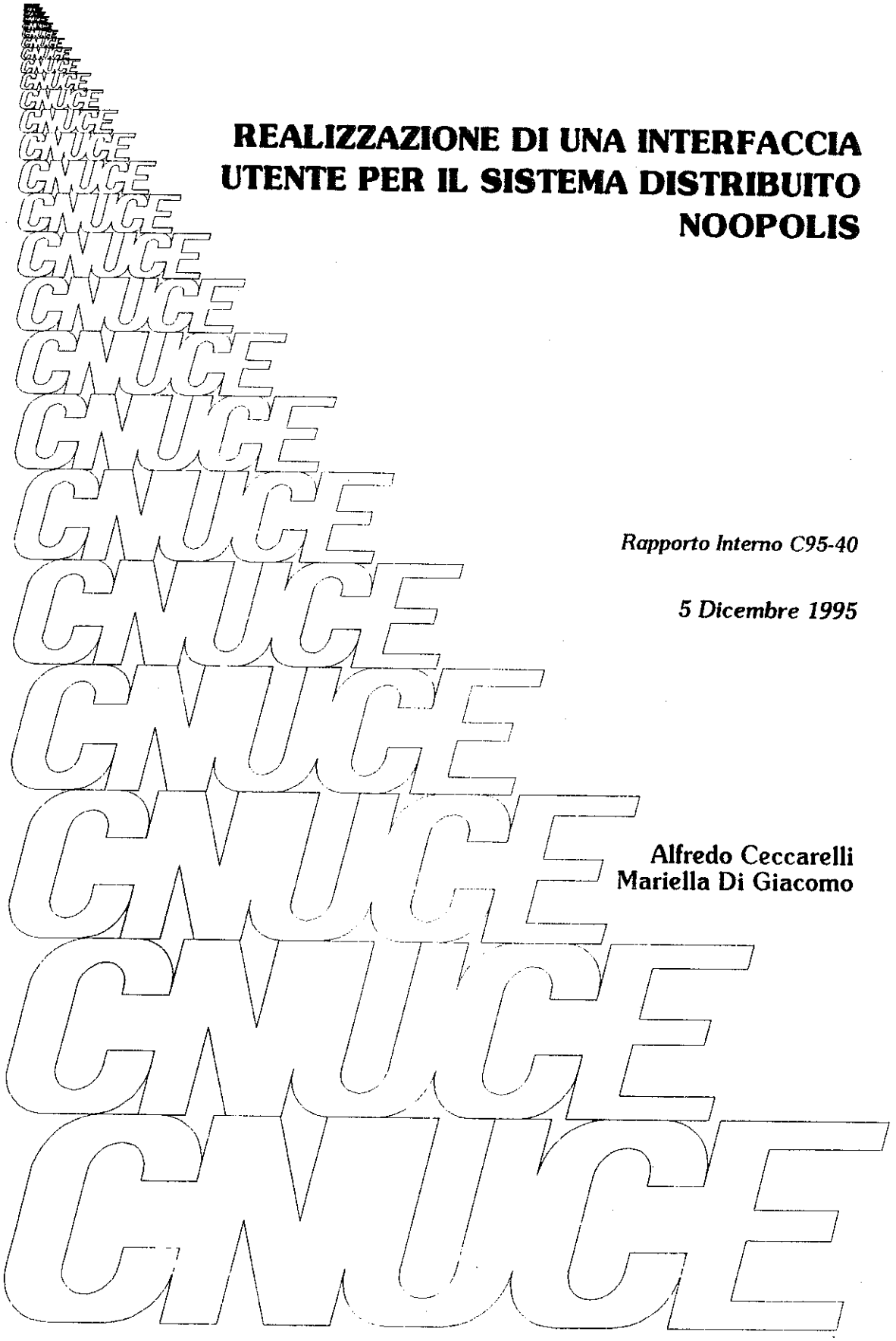
REALIZZAZIONE DI UNA INTERFACCIA UTENTE PER IL SISTEMA DISTRIBUITO NOOPOLIS



Rapporto Interno C95-40

5 Dicembre 1995

Alfredo Ceccarelli
Mariella Di Giacomo

Realizzazione di una interfaccia utente per il sistema distribuito Noopolis

Alfredo Ceccarelli *(CNUCE - Istituto del CNR)*

Mariella Di Giacomo *(Consorzio Pisa Ricerche)*

Rapporto Interno C95-40

Pisa, Dicembre 1995

Indice

Introduzione

Il rapporto descrive l'attività svolta nell'ambito del progetto Noopolis. Il progetto nasce dalla necessità di rendere disponibile, alla popolazione giovanile, le informazioni di loro specifico interesse, mantenute nella Banca Dati (BD) [Albano 85] Noopolis. Nella BD si trovano memorizzate le opportunità di studio nel mondo, nell'ambito di istituzioni, progetti di ricerca o corsi di alta formazione. La BD costituisce quindi una fonte di informazione particolarmente interessante per l'attività di studio e ricerca di giovani laureati e laureandi.

Il sistema distribuito Noopolis (sistema costituito di *personal computer*, stazioni di lavoro, *computer* Apple Macintosh, interconnessi tra loro tramite rete locale o geografica [Tanenbaum 91]) utilizza il modello cliente-servente (modello asimmetrico di comunicazione in cui esiste un processo servente in grado di fornire servizi, ed altri processi, clienti, che li richiedono, funzionanti anche su macchine diverse) [Stevens 90] per accedere alla BD; esso può essere quindi pensato come un sistema integrante un processo cliente, un processo servente ed il meccanismo che permette loro di comunicare.

Il servente Noopolis gestisce le richieste inoltrate dal cliente e interagisce con il gestore della BD, mentre il cliente è utilizzato da un utente per comporre una interrogazione e visualizzarne i risultati.

L'interazione è semplice e completamente guidata tramite menù. Il cliente può essere usato, anche da *computer* con video terminale tipo VT100 (terminale che accetta i comandi propri del VT100, un glorioso modello iniziale della Digital) o che emulano il tipo VT100, in collegamento remoto *(telnet)* ad una stazione di lavoro SUN *Microsystem*, collegata in rete locale al *personal computer* che ospita il servente e la BD Noopolis.

