul lucrării de față este de a furniza o soluție la problema dezvoltării sistemelor de dialog, problemă care ridică dificultăți specialiștilor din domeniu.

Deși pe plan mondial există unele instrumente de dezvoltare, ele nu sunt din păcate disponibile public sau sunt disponibile doar parțial și pentru numai câteva limbi.

Această lucrare este un început, o bază de pornire pentru a pune la punct un prim set de instrumente care să permită construcția unor sisteme de dialog.

De-a lungul capitolelor ce urmează, se arată care sunt principalele dificultăți în realizarea unor astfel de sisteme și ce soluții există; se prezintă o realizare a unui set de instrumente de dezvoltare și se analizează plusurile și minusurile sale; se indică aspectele nerezolvate încă, dar care se speră a fi soluționate în continuare.

“As Dorothy gazed upon this in wonder and fear, the eyes turned slowly and looked at her sharply and steadily. Then the mouth moved, and Dorothy heard a voice say, *‘I am Oz, the Great and Terrible. Who are you, and why do you seek me?’* “

L. Frank Baum, *The Wizard of Oz*

Cuprinsul

Cuprinsul	3
Lista figurilor	4
Lista tabelelor	5
1 Sisteme de dialog	6
1.1 Interfețe avansate bazate pe dialog	6
1.2 Structura sistemelor de dialog	7
1.3 Limite ale sistemelor de dialog	10
1.3.1 Limitări la nivelul interacțiunii	10
1.3.2 Limitări în proiectare	11
2 Dezvoltarea Sistemelor de dialog	13
2.1 Principii ale dezvoltării sistemelor de dialog . . .	13
2.2 Modelarea dialogului	15
2.3 Metoda Vrajitorului din Oz	18
2.3.1 Necesitatea Vrajitorului din Oz	18
2.3.2 Modalități de comunicare	19
2.3.3 Variabilele metodei	19
2.3.4 Erori tipice	22
3 Mediul de dezvoltare	23
3.1 Structura mediului de dezvoltare	24
3.1.1 Aspecte software	26

<i>CUPRINSUL</i>	3
3.1.2 Utilitatea mediului de dezvoltare	29
3.2 Interfața cu Vrăjitorul	29
3.2.1 Interfața SQL	30
3.2.2 Managementul dialogului	32
3.2.3 Interfața pentru răspuns	34
3.3 Interfața cu utilizatorul	35
3.4 Sistemul de comunicație	36
3.5 Colectarea datelor	37
4 Experimente	39
5 Concluzii și continuări	42
Referințe	45
Anexe	48
A Codul sursă al programelor	48
B Fișiere de configurație	131

Lista figurilor

1.1	Structura generală a unui sistem de dialog	8
2.1	Taxonomia simulărilor prin metoda Vrajitorului din Oz	20
3.1	Structura mediului de dezvoltare	25
3.2	Diagramele de secvență ale programelor	28
3.3	Implementarea interfeței cu baza de date	30
3.4	Interfața de acces la baza de date	31
3.5	Modelarea dialogului – exemplu	33
3.6	Mediul daVinci	34
3.7	Sistemul de comunicație	36
3.8	Fluxul de control prin rețea	37
4.1	Ilustrare a deteriorării calității semnalului și ecoului	40

Lista tabelelor

1.1	Caracteristici ale conversației	11
3.1	Parametrii înregistrării	38

Capitolul 1

Sisteme de dialog

1.1 Interfețe avansate bazate pe dialog

De la apariția calculatoarelor, a fost evident faptul că, pentru a putea fi folosite în cele mai diverse domenii, ele au nevoie de o interfață cu utilizatorul cât mai ușor de folosit, dar care totuși să ofere posibilitatea unei interacțiuni cât mai complexe și mai eficiente, în scopul execuției de către calculator a unor sarcini dificile. În acest context, cercetarea științifică a pus un accent covârșitor pe dezvoltarea interfețelor om-mașină. Istoria calculatoarelor cunoaște multe momente cruciale în evoluția interfețelor, de la panouri de comandă cu comutatoare și cartele perforate, la console și mijloace de afișare în mod text, ajungându-se la paradigma interfețelor grafice, viitorul nu foarte îndepărtat promițând realitatea virtuală.

În momentul de față, importante eforturi pe plan mondial sunt direcționate spre proiectarea interfețelor bazate pe vorbire. Fără o interfață performantă bazată pe vorbire, sistemele de dialog nu își au rostul. Deși, aparent, aceste interfețe sunt foarte utilizate sub diferite forme, nu există o metodă științifică de abordare a proiectării sistemelor de dialog unanim acceptată de către cercetători. În general, majoritatea aplicațiilor care au la bază un sistem de dialog folosind interfețe bazate pe vorbire au

un caracter comercial, realizările cele mai cunoscute fiind din sfera sistemelor automate de furnizare a unor informații, prin telefon în majoritatea cazurilor (Air Travel Information Service — ATIS [2] în S.U.A. și Franța, Railway Telephone Information Service — RAILTEL [3] și Automatic Railways Information Systems for Europe — ARISE [4] în Europa fiind unele din cele mai cunoscute). Un sistem nu foarte complex, dar care are și interfață grafică interactivă este sistemul suedez WAXHOLM [5], pentru informații despre orarul vapoarelor.

Interfețele bazate pe vorbire sunt un prim pas în dezvoltarea unor interfețe interactive complexe, în care sistemul să fie capabil de recunoașterea și înțelegerea nu numai a vorbirii împreună cu elemente lingvistice suplimentare (prozodie, anaforă), dar și a mimicii și gesturilor. Deocamdată, deși îmbunătățirile în recunoașterea vorbirii au dus la scăderea ratei de erori produsă de aceasta (atât prin metodele de recunoaștere, cât și prin creșterea performanțelor calculatoarelor), nu se explică nemulțumirile utilizatorilor care nu reușesc îndeplinirea unor sarcini cu ajutorul unui sistem de dialog, folosind actuala tehnologie de recunoaștere a vorbirii [6]. Problema este deci mult mai complexă, domeniul sistemelor interactive prezentând încă multe necunoscute.

1.2 Structura sistemelor de dialog

Având în vedere cele menționate la 1.1, putem spune că domeniul sistemelor de dialog interactive este încă la începuturile sale, ceea ce nu ne permite să conturăm mai precis o definiție și o structură a sistemelor de dialog. Totuși, vom pleca de la descrierea făcută de N. O. Bernsen, H. Dybkjaer și L. Dybkjaer: “Sistemele interactive bazate pe dialog vor fi caracterizate ca sisteme de calcul ce permit oamenilor să îndeplinească cel puțin o parte din sarcini printr-o anumită formă de comunicare verbală” [7]. Puțini autori indică în literatura de specialitate o

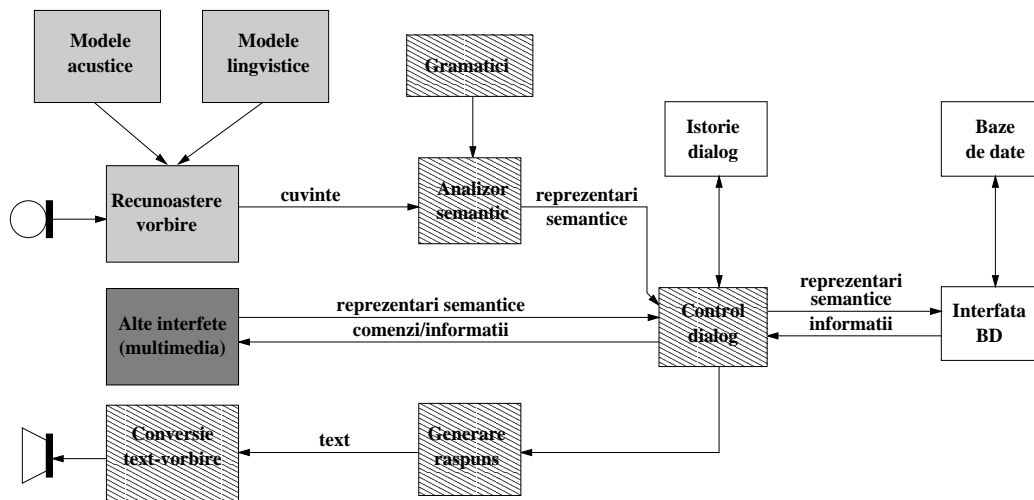


Figura 1.1: Structura generală a unui sistem de dialog

structură precisă a unui astfel de sistem, majoritatea fiind concretizate în funcție de aplicația pentru care au fost proiectate, o structură general valabilă nefiind încă pusă la punct; putem încerca o definiție generală a structurii de forma prezentată în figura 1.1 [8].

Elementele componente ale structurii se pot grupa în cinci categorii, în ordinea dată de prelucrarea semnalului vocal până la accesarea informației dorite:

- 1) **Dispozitive de Intrare/Ieșire.** Uzual, includ un microfon și un difuzor (boxe, căști). Pentru un sistem avansat, mai pot apare o cameră video, touch-screen ș.a. De asemenea, dacă destinația sistemului este colectarea semnalului vocal sau diferite măsurători la nivelul semnalului sau al componentelor bio-mecanice, se pot folosi instrumente specifice (laringograf, endoscop pentru monitorizarea tracului vocal, diferiți senzori). Alte interfețe multimedia pot

fi adăugate pentru a conecta dispozitive speciale de intrare/ieșire.

2) **Module de interfață.** Cuprind patru părți esențiale:

- *Modele acustice* folosite în recunoașterea la nivel de unități acustice (uzual la nivel de fonem într-un sistem general, independent de vorbitor).
- *Modelele lingvistice* completează modelele acustice pentru a realiza recunoașterea cuvintelor.
- *Recunoașterea vorbirii:* folosind semnalul vocal de la intrare, pe baza modelelor acustice și lingvistice, generează cuvintele (text) presupuse a fi fost produse de utilizator.
- *Conversia text-vorbire* realizează acțiunea inversă recunoașterii, producând răspunsul sistemului.

3) **Module Semantice:** partea care realizează “înțelegerea” acțiunilor utilizatorului (vorbire, eventual gesturi); are ca și componente:

- *Gramatici*, reguli pentru a descrie gramatica limbajului natural.
- *Analizorul semantic* primește la intrare textul generat de modulul de recunoaștere a vorbirii și pe baza gramaticilor generează reprezentări semantice.
- *Generarea răspunsului* este componenta utilizată pentru a sintetiza răspunsul adresat utilizatorului; poate fi implementată pe baza unui sistem TTS (text-to-speech) sau folosind răspunsuri preînregistrate.

4) **Managementul Dialogului:** partea esențială a unui sistem de dialog

- *Istoria dialogului* înregistrează reprezentări ale interacțiunilor sistemului (propozițiile utilizatorului și ale vrăjitorului, precum și secvența momentelor dialogului).
 - *Controlul dialogului* are rolul de a gestiona desfășurarea dialogului și a interacțiunii între utilizator și sistem.
- 5) **Managementul Bazei de Date:** facilitează accesul la datele pe care utilizatorul le cere sistemului
- *Baza de Date* gestionează informațiile efective.
 - *Interfața B.D.* asigură legătura între sistemul de dialog și baza de date, preluând cereri și transmițând rezultate.

1.3 Limite ale sistemelor de dialog

Limitările actuale în domeniul sistemelor de dialog în ansamblu (deci în afara celor ce țin exclusiv de limitările tehnologice sau teoretice ale anumitor părți componente, cum ar fi recunoașterea sau generarea vorbirii) au la bază elemente ale teoriei interacțiunii și elemente legate de proiectarea și realizarea lor.

1.3.1 Limitări la nivelul interacțiunii

Pentru a aplica teoria interacțiunii asupra unui sistem de dialog trebuie studiat dialogul între persoane. Oamenii sunt capabili să susțină un dialog fluent, fără efort și eficient despre aproape orice subiect. În plus, prin conversație, oamenii realizează și anumite scopuri. Pentru a putea susține o conversație, sunt necesare anumite calități, naturale oamenilor (Tabelul 1.1). Dialogul om-calculator este restricționat însă de limitele calculatorului. Aceasta înseamnă că modelul de interacțiune trebuie

Tabelul 1.1: Caracteristici ale conversației

Recunoaștere	Înțelegere		Dialog	Generare
	Sintactic și semantic	Contextul discursului		
Recunoaștere vorbire spontană independentă de vorbitor, zgomote, variații de intensitate; Vocabular foarte mare	Prosodie; Propoziții incomplete; Ezitări; Repetiții	Anafore; Focalizare; Istorie discurs	Schimbarea dialogului funcție de indicii, semantică; Reacție în timp real	Expresii semantice complexe; Stil adaptat situației; Intonație

construit cu mare atenție, chiar pentru un domeniu restrâns, altfel sistemul nu va funcționa corespunzător. Din păcate, majoritatea teoriei interacțiunii (scrise și vorbite) a urmărit doar analiza dialogului general între oameni, fără a urmări un dialog orientat pe executarea unei sarcini, cum este dialogul om-mașină [7]. Acesta este un motiv suplimentar pentru a dezvolta sisteme de dialog care să permită colectarea a cât mai multe date experimentale.

1.3.2 Limitări în proiectare

Deși multe aspecte ale tehnologiei în domeniul vorbirii ridică probleme — cum ar fi recunoașterea, vocabularul, gestionarea dialogului — un număr destul de mare de sisteme de dialog funcționează cu succes. Din păcate, aceste sisteme sunt caracterizate de un vocabular limitat și gramatici simple, fiind orientate pe aplicația pentru care au fost proiectate. Toate aceste neajunsuri ale sistemelor de dialog au la bază diferite cauze [9]:

- **Lipsa expertizei.** Pentru ca un sistem de dialog să func-

ționeze, este nevoie de o expertiză tehnică (în fazele de proiectare și testare) care acoperă mai multe domenii, ceea ce duce la limitarea cercetării mai mult la nivelul laboratoarelor.

- **Costurile mari de realizare.** Procesul de realizare durează în timp și este scump (durata realizării unui sistem obișnuit depășește de obicei un an). Colectarea datelor pentru antrenarea modelelor acustice necesită timp, iar pentru modelele semantice și de dialog este costisitoare.
- **Neportabilitatea.** Tehnologia folosită la implementarea unui sistem de dialog nu permite portarea sa pe alte arhitecturi sau sisteme de calcul fără a suferi modificări esențiale, datorită neuniformității resurselor hardware și software existente în domeniu.
- **Dificultatea reutilizării.** Un sistem de dialog este destinat unei aplicații concrete, realizarea unui nou sistem, folosind componente existente, este dificil de realizat.

Capitolul 2

Dezvoltarea Sistemelor de dialog

În ciuda existenței structurii generale prezentate la 1.1, domeniul sistemelor de dialog se află încă la limita dintre artă și tehnică. Etapele bine definite în dezvoltarea unui sistem de dialog, de la specificarea cerințelor până la testarea în funcționare, se întrepătrund cu situații necunoscute cauzate de suportul teoretic și practic inexistent sau incomplet, astfel încât multe probleme nu și-au găsit încă o rezolvare general valabilă [7].

2.1 Principii ale dezvoltării sistemelor de dialog

Dezvoltarea sistemelor de dialog, ca parte esențială a dezvoltării sistemelor interactive avansate bazate pe vorbire, concentrează o mare parte a efortului de proiectare. Această dezvoltare se poate face urmând două căi principale: iterativă și prin simulare. Metoda iterativă presupune fazele ciclice: implementare–testare–revizuire, iar metoda prin simulare : simulare–evaluare–revizuire.

Ambele metode au la bază un model al dialogului, numit *prim-model de interacțiune* [7]. Construirea acestui prim-model

este decisivă pentru întreaga fază de proiectare și este determinată de metoda de proiectare aleasă. Factorii importanți de care se ține seama în implementarea prim-modelului aparțin de teoria vorbirii; nu există metode concrete pentru dezvoltarea modelului, experiența proiectantului este decisivă. Deoarece fără un prim-model de interacțiune nu se poate dezvolta un sistem de dialog, este necesar ca modelul să fie specificat cât mai complet posibil; pentru a satisface această cerință este necesară folosirea oricăror instrumente care pot eficientiza proiectarea prim-modelului. Această constatare vine în sprijinul ideii că un mediu de dezvoltare care să reducă eforturile (și erorile) inițiale este cât se poate de necesar.

Orice metodă de dezvoltare presupune existența unor *criterii de evaluare*. Este dificilă stabilirea unor criterii exacte de evaluare, vorbirea și dialogul fiind complexe și dificil de specificat prin chiar natura lor. O clasificare a parametrilor de evaluare ar putea fi în *obiective* și *subiective* [10].

Criteriile obiective includ factori ca: timpul total al propozițiilor, numărul de schimbări de dialog, rata corecțiilor, succesul interacțiunii. Deși termenul de succes e departe de a fi obiectiv, putem considera *succesul interacțiunii* ca o cantitate booleană, indicând dacă utilizatorul a rezolvat problema (a obținut informațiile dorite) fără să primească informații inutile din partea sistemului.

Evaluarea sistemului pe baza criteriilor obiective nu este suficientă din punctul de vedere al utilizării acestuia. De aceea, criteriile subiective au ca scop aflarea opiniei utilizatorului despre sistem. Cea mai simplă metodă este prin chestionare completate în faza de testare-simulare. Chestionarele includ întrebări și aprecieri despre dificultatea utilizării, naturalețe, claritate, erori, efortul depus în aflarea informațiilor ș.a.

2.2 Modelarea dialogului

Partea cea mai importantă într-un sistem interactiv bazat pe vorbire este componenta de dialog. Importanța componentei de dialog este evidentă: de cele mai multe ori, utilizatorul nu poate exprima într-o propoziție simplă cererea adresată sistemului, iar sistemul intervine ajutând utilizatorul, creându-se astfel un dialog. Mai mult, controlul dialogului este răspunzător de identificarea și corectarea erorilor cauzate de recunoașterea și înțelegerea vorbirii.

Studiile asupra dialogului om-mașină au urmat două linii principale date de analiza dialogului între persoane [10]:

- **Analiza discursului** privește dialogul ca o cooperare rațională și presupune că textul rostit de vorbitor este compus din propoziții bine formate;
- **Analiza conversației** studiază dialogul ca o interacțiune socială în care trebuie luate în considerație fenomene ale limbajului natural (cum ar fi prozodia, anafora, întreruperi, ezitări ș.a.).

În modelarea dialogului trebuie ținut cont de stadiul actual al recunoașterii vorbirii și de timpul relativ scurt alocat dezvoltării prototipului unui sistem [11]. Primul criteriu intervine mai puțin în metoda discutată aici (vezi 2.3). Al doilea criteriu determină implementarea interfeței ca ”direcționată”, adică utilizatorul are o foarte mică inițiativă, sistemul controlând dialogul. Sub aspectul dialogului, se pot distinge trei tipuri de dialog, în ordinea complexității fiind: *single-word*, la nivel de cuvânt, pentru sisteme simple, în care utilizatorul dă comenzi sistemului; *system-directed*, direcționat, în care sistemul controlează desfășurarea dialogului; *user-oriented*, în care utilizatorul poate lua inițiativa [12]. Dialogul direcționat are două tipuri de puncte de decizie:

interpretarea răspunsului dat de utilizator și interpretarea rezultatului consultării bazei de date, și trei tipuri de acțiuni posibile: captarea răspunsului utilizatorului, consultarea bazei de date și transmiterea unui mesaj către utilizator.