Per accedere a questo tipo di interfaccia possono inoltre essere usate le seguenti piattaforme: *computer* Apple Macintosh, *personal computer* o stazione di lavoro.

In questo modo è possibile, ad esempio, aprire un'applicazione su un *computer* in rete situato fisicamente a molti chilometri di distanza ed utilizzarla sul video del proprio computer.

La prima parte del documento (Capitolo 1) è rivolta all'utente finale del sistema distribuito Noopolis, e descrive le modalità di accesso alla BD accessibile tramite *Internet* presso il Consorzio Pisa Ricerche e come formulare ricerche sui documenti in essi contenuti.

La seconda parte (Capitolo 2 e 3) contiene le specifiche architetturali ed implementative del sistema distribuito Noopolis.

Nella parte finale (Capitolo 4 e 5) vengono riportate alcune informazioni relative al protocollo *Z39.50*, al protocollo *Wais* (protocolli di livello applicativo per interrogazioni a Banche Dati e per il recupero delle informazioni per sistemi basati sul modello cliente-servente) ed alla loro interazione nel trattamento della informazione.

1. Modalità di accesso e di ricerca del sistema distribuito Noopolis.

1.1 Noopolis: generalità

Noopolis è un Centro istituito per dare un concreto contributo al problema della formazione dei giovani e al loro inserimento nel mondo del lavoro.

Nato come Associazione, in seguito gli è stata affiancata l'omonima Fondazione, che è in attesa del riconoscimento da parte della Presidenza della Repubblica Italiana.

L'attività di Noopolis si articola su una serie di interventi tra loro interconnessi, riguardanti in particolare i seguenti settori:

- Banche Dati contenenti informazioni di specifico interesse per la popolazione giovanile, in particolare la BD sulle Borse di Studio realizzata direttamente da Noopolis per consentire il reperimento di informazioni su opportunità di studio nel mondo, nell'ambito di istituzioni, progetti di ricerca o corsi di alta formazione.

- Corsi, convegni, sistemi esperti su temi riguardanti la formazione giovanile e l'apporto della ricerca scientifica al soddisfacimento dei bisogni dell'uomo.

- Ricerche scientifiche, prestando particolare attenzione alla valorizzazione dei risultati ottenuti da giovani ricercatori, attraverso il Premio Noopolis e altre iniziative analoghe.

Le Banche Dati di Noopolis si propongono di raccogliere e di rendere facilmente accessibili ai giovani le informazioni che possono aiutare a completare e valorizzare la propria preparazione, anche in vista del loro inserimento nel mondo del lavoro.

La BD Noopolis sulle Borse di Studio è quella più completa e fornisce un quadro delle opportunità offerte in questo campo sia in Italia che all'estero. La BD si è considerevomente ampliata dal 1985, anno della sua istituzione, al 1994. Il numero degli Enti Erogatori, infatti, è quasi raddoppiato e quello dei bandi, delle singole borse di studio e dei paesi erogatori è aumentato di 10 -20 volte.

Essa viene continuamente aggiornata e può essere raggiunta attraverso i suoi sportelli aperti presso numerose Università, Centri Culturali ed Enti locali nelle città di: Alghero, Ancona, Bari, Bergamo, Bologna, Brescia, Cagliari, Carrara, Catania, Cesena, Firenze, Forlì, Genova, Lamezia T., Lecce, Macerata, Mestre, Milano, Messina, Modena, Napoli, Pagani, Palermo, Parma, Pisa, Pistoia, Ragusa, Reggio Emilia, Rende, Rimini, Roma, Salerno, Siena, Terni, Torino, Trento, Udine, Verona.

Un'indagine condotta nell'ambito del Programma Erasmus ha dimostrato che la BD Noopolis sulle Borse di Studio (BDN/BS) è l'unica BD esistente in Europa.

Noopolis sta inoltre preparando altre due Banche Dati, contenenti rispettivamente i corsi universitari e para-universitari esistenti in Italia ed una serie di informazioni sulla sicurezza della ricerca scientifica.

1.2 Modalità di accesso al sistema distribuito Noopolis

Le interrogazioni alla BD possono essere eseguite secondo due diverse modalità di collegamento alla macchina che ospita i dati.

La prima modalità consiste in una interfaccia grafica che è possibile eseguire tramite l'applicazione *Netscape (browser* della rete *Internet)* [Krol 94]. L'interfaccia, molto semplice da utilizzare, è realizzata tramite finestre, dove vengono inserite le informazioni oggetto della ricerca e contenente i bottoni per l'esecuzione dei relativi comandi. Questo tipo di interfaccia viene utilizzata con terminali grafici.

Il secondo tipo di interfaccia consente di interrogare la BD anche tramite un terminale di tipo VT100. Dopo l'accesso remoto alla stazione di lavoro che ospita il cliente Noopolis, interfaccia di comunicazione fra l'utente ed il servente Noopolis, si accede automaticamente al cliente attraverso successivi menù, che saranno specificati di seguito, consentendo in modo abbastanza semplice la ricerca delle informazioni desiderate.

I tipi di piattaforma che possono essere usati per accedere a tale tipo di interfaccia sono: una stazione di lavoro, un *computer* Apple Macintosh, un *personal computer* con terminali di tipo o che emulano il tipo VT100.
In questo documento vengono descritte le modalità per formulare la ricerca tramite l'interfaccia per video di tipo VT100.

1.3 Struttura di una interrogazione

Il tipo di interrogazione che si vuole impostare, è realizzato tramite la visualizzazione di menù successivi di guida.

Inizialmente, sul video appare il menù "Lingua" (vedi fig.1) che consente, inserendo il valore corrispondente, di selezionare la lingua di colloquio con l'interfaccia.