În general, din punct de vedere al design-ului, proiectarea unui dialog poate fi: *la nivelul sistemului*, în care se urmăresc aspecte tehnice, se consideră că utilizatorul urmărește sarcini precise; *la nivelul aplicației* are în centrul atenției îndeplinirea sarcinilor ce se propun a fi executate de către sistem; *la nivel utilizator*, unde apar mai puține constrângeri din partea sistemului [13].

Managementul dialogului se bazează pe două subcomponențe [10]:

- **Istoricul interacțiunii** este folosit pentru a interpreta propoziții în care apar anafore sau alte elemente ale vorbirii spontane care nu pot fi înțelese decât în funcție de context. Contextul dialogului (denumit și *focalizare activă*) se schimbă în decursul dialogului, ceea ce duce la necesitatea ca sistemul să aibă acces la un istoric actualizat, într-o reprezentare eficientă (cum ar fi arbori ierarhici).
- **Modelul de interacțiune** (în fazele de simulare, primul model, vezi 2.1) definește strategia care ghidează dialogul. Strategia poate fi plasată între două extreme: completă libertate a utilizatorului în contrast cu controlul total al sistemului. Proiectarea unei strategii potrivite este esențială, deoarece succesul interacțiunii depinde în principal de aceasta.

Importanța modelului de dialog în cadrul sistemului impune existența unor criterii de evaluare și pentru acesta. Privind aceste criterii din punct de vedere al categoriei din care fac parte, am putea delimita următoarele categorii:

- Integrarea sistemului în domeniul pentru care a fost proiectat;
- Factorii umani care influențează comportarea sistemului;
- Aspecte ergonomice;
- Interacțiunea om–mașină.

Prioritatea principiilor proiectării dialogului față de alte componente și importanța dialogului în interacțiunea cu utilizatorul sugerează mai multe criterii posibile pentru evaluarea componentei de dialog [13]:

- **Flexibilitate.** Utilizatorul trebuie să aibă posibilitatea de a alege modalitatea de rezolvare a sarcinilor.
- **Controlabilitate.** Utilizatorul să poată controla interacțiunea astfel încât sistemul să nu ajungă în stări nedorite.
- **Feedback.** Sistemul să informeze utilizatorul despre acțiunile pe care le-a inițiat.
- **Transparență.** Utilizatorul să obțină rezultatele dorite fără să cunoască detalii ale structurii sistemului.
- **Robustețe.** Un sistem de dialog nu trebuie niciodată să ajungă în stări nedefinite.
- **Compatibilitate.** Secvențele dialogului să se desfășoare în conformitate cu cunoștințele utilizatorului despre sarcinile pe care dorește să le execute.

La limită, componenta de dialog poate fi văzută ca un scenariu al acțiunilor sistemului [14]. Scenariul concentrându-se asupra utilizării sistemului, proiectarea managerului dialogului unui sistem *task-oriented* este mult mai facilă sub această formă.

2.3 Metoda Vrăjitorului din Oz

În povestirea “Vrăjitorul din Oz” (L. Frank Baum, 1900) [1], se arată cum în spatele unui atotputernic vrăjitor se ascundea un om. La fel, în metoda Vrăjitorului din Oz, un om interpretează rolul calculatorului în dialogul cu alți oameni.

Vrăjitorul din Oz (Wizard of Oz — WOZ) este o metodă experimentală de prototipizare/simulare în care o persoană (vrăjitor) simulează în întregime sau parțial modelul de interacțiune al sistemului de dialog în curs de dezvoltare, purtând dialogul cu utilizatorul ce este convins că interacționează cu un sistem real [7].

2.3.1 Necesitatea Vrăjitorului din Oz

Această metodă a apărut ca necesară în urma observațiilor că oamenii se comportă diferit în dialogul cu un calculator, față de un dialog purtat cu alți oameni. S-a constatat că, în timp ce calculatorul interacționează normal din punct de vedere al dialogului, utilizatorii umani tind să simplifice frazele rostite, acceptând în același timp și un răspuns simplificat din partea calculatorului. Cea mai bună metodă de a analiza comportamentul uman în dialogul om–mașină este de a face observații asupra unui asemenea dialog real; este dificil, dacă nu imposibil, să se transpună cercetările asupra dialogului inter–uman în domeniul dialogului cu sisteme automate.

În acest context, scopul prototipizării prin metoda Vrăjitorului din Oz este de a colecta informații despre comportamentul utilizatorilor cu scopul de a evalua modelul de interacțiune al sistemului, despre fiabilitatea modelului dialogului, și nu în ultimul rând, de a colecta o bază de text vorbit, utilă în implementarea ulterioară a părților de recunoaștere a vorbirii. Din punct de vedere al costurilor și timpului necesar implementării,

metoda este foarte eficientă. Ultimii ani au adus o dezvoltare rapidă a sistemelor de dialog, iar pentru ca aceste sisteme să devină cât mai fiabile și performante, este necesară o cantitate considerabilă de text scris și vorbit. O simulare a unui sistem de dialog prin metoda Vrăjitorului din Oz este utilă (de fapt, singura soluție puțin costisitoare) chiar dacă singurul scop este colectarea acestor date.

2.3.2 Modalități de comunicare

Ca în orice sistem de dialog, și în metoda Vrăjitorului din Oz pot apare combinații ale diverselor modalități de comunicare: vorbire, scriere, gesturi. Excepție fac gesturile pe partea de Vrăjitor — ar fi într-adevăr ciudat ca un om să simuleze prin intermediul calculatorului gesturi umane — deși cercetările în domeniul realității virtuale poate vor contrazice această excepție [15]. O clasificare generală pe baza acestor combinații este dată în Figura 2.1. Mediul de dezvoltare prezentat pe larg în Capitolul 3 este reprezentat hașurat.

2.3.3 Variabilele metodei

În cadrul dezvoltării unui sistem de dialog folosind metoda Vrăjitorului din Oz, apare un set de variabile. Prin variabilele metodei înțelegem anumiți factori a căror valoare poate varia și pe care proiectantul trebuie să îi aibă în vedere în fazele preliminare ale proiectării sistemului. Variabilele pot fi clasificate în trei grupe: Utilizator, Vrăjitor și de Comunicație [16, 15].

Variabile Utilizator

- **Variabile de recunoaștere a vorbirii:** se referă la capacitatea utilizatorului de a recunoaște textul produs de Vrăjitor.

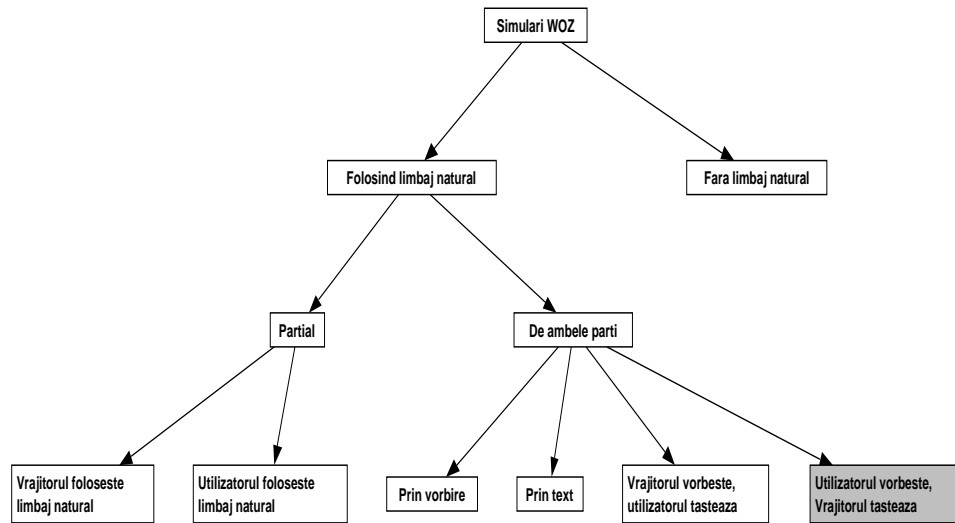


Figura 2.1: Taxonomia simulărilor prin metoda Vrajitorului din Oz

- *La nivel acustic*: determină dacă semnalul acustic este inteligibil pentru utilizator.
- *La nivel lexical*: cât de inteligibil este textul produs de Vrajitor ?
- **Variabile de producere a vorbirii**: sunt factori care pot afecta calitatea pronunțiilor utilizatorului.
 - *Accent*
 - *Calitatea vocii*
 - *Dialect*
- **Variabile dependente de cunoștințele utilizatorului**
 - *Cunoașterea domeniului*: cât de bine cunoaște utilizatorul domeniul pe care îl deservește sistemul; un domeniu mai puțin cunoscut ridică dificultăți în formularea cererilor.

- *Cunoașterea sistemului.*, un utilizator obișnuit cu un sistem de dialog formulează cereri mai eficiente.

Variabile Vrăjitor

- **Recunoaștere:** corespund variabilelor de producere a vorbirii pe partea de utilizator. Definesc posibilele erori în recunoașterea vorbirii, cauzate de fenomene ale limbajului natural și deficiențe tehnologice.
- **Producerea vorbirii.** Spre deosebire de utilizator, calitatea textului produs de vrăjitor poate fi influențată de factori diferiți:
 - *Timpul de reacție*, dependent de durata necesară formulării cererii pentru baza de date și de timpul necesar generării răspunsului. Este una din variabilele critice ale sistemului, dacă acest timp este mult mai mare decât timpul de răspuns al unui sistem real, încrederea utilizatorului scade; în acest caz vrăjitorul poate fi ajutat de un asistent.
- **Modelul dialogului.** Influențează modul în care vrăjitorul folosește managerul dialogului.
- **Pregătire.** Include variabile legate de experiența vrăjitorului în simularea unui sistem, existența unor instrumente care să-i ușureze sarcinile, personalitate și alți factori subiectivi.

Variabile de comunicație

Calitatea canalului de comunicație poate afecta funcționarea unui sistem de dialog. Variabilele din această categorie devin critice când se folosesc mijloace reale de comunicație (linie telefonică, difuzoare amplasate în locuri zgomotoase ș.a.)

- **Pe direcția utilizator → Vrăjitor.** În cazul unor simulări de laborator, pentru a crea condiții cât mai reale, se poate afecta cu ajutorul unui mixer calitatea semnalului de la utilizator. Într-un mediu real, zgomotele pe acest canal pot fi mari, putând influența recunoașterea vorbirii.
- **Pe direcția Vrăjitor → utilizator.** Un sistem real nu poate “vorbi” la fel de natural ca un om, deci textul produs de vrăjitor trebuie să respecte acest criteriu, atât din punct de vedere al vorbirii propriu-zise (intonație, accent) cât și din punctul de vedere al limbajului (trebuie să aibă o anumită rigiditate a exprimării).
- **Interacțiunea canalelor.** Este preferabilă o modalitate de comunicare full-duplex pentru a putea asigura comunicarea simultană.

2.3.4 Erori tipice

Dezavantajul acestei metode constă în experiența necesară ”vrăjitorului”, factor important în a realiza o simulare fără ca utilizatorul să își dea seama că nu dialoghează cu un sistem real; în caz contrar erorile apărute pot sugera natura umană a sistemului. Dacă nu sunt în număr foarte mare, erorile sunt chiar utile, pentru a urmări reacția utilizatorilor în fața comportării neașteptate a sistemului. Erorile ce apar trebuie să se încadreze ca frecvență și ca tip în cele produse de un sistem real, determinate de partea de recunoaștere a vorbirii și de partea de înțelegere a vorbirii, precum și eșecuri în interogarea bazei de date. Ca termen de comparație, putem menționa ratele de eroare obținute în proiectul ”MASK Kiosk”: 16.2% la recunoaștere și 5.4% la înțelegere [17].

Capitolul 3

Mediul de dezvoltare

Scopul mediului de dezvoltare prin metoda Vrăjitorului din Oz (MDWOZ) este de a oferi proiectanților de sisteme de dialog interactive un instrument pentru a dezvolta un prototip, printr-o metodă cât mai rapidă și fără a intra, pe cât posibil, în detalii legate de implementare, putându-se astfel concentra asupra părților specifice sistemelor bazate pe vorbire.

Printre puținele sisteme de acest tip existente se numără “CSLU Rapid Prototyper“, dezvoltat de *Center for Spoken Language Understanding*, Oregon Graduate Institute, S.U.A. [9], dar acesta oferă un mediu mai limitat pentru dezvoltarea sistemelor de dialog în general, fiind mai mult un sistem de dialog configurabil.

La ora actuală, tehnica dezvoltării sistemelor de dialog prin metoda Vrăjitorului din Oz este foarte răspândită. Avantajele acestei tehnici constau în costurile reduse, comparativ cu dezvoltarea folosind o metodă prin prototipizare. De asemenea, este foarte importantă culegerea de date experimentale cât mai relevante pentru funcționarea ulterioară a sistemului. Asemenea date nu pot fi colectate prin simulări în laborator sau din alte experimente desfășurate în condiții departe de realitate. Apare deci ca necesară colectarea datelor într-un mediu real. Față de un sistem ideal, de laborator, un sistem real se confruntă

cu multe probleme legate de calitatea semnalului vocal, calitate afectată de mediu, canalul de comunicație (linia telefonică prin lățimea de bandă redusă, dacă sistemul este destinat furnizării unor informații prin telefon), zgomot, faptul că un asemenea sistem trebuie să fie independent de vorbitor. Acești factori cresc rata de eroare pentru un sistem de dialog; de exemplu pentru sistemul automat de informații realizat de firma Philips rata de eroare la recunoaștere este de până la 25% [18]. Pe baza datelor culese dintr-un asemenea mediu real, se poate implementa partea de recunoaștere ținând cont de toate perturbațiile posibile.

În paragrafele ce urmează se vor prezenta considerații referitoare la proiectarea efectivă și implementarea mediului de dezvoltare, cât și utilizarea sa pentru un prim sistem de dialog. Sistemul ales pentru implementare (sistem pilot) își propune să ofere informații despre programarea examenelor din sesiune, într-o manieră *task-oriented* cu 4 schimbări de dialog utilizator-sistem, inițiativa aparținând sistemului.

3.1 Structura mediului de dezvoltare

Structura aleasă pentru mediul de dezvoltare prin metoda Vrăjitorului din Oz (MDWOZ) este prezentată în figura 3.1. MDWOZ este conceput modular, pentru a oferi posibilitatea utilizării doar a unor părți și pentru a putea fi transformat incremental din faza de simulare în produs final. Se disting patru module: Utilizator, Vrăjitor, Comunicație, Colectare Date. Modulul Vrăjitor cuprinde submodulele: Interfața cu Baza de Date (Interfața SQL), Managementul Dialogului (Descriere Dialog) și Interfața pentru Răspuns. Modulul de Comunicație este compus din Legătura Sunet și Rețea (pentru fluxul de control). Modulul de Colectare a Datelor înglobează Istoria Dialogului și partea de Redare/Înregistrare vorbire.

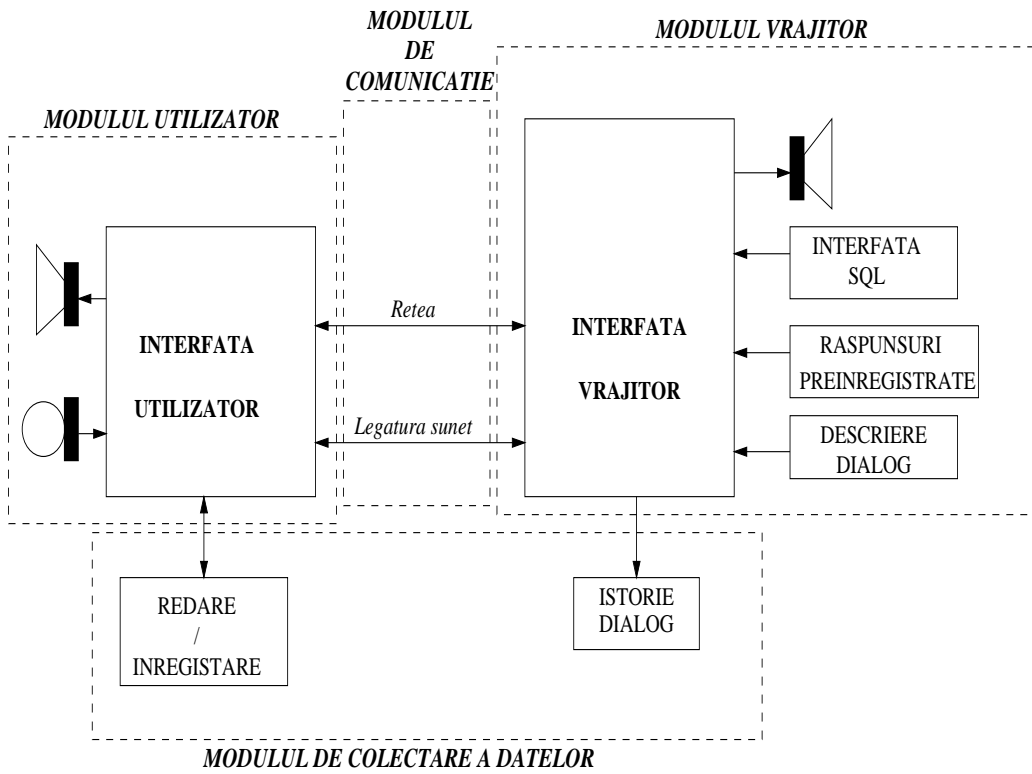


Figura 3.1: Structura mediului de dezvoltare

Din structura generală a unui sistem de dialog, prezentată în figura 1.1, părțile reprezentate hașurat nu au fost implementate în mediul de dezvoltare, în mod uzual simularea acestor componente căzând în sarcina vrăjitorului. Interfețele multimedia (hașura mai închisă) nu fac parte dintr-un sistem standard, iar Analizorul semantic (hașura mai deschisă) a fost implementat ca o componentă separată, nefiind inclus în mediul de dezvoltare (realizează o deducere a contextului pe baza textului rostit de utilizator, funcționează mai mult ca un sistem expert care asistă vrăjitorul, ajutându-l în alegerea stării următoare în graful de dialog).

3.1.1 Aspecte software

Din punct de vedere software, mediul de dezvoltare se prezintă sub forma unui pachet implementat în cea mai mare parte în limbajul C. În afara bibliotecilor standard aflate în distribuția Linux, au mai fost folosite: XForms v.0.86 pentru interfețele grafice sub mediul X Window [19], MySQL 9.29 pentru interfața cu SQL [20], daVinci pentru managementul dialogului [21].

Pachetul de programe realizat include unsprezece executabile și două fișiere de configurație (vezi Anexele A și B pentru listin-gul complet).

Executabilele sunt:

- **scenario** – este aplicația principală care gestionează dia-logul, folosind ca interfață grafică programul daVinci; in-clude și partea de comunicare prin rețea și interfața pentru generarea răspunsurilor, implementată folosind biblioteca XForms;
- **dbask** – este interfața cu baza de date, prezentată pe larg în secțiunea 3.2.1;

- **dbinput** – este folosit pentru introducerea datelor în baza de date, în caz că proiectantul sistemului nu dorește să folosească în mod direct limbajul SQL;
- **dbinit** – reinițializează baza de date, ștergând toate datele; este implementată sub forma unui script shell;
- **dbcreate** – crează o bază de date nouă; de asemenea, este implementată ca script shell;
- **wordplay** – redă un cuvânt din setul de cuvinte preînregistrate;
- **wordrec** – folosit la înregistrarea cuvintelor;
- **stereoplay** – program pentru redarea înregistrării efectuate pe durata unei sesiuni de lucru;
- **lexicon** – script shell care generează lista cuvintelor folosite în baza de date și în descrierea dialogului;
- **listword** – extrage din fișierul care memorează graful cuvintele folosite în descrierea dialogului.