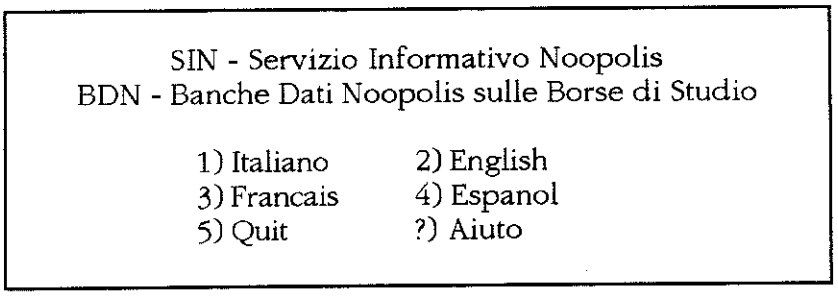


fig.1 - Menù "Lingua"

In alternativa alla scelta della lingua di colloquio preferita esistono la scelta associata al valore 5 che consente di uscire dal menù e dall'applicazione cliente, e quella associata al carattere ?, con cui vengono fornite le informazioni per effettuare l'interrogazione.

Il menù successivo, riportato nella fig.2 consente di effettuare la scelta della BD su cui si vuole effettuare la ricerca.

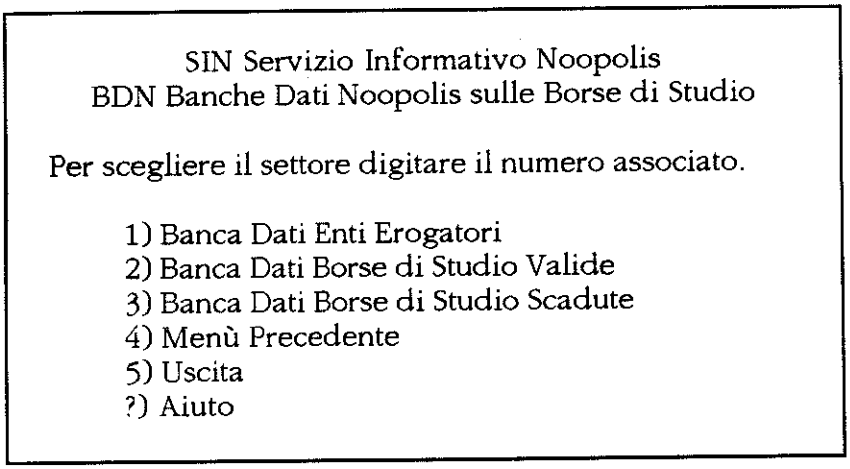


fig. 2 - Menù "Banca Dati"

La selezione della BD "Enti Erogatori" fornisce la visualizzazione del menù, riportato nella fig.3, che consente di formulare la ricerca.

Per utilizzare in modo corretto l'interfaccia si consiglia, prima di eseguire altri comandi, di scegliere l'opzione associata al carattere ? per avere le informazioni di codifica dei parametri.

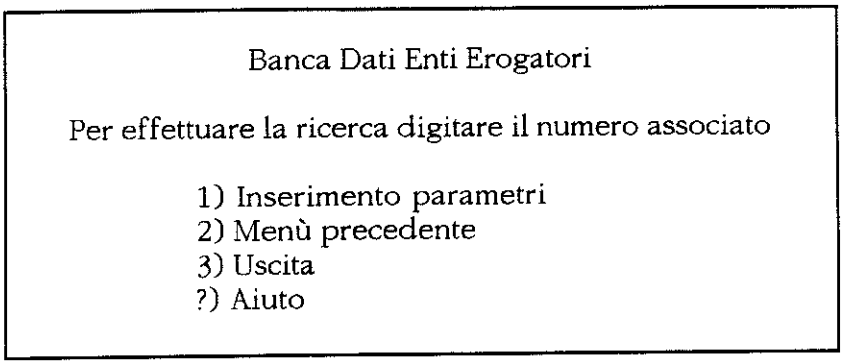


fig.3 - Menù "Banca Dati Enti Erogatori"

Con la scelta dell'opzione 1 vengono visualizzati i campi mostrati nell fig.4.