Fișierele de configurație sunt:

- **woz.dialog** – conține descrierea fiecărui nod al grafului de dialog;
- **dbask.conf** – fișierul de configurație pentru baza de date; conține numele bazei de date, al utilizatorului SQL, numele tabelii, numărul și numele câmpurilor.

Diagramele de secvență ale MDWOZ sunt prezentate în Figura 3.2. Se desfășoară în paralel 3 procese pe 2 calculatoare. Părțile importante sunt buclele de interacțiune cu vrăjitorul și de interogare a bazei de date. Apar două etape de inițializare: sincronizarea prin rețea (pornirea înregistrării) și parcurgerea inițială a grafului de descriere a dialogului (necesară pentru a

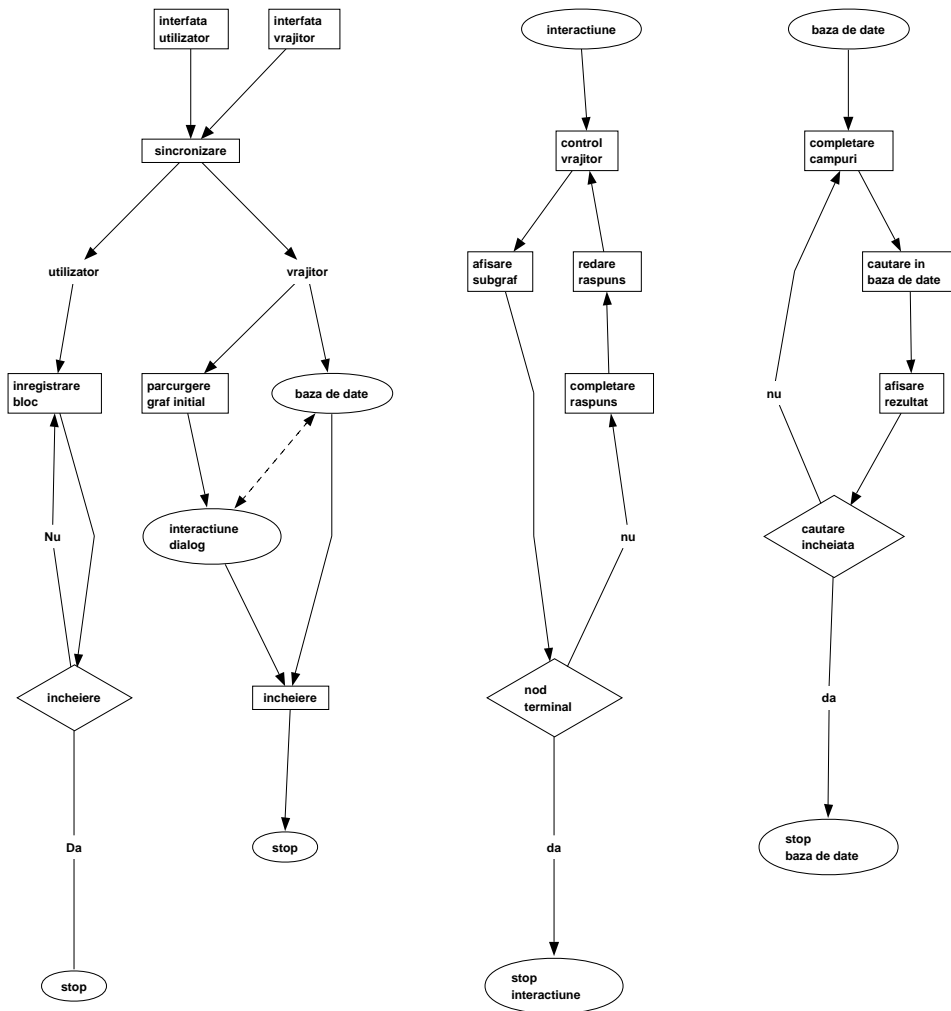


Figura 3.2: Diagramele de secvență ale programelor

converti graful din reprezentarea daVinci într-o reprezentare internă, sub forma unui arbore generalizat și a unor liste)

3.1.2 Utilitatea mediului de dezvoltare

În proiectarea acestui mediu de dezvoltare s-a urmărit în primul rând realizarea unui suport pentru cei ce dezvoltă diferite sisteme de dialog, care să îi scutească de efortul implementării: după cum s-a arătat în capitolele anterioare, nu există o metodologie în abordarea proiectării unui sistem de dialog, ambiguitățile crescând odată cu gradul de generalitate al sistemului.

MDWOZ trebuie să ofere un mediu nu doar pentru construirea sistemului de dialog, ci să asigure și cerințele de evaluare ale unui asemenea sistem (cum s-a arătat în 2.1) [13]. Fiabilitatea ca sistem general este limitată de implementare (vezi capitolul 5), iar robustețea dialogului depinde exclusiv de cel care configurează sistemul de dialog (este importantă includerea în graful ce reprezintă dialogul de noduri care, la fiecare interacțiune cu utilizatorul, să ofere posibilitatea unei opțiuni de întoarcere, pentru a relua faza anterioară). Flexibilitatea este asigurată atât prin modul de proiectare a dialogului (pentru a defini sistemul ca *user-oriented* sau *task-oriented*), cât și de configurarea bazei de date.

3.2 Interfața cu Vrăjitorul

Interfața cu Vrăjitorul este partea principală a mediului de dezvoltare MDWOZ. În unele situații, când complexitatea simulării este mare, o parte din părțile componente pot fi preluate de un asistent al vrăjitorului [7]. MDWOZ are prevăzut un singur operator, complexitatea în această fază nefiind foarte mare. În funcție de teste ulterioare, se va vedea dacă este necesară prezența unui asistent, pe baza întârzierii răspunsului dat de

sistem.

3.2.1 Interfața SQL

Interfața cu Baza de Date este implementată folosind limbajul specializat SQL, mai precis MySQL Ver. 9.29 Distrib. 3.22.18, varianta Linux [20]. Pentru a facilita un acces rapid la baza de date și a nu impune ca potențialii utilizatori ai MDWOZ să cunoască SQL, s-a proiectat o interfață grafică care folosește atât la construirea bazei de date, cât și la consultarea ei.

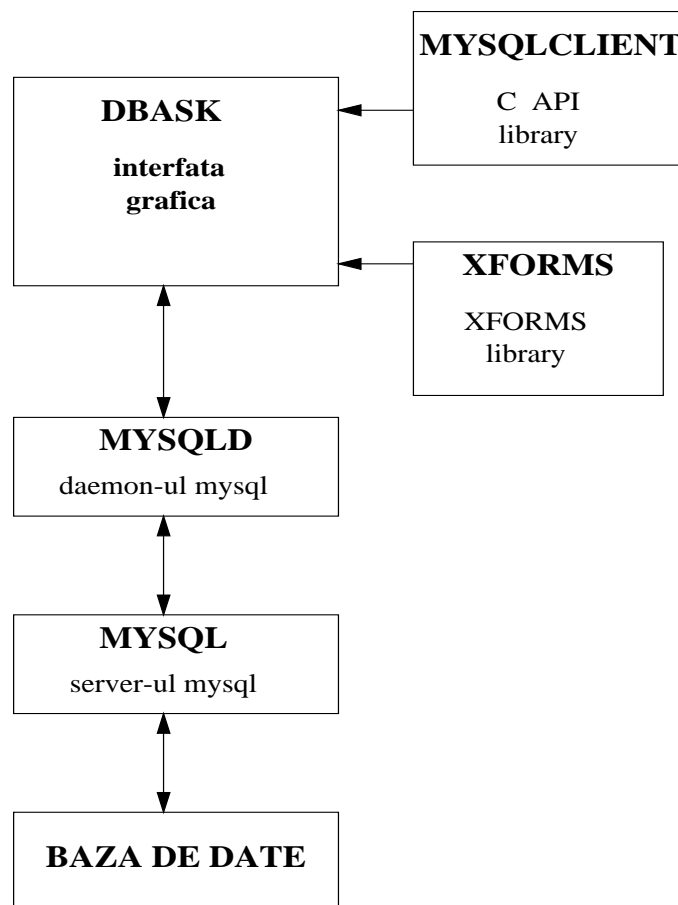


Figura 3.3: Implementarea interfeței cu baza de date

MySQL a fost preferat altor sisteme SQL (cum ar fi, de e-

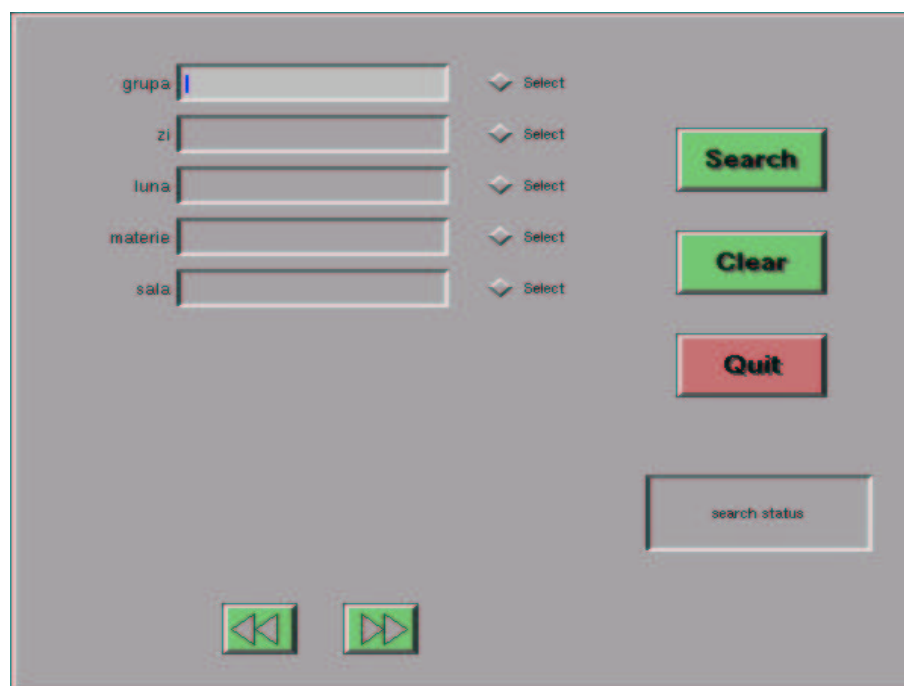


Figura 3.4: Interfața de acces la baza de date

xemplu, miniSQL, Postgres) datorită suportului pe care îl oferă pentru C, prin biblioteci care asigură accesul la mediul SQL [20]. Interogarea și accesul la rezultat se fac într-un mod facil, prin intermediul unor structuri predefinite. Modul cum interacționează interfața cu sistemul este prezentat în Figura 3.3. Din programul **dbask** pe baza structurilor specifice MySQL se realizează comunicarea cu daemon-ul **mysqld**. Mysqld este programul de tip client folosit de sistemul SQL pentru a oferi acces utilizatorilor. Accesul efectiv la înregistrările din baza de date este făcut de programul-server **mysql**. Pentru a asigura o interfață *user-friendly*, se folosesc funcții din biblioteca XForms.

Structura bazei de date este configurabilă, pentru ca sistemul să poată fi folosit în aplicații cât mai diverse. Interfața grafică pentru accesul la baza de date este prezentată în Figura 3.4.

Proiectantul sistemului de dialog alege numele și numărul

câmpurilor prin intermediul unui fișier de configurație, interfața urmând a afișa doar câmpurile necesare. Interogarea bazei de date se face printr-o cerere de tip *select* pe baza câmpurilor selectate. Se oferă posibilitatea afișării a până la patru rezultate și a vizualizării secvențiale a acestora. De asemenea, se pot rafina rezultatele obținute prin cereri suplimentare, modificând doar valoarea unor câmpuri și măbind numărul câmpurilor selectate.

3.2.2 Managementul dialogului

Pentru a modela dialogul se folosește o interfață intuitivă care reprezintă dialogul sub forma unui graf orientat, nodurile fiind stări ale sistemului în care se produc ieșiri (text vorbit) din partea utilizatorului sau al vrăjitorului. Pentru sistemul pilot folosit, graful complet este ilustrat în Figura 3.5. Vrăjitorul alege starea următoare a sistemului pe baza frazelor rostite de utilizator.

Este deosebit de important modul în care proiectanții viitorului sistem descriu dialogul, eventualele greșeli ducând la limitarea capacității de răspuns a sistemului. Pentru a simplifica proiectarea dialogului, se folosește mediul daVinci V2.1 [21].

În parcurgerea dialogului, daVinci este legat, folosind un pipe ca suport de comunicare inter-procese, de aplicația *scenario* care gestionează dialogul, permițând vrăjitorului să specifice starea următoare în mod interactiv (Figura 3.6 prezintă graful de dialog în mediul daVinci). În graf alternează textul produs de vrăjitor cu textul produs de utilizator. Nodurile simbolizate prin elipse reprezintă text produs de vrăjitor, celelalte posibilele intrări din partea utilizatorului (Figura 3.5). În starea inițială, se prezintă întreg graful, fiecare stare având asociat un cod dat de proiectantul sistemului. Codificărilor le corespund într-un fișier *woz.dialog* textul pe care îl reproduce sistemul corespunzător stării, sau o descriere a posibilelor cereri/întrebări ale uti-

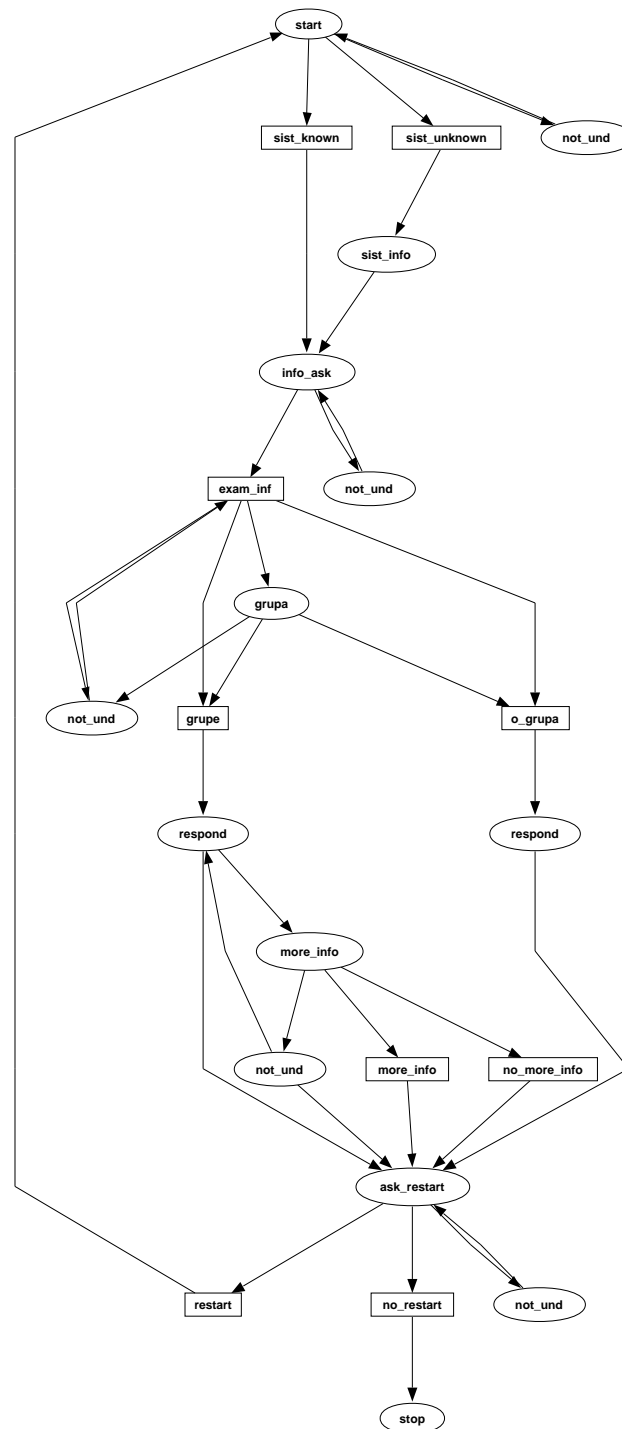


Figura 3.5: Modelarea dialogului – exemplu

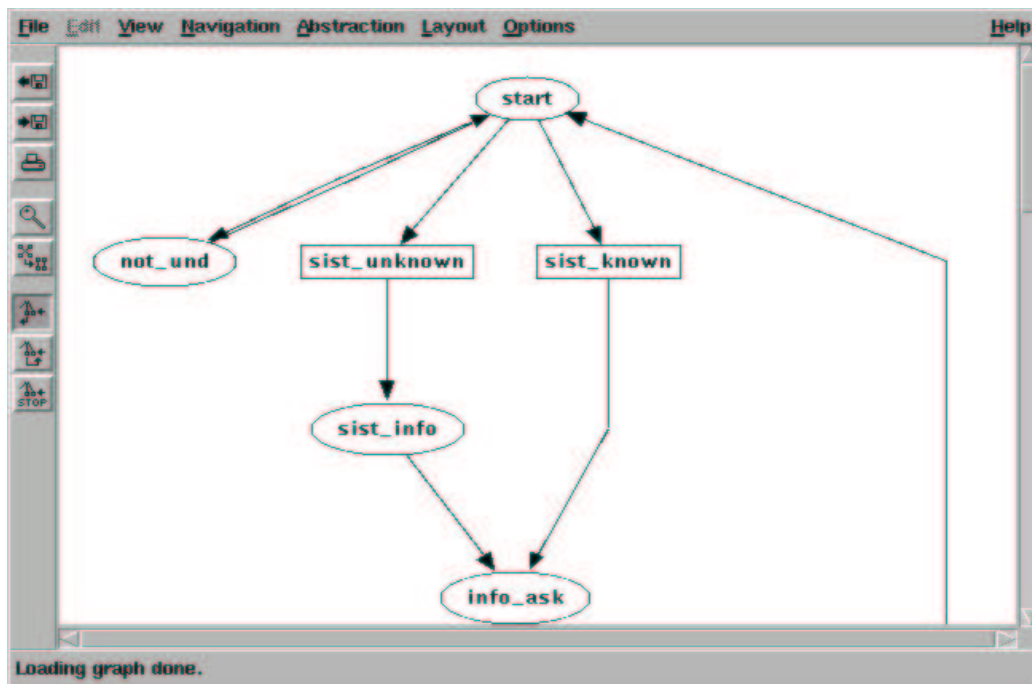


Figura 3.6: Mediul daVinci

lizadorului (pentru modul de completare a informațiilor din acest fișier vezi Anexa B); în continuare vor fi afișate doar starea curentă și stările succesoare, etichetele conținând codul stării fiind înlocuite cu textul respectiv din fișier. După analizarea cererii, sistemul trebuie să producă un răspuns sau o întrebare, depinzând de cât de completă este cererea utilizatorului. Textului corespunzător îi trebuie, dacă este cazul, adăugate rezultate obținute în urma interogării bazei de date.

3.2.3 Interfața pentru răspuns

În general, după cum am arătat în paragraful 1.2, răspunsul sistemului poate fi sintetizat prin conversie TTS (text-to-speech) sau din cuvinte preînregistrate. Interfața pentru răspuns folosește cuvinte preînregistrate; în funcție de starea următoare aleasă se generează un tipar de răspuns, vrăjitorul, dacă este necesar,

completând acest tipar cu date rezultate în urma consultării bazei de date, așa cum s-a arătat anterior.

Deoarece textul poate conține și numere, are loc o *conversie* a numerelor în cuvintele corespunzătoare; înainte de realizarea conversiei, textul este *normalizat* (separatorul “:” dintre ore și minute este convertit în “și”, iar virgula zecimală în cuvântul “virgulă”). După conversie și normalizare, fiecare cuvânt separat din text poate fi redat, ignorându-se semnele de punctuație, folosind înregistrarea respectivului cuvânt.

Textul astfel obținut trebuie transformat în semnal vocal, pe baza înregistrărilor existente. Înregistrările sunt la nivel de cuvânt și se fac separat. Pentru a nu lipsi cuvinte, se generează o listă a tuturor cuvintelor folosite, în modulul de dialog (graf) și în baza de date. Pentru numere se înregistrează unitățile componente (toate cifrele și numerele până la 20, ambele genuri pentru 1 și 2, apoi multiplicatorii: zeci, sută, sute, mie, mii, milion, milioane).

Pentru a da o mai mare credibilitate sistemului, după înregistrare se pot aplica diverse efecte, cum ar fi modificare frecvenței fundamentale, pentru a elimina inflexiunile vocii. O voce “metalică” și monotonă poate convinge un utilizator că sistemul chiar “vorbește”, pe când o voce umană, chiar dacă reprezintă o limită de performanță a sintezei vorbirii artificiale, trezește “suspiciuni” [7].

3.3 Interfața cu utilizatorul

Interfața cu utilizatorul rulează pe un calculator aflat de preferabil în altă încăpere sau măcar izolat de calculatorul vrăjitorului, pentru a păstra impresia de sistem real. Comunicarea utilizatorului cu calculatorul este doar prin voce, fără a fi nevoie ca utilizatorul să folosească și alte mijloace; pe ecran se afișează doar informații generale de utilizare (vezi Capitolul 4 pentru

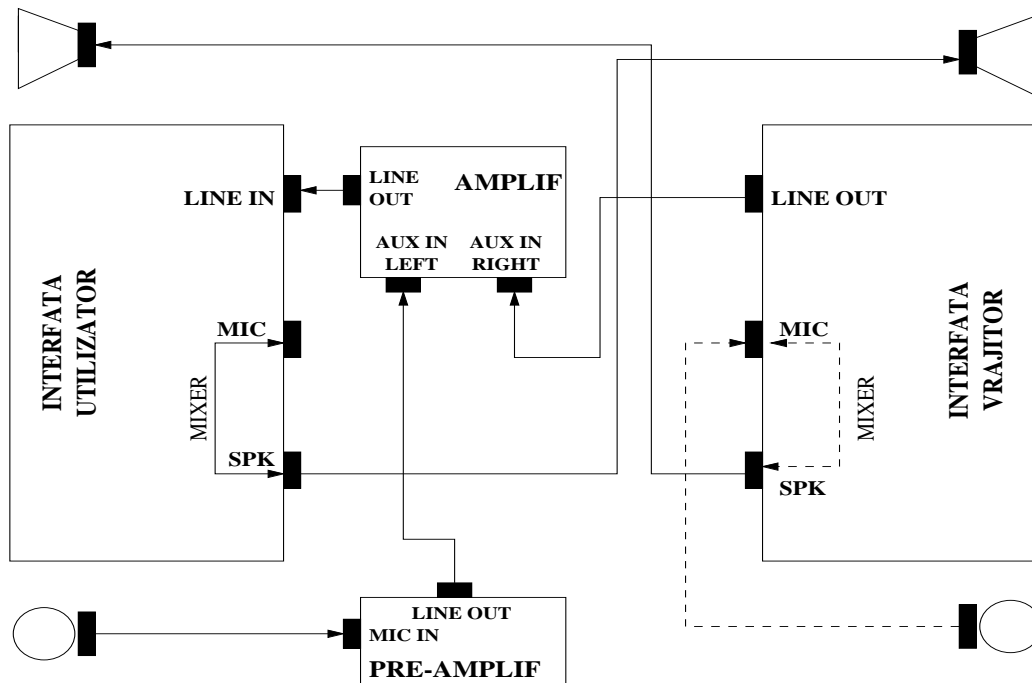


Figura 3.7: Sistemul de comunicație

variante experimentale de comunicare). La inițializare se setează placa de sunet la parametrii doriți, alte operații nemaifiind necesare. Se poate afișa și textul produs de sistem, concomitent cu redarea lui la nivel audio (vezi paragrafele 3.4 și 3.5).

3.4 Sistemul de comunicație

Sistemul de comunicație are două părți principale: Legătura de sunet și Legătura de control. Calea de sunet transmite semnalul sonor între cele două calculatoare, folosind o legătură fizică prin cablu (Figura 3.7), făcând uz de un mixer pentru preamplificarea semnalului produs de microfon și unificarea celor două canale mono provenite de la utilizator și vrăjitor într-un canal stereo. Opțional, peste legătura de control poate exista o legătură de date, pentru a afișa în interfața utilizator textul produs de

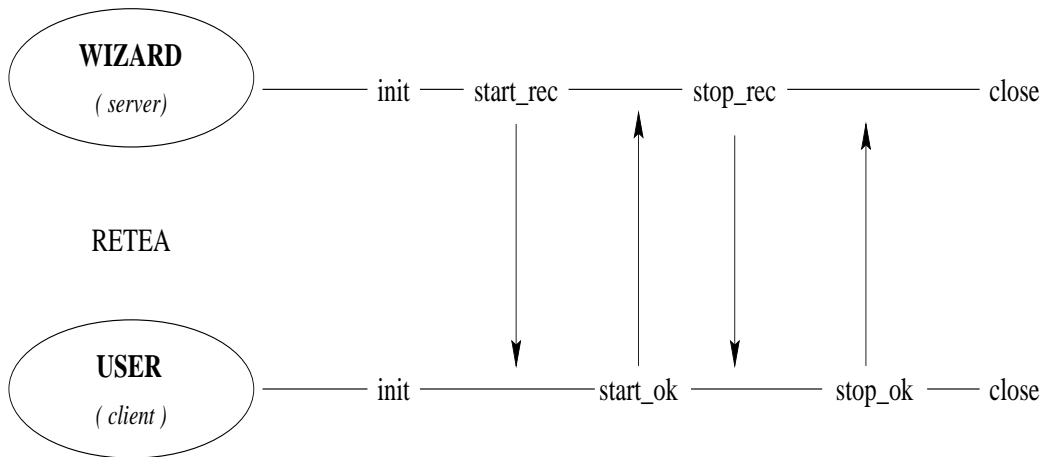


Figura 3.8: Fluxul de control prin rețea

vrăjitor (vezi capitolul 4).

Pentru a putea sincroniza începutul și sfârșitul înregistrării este nevoie și de o comunicare de control, realizată prin socket-uri ca interfață de comunicare prin rețea, în regim client-server. Pe partea de utilizator, se așteaptă pornirea programului care gestionează interfața vrăjitor, după care are loc sincronizarea proceselor, sub forma unui dialog de control (Figura 3.8). Programul utilizator așteaptă un cod de terminare de la server, pentru a opri înregistrarea.

3.5 Colectarea datelor

Împreună cu dezvoltarea modelului de interacțiune, colectarea datelor este unul din motivele care impun folosirea metodei Vrajitorului din Oz și pentru care a fost creat acest mediu de dezvoltare.

Se colectează date atât la nivel audio (înregistrarea ieșirilor la nivel utilizator și vrăjitor) cât și date despre evoluția dialogului, sub forma textelor produse de vrăjitor. Ulterior, textul și semnalul audio vor fi prelucrate folosind instrumente specifice, din

Tabelul 3.1: Parametrii înregistrării

Frecvența de eșantionare	Biți/Eșantion	Număr canale	Format
16000 Hz	16	2	Raw

semnal urmând a fi extras zgomotul și ecoul, iar textul folosit la etichetarea manuală sau automată a semnalului.

La nivel audio are loc înregistrarea semnalului în format raw-audio, parametrii înregistrării fiind prezentați în Tabelul 3.1.

Colectarea datelor are loc pe ambele calculatoare: la utilizator se face înregistrarea audio, iar la vrăjitor se înregistrează istoria dialogului.

Istoria dialogului constă din două fișiere (*log files*), unul pentru textul produs de vrăjitor (*sentences.history*), iar altul pentru memorarea drumului parcurs în graful de descriere a dialogului (*nodes.history*) cu scopul de a stabili eficiența proiectării dialogului. Folosind graful se poate identifica și momentul (nodul) în care a fost generată o anumită propoziție.

Capitolul 4

Experimente

Dezvoltarea unui sistem de dialog folosind metoda Vrăjitorului din Oz are trei etape [16]:

- etapa pre-experimentală, în care se definesc structurile de dialog folosite și constrângerile impuse de vocabular, gramatică și complexitatea aplicației;
- etapa experimentală primară, în care nu se aplică toate restricțiile, folosită pentru o primă evaluare a sistemului; în această fază, nu este necesar ca utilizatorii să creadă că sistemul este real, rolul ei fiind în principal acela de a pune la punct un prim model al interacțiunii;
- etapa experimentală secundară, în care are loc procesul iterativ simulare-evaluare-revizuire; tot acum are loc colectarea datelor necesare pentru antrenarea modelelor acustice și lingvistice folosite în recunoașterea vorbirii.

Pentru punerea la punct a unui mediu de dezvoltare, relevante sunt primele două faze, ultima urmând a fi parcursă doar pentru sisteme construite cu ajutorul lui.

Prima etapă a presupus stabilirea cerințelor funcționale și a structurii generale a mediului de dezvoltare, cu accent pe definirea unei metode generale de reprezentare a dialogului care

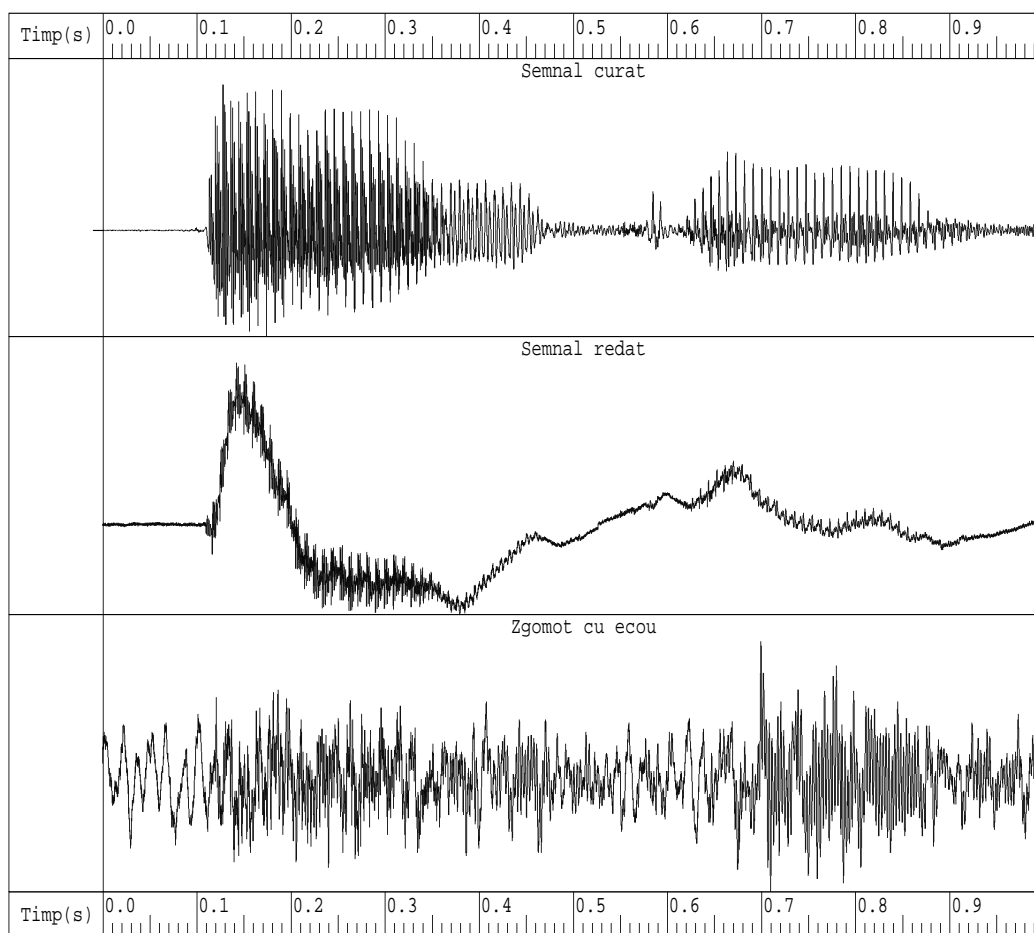


Figura 4.1: Ilustrare a deteriorării calității semnalului și ecoului

să poată fi utilizată pentru fiecare sistem dezvoltat cu MDWOZ (vezi 3.2.2).

În a doua etapă au fost testate funcționarea și modul de utilizare ale mediului pe cazul unui sistem pilot de mici dimensiuni. Concret, aplicația avută în vedere a fost un sistem de dialog pentru furnizarea de informații despre programarea examenelor anului IV în sesiunea iunie 1999, pentru care s-a construit off-line un prim model de interacțiune. Odată acest model disponibil și baza de date necesară creată, s-a generat vocabularul și s-au

realizat înregistrările necesare sintezei răspunsurilor sistemului. De asemenea, pentru a realiza un feed-back cât mai complet, legăturii de control (vezi 3.4) i s-a adăugat un canal de date prin care interfața utilizatorului afișează textul emis de vrăjitor.

Testarea sistemului s-a făcut considerând ca scenariu încercarea unui utilizator de a afla data și locul unui examen pentru o anumită disciplină și grupă. S-a urmărit în special funcționarea corectă a managerului de dialog și modul în care mediul poate fi utilizat pentru rafinarea modelului de interacțiune, dar au fost avute de asemeni în vedere aspecte legate de fiabilitatea software a componentelor și calitatea semnalelor colectate.

Simularea a decurs în condiții realiste, într-un mediu neizolat fonic și a pus în evidență problemele deosebite pe care un asemenea sistem le poate avea la nivel acustic. Pentru o comparație elocventă, în figura 4.1 se prezintă un semnal original folosit pentru sinteză, semnalul sintetizat așa cum a fost redat prin interfața utilizator și zgomotul peste care s-a suprapus ecoul, înregistrat de interfața utilizator: deși conținutul (cuvântul "pentru") este inteligibil în toate semnalele, variațiile foarte mari de calitate oferă un indiciu despre dificultățile menționate.

Capitolul 5

Concluzii și continuări

În urma experienței proiectării mediului de dezvoltare și a testării lui, chiar pe cazul unui sistem pilot de mici dimensiuni cum a fost cel prezentat în capitolul 4, se pot trage următoarele concluzii:

- un asemenea mediu oferă un suport foarte necesar pentru modelarea dialogului și punerea la punct a modelelor de interacțiune pentru sisteme de dialog; această concluzie se bazează pe experiența testării mediului pe cazul sistemului pilot, care a pus în evidență limitele severe ale unui model de interacțiune generat off-line, pe baza unei presupuse expertize umane: deși oamenii sunt capabili de comportamente foarte complexe, formalizarea acestor comportamente nu este deloc simplă, iar observarea lor este esențială pentru o modelare cât mai bună;
- mediul se dovedește de asemeni o soluție ieftină și rapidă de colectare a datelor necesare pentru punerea la punct a unor module componente ale unui sistem de dialog, în primul rând cele destinate recunoașterii automate a vorbirii;
- o primă analiză foarte sumară a unora din aceste date pune în evidență dificultățile pe care aceste module le vor avea

de rezolvat și indică direcții de urmat pentru ca punerea lor la punct să se facă anticipând aceste dificultăți.

Dezvoltările ulterioare plecând de la acest sistem pot fi direcționate pe două căi: corectare actualelor disfuncționalități și îmbunătățiri ale sistemului.

Principalele disfuncționalități sunt legate de fiabilitatea sistemului. Deoarece în cadrul său interacționează componente extrem de diverse, în fazele de testare au apărut diverse erori. Cele mai multe sunt cauzate de pachetele soft folosite (mediul daVinci prezintă suficiente asemenea cazuri) sau de interacțiunea cu interfața grafică a mediului Xwindows (bibliotecile xforms). Acestea pot fi remediate (sperăm) odată cu apariția noilor versiuni ale bibliotecilor folosite.

Pe partea de îmbunătățiri, lista ar putea continua la nesfârșit, deoarece nu există un standard despre cum trebuie să arate și să se comporte un astfel de mediu. Putem însă spune ce ar mai trebui făcut pentru a asigura o bază mai puternică pentru dezvoltarea unor sisteme de dialog.

Punctul critic al oricărui sistem de dialog este managementul dialogului: pentru aceasta, ar trebui renunțat la utilizarea daVinci în design-ul dialogului, păstrându-l doar pentru afișare, și trecut la o implementare pe baza unui limbaj de descriere; acest pas trebuie făcut în conjuncție cu punerea la punct a unor elemente ale unei gramatici computaționale a limbii române, pentru a facilita partea de înțelegere a limbajului natural. Gramatica, împreună cu descrierea dialogului, sunt părți esențiale în cadrul unui sistem de dialog, intrând în componența unui parser pe baza căruia se poate realiza înțelegerea pronunțiilor utilizatorilor.

Asemenea parsere, orientate pe aplicație, există în unele sisteme. De exemplu sistemul WAXHOLM, dezvoltat la Department of Speech Communication, KTH, Stockholm, Suedia, include un parser numit STINA (construit plecând de la TINA al

M.I.T.) [5] care combină elemente gramaticale cu unele probabiliste în analiza semantică.

Deși partea de înțelegere este extrem de importantă, pentru ca Vrăjitorul din Oz să simuleze cât mai bine un sistem real, adică subiecții experimentelor să nu realizeze că dialoghează cu un operator uman, răspunsurile preînregistrate se pot înlocui cu o componentă TTS (Text–To–Speech), mult mai realistă în cazul unui sistem de dialog. În acest sens se urmărește integrarea în mediu a unui asemenea sistem, dezvoltat în cadrul unor dizertații de studii aprofundate anterioare [22], [23].

Bibliografie

- [1] L. F. Baum. *The Wizard of Oz*. Wordsworth, Ware, Hertfordshire, UK, 1993.
- [2] P. Price. Evaluation of spoken language systems: the ATIS domain. In *Proceedings of Third ARPA Speech and Natural Language Workshop*, Hidden Valley, Pennsylvania, 1990.
- [3] R. Billi and L. Lamel. RAILTEL: Railway Telephone Services. In *Speech Communication*, volume 23, pages 63–65. 1997.
- [4] L. Lamel, S. Rosset, et al. The LIMSI ARISE system. In *Proceedings of IVITTA '98 – IEEE Workshop on Interactive Voice Technology for Telecommunications*, pages 209–214, Torino, Italia, 1998.
- [5] R. Carlson, S. Hunnicutt, and J. Gustafsson. Dialog management in the Waxholm system. In *Proceedings ESCA Tutorial and Research Workshop on Spoken Dialogue Systems*, Visgo, Danemarca, 1995.
- [6] S. Oviatt. Usability and Interface design. In *Survey of the State of the Art in Human Language Technology*, chapter 13: Evaluation. Cambridge University Press, 1998.
- [7] N. O. Bernsen, H. Dybkjaer, and L. Dybkjaer. *Designing Interactive Speech Systems*. Springer-Verlag, Londra, 1998.