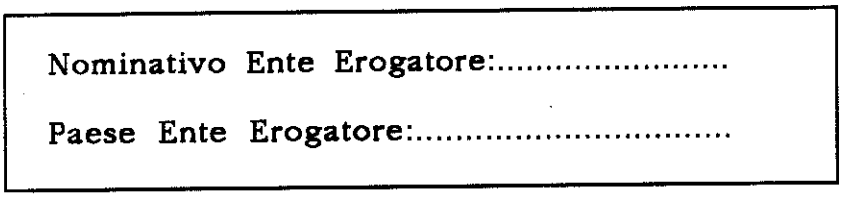


fig. 4 - "Nome dell'Ente Erogatore"

I parametri indicano, rispettivamente, il nome dell'Ente Erogatore delle Borse di Studio ed il paese dell'Ente medesimo.

I valori che possono essere specificati nei due campi sono riportati nell'Appendice A.

Le informazioni fornite sono memorizzate in una struttura dati del cliente per poter essere inviate al servente Noopolis, gestore delle interrogazioni.

Il menù successivo, riportato in fig.5, consente di inviare la ricerca specificata, annullare la ricerca medesima e fornire all'utente la possibilità di codificare una ricerca diversa, o tornare al menù precedente.

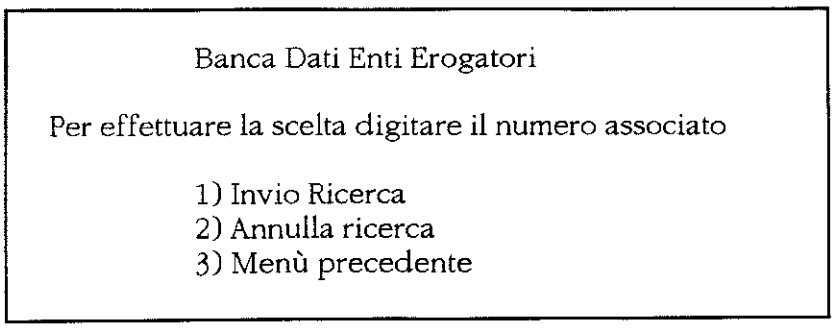


fig.5 - Menù "Invio Ricerca"

Nel caso in cui l'utente abbia selezionato l'opzione associata al valore 1, il servente Noopolis riceverà tale richiesta procedendo alla sua elaborazione e relativamente all'esito della stessa, restituirà o meno un elenco di documenti, contenenti le informazioni richieste.

A tal punto l'elenco verrà visualizzato all'utente, in ordine decrescente di peso, dove il documento in testa alla lista conterrà il maggior numero di informazioni richieste. L'utente a tal proposito potrà, se lo desidera, selezionare un documento, in modo da visualizzarne il contenuto. Tale operazione richiede che il cliente comunichi la richiesta di selezione al servente e riceva da esso il contenuto del documento da visualizzare.

Terminata la ricerca sulla BD degli Enti Erogatori, ritornando al menù della fig. 2, è possibile effettuare una ricerca diversa, per esempio relativamente alla BD delle Borse di Studio Valide oppure alla BD delle Borse di Studio Scadute. Nel primo caso è necessario selezionare il codice 2, mentre il valore 3 apre il percorso per il secondo caso. Per la scelta del codice 2, l'interfaccia fornisce il menù riportato nella fig.6.

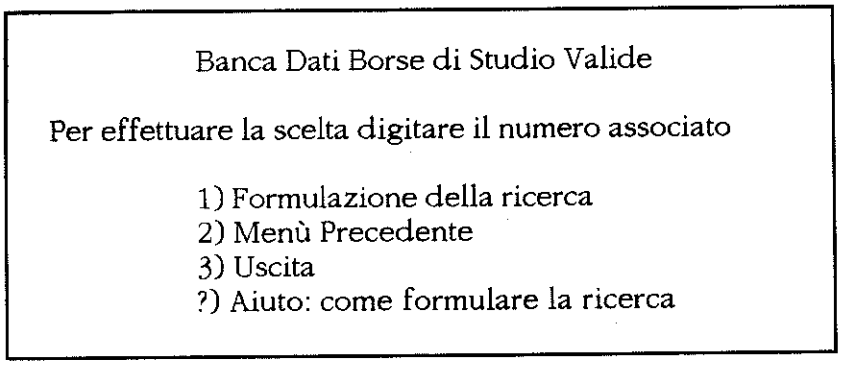


fig.6 - Menù "Borse di Studio Valide"