- [8] L. Lamel. Spoken language dialog system development and evaluation at LIMSI. In *Proceedings of International Symposium on Spoken Dialogue*, Sydney, November 1998.
- [9] S. Sutton, D. G. Novick, R. Cole, P. Vermeulen, J. de Villiers, J. Schalkwyk, and M. Fauty. Building 10000 spoken dialogue system. In *Proceedings ICSLP'96*, Philadelphia, 1996.
- [10] E. Giachin. Spoken language dialogue. In *Survey of the State of the Art in Human Language Technology*, chapter 6: Discourse and Dialogue. Cambridge University Press, 1998.
- [11] T. Andernach, G. Deville, and L. Mortier. The design of a real world Wizard of Oz experiment for a speech driven telephone directory information system. In *Proceedings of EUROSPEECH'93*, Berlin, 1993.
- [12] N. O. Bernsen, L. Dybkjaer, and H. Dybkjaer. A dedicated task-oriented dialogue theory in support of spoken language dialogue systems design. In *Proceedings of ICSLP'94*, Yokohama, 1994.
- [13] C. Müller and F. Runge. Dialogue design principles – key for usability of voice processing. In *Proceedings of EUROSPEECH'93*, Berlin, 1993.
- [14] M. Jarke. Scenarios for modeling. *Communications of the ACM*, 42(1):47–48, January 1999.
- [15] N. M. Fraser and G. N. Gilbert. Simulating speech systems. *Computer Speech and Language*, 5:81–99, 1991.
- [16] D. Gibbon, R. Moore, and R. Winski. *Handbook of Standards and Resources for Spoken Language Systems*. Mouton de Gruyter, New York, 1997.

- [17] A. Life, I. Salter, J. N. Temem, F. Bernard, S. Rosset, S. Bennacef, and L. Lamel. Data collection for the MASK Kiosk: WOZ vs Prototype System. In *Proceedings IC-SLP'96*. Philadelphia, 1996.
- [18] H. Aust, M. Oerder, F. Seide, and V. Steinbiss. The Philips automatic train timetable information system. *Speech Communication*, 17, 1995.
- [19] T. C. Zhao and M. Overmars. Forms library, a graphical user interface toolkit for X. <http://bragg.phys.uwm.edu/xforms>, 1996.
- [20] *MySQL Reference Manual*. www.mysql.com, 1999.
- [21] *daVinci Online Documentation V2.1*. www.tzi.de/~davinci, 1998.
- [22] M. Pescaru. Prelucrări de texte pentru sinteza vorbirii din text în limba română. Dizertație de studii aprofundate, Departamentul de Calculatoare, Universitatea "Politehnica" Timișoara, iunie 1996.
- [23] T. Dumitrescu. Elemente pentru sinteza vorbirii în limba română. Dizertație de studii aprofundate, Departamentul de Calculatoare, Universitatea "Politehnica" Timișoara, iunie 1996.

Anexa A

Codul sursă al programelor

Fișierul **alphanum_play.h**

```
char * d2t(char *);  
char * sute(char *);  
char * mii(char *);  
char * milioane(char *);  
char * nr2text(char *);  
char * normal_text(const char *);  
extern void play_response(const char *);
```

Fișierul **alphanum_play.c**

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <linux/soundcard.h>
#include <unistd.h>
#include <fcntl.h>
#include "alphanum_play.h"

#define DSP_NAME "/dev/dsp"

int dspfd;

char * d2t(char * s)
{
    //digits to text 0..19
    char * digits[20]={"zero","unu","doi","trei","patru","cinci",
"sase","sapte","opt","noua","zece","unsprezece","doisprezece",
"treisprezece","paisprezece","cincisprezece","saisprezece",
"saptesprezece","optsprezece","nouasprezece"};
    return digits[(int)strtol(s,NULL,10)];
}

char * sute(char * s)
{
    int l,i;
    char * text;
    char temp[4];

    l=strlen(s);

    for (i=0;i<l;i++)
        if ((s[i]<'0') || (s[i]>'9'))
            {
                printf("Not a digit: %c\n",s[i]);
                exit(1);
            }
    text=(char *)malloc(1024);
    strcpy(text,"");
}
```

```
if (l==3)
{
    switch(s[0])
    {
        case '0':
            break;
        case '1':
            strcat(text,"o suta ");
            break;
        case '2':
            strcat(text,"doua sute ");
            break;
        default:
            strncpy(temp,s,1);
            strcat(text,d2t(temp));
            strcat(text," sute ");
            break;
    }

    switch(s[1])
    {
        case '0':
            break;
        case '1':
            strcat(text,d2t(s+1));
            break;
        case '2':
            strcat(text,"doua zeci ");
            break;
        case '6':
            strcat(text,"saizeci ");
            break;
        default :
            strncpy(temp,s+1,1);
            temp[1]='\0';
            strcat(text,d2t(temp));
            strcat(text," zeci ");
            break;
    }

    if ((s[1]!='1') && (s[2]!='0'))
```

```
        {
            if (s[1]!='0')
                strcat(text,"si ");
            strcat(text,d2t(s+2));
        }

    }
else
    if (l==2)
        {
            switch(s[0])
            {
                case '0':
                    break;
                case '1':
                    strcat(text,d2t(s));
                    break;
                case '2':
                    strcat(text,"doua zeci ");
                    break;
                case '6':
                    strcat(text,"saizeci ");
                    break;
                default :
                    strncpy(temp,s,1);
                    temp[1]='\0';
                    strcat(text,d2t(temp));
                    strcat(text," zeci ");
                    break;
            }

            if ((s[0]!='1') && (s[1]!='0'))
                {
                    if (s[0]!='0')
                        strcat(text,"si ");
                    strcat(text,d2t(s+1));
                }
        }
else

    return(d2t(s));
```

```
    return text;
}

char * mii(char * s)
{
    int l,i;
    char * text;

    l=strlen(s);
    for (i=0;i<l;i++)
        if ((s[i]<'0') || (s[i])>'9')
            {
                printf("Not a digit: %c\n",s[i]);
                exit(1);
            }
    text=(char *)malloc(1024);
    strcpy(text,"");
    if (l==3)
        {
            if ((s[0]=='0') && (s[1]=='0'))
                switch(s[2])
                {
                    case '0':
                        return "";
                        break;
                    case '1':
                        return "o mie";
                        break;
                    case '2':
                        return "doua mii";
                        break;
                    default:
                        strcpy(text,d2t(s+2));
                        strcat(text," mii");
                        return text;
                        break;
                }
            if (s[0]=='0')
                {
                    if (s[0]=='1')
```

```
        {
            strcpy(text,d2t(s+1));
            strcat(text, " mii");
        }
    else
    {
        strcpy(text,sute(s+1));
        strcat(text," de mii");
    }
    return text;
}
strcpy(text,sute(s));
if ((s[1]!='0')&&(s[1]!='1'))
    strcat(text," de");
strcat(text," mii");
return text;
}
if (l==2)
{
    if (s[0]=='0')
        switch(s[1])
        {
            case '0':
                return "";
                break;
            case '1':
                return "o mie";
                break;
            case '2':
                return "doua mii";
                break;
            default:
                strcpy(text,d2t(s+1));
                strcat(text," mii");
                return text;
                break;
        }
    if (s[0]=='1')
    {
        strcpy(text,d2t(s));
        strcat(text," mii");
    }
}
```

```
        return text;
    }
    strcpy(text,sute(s));
    strcat(text," de mii");
    return text;
}

switch(s[0])
{
    case '0':
        return "";
        break;
    case '1':
        return "o mie";
        break;
    case '2':
        return "doua mii";
        break;
    default:
        strcpy(text,d2t(s));
        strcat(text," mii");
        return text;
        break;
}
}

char * milioane(char * s)
{
    int l,i;
    char * text;

    l=strlen(s);
    for (i=0;i<l;i++)
        if ((s[i]<'0') || (s[i]>'9'))
            {
                printf("Not a digit: %c\n",s[i]);
                exit(1);
            }
    text=(char *)malloc(1024);
    strcpy(text,"");
    if (l==3)
```

```
{
  if ((s[0]=='0') && (s[1]=='0'))
    switch(s[2])
    {
      case '0':
        return "";
        break;
      case '1':
        return "un milion";
        break;
      case '2':
        return "doua milioane";
        break;
      default:
        strcpy(text,d2t(s+2));
        strcat(text," milioane");
        return text;
        break;
    }
  if (s[0]=='0')
  {
    if (s[0]=='1')
    {
      strcpy(text,d2t(s+1));
      strcat(text, " milioane");
    }
    else
    {
      strcpy(text,sute(s+1));
      strcat(text," de milioane");
    }
    return text;
  }
  strcpy(text,sute(s));
  if ((s[1]!='0')&&(s[1]!='1'))
    strcat(text," de");
  strcat(text," milioane");
  return text;
}
if (l==2)
{
```

```
    if (s[0]=='0')
        switch(s[1])
        {
            case '0':
                return "";
                break;
            case '1':
                return "un milion";
                break;
            case '2':
                return "doua milioane";
                break;
            default:
                strcpy(text,d2t(s+1));
                strcat(text," milioane");
                return text;
                break;
        }
    if (s[0]=='1')
    {
        strcpy(text,d2t(s));
        strcat(text," milioane");
        return text;
    }
    strcpy(text,sute(s));
    strcat(text," de milioane");
    return text;
}

switch(s[0])
{
    case '0':
        return "";
        break;
    case '1':
        return "un milion";
        break;
    case '2':
        return "doua milioane";
        break;
    default:
```

```
        strcpy(text,d2t(s));
        strcat(text," milioane");
        return text;
        break;
    }
}

char * nr2text(char * s)
{
    char * text;
    char temp[4];
    int l;

    text=(char *)malloc(1024);
    l=strlen(s);
    if (l>9)
    {
        printf("Max. number of digits is 9\n");
        exit(1);
    }
    strcpy(text,"");
    if (l>6)
    {
        strncpy(temp,s,l-6);
        temp[l-6]='\0';
        strcat(text,milioane(temp));
        strcat(text," ");
        strncpy(temp,s+l-6,3);
        temp[3]='\0';
        strcat(text, mii(temp));
        strcat(text," ");
        strncpy(temp,s+l-3,3);
        temp[3]='\0';
        strcat(text,sute(temp));
    }
    if ((l>3) && (l<=6))
    {
        strncpy(temp,s,l-3);
        temp[l-3]='\0';
        strcat(text,mii(temp));
        strcat(text," ");
    }
}
```

```

        strncpy(temp,s+1-3,3);
        temp[3]='\0';
        strcat(text,sute(temp));
    }
    if (l<=3)
        strcat(text,sute(s));
    return text;
}

char * normal_text(const char * s)
{
    char * text;
    int i,j=0,k;
    char last;
    char *t[80];
    char * rez;

    for (i=0;i<80;i++)
        t[i]=(char *)malloc(256);

    text=(char *)malloc(2*strlen(s));
    rez=(char *)malloc(4*strlen(s));

    for (i=0;i<strlen(s);i++)
    {
        if (s[i]==':')
            if ((i>0)&&(i<strlen(s)-1)&&(isdigit(s[i-1]))&&(isdigit(s[i+1])))
            {
                strcat(text+j," si ");
                j+=4;
            }
            else
                text[j++]=' ';
        else
            if (s[i]=='.')
                if ((i>0)&&(i<strlen(s)-1)&&(isdigit(s[i-1]))&&(isdigit(s[i+1])))
                {
                    strcat(text+j," virgula ");
                    j+=9;
                }
                else

```

```
        text[j++]=' ';
    else
        if (!isalnum(s[i]))
            text[j++]=' ';
        else
            text[j++]=s[i];
    }
text[j]='\0';

//tokenize
j=0;
k=0;
last=text[0];
for (i=0;i<strlen(text);i++)
    {
        if (text[i]==' ')
            if (last!=' ')
                {
                    t[j++][k]='\0';
                    k=0;
                    last=text[i];
                }
            else
                {
                }
        else
            {
                t[j][k++]=text[i];
                last=text[i];
            }
    }
if (last!=' ')
    t[j++][k]='\0';

strcpy(rez,"");
for (i=0;i<j;i++)
    {
        if(isdigit(t[i][0]))
            strcat(rez,nr2text(t[i]));
        else
            strcat(rez,t[i]);
    }
```

```
        if (i<j-1)
            strcat(rez," ");
    }
    for (i=0;i<80;i++)
        free(t[i]);

    return rez;
    // return text;
}

void word_play(char * s)
{
    char * REC_FILE;
    int l;
    char buf[256];
    FILE * f;

    REC_FILE=(char *)malloc(strlen(s)+5);
    strcpy(REC_FILE,s);
    strcat(REC_FILE,".rec");

    if ((f=fopen(REC_FILE,"r"))==NULL)
    {
        printf("Cant't open file %s\n",REC_FILE);
        return;
    }
    while((l=fread(buf,1,1,f))>0)
    {
        if (write(dspfd,buf,l)<0)
            perror("Playing error on /dev/dsp");
    }
    if (l==--1)
    {
        perror("Internal buffer error or file error");
        exit(-1);
    }
    fclose(f);
}
```

```
void play_response(const char * rasp)
{
    char * s;
    char * s1[40];
    FILE * logfile;
    FILE * socks;
    int param, original;
    int i,j=0,k=0;
    char last;
    int sockfd;
    int numar;
    char mesaj[1024];

    s=(char *)malloc(1024);
    for (i=0;i<40;i++)
        s1[i]=(char *) malloc(256);

    if((logfile=fopen("sentences.history","a"))==NULL)
        printf("Error on file : sentence.history. There will be
no session log\n");
    if ((dspfd=open(DSP_NAME, O_WRONLY))===-1)
    {
        perror("Can't open /dev/dsp");
    }
    //setting record parameters
    /*
    original=param=2;
    if (ioctl(dspfd, SOUND_PCM_WRITE_CHANNELS, &param)==-1)
        perror("Couldn't set parameters to /dev/dsp");
    if (param!=original)
        perror("Your soundcard doesn't accept stereo mode");
    */
    original=param=16;
    if (ioctl(dspfd, SOUND_PCM_WRITE_BITS, &param)==-1)
        perror("Couldn't set parameters to /dev/dsp");
    if (param!=original)
        perror("Your soundcard doesn't accept the required parameters");

    original=param=16000;
    if (ioctl(dspfd, SOUND_PCM_WRITE_RATE, &param)==-1)
```

```
    perror("Couldn't set parameters to /dev/dsp");
    if (param!=original)
        printf("Frequency of %d Hz not supported by your soundcard\nPlaying will
be at %dHz\n",original,param);
```

```
fprintf(logfile,"&&\n");
s=normal_text(rasp);
if ((socks=fopen("socket.used","r"))==NULL)
    {
        printf("Couldn't read temporary file\n");
        exit(1);
    }
fscanf(socks,"%d",&sockfd);
strcpy(mesaj,"woz_text");
if (send(sockfd,mesaj,9,0)<0)
    {
        printf("Can't write to socket\n");
        exit(1);
    }
numar=strlen(s)+1;
if (send(sockfd,&numar,2,0)<0)
    {
        printf("Can't write to socket\n");
        exit(1);
    }
if (send(sockfd,s,numar,0)<0)
    {
        printf("Can't write to socket\n");
        exit(1);
    }
```

```
//tokenize
last=s[0];
for (i=0;i<strlen(s);i++)
    {
        if (s[i]==' ')
            if (last!=' ')
                {
```

```
        s1[j++][k]='\0';
        k=0;
        last=s[i];
    }
    else
    {
    }
    else
    {
        s1[j][k++]=s[i];
        last=s[i];
    }
}
if (last!=' ')
    s1[j++][k]='\0';

for(i=0;i<j;i++)
{
    word_play(s1[i]);
    //          execlp("wordplay", "wordplay", s1, NULL);
    printf("%s\n", s1[i]);
    fprintf(logfile, "%s ", s1[i]);
}
fprintf(logfile, "\n"),
free(s);
for (i=0;i<40;i++)
    free(s1[i]);
fclose(logfile);
close(dspfd);
}
```

Fișierul **answer.h**

```
#include <string.h>
#include <stdlib.h>
#include "parsing.h"

char * node_text (nodestr *);
char * get_woz_label (nodestr *, int ,char **, int, int);
char * get_user_label (nodestr *);
```

Fișierul **answer.c**

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <forms.h>
#include "parsing.h"
#include "respond.h"
#include "raspunde.h"

#define WOZDIALOGFILE "woz.dialog"

int freadline(FILE * f, char * s)
{
    int i=0;

    while ((fread(s+i, 1, 1, f)>0)&&(s[i]!='\n'))
        i++;
    if (feof(f))
        return EOF;
    if (s[i]=='\n')
    {
        s[i]='\0';
        return 1;
    }
    else
        return -2;
}

char * node_text (nodestr * selected)
{
    //returneaza eticheta asociata nodului
    char * attr;
    char * text;
    int i;

    attr=(char *)malloc(strlen(selected->attr)+1);
    text=(char *)malloc(strlen(selected->attr)+1);

    strcpy(attr, selected->attr);
```

```

    text=strstr(attr,"OBJECT");
    text=text+9;
    text[(int)strcspn(text,"")-1]='\0';
    for (i=0;i<strlen(text);i++)
        if (text[i]=='\\')
            {
                text[i]=' ';
                text[++i]='\n';
            }
    return text;
}

char * get_woz_label(nodestr * node, int argc, char *argv[], int parent, int
first_node)
{
    //returneaza textul asociat unui nod
    FILE * f;
    char * s,*p;
    int i,j;
    char * text;
    char * attr;
    char * repl;
    char c;

    text=(char *)malloc(strlen(node->attr)+1);
    attr=(char *)malloc(strlen(node->attr)+1);
    s=(char *)malloc(1024);
    p=(char *)malloc(1024);
    repl=(char *)malloc(1024);

    text=node_text(node);
    printf("Get_woz, text: %s %d\n",text, strlen(text));
    if (!(f=fopen(WOZDIALOGFILE,"r")))
        {
            printf("Couldn't open %s\n",WOZDIALOGFILE);
            exit(1);
        }

    while( ((i=freadline(f,s))!=EOF) && ( (strcmp(s+1,text)!=0) || !((s[0]=='&')
|| (s[0]=='%')) ) ) )

```

```

    {
        printf("Scan: %s 0: %c\n",s,s[0]);
    }
    if (i==EOF)
    {
        printf("Eroare in fisierul %s sau la graf (verificati
label-urile)\n",WOZDIALOGFILE);
        exit(1);
    }
    c=s[0];
    strcpy(p,"");
    strcpy(s,"");
    while((p[0]!='%')&&(p[0]!='&'))
    {
        if (strcmp(p,"")!=0)
            strcat(s,p);
        if(freadline(f,p)<0)
        {
            printf("Eroare in fisierul %s\n",WOZDIALOGFILE);
            exit(1);
        }
        if (strcmp(s,"")!=0)
            strcat(s,"\\n");
    }
    if ((s[strlen(s)-2]=='\\') && (s[strlen(s)-1]=='n'))
        s[strlen(s)-2]='\0';
    printf("NextScan: %s\n",s);
    if ((c=='&') && parent)
        do_respond(s, argc, argv, first_node);
    strcpy(attr,node->attr);
    i=0;
    j=0;

    while(strncmp(attr+i,"OBJECT",6)!=0)
        i++;

    j=i;
    while(strncmp(attr+j,")",1)!=0)
        j++;
    strcpy(repl,attr);
    strcpy(repl+i+9,s);