L'opzione associata al carattere ? consente di avere, in linea, le informazioni necessarie su come effettuare la ricerca in modo corretto. Si consiglia perciò, prima di ogni altro, di selezionare questa opzione.

L'inserimento del codice 1 consente di visualizzare il menù Borse di Studio Valide riportato nella fig.7.

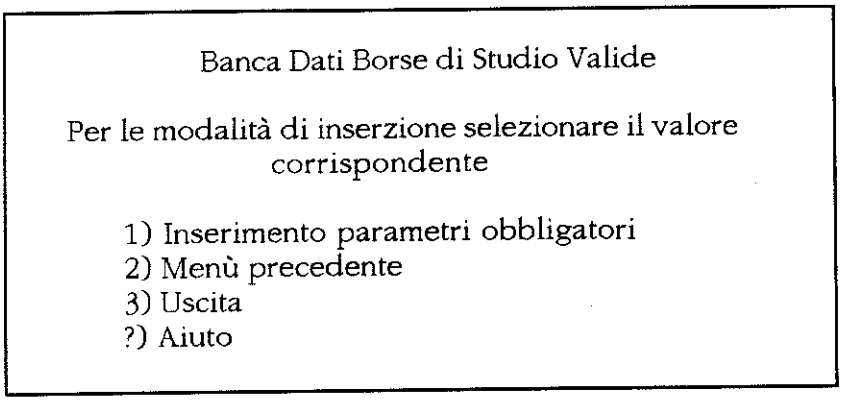


fig.7 - Menù "Borse di Studio Valide"

La selezione del codice ? fornisce il significato ed il valore che possono assumere i parametri obbligatori.

Specificando, invece, il valore 1 viene visualizzato il menù interattivo che consente di inserire i valori da associare ai parametri obbligatori, per impostare correttamente la ricerca desiderata. Il menù associato ai parametri obbligatori è riportato di seguito nella fig. 8.

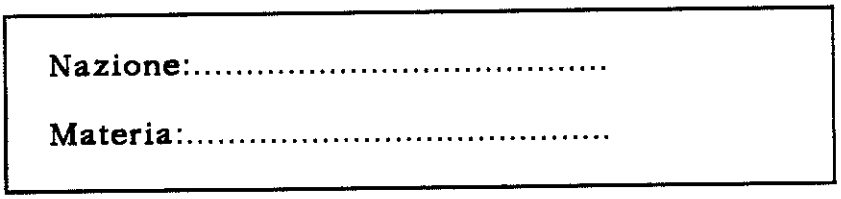


fig. 8 - "Parametri Obbligatori"

Il campo <Nazione> assumerà un valore che dovrà indicare se devono essere individuate le Borse di Studio erogate per studenti di ogni nazionalità, se la ricerca deve essere effettuata per le Borse di Studio riservate a studenti stranieri oppure a studenti italiani.

Nel campo <Materia> è possibile specificare fino a quattro valori.

Si fa presente che, qualunque informazione fornita in questo parametro, con esclusione del carattere *RETURN* è considerato valido ai fini della ricerca. Un eventuale errore comporta un prevedibile risultato errato.

I valori che possono essere specificati nei due campi sono riportati nell'Appendice B.

La prima parte della ricerca può essere formulata come descritto precedentemente. Viene di seguito visualizzato il menù successivo delle Borse di Studio Valide (vedi fig.9).