```

```
    strcat(repl,attr+j-1);  
  
    fclose(f);  
    return repl;  
}
```

Fișierul **dbask.c**

```
#include <stdio.h>
#include <forms.h>
#include <stdlib.h>
#include "query.h"

void main(int argc, char * argv[])
{
    //initializare fereastră pt. consultare baza de date
    FD_sql * fsql;
    fsql=(FD_sql *)malloc(sizeof(FD_sql));
    fl_initialize(&argc, argv, "Sql", 0, 0);
    fsql_init();
    fsql=create_form_sql();
    fl_show_form(fsql->sql, FL_PLACE_CENTER, 10, "DBase Query");
    while(fl_do_forms()!=fsql->quit);
    fl_hide_form(fsql->sql);
}
```

Fișierul **dbinput.c**

```
#include <stdio.h>
#include <forms.h>
#include <stdlib.h>
#include <mysql.h>
#include "input.h"

void main(int argc, char * argv[])
{
    //initializare fereastră pt. actualizat baza de date
    FD_sql * fsql;
    fsql=(FD_sql *)malloc(sizeof(FD_sql));
    fl_initialize(&argc, argv, "Sql", 0, 0);
    fsql_init();
    fsql=create_form_sql();
    fl_show_form(fsql->sql, FL_PLACE_CENTER, 10, "DBase Input");
    while(fl_do_forms()!=fsql->quit);
    fl_hide_form(fsql->sql);
}
```

Fișierul **input.h**

```
#ifndef FD_sql_h_
#define FD_sql_h_
/* Header file generated with fdesign. */

/**** Callback routines ****/

extern void seteaza(FL_OBJECT *, long);
extern void sterge(FL_OBJECT *, long);
extern void left(FL_OBJECT *, long);
extern void right(FL_OBJECT *, long);
extern void modify(FL_OBJECT *, long);
extern void add(FL_OBJECT *, long);
extern void search(FL_OBJECT *, long);

/**** Forms and Objects ****/

typedef struct {
    FL_FORM *sql;
    void *vdata;
    long ldata;
    FL_OBJECT *cauta;
    FL_OBJECT *quit;
    FL_OBJECT *but_sterge;
    FL_OBJECT *bleft;
    FL_OBJECT *bright;
    FL_OBJECT *bmodify;
    FL_OBJECT *badd;
    FL_OBJECT *bsearch;
    FL_OBJECT *status_box;
    FL_OBJECT *i[6];
    FL_OBJECT *b[6];
} FD_sql;

extern FD_sql * create_form_sql(void);

#endif /* FD_sql_h_ */
```

Fișierul **input.c**

```
#include <mysql.h>
#include <forms.h>
#include <stdlib.h>
#include <string.h>
#include "input.h"
#define MAX 10 //number of fields
#define MAX_LENGTH 40 //field length
#define MAX_RES 4 //max of returned rows

//15 = max field name
char inp_label[MAX][15];
char inp_text[MAX][MAX_LENGTH];
char selected[MAX];
char result[MAX_RES][MAX][MAX_LENGTH];
int res_pos, res_max;
char q[(MAX_LENGTH+30+8)*MAX+15+13]; //query
int modif_pressed=0;

MYSQL * mys;
MYSQL_RES * mysres;
MYSQL_ROW mysrow;

char dbname[30], tblname[30], username[8];
int N=10;

void fsql_init(void)
{
    int i,j;
    char s[MAX_LENGTH];
    FILE * f;

    if ((f=fopen("dbask.conf", "r"))==NULL)
    {
        printf("Missing dbask.conf\n");
        exit(1);
    }
    if (fscanf(f, "DBNAME=%s\n", dbname)<=0)
    {
```

```
        printf("Error in dbask.conf\n");
        exit(1);
    }
    if (fscanf(f, "TBLNAME=%s\n", tblname) <= 0)
    {
        printf("Error in dbask.conf\n");
        exit(1);
    }
    if (fscanf(f, "USERNAME=%s\n", username) <= 0)
    {
        printf("Error in dbask.conf\n");
        exit(1);
    }
    if (fscanf(f, "FIELDS=%d\n", &N) <= 0)
    {
        printf("Error in dbask.conf\n");
        exit(1);
    }

    for(i=0; i<N; i++)
    {
        if (fscanf(f, "FLDNAME=%s\n", inp_label[i]) <= 0)
        {
            printf("Error in dbask.conf\n");
            exit(1);
        }
        selected[i]=0;
    }
    fclose(f);

    //porneste mysql si deschide baza de date

    mys=(MYSQL *)malloc(sizeof(MYSQL));
    mysres=(MYSQL_RES *)malloc(sizeof(MYSQL_RES));

    mysql_init(mys);
    if (!mysql_real_connect(mys, "localhost", username, NULL,
    dbname, 0, NULL, 0)){
        printf("SQL: Failed to connect to database\n");
        exit(1);
    }
}
```

```

}

FD_sql *create_form_sql(void)
{
    //create form
    FL_OBJECT *obj;
    FD_sql *fdui = (FD_sql *) fl_malloc(1, sizeof(*fdui));
    int j;

    fdui->sql = fl_bgn_form(FL_NO_BOX, 600, 530);
    fdui->cauta = obj = fl_add_box(FL_UP_BOX,0,0,600,530,"");

    for (j=0;j<N;j++)
    {
        fdui->i[j] = obj = fl_add_input(FL_NORMAL_INPUT,110,40+j*40,180,30,
inp_label[j]);
        fl_set_object_lsize(obj,FL_NORMAL_SIZE);
        fdui->b[j] = obj = fl_add_checkbutton(FL_PUSH_BUTTON,310,40+j*40,50,30,
"Select");
        fl_set_object_callback(obj,seteaza,j);
    }

    fdui->quit = obj = fl_add_button(FL_NORMAL_BUTTON,440,250,100,50,"Quit");
    fl_set_object_color(obj,FL_INDIANRED,FL_GREEN);
    fl_set_object_lsize(obj,FL_LARGE_SIZE);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE+FL_SHADOW_STYLE);
    fdui->but_sterge = obj = fl_add_button(FL_NORMAL_BUTTON,440,170,100,50,
"Clear");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lsize(obj,FL_LARGE_SIZE);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE+FL_SHADOW_STYLE);
    fl_set_object_callback(obj,sterge,(long)fdui);
    fdui->bleft = obj = fl_add_button(FL_NORMAL_BUTTON,140,460,50,40,"@<<");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lcolor(obj,FL_COL1);
    fl_set_object_callback(obj,left,(long)fdui);
    fdui->bright = obj = fl_add_button(FL_NORMAL_BUTTON,220,460,50,40,"@>>");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lcolor(obj,FL_COL1);
    fl_set_object_callback(obj,right,(long)fdui);

```

```

    fdui->bmodify = obj = fl_add_button(FL_NORMAL_BUTTON,510,460,50,
40,"MODIFY");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lcolor(obj,FL_BOTTOM_BCOL);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_object_callback(obj,modify,(long)fdui);
    fdui->badd = obj = fl_add_button(FL_NORMAL_BUTTON,430,460,50,40,
"ADD");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lcolor(obj,FL_BOTTOM_BCOL);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_object_callback(obj,add,(long)fdui);
    fdui->bsearch = obj = fl_add_button(FL_NORMAL_BUTTON,350,460,50,
40,"SEARCH");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lcolor(obj,FL_BOTTOM_BCOL);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_object_callback(obj,search,(long)fdui);
    fdui->status_box = obj = fl_add_box(FL_DOWN_BOX,420,360,150,60,
"search status");
    fl_end_form();
    fdui->sql->fdui = fdui;
    return fdui;
}
/*-----*/

```

```

void seteaza(FL_OBJECT * o, long p)
{
    //preseteaza butoanele "Select"
    if (selected[(int)p]==1)
        selected[(int)p]=0;
    else
        selected[(int)p]=1;
}

```

```

void search(FL_OBJECT * o, long p)
{
    //cautare in baza de date
    int i,j,k;
    char s[2];

```

```
strcpy(q,"select * from ");
strcat(q,tblname);
strcat(q," where ");

if (modif_pressed)
    return;
for (i=0;i<N;i++)
{
    if(selected[i])
    {
        strcpy(inp_text[i],fl_get_input(((FD_sql *)p)->i[i]));
        strcat(q,inp_label[i]);
        strcat(q,"=\");
        strcat(q,inp_text[i]);
        strcat(q,"\" and ");
    }
}
q[strlen(q)-5]='\0';
if(!mysql_query(mys,q))
{
    if(!(mysres=mysql_store_result(mys)))
    {
        printf("SQL: Returning Result Error\n");
        exit(1);
    }
    k=mysql_num_rows(mysres);
    if(k)
    {
        if (k>MAX_RES)
            k=MAX_RES;
        for (j=0;j<k;j++)
        {
            mysrow=mysql_fetch_row(mysres);
            if (mysql_num_fields(mysres)!=N)
                printf("SQL: unmatched number of fields\n");
            for (i=0;i<N;i++)
            {
                if (i<mysql_num_fields(mysres))
                    strcpy(result[j][i],mysrow[i]);
                else
```

```

        strcpy(result[j][i],"");
    }
}

for (i=0;i<N;i++)
    fl_set_input(((FD_sql *)p)->i[i],result[0][i]);
if (MAX_RES>=mysql_num_rows(mysres))
    fl_set_object_label(((FD_sql *)p)->status_box,"SQL:
Query OK");
else
{
    strcpy(q,"SQL: displayed ");
    sprintf(s,"%d",MAX_RES);
    strcat(q,s);
    strcat(q," of ");
    sprintf(s,"%d",mysql_num_rows(mysres));
    strcat(q,s);
    strcat(q," results");
    fl_set_object_label(((FD_sql *)p)->status_box,q);
} }
else
{
    fl_set_object_label(((FD_sql *)p)->status_box,"SQL: 0
results found");
}
}
else
{
    printf("SQL: Query Error\n");
    fl_set_object_label(((FD_sql *)p)->status_box,"SQL:
Query Error");
}
strcpy(q,"");
res_max=j-1;
res_pos=0;
}
void sterge(FL_OBJECT * o, long p)
{
    //reinitializeaza input-box-urile
    int i;
    for (i=0;i<N;i++)

```

```
    fl_set_input(((FD_sql *)p)->i[i], "");
}

void left(FL_OBJECT * o, long p)
{
    //afisare rezultat anterior
    int i;
    if (modif_pressed)
        return;
    if (res_pos==0)
        res_pos=res_max+1;

    res_pos--;
    for (i=0;i<N;i++)
    {
        fl_set_input(((FD_sql *)p)->i[i], result[res_pos][i]);
    }
}

void right(FL_OBJECT * o, long p)
{
    //afisare rezultat urmator
    int i;
    if (modif_pressed)
        return;
    if (res_pos==res_max)
        res_pos=-1;

    res_pos++;
    for (i=0;i<N;i++)
    {
        fl_set_input(((FD_sql *)p)->i[i], result[res_pos][i]);
    }
}

void add(FL_OBJECT * o, long p)
{
    //adauga o noua inregistrare
    int i,j;
    char s[2];
    if (modif_pressed)
```

```

    return;
    strcpy(q,"insert into ");
    strcat(q,tblname);
    strcat(q," (");
    for (i=0;i<N;i++)
    {
        strcat(q,inp_label[i]);
        strcat(q,",");
    }
    q[strlen(q)-1]='\0';
    strcat(q," values (");
    for (i=0;i<N;i++)
    {
        strcpy(inp_text[i],fl_get_input(((FD_sql *)p)->i[i]));
        strcat(q,"\"");
        strcat(q,inp_text[i]);
        strcat(q,"\",");
    }
    q[strlen(q)-1]='\0';
    strcat(q,")");
    if(mysql_query(mys,q))
    {
        printf("SQL: Query Error\n");
        fl_set_object_label(((FD_sql *)p)->status_box,"SQL:
Query Error");
    }
    else
    {
        strcpy(q,"SQL: Added ");
        sprintf(s,"%d",mysql_affected_rows(mys));
        strcat(q,s);
        strcat(q," rows");
        fl_set_object_label(((FD_sql *)p)->status_box,q);
    }
    strcpy(q,"");
}
void modify(FL_OBJECT * o, long p)
{
    int i,j;
    char s[2];
    char mq[(MAX_LENGTH+30+8)*MAX+15+13]; //modify query

```

```
if (modif_pressed)
{
    strcpy(mq,"update ");
    strcat(mq,tblname);
    strcat(mq," set ");
    for (i=0;i<N;i++)
    {
        strcpy(inp_text[i],fl_get_input(((FD_sql *)p)->i[i]));
        strcat(mq,inp_label[i]);
        strcat(mq,"=\"");
        strcat(mq,inp_text[i]);
        strcat(mq,"\", ");
    }
    mq[strlen(mq)-2]='\0';
    strcat(mq,q);
    if(mysql_query(mys,mq))
    {
        printf("SQL: Query Error\n");
        fl_set_object_label(((FD_sql *)p)->status_box,"SQL: Query Error");
    }
    else
    {
        strcpy(q,"SQL: Modified ");
        sprintf(s,"%d",mysql_affected_rows(mys));
        strcat(q,s);
        strcat(q," rows");
        fl_set_object_label(((FD_sql *)p)->status_box,q);
    }
    modif_pressed=0;
    fl_set_object_color(o,FL_PALEGREEN,FL_INDIANRED);
    strcpy(q,"");
}
else
{
    strcpy(q," where ");
    for (i=0;i<N;i++)
    {
        if(selected[i])
        {
            strcpy(inp_text[i],fl_get_input(((FD_sql *)p)->i[i]));
```

```
        strcat(q,inp_label[i]);
        strcat(q,"=\"");
        strcat(q,inp_text[i]);
        strcat(q, "\" and ");
    }
}
q[strlen(q)-5]='\0';
fl_set_object_color(o,FL_GREEN,FL_INDIANRED);
fl_set_object_label(((FD_sql *)p)->status_box,"Input now
the new values");
    modif_pressed=1;
}
}
```

Fișierul **listword.c**

```
#include <stdio.h>
#include <string.h>

void main(int argc, char ** argv)
{
    //citeste cate un cuvant din fisier
    FILE * f;
    char s[255];
    char c;

    if ((f=fopen(argv[1],"r"))==NULL)
        exit(1);
    while(fscanf(f,"%s ",s)!=EOF)
    {
        c=s[strlen(s)-1];
        if ((c==' ') || (c=='.') || (c=='!') || (c=='?') || (c==':') || (c==';')
|| (c=='*'))
            s[strlen(s)-1]='\0';
        if (strcmp(s,"")!=0)
            printf("%s\n",s);
    }
}
```

Fișierul `parsing.h`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define GRFILE "woz.daVinci"
#define MAXSIZE 65000

typedef struct
{
    char * nodeId;
    char * attr;
    struct edgestr * edges;//lista muchilor adiacente
    struct nodestr * nextnode;
} nodestr;
typedef struct
{
    char * edgeId;
    char * attr;
    char * noderef;
    struct nodestr * node; //nod fiu
    struct edgestr * nextedge; //pointer in lista tuturor muchiilor
    struct edgestr * nextlistedge; //pt. lista muchiilor
    //care ies dintr-un nod
} edgestr;
nodestr * process_node(char *);
edgestr * process_edge(char *);
edgestr * process_edgelist(char *);
extern nodestr * process_nodelist(char *);
extern nodestr * find_node(char *);
extern edgestr * link_references(void);
extern void afis_nod(nodestr *);
extern void afis_edge(edgestr *);
extern void tree(nodestr *);
```

Fișierul `parsing.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parsing.h"

nodestr * noduri;
nodestr * cn;
edgestr * muchii;
edgestr * ce;
//cn,ce - current node, edge

nodestr * process_node(char *);
edgestr * process_edge(char *);
edgestr * process_edgelist(char *);

void new_edge(edgestr * e,char * edgeid, char * attr)
{
    e=(edgestr *)malloc(sizeof(edgestr));
    e->edgeId=(char *)malloc(strlen(edgeid)+1);
    e->attr=(char *)malloc(strlen(attr)+1);
    strcpy(e->edgeId,edgeid);
    strcpy(e->attr,attr);
}

void new_node(nodestr * n, char * nodeid, char * attr)
{
    n=(nodestr *)malloc(sizeof(nodestr));
    n->nodeId=(char *)malloc(strlen(nodeid)+1);
    n->attr=(char *)malloc(strlen(attr)+1);
    strcpy(n->nodeId,nodeid);
    strcpy(n->attr,attr);
}

nodestr * process_nodelist(char * string)
{
    //initializeaza parcurgerea nodurilor
    //prin apelul process_node pe fiecare nod radacina
```

```

char subsir[strlen(string)];
int i=0,j=0,par=0;

if (!strcmp(string,"[]"))
    return;
while(string[i++]!='\0')
{
    subsir[j++]='1';
    i++;
    do{
        if (string[i]=='(')
            par++;
        if (string[i]==')')
            par--;
        subsir[j++]=string[i++];
    }while(par);
    subsir[j]='\0';
    process_node(subsir);
    j=0;
}
return noduri;
}

nodestr * process_node(char * string)
{
    //parcurge descendent arborele, transformad-ul
    //in reprezentarea interna

    char subs[strlen(string)];
    char nodeid[strlen(string)];
    nodestr * n;
    int i=2,j=0,par=1;

    //NodeId
    subs[j++]=string[i];
    while(string[++i]!='\0')
    {
        subs[j++]=string[i];
    }
    subs[j++]='\0';
    subs[j]='\0';
}

```

```
printf("Nod: %s ",subs);
strcpy(nodeid,subs);
i+=4;
j=0;
//Type
subs[j++]=string[i];
while(string[++i]!='')
{
    subs[j++]=string[i];
}
subs[j++]='';
subs[j]='\0';
printf("Tip: %s ",subs);
i+=2;
j=0;
//Attr
subs[j++]=string[i];
while(par)
{
    subs[j++]=string[++i];
    if (string[i]=='')
        par--;
    if (string[i]=='[')
        par++;
}
subs[j]='\0';
printf("Attr: %s ",subs);
i+=2;
j=0;
par=1;

n=(nodestr *)malloc(sizeof(nodestr));
n->nodeId=(char *)malloc(strlen(nodeid)+1);
n->attr=(char *)malloc(strlen(subs)+1);
strcpy(n->nodeId,nodeid);
strcpy(n->attr,subs);

if (!noduri)
{
```

```

        noduri=n;
        cn=noduri;
    }
else
    {
        cn->nextnode=(struct nodestr *)n;
        cn=n;
    }

//Edges
subs[j++]=string[i];

while(par)
    {
        subs[j++]=string[++i];
        if (string[i]=='')
            par--;
        if (string[i]=='[')
            par++;
    }
subs[j]='\0';
printf("Edges: %s ",subs);
printf("\n");
n->edges=(struct edgestr *)process_edgelist(subs);
return n;
}

edgestr * process_edgelist(char * string)
{
    //parcurge muchiile
    char subsir[strlen(string)];
    int i=0,j=0,par=0;
    edgestr * e=NULL, * prim_e;
    edgestr * r;

    if (!strcmp(string,"[]"))
        return NULL;

    while(string[i++]!='')