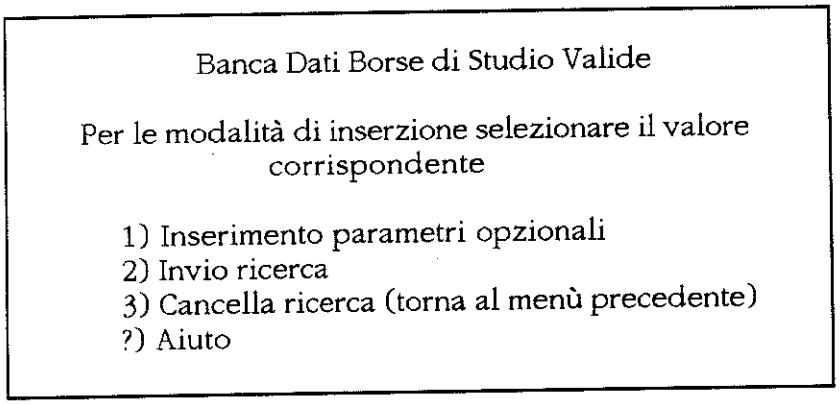


fig.9 - Menù "Borse di Studio Valide"

La ricerca viene inviata al servente tramite la selezione dell'opzione 2, mentre un'eventuale riformulazione della stessa può essere eseguita selezionando il valore 3. Nel caso in cui si vogliano inserire i parametri opzionali si consiglia di scegliere, prima, l'opzione associata al valore ?. Questa scelta fornisce le informazioni per la codifica dei parametri stessi. A questo punto, selezionando il valore 1 viene visualizzato il menù interattivo riportato nella fig. 10.

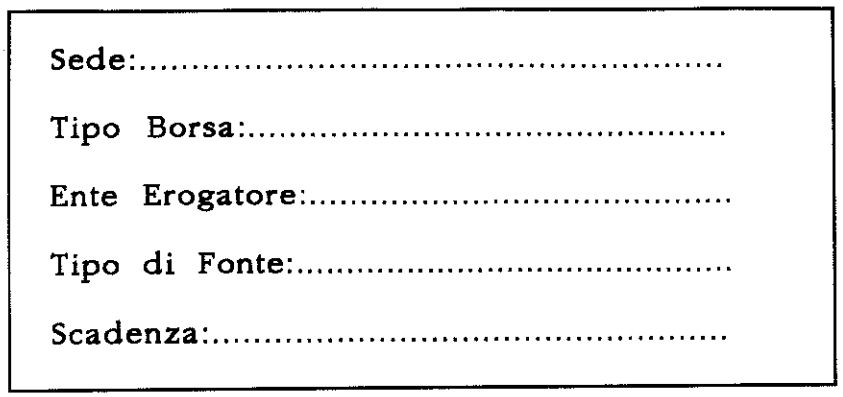


fig. 10 - Menù "Parametri Opzionali"

Il significato di ciascun parametro è il seguente:

Sede	indica la sede (continente o stato o città) dove l'utente desidera svolgere l'attività di studio o di ricerca. In questo campo si possono specificare fino a quattro valori.
Tipo Borsa	specifica il tipo di erogazione o altre possibilità. Il parametro può assumere soltanto un valore.
Ente Erogatore	specifica l'Ente a cui rivolgersi per maggiori informazioni sul concorso. È possibile specificare un valore soltanto.
Tipo di Fonte	indica l'Ente o gli Enti di riferimento per partecipare ai bandi a cui l'utente è interessato. In questo campo è possibile specificare soltanto un valore.
Scadenza	Indica la data di scadenza per la presentazione della domanda. Si può specificare un valore soltanto.

I valori che possono essere assegnati ai parametri opzionali sono specificati nell'Appendice C.

Al termine dell'assegnazione viene visualizzato il menù della successiva fig. 11.