```

```
{
    subsir[j++]='1';
    i++;
    do{
        if (string[i]=='(')
            par++;
        if (string[i]==')')
            par--;
        subsir[j++]=string[i++];
    }while(par);
    subsir[j]='\0';

    if (!e)
    {
        e=process_edge(subsir);
        prim_e=e;
    }
    else
    {
        e->nextlistedge=(struct edgestr *)process_edge(subsir);
        if (!e->nextlistedge)
            return prim_e;
        e=(edgestr *)e->nextlistedge;
    }

    j=0;
}
return prim_e;
}

nodestr * find_node(char * s)
{
    //gazește un nod după identificatorul său
    nodestr * n;

    for(n=noduri;n->nextnode!=NULL;n=(nodestr *)n->nextnode)
        if(!strcmp(n->nodeId,s))
            return n;
    if(!strcmp(n->nodeId,s))
        return n;
    return NULL;
}
```

```
}

edgestr * process_edge(char * string)
{
    //parcurge muchiile ce ies dintr-un nod
    char subs[strlen(string)];
    char edgeid[strlen(string)];
    int i=2,j=0,par=1,ref=0;
    edgestr * e;

    if (strcmp(string,"[]")==0)
        return NULL;

    //EdgeId
    subs[j++]=string[i];
    while(string[++i]!='')
    {
        subs[j++]=string[i];
    }
    subs[j++]='';
    subs[j]='\0';
    printf("Edge: %s ",subs);
    i+=4;
    j=0;
    strcpy(edgeid,subs);
    //Type
    subs[j++]=string[i];
    while(string[++i]!='')
    {
        subs[j++]=string[i];
    }
    subs[j++]='';
    subs[j]='\0';
    printf("Tip: %s ",subs);
    i+=2;
    j=0;
    //Attr
    subs[j++]=string[i];
    while(par)
    {
        subs[j++]=string[++i];
    }
}
```

```
        if (string[i]=='')
            par--;
        if (string[i]=='(')
            par++;

    }
    subs[j]='\0';
    printf("Attr: %s ",subs);
    i+=2;
    j=0;
    par=1;

    e=(edgestr *)malloc(sizeof(edgestr));
    e->edgeId=(char *)malloc(strlen(edgeid)+1);
    e->attr=(char *)malloc(strlen(subs)+1);
    strcpy(e->edgeId,edgeid);
    strcpy(e->attr,subs);

//Node
e->noderef=(char *)malloc(20);
if(string[i]=='r')
    ref=1;
subs[j++]=string[i++];
subs[j++]=string[i];

while(par)
    {
        subs[j++]=string[++i];
        if (string[i]=='')
            par--;
        if (string[i]=='(')
            par++;
    }
    subs[j]='\0';
    printf("Nodes: %s ",subs);
    printf("\n");

if (!muchii)
    {
        muchii=e;
        ce=muchii;
```

```

    }
else
    {
        ce->nextedge=(struct edgestr *)e;
        ce=e;
    }

if(!ref)
    {
        e->node=(struct nodestr *)process_node(subs);
        strcpy(e->noderef,"1");
    }
else
    {
        subs[strlen(subs)-1]='\0';
        strcpy(subs,subs+2);
        strcpy(e->noderef,subs);
    }
printf("\n");
return e;
}

edgestr * link_references(void)
{
    //inlocuieste referintele la identificatorii
    //nodurilor deja definiti cu pointeri spre acestia
    edgestr * e;
    int i=0;
    for (e=muchii;e->nextedge!=NULL;e=(edgestr *)e->nextedge)
        if (e->noderef[0]!='1')
            e->node=(struct nodestr *)find_node(e->noderef);
    if (e->noderef[0]!='1')
        e->node=(struct nodestr *)find_node(e->noderef);
    return muchii;
}

void afis_nod(nodestr * n)
{
    //afiseaza nodul cu toti succesorii
    nodestr * n1;

```

```

    printf("\n\n");
    for (n1=n;n1->nextnode!=NULL;n1=(nodestr *)n1->nextnode)
        printf("%s ",n1->nodeId);
    printf("%s ",n1->nodeId);
    printf("\n");
}

void afis_edge(edgestr * n)
{
    //afiseaza muchia si muchiile urmatoare
    edgestr * n1;
    printf("\n\n");
    for (n1=n;n1->nextedge!=NULL;n1=(edgestr *)n1->nextedge)
        {
            printf("%s -> ",n1->edgeId);
            printf("%s \n",((nodestr *)n1->node)->nodeId);
        }
    printf("%s -> ",n1->edgeId);
    printf("%s \n",((nodestr *)n1->node)->nodeId);
    printf("\n\n");
}

void tree(nodestr * n)
{
    //parcurge arborele transformat in reprezentarea interna
    edgestr * e;
    printf("NodId: %s ",n->nodeId);
    printf("Attr: %s ",n->attr);
    if (!n->edges)
        {
            printf("\n");
            return;
        }
    printf("\n Edges:\n");
    for (e=(edgestr *)n->edges;e->nextlistedge!=NULL;e=(edgestr *)e->
nextlistedge)
        printf(" %s: -> Node: %s\n",e->edgeId,((nodestr *)e->node)->nodeId);
    printf(" %s: -> Node: %s\n",e->edgeId,((nodestr *)e->node)->nodeId);
    for (e=(edgestr *)n->edges;e->nextlistedge!=NULL;e=(edgestr *)e->
nextlistedge)

```

```
    if (e->noderef[0]=='1')
        tree((nodestr *)e->node);
if (e->noderef[0]=='1')
    tree((nodestr *)e->node);
}
```

Fișierul `query.h`

```
#ifndef FD_sql_h_
#define FD_sql_h_
/* Header file generated with fdesign. */

/**** Callback routines ****/

extern void seteaza(FL_OBJECT *, long);
extern void start_query(FL_OBJECT *, long);
extern void sterge(FL_OBJECT *, long);
extern void left(FL_OBJECT *, long);
extern void right(FL_OBJECT *, long);
extern void fsql_init(void);

/**** Forms and Objects ****/

typedef struct {
    FL_FORM *sql;
    void *vdata;
    long ldata;
    FL_OBJECT *cauta;
    FL_OBJECT *but_ok;
    FL_OBJECT *quit;
    FL_OBJECT *but_sterge;
    FL_OBJECT *bleft;
    FL_OBJECT *bright;
    FL_OBJECT *i[6];
    FL_OBJECT *b[6];
    FL_OBJECT *status_box;
} FD_sql;

extern FD_sql * create_form_sql(void);

#endif /* FD_sql_h_ */
```

Fișierul `query.c`

```
#include <forms.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include "query.h"
#define MAX 10 //number of fields
#define MAX_LENGTH 40 //field length
#define MAX_RES 4 //max of returned rows

//15 = max field nam
char inp_label[MAX][15];
char inp_text[MAX][MAX_LENGTH];
char selected[MAX];
char result[MAX_RES][MAX][MAX_LENGTH];
int res_pos, res_max;
char q[(MAX_LENGTH+30+8)*MAX+15+13]; //query

MYSQL * mys;
MYSQL_RES * mysres;
MYSQL_ROW mysrow;

char dbname[30],tblname[30], username[8];
int N=10;

void fsql_init(void)
{
    //citire fisier dbask.conf
    int i;
    char s[MAX_LENGTH];
    FILE * f;

    if ((f=fopen("dbask.conf","r"))==NULL)
    {
        printf("Missing dbask.conf\n");
        exit(1);
    }
    if (fscanf(f,"DBNAME=%s\n",dbname)<=0)
    {
```

```
        printf("Error in dbask.conf\n");
        exit(1);
    }
    if (fscanf(f, "TBLNAME=%s\n", tblname) <= 0)
    {
        printf("Error in dbask.conf\n");
        exit(1);
    }
    if (fscanf(f, "USERNAME=%s\n", username) <= 0)
    {
        printf("Error in dbask.conf\n");
        exit(1);
    }
    if (fscanf(f, "FIELDS=%d\n", &N) <= 0)
    {
        printf("Error in dbask.conf\n");
        exit(1);
    }

    for(i=0; i<N; i++)
    {
        if (fscanf(f, "FLDNAME=%s\n", inp_label[i]) <= 0)
        {
            printf("Error in dbask.conf\n");
            exit(1);
        }
        selected[i]=0;
    }
    fclose(f);

    //starting mysql and opening databases

    mys=(MYSQL *)malloc(sizeof(MYSQL));
    mysres=(MYSQL_RES *)malloc(sizeof(MYSQL_RES));

    mysql_init(mys);
    if (!mysql_real_connect(mys, "localhost", username, NULL, dbname, 0, NULL, 0))
    {
        printf("SQL: Failed to connect to database\n");
        exit(1);
    }
```

```

}

FD_sql *create_form_sql(void)
{
    //create form
    FL_OBJECT *obj;
    FD_sql *fdui = (FD_sql *) fl_malloc(1, sizeof(*fdui));
    int j;
    fdui->sql = fl_bgn_form(FL_NO_BOX, 600, 530);
    fdui->cauta = obj = fl_add_box(FL_UP_BOX,0,0,600,530,"");
    for (j=0;j<N;j++)
    {
        fdui->i[j] = obj = fl_add_input(FL_NORMAL_INPUT,110,
40+j*40,180,30,inp_label[j]);
        fl_set_object_lsize(obj,FL_NORMAL_SIZE);
        fdui->b[j] = obj = fl_add_checkbox(FL_PUSH_BUTTON,310,
40+j*40,50,30,"Select");
        fl_set_object_callback(obj,seteaza,j);
    }
    fdui->but_ok = obj = fl_add_button(FL_NORMAL_BUTTON,440,90,100,
50,"Search");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lsize(obj,FL_LARGE_SIZE);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE+FL_SHADOW_STYLE);
    fl_set_object_callback(obj,start_query,(long)fdui);
    fdui->quit = obj = fl_add_button(FL_NORMAL_BUTTON,440,250,100,
50,"Quit");
    fl_set_object_color(obj,FL_INDIANRED,FL_GREEN);
    fl_set_object_lsize(obj,FL_LARGE_SIZE);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE+FL_SHADOW_STYLE);
    fdui->but_sterge = obj = fl_add_button(FL_NORMAL_BUTTON,440,170,
100,50,"Clear");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lsize(obj,FL_LARGE_SIZE);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE+FL_SHADOW_STYLE);
    fl_set_object_callback(obj,sterge,(long)fdui);
    fdui->bleft = obj = fl_add_button(FL_NORMAL_BUTTON,140,460,50,
40,"@<<");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lcolor(obj,FL_COL1);

```

```

    fl_set_object_callback(obj, left, (long)fdui);
    fdui->bright = obj = fl_add_button(FL_NORMAL_BUTTON, 220, 460, 50, 40, "@>>");
    fl_set_object_color(obj, FL_PALEGREEN, FL_INDIANRED);
    fl_set_object_lcolor(obj, FL_COL1);
    fl_set_object_callback(obj, right, (long)fdui);
    fdui->status_box = obj = fl_add_box(FL_DOWN_BOX, 420, 360, 150, 60, "search
status");
    fl_end_form();

    fdui->sql->fdui = fdui;

    return fdui;
}
/*-----*/

void seteaza(FL_OBJECT * o, long p)
{
    //seteaza butoanele Select
    if (selected[(int)p]==1)
        selected[(int)p]=0;
    else
        selected[(int)p]=1;
}

void start_query(FL_OBJECT * o, long p)
{
    //interogare baza de date
    int i,j,k;
    char s[2];

    strcpy(q, "select * from ");
    strcat(q, tblname);
    strcat(q, " where ");

    for (i=0; i<N; i++)
    {
        if(selected[i])
        {
            strcpy(inp_text[i], fl_get_input(((FD_sql *)p)->i[i]));

```

```

        strcat(q,inp_label[i]);
        strcat(q,"=\"");
        strcat(q,inp_text[i]);
        strcat(q,"\" and ");
    }
}
q[strlen(q)-5]='\0';
if(!mysql_query(mys,q))
{
    if(!(mysres=mysql_store_result(mys)))
    {
        printf("SQL: Returning Result Error\n");
        exit(1);
    }
    k=mysql_num_rows(mysres);
    if (k)
    {
        if (k>MAX_RES)
            k=MAX_RES;
        for (j=0;j<k;j++)
        {
            mysrow=mysql_fetch_row(mysres);
            if (mysql_num_fields(mysres)!=N)
                printf("SQL: unmatched number of fields\n");
            for (i=0;i<N;i++)
            {
                if (i<mysql_num_fields(mysres))
                    strcpy(result[j][i],mysrow[i]);
                else
                    strcpy(result[j][i],"");
            }
        }
        for (i=0;i<N;i++)
            fl_set_input(((FD_sql *)p)->i[i],result[0][i]);
        if (MAX_RES>=mysql_num_rows(mysres))
            fl_set_object_label(((FD_sql *)p)->status_box,"SQL:
            Query OK");
        else
        {
            strcpy(q,"SQL: displayed ");
            sprintf(s,"%d",MAX_RES);

```

```

        strcat(q,s);
        strcat(q," of ");
        sprintf(s,"%d",mysql_num_rows(mysres));
        strcat(q,s);
        strcat(q," results");
        fl_set_object_label(((FD_sql *)p)->status_box,q);
    }
}
else
{
    fl_set_object_label(((FD_sql *)p)->status_box,"SQL: 0 results found");
}
}
else
{
    fl_set_object_label(((FD_sql *)p)->status_box,"SQL: Query Error");
}
strcpy(q,"");
res_max=j-1;
res_pos=0;
}

```

```

void sterge(FL_OBJECT * o, long p)
{
    //reinitializeaza input-box-urile
    int i;
    for (i=0;i<N;i++)
        fl_set_input(((FD_sql *)p)->i[i],"");
}

```

```

void left(FL_OBJECT * o, long p)
{
    //afisare rezultat anterior din lista rezultatelor curente
    int i;
    if (res_pos==0)
        res_pos=res_max+1;
}

```

```
    res_pos--;
    for (i=0;i<N;i++)
        {
            fl_set_input(((FD_sql *)p)->i[i],result[res_pos][i]);
        }
}

void right(FL_OBJECT * o, long p)
{
    //afisare rezultat urmator din lista rezultatelor curente
    int i;

    if (res_pos==res_max)
        res_pos=-1;

    res_pos++;
    for (i=0;i<N;i++)
        {
            fl_set_input(((FD_sql *)p)->i[i],result[res_pos][i]);
        }
}
```

Fișierul **raspunde.h**

```
extern void do_respond(char * , int , char **, int);
```

Fișierul **raspunde.c**

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <forms.h>
#include "respond.h"
#include "raspunde.h"

void do_respond(char * text_raspuns, int argc, char *argv[],int first_node)
{
    //afisare fereastra de raspuns
    FD_respond * frespond;

    frespond=(FD_respond *)malloc(sizeof(FD_respond));

    if (first_node)
        fl_initialize(&argc, argv, "Respond", 0, 0);

    frespond=create_form_respond();
    fl_show_form(frespond->respond, FL_PLACE_CENTER, 10, "Send Response");
    fl_set_input(frespond->raspuns,text_raspuns);
    while(fl_do_forms()!=frespond->but_quit);
    fl_hide_form(frespond->respond);
    fl_free_form(frespond->respond);
}
```

Fișierul **respond.h**

```
#ifndef FD_respond_h_
#define FD_respond_h_
/* Header file generated with fdesign. */

/**** Callback routines ****/

extern void send_response(FL_OBJECT *, long);

/**** Forms and Objects ****/

typedef struct {
    FL_FORM *respond;
    void *vdata;
    long ldata;
    FL_OBJECT *raspuns;
    FL_OBJECT *but_response;
    FL_OBJECT *but_quit;
} FD_respond;

extern FD_respond * create_form_respond(void);

#endif /* FD_respond_h_ */
```

Fișierul **respond.c**

```

#include <stdlib.h>
#include "forms.h"
#include "respond.h"
#include "alphanum_play.h"

FD_respond *create_form_respond(void)
{
    //creare form pt. raspuns
    FL_OBJECT *obj;
    FD_respond *fdui = (FD_respond *) fl_malloc(1, sizeof(*fdui));

    fdui->respond = fl_bgn_form(FL_NO_BOX, 550, 110);
    obj = fl_add_box(FL_UP_BOX,0,0,550,110,"");
    fdui->raspuns = obj = fl_add_input(FL_NORMAL_INPUT,60,20,480,30,"Raspuns");
    fl_set_object_lsize(obj,FL_NORMAL_SIZE);
    fdui->but_response = obj = fl_add_button(FL_NORMAL_BUTTON,210,70,180,30,
"RESPOND !");
    fl_set_object_color(obj,FL_INDIANRED,FL_PALEGREEN);
    fl_set_object_lsize(obj,FL_MEDIUM_SIZE);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);
    fl_set_object_callback(obj,send_response,(long)fdui);
    fdui->but_quit = obj = fl_add_button(FL_NORMAL_BUTTON,70,70,70,30,"CLOSE");
    fl_set_object_color(obj,FL_PALEGREEN,FL_INDIANRED);
    fl_set_object_lsize(obj,FL_MEDIUM_SIZE);
    fl_set_object_lstyle(obj,FL_BOLD_STYLE);

    fl_end_form();
    fdui->respond->fdui = fdui;
    return fdui;
}
/*-----*/
void send_response(FL_OBJECT * o, long p)
{
    //trimite textul introdus pentru a fi prelucrat
    //in vederea raspunsului
    play_response(fl_get_input(((FD_respond *)p)->raspuns));
}

```

Fișierul `scenario.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include "answer.h"

char graf[MAXSIZE];
nodestr * noduri;
edgestr * muchii;
nodestr * selected;
FILE * logfile;
int sockfd, newsockfd=-2, cli_len;
struct sockaddr_in * serv_addr, * cli_addr;
char mesaj[10];

void node_log(void)
{
    //writes the curent node in the log file
    if (!selected)
        return;
    fprintf(logfile,"&&Node: %s\n",node_text(selected));
}

int menu_click(int fd, int argc, char * argv[], int first_node)
{
    edgestr * e;
    char * node_label;
    char subgraf[MAXSIZE];
    char sentence[MAXSIZE];

    if (!selected)
        return 1;
    printf("Click pe : %s\n",selected->nodeId);
```

```

printf("Text Nod: \n%s\n",node_text(selected));

strcpy(subgraf,"window(deactivate)\n");
write(fd,subgraf,strlen(subgraf));
strcpy(subgraf,"window(activate)\n");
write(fd,subgraf,strlen(subgraf));
node_log();

if (!selected->edges)
{
    printf("No childs\n");
    get_woz_label(selected,argc,argv,1,first_node);
    write(fd,"menu(file(exit))\n",17);
    return 0; //close all programs
}
strcpy(subgraf,"");
strcat(subgraf,"graph(new([");
strcat(subgraf,"l(");
strcat(subgraf,selected->nodeId);
strcat(subgraf,",n(\"node\",");
strcat(subgraf,get_woz_label(selected,argc,argv,1,first_node));
strcat(subgraf,",[");
for(e=(edgestr *)selected->edges;e->nextlistedge!=NULL;e=(edgestr *)e->nextli
stedge)
{
    strcat(subgraf,"l(");
    strcat(subgraf,e->edgeId);
    strcat(subgraf,",e(\"edge\",");
    strcat(subgraf,e->attr);
    if(exists(((nodestr *)e->node)->nodeId,subgraf))
    {
        strcat(subgraf,",r(");
        strcat(subgraf,(((nodestr *)e->node)->nodeId);
        strcat(subgraf,"))))");
    }
    else
    {
        strcat(subgraf,",l(");
        strcat(subgraf,(((nodestr *)e->node)->nodeId);
        strcat(subgraf,",n(\"node\",");
        strcat(subgraf,get_woz_label(((nodestr *)e->node,argc,argv,0, 0));

```

```

        strcat(subgraf, "[ ])))))");
    }
}

strcat(subgraf, "l(");
strcat(subgraf, e->edgeId);
strcat(subgraf, "e(\"edge\",");
strcat(subgraf, e->attr);
if(exists(((nodestr *)e->node)->nodeId, subgraf))
{
    strcat(subgraf, "r(");
    strcat(subgraf, ((nodestr *)e->node)->nodeId);
    strcat(subgraf, "))))))\n");
}
else
{
    strcat(subgraf, "l(");
    strcat(subgraf, ((nodestr *)e->node)->nodeId);
    strcat(subgraf, "n(\"node\",");
    strcat(subgraf, get_woz_label((nodestr *)e->node, argc,
    argv, 0, 0));
    strcat(subgraf, "[ ])))))\n");
}
printf("Subgraf : %s \n", subgraf);
write(fd, subgraf, strlen(subgraf));
return 1;
}
int exists(char * s, char * r)
{
    int i, l;

    l = strlen(s);
    for (i = 0; i < strlen(r); i++)
        if (strncmp(s, r + i, l) == 0)
            return 1;
    return 0;
}

void menu_select(char * string)
{
    char s[50];

```

```
int i;

if (strcmp(string,"node_selections_labels([])\n")==0)
{
    selected=NULL;
    return;
}
strcpy(s,string+24);
s[strlen(s)-3]='\0';

for (i=0;i<strlen(s);i++)
    if (s[i]==',')
        return;
selected=find_node(s);
printf("Selectie: %s\n",selected->nodeId);
}

void main (int argc, char * argv[])
{
    char inmes[50];
    char buf[MAXSIZE+50];
    int i=0;
    FILE * f;
    FILE * g;
    pid_t pid;
    int inp[2],outp[2];
    int first_node=1;

    if(pipe(inp))
    {
        printf("Pipe error\n");
        exit(1);
    }
    if(pipe(outp))
    {
        printf("Pipe error\n");
        exit(1);
    }

    pid=fork();
```

```
if (pid==-1)
{
    printf("Fork error\n");
    exit(1);
}
if (pid==0)
{
    //child
    dup2(outp[0],0);
    dup2(inp[1],1);
    close(outp[1]);
    close(inp[0]);
    if (execlp("daVinci","daVinci","-pipe",NULL)==-1)
        exit(1);
    close(outp[0]);
    close(inp[1]);
}
else
{

    close(outp[0]);
    close(inp[1]);
    if (!(logfile=fopen("nodes.history","a")))
        exit(1);
    if (!(f=fopen(GRFILE,"r")))
        exit(1);

    //initialize network socket
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("server: can't open stream socket\n");
        exit(1);
    }
    serv_addr=(struct sockaddr_in *)malloc(sizeof(struct
sockaddr_in));
    bzero((char *) serv_addr, sizeof(struct sockaddr_in));
    serv_addr->sin_family=AF_INET;
    serv_addr->sin_addr.s_addr=htonl(INADDR_ANY);
    serv_addr->sin_port=htons(6999);
    if (bind(sockfd, (struct sockaddr *)serv_addr, sizeof(struct
sockaddr_in)) < 0)
```

```
    {
        printf("server: can't bind local address\n");
        exit(1);
    }

listen(sockfd, 5);

cli_len=sizeof(cli_addr);

while(newsockfd!=-2)
    {
        newsockfd=accept(sockfd,(struct sockaddr *) cli_addr, &cli_len);
    }
if (newsockfd<0)
    {
        printf("server: accept error\n");
        exit(1);
    }
if ((g=fopen("socket.used","w"))==NULL)
    {
        printf("Couldn't write temporary file\n");
        exit(1);
    }
fprintf(g,"%d",newsockfd);
fclose(g);
//send start command

strcpy(mesaj,"start_rec");
if (send(newsockfd,mesaj,10,0)<0)
    {
        printf("Can't write to socket\n");
        exit(1);
    }

//wait for confirmation

strcpy(mesaj,"");
while (strcmp(mesaj,"start_ok")!=0)
    {
        strcpy(mesaj,"");
        if (recv(newsockfd,mesaj,9,0)<0)
```

```
        {
            printf("Can't read from socket\n");
            exit(1);
        }
    }

//start the real work

strcpy(buf,"graph(new_placed(");
i=0;
while(!feof(f))
    fread(&graf[i++],1,1,f);
strcat(buf,graf);
strcat(buf,")\n");

strcpy(inmes,"");

inmes[read(inp[0],inmes,50)]='\0';
while(strcmp(inmes,"ok\n"))
    {
        inmes[read(inp[0],inmes,50)]='\0';
    }
for(i=0;i<5;i++)
    {
        write(outp[1],"nothing\n",8);
        inmes[read(inp[0],inmes,50)]='\0';
        if(strcmp(inmes,"ok\n"))
            exit(1);
    }
write(outp[1],buf,strlen(buf));
inmes[read(inp[0],inmes,50)]='\0';
printf("%s",inmes);

//Graph parsing
noduri=process_nodelist(graf);
muchii=link_references();

tree(noduri);

while(strcmp(inmes,"quit\n"))
    {
```

```
//User interaction
printf("%s",inmes);

if (strlen(inmes)>22)
    if (strncmp(inmes,"node_selections_labels",22)==0)
        menu_select(inmes);
if (strlen(inmes)>=17)
    if (strncmp(inmes,"node_double_click",17)==0)
    {
        //send subgraph
        if (menu_click(outp[1],argc,argv,first_node)==0)
        {
            inmes[read(inp[0],inmes,50)]='\0';
        }
        if (first_node==1)
            first_node=0;
    }

    inmes[read(inp[0],inmes,50)]='\0';

}

//send stop rec command

strcpy(mesaj,"stop_rec");
if (send(newsockfd,mesaj,9,0)<0)
{
    printf("Can't write to socket\n");
    exit(1);
}

//wait for confirmation

strcpy(mesaj,"");
while (strcmp(mesaj,"stop_ok")!=0)
    if (recv(newsockfd,mesaj,8,0)<0)
    {
        printf("Can't read from socket\n");
        exit(1);
    }
```

```
    }  
  
    fclose(f);  
    fclose(logfile);  
    close(outp[0]);  
    close(inp[1]);  
    close(newsockfd);  
}  
}
```

Fișierul **stereoplay.c**

```
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <linux/soundcard.h>

#define DSP_NAME "/dev/dsp"

//Redare inregistrare cu parametrii:
//inreg: 16bit/sample
//esantionare 16Khz
//canale: stereo

int dspfd;

void main(int argv, char ** argc)
{
    char buf[256];
    int i,l;
    FILE * f;
    char * REC_FILE;
    int param, original;

    REC_FILE=(char *)malloc(strlen(argc[1])+1);
    strcpy(REC_FILE,argc[1]);

    if ((f=fopen(REC_FILE,"r"))==NULL)
    {
        printf("Cant't open file %s\n",REC_FILE);
        exit(1);
    }

    if ((dspfd=open(DSP_NAME, O_WRONLY))===-1)
    {
        perror("Can't open /dev/dsp");
    }
}
```

```
//setting record parameters

original=param=2;
if (ioctl(dspfd, SOUND_PCM_WRITE_CHANNELS, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    perror("Your soundcard doesn't accept stereo mode");

original=param=16;
if (ioctl(dspfd, SOUND_PCM_WRITE_BITS, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    perror("Your soundcard doesn't accept the required parameters");

original=param=16000;
if (ioctl(dspfd, SOUND_PCM_WRITE_RATE, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    printf("Frequency of %d Hz not supported by your soundcard\n
Playing will be at %dHz\n",original,param);

while((l=fread(buf,1,2,f))>0)
    {
        if (write(dspfd,buf,l)<0)
            perror("Playing error on /dev/dsp");
    }
if (l==-1)
    {
        perror("Internal buffer error or file error");
        exit(-1);
    }
}
```

Fișierul **wordplay.c**

```
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <linux/soundcard.h>

#define DSP_NAME "/dev/dsp"
#define DSP_MAXCOUNT 400000//=25 sec la esant. de 16KHz

//Redare un cuvânt înregistrat la :
//inreg: 16bit/sample
//esantionare 16Khz
//canale: mono

int dspfd;

void main(int argv, char ** argc)
{
    char buf[256];
    int i,l;
    long int max_count=DSP_MAXCOUNT;
    FILE * f;
    char * REC_FILE;
    int param, original;

    REC_FILE=(char *)malloc(strlen(argc[1])+5);
    strcpy(REC_FILE,argc[1]);
    strcat(REC_FILE, ".rec");

    if ((f=fopen(REC_FILE,"r"))==NULL)
    {
        printf("Cant't open file %s\n",REC_FILE);
        exit(1);
    }

    if ((dspfd=open(DSP_NAME, O_WRONLY))===-1)
```

```
    {
        perror("Can't open /dev/dsp");
    }
//setting record parameters
/*
original=param=2;
if (ioctl(dspfd, SOUND_PCM_WRITE_CHANNELS, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    perror("Your soundcard doesn't accept stereo mode");
*/
original=param=16;
if (ioctl(dspfd, SOUND_PCM_WRITE_BITS, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    perror("Your soundcard doesn't accept the required parameters");

original=param=16000;
if (ioctl(dspfd, SOUND_PCM_WRITE_RATE, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    printf("Frequency of %d Hz not supported by your soundcard\n
Playing will be at %dHz\n",original,param);

while((max_count > 0) && ((l=fread(buf,1,1,f))>0))
    {
        if (write(dspfd,buf,l)<0)
            perror("Playing error on /dev/dsp");
        max_count--;
    }
if (l==-1)
    {
        perror("Internal buffer error or file error");
        exit(-1);
    }
}
```

Fișierul **wordrec.c**

```
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <linux/soundcard.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/time.h>

#define DSP_NAME "/dev/dsp"
#define DSP_MAXCOUNT 400000//=25 sec la esant. de 16KHz

//Inregistrare un cuvânt cu parametrii:
//inreg: 16bit/sample
//esantionare 16Khz
//canale: mono

int dspfd;
fd_set sready;
struct timeval nowait;

void main(int argv, char ** argc)
{
    char buf[256];
    int i,l;
    long int max_count=DSP_MAXCOUNT;
    FILE * f;
    char * REC_FILE;
    int param, original;
    struct termios * termios_p = (struct termios*) malloc(sizeof (struct termios)
);
    struct termios * termios_init = (struct termios*) malloc(sizeof (struct
termios));
    int q, retval;
    char c[1];
```

```
REC_FILE=(char *)malloc(strlen(argc[1])+5);
strcpy(REC_FILE,argc[1]);
strcat(REC_FILE, ".rec");

if ((f=fopen(REC_FILE,"w"))==NULL)
{
    printf("Can't open file %s\n",REC_FILE);
    exit(1);
}

if ((dspfd=open(DSP_NAME, O_RDONLY))===-1)
{
    perror("Can't open /dev/dsp");
}
//setting record parameters
/*
original=param=2;
if (ioctl(dspfd, SOUND_PCM_WRITE_CHANNELS, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    perror("Your soundcard doesn't accept stereo mode");
*/
original=param=16;
if (ioctl(dspfd, SOUND_PCM_WRITE_BITS, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    perror("Your soundcard doesn't accept the required parameters");

original=param=16000;
if (ioctl(dspfd, SOUND_PCM_WRITE_RATE, &param)==-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    printf("Frequency of %d Hz not supported by your soundcard\n
Playing will be at %dHz\n",original,param);

//set terminal non-buffered
tcgetattr (0, termios_p);
tcgetattr (0, termios_init);
termios_p->c_lflag &=~ICANON;
tcsetattr(0, TCSANOW, termios_p);
```

```
c[0]=' ';
q=0;
while((q>=0) && (c[0]!=27) && (max_count > 0) && ((l=read(dspfd,buf,1))>0))
{
    if (fwrite(buf,1,l,f)==-1)
        perror("Internal buffer error or file error");
    max_count--;

    FD_ZERO(&sready);
    FD_SET((unsigned int)0, &sready);
    memset((char *)&nowait,0,sizeof(nowait));

    if ((retval=select(1,&sready,NULL,NULL,&nowait))<0)
    {
        printf("Terminal error, can't read status\n");
        exit(1);
    }
    if (FD_ISSET(0,&sready))
        q=read(0,c,1);

}
if (q<0)
{
    perror("Terminal error");
    exit(1);
}
if (l==-1)
{
    perror("Recording error on /dev/dsp");
    exit(-1);
}
//restore terminal attributes
tcsetattr(0, TCSANOW, termios_init);
}
```

Fișierul **user.c**

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <linux/soundcard.h>
#include <sys/types.h>
#include <sys/time.h>

#define DSP_NAME "/dev/dsp"

fd_set sready;
struct timeval nowait;
int sockfd;
struct sockaddr_in * serv_addr;

void main(void)
{
    int r,q;
    char mesaj[1024];
    int dspfd;
    char buf[256];
    int i,l;
    FILE * f;
    int param, original;
    long int sock_arg;
    int retval;
    int numar;

    printf("=====\n");
    printf("|          Buna ziua,          |\n");
    printf("|          |\n");
    printf("|      Va rugam sa asteptati      |\n");
    printf("|  instructiunile pe care vi le  |\n");
```

```
printf("|   va comunica sistemul, dupa   |\n");
printf("|   care puteti incepe dialogul. |\n");
printf("|                                   |\n");
printf("=====\n\n");

//initialize sockets

if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("client: can't open stream socket\n");
    exit(1);
}

serv_addr=(struct sockaddr_in *)malloc(sizeof(struct sockaddr_in));
bzero((char *) serv_addr, sizeof(struct sockaddr_in));
serv_addr->sin_family=AF_INET;
serv_addr->sin_addr.s_addr=inet_addr("192.168.2.5");
serv_addr->sin_port=htons(6999);

if (connect(sockfd, (struct sockaddr_in *)serv_addr, sizeof(struct
sockaddr_in))<0)
{
    printf("client: can't connect to server\n");
    exit(1);
}

//wait for start command

strcpy(mesaj,"");
while (strcmp(mesaj,"start_rec")!=0)
{
    strcpy(mesaj,"");
    if (recv(sockfd,mesaj,10,0)<0)
    {
        printf("Can't read from socket\n");
        exit(1);
    }
}

//send confirmation
```

```
strcpy(mesaj,"start_ok");
if (send(sockfd,mesaj,9,0)<0)
{
    printf("Can't write to socket\n");
    exit(1);
}

//preparing files for recording

if ((f=fopen("session.raw","w"))==NULL)
{
    printf("Cant't open file session.raw\n");
    exit(1);
}

if ((dspfd=open(DSP_NAME, O_RDONLY))===-1)
{
    perror("Can't open /dev/dsp");
}

//setting record parameters
//inreg: 16bit/sample
//esantionare 16Khz
//canale: stereo

original=param=2;
if (ioctl(dspfd, SOUND_PCM_WRITE_CHANNELS, &param)===-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    perror("Your soundcard doesn't accept stereo mode");

original=param=16;
if (ioctl(dspfd, SOUND_PCM_WRITE_BITS, &param)===-1)
    perror("Couldn't set parameters to /dev/dsp");
if (param!=original)
    perror("Your soundcard doesn't accept the required parameters");

original=param=16000;
if (ioctl(dspfd, SOUND_PCM_WRITE_RATE, &param)===-1)
    perror("Couldn't set parameters to /dev/dsp");
```

```
    if (param!=original)
        printf("Frequency of %d Hz not supported by your soundcard\nPlaying will
be at %dHz\n",original,param);

//record until stop command

strcpy(mesaj,"");
while (strcmp(mesaj,"stop_rec")!=0)
    {
        strcpy(mesaj,"");
//        printf(".");

        if((l=read(dspfd,buf,16))<0)
            {
                perror("Recording error on /dev/dsp");
                exit(-1);
            }
        if (fwrite(buf,1,l,f)==-1)
            perror("Internal buffer error or recording file error");

//check if there is data on sockfd

        FD_ZERO(&sready);
        FD_SET((unsigned int)sockfd, &sready);
        memset((char *)&nowait,0,sizeof(nowait));

        if ((retval=select(sockfd+1,&sready,NULL,NULL,&nowait))<0)
            {
                printf("Socket error, can't read status\n");
                exit(1);
            }
        if (FD_ISSET(sockfd,&sready))
            {
//                printf("socket modified\n");
                strcpy(mesaj,"");
                if ((r=recv(sockfd,mesaj,9,0))<0)
                    {
                        printf("Can't read from socket\n");
                        exit(1);
                    }
            }
    }
```


Fișierul **dbcreate.sh**

```
#!/bin/sh

if test $# -ne 2
then
echo "usage: lexicon DBNAME USERNAME"
echo "DBNAME USERNAME are the values from dbask.conf"
else
db=$1
user=$2
mysql --user=root -e "drop database if exists $db; create database $db;
grant select, insert, update, delete, create, drop on $db.* to
$user@localhost; flush privileges;"
fi
```

```
//Fisierul dbinit.sh

#!/bin/sh

if test $# -lt 4
then
echo "usage: lexicon DBNAME TBLNAME USERNAME FIELD1 FIELD2 ..."
echo "DBNAME TBLNAME USERNAME FIELD1 ... are the values from
      dbask.conf"
else
db=$1
tbl=$2
user=$3
fields=$#
createdef=""
i=4
while test $i -le $fields
do
cmd="echo -ne \$$i"
field='eval $cmd'
createdef='echo -n $createdef $field char\ (40\)'
if test $i -ne $fields
then
createdef='echo -n $createdef, '
fi
i='expr $i + 1'
done
mysql --user=$user -e "use $db; create table $tbl ($createdef)"
fi
```

Fișierul **lexicon.sh**

```
#!/bin/sh

if test $# -ne 4
then
echo "usage: lexicon DBNAME TBLNAME USERNAME FIELDS"
echo "DBNAME TBLNAME USERNAME FIELDS are the values from dbask.conf"
else
db=$1
tbl=$2
user=$3
fields=$4
mysql -B -u $user -e "use $db; select * from $tbl;" > content.db
lines='wc -l content.db'
lines='basename $lines'
lines='expr $lines - 1'
tail --lines $lines content.db > content.dbase
mv content.dbase content.db
i=1
while test $i -le $fields
do
cut -f $i-$i content.db >> lexicon.list
i='expr $i + 1'
done
rm content.db
sort lexicon.list -o lexicon.word
sort woz.dialog -o woz.word
lines='grep "&" woz.word | wc -l'
ll='grep "%" woz.word | wc -l'
lines='basename $lines'
ll='basename $ll'
totlines='wc -l woz.word'
totlines='basename $totlines'
lines='expr $totlines - $lines - $ll'
tail --lines $lines woz.word > woz.word1
listword woz.word1 > woz.word2
rm woz.word1
sort woz.word2 > woz.word
rm woz.word2
```

```
cat woz.word >> lexicon.word  
rm woz.word  
uniq lexicon.word lexicon.list  
rm lexicon.word
```

```
fi
```

Anexa B

Fișiere de configurație

Fișierul de configurație **dbask.conf**

```
DBNAME=woz
TBLNAME=orar
USERNAME=woz
FIELDS=5
FLDNAME=grupa
FLDNAME=zi
FLDNAME=luna
FLDNAME=materie
FLDNAME=sala
FLDNAME=Input6
FLDNAME=Input7
FLDNAME=Input8
FLDNAME=Input9
FLDNAME=Input10
```

Fișierul de configurație **woz.dialog**

```
&woz_node_label
text
&woz_node_label
text
%user_node_label
text
%user_node_label
text
&
```