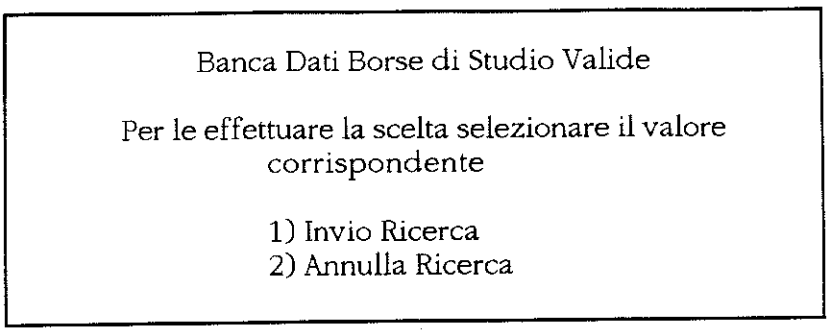


fig. 11 - Menù "Borse di Studio Valide"

A questo punto, la formulazione della ricerca si intende terminata e viene visualizzato un menù con due sole possibilità di scelta. Si può selezionare il comando per inviare la ricerca al servente oppure annullare la ricerca specificata. In quest'ultimo caso, viene riproposto il menù interattivo per l'inserimento dei parametri opzionali.

Nel caso in cui l'utente selezioni l'opzione 1, si procederà nello stesso modo del caso di ricerca sulla BD degli Enti Erogatori.

Se, infine, si desidera effettuare una ricerca nella BD delle Borse di Studio Scadute, sarà sufficente specificare l'opzione associata al valore 3 nel "Menù Banca Dati" della fig. 2. Il percorso per la formulazione della ricerca, è identico alla scelta "Banca Dati delle Borse di Studio Valide".

2. Architettura del sistema distribuito Noopolis.

2.1 Architettura generale del sistema distribuito Noopolis

L'architettura software del sistema distribuito Noopolis (vedi fig. 12) comprende uno o più Clienti Noopolis (il cui software è stato realizzato nell'ambito del progetto, su piattaforma SUN *Microsystem*), il Servente Noopolis ed il Gestore della Banca Dati (questi ultimi supportati in un *personal computer* Pentium con sistema operativo Linux [Zanetti 94], un clone del sistema operativo UNIX funzionante su *personal computer* 386/486/Pentium), interconnessi tramite rete locale [Martin 94].

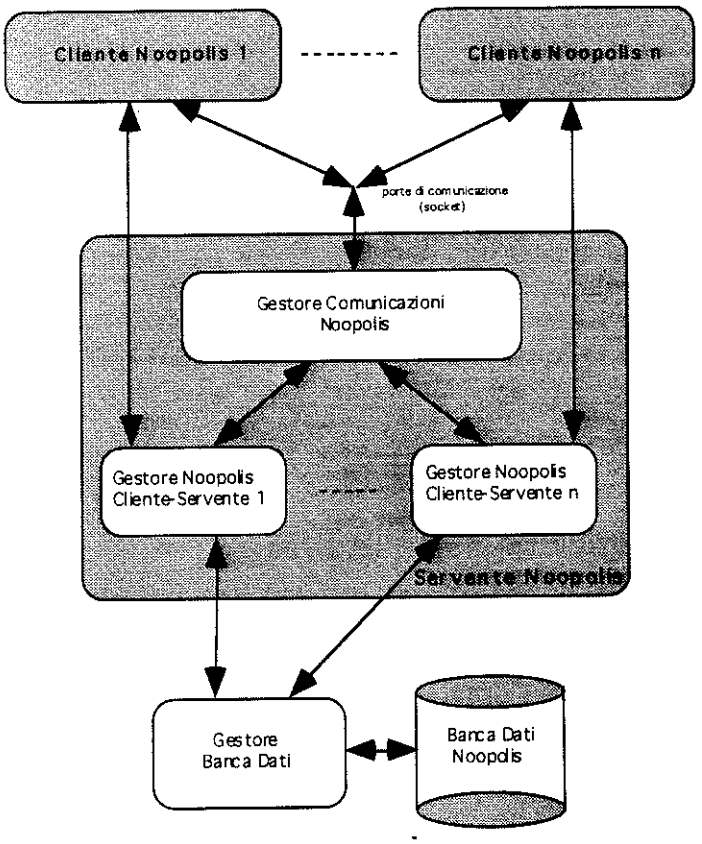


fig. 12 - Architettura Distribuita di Noopolis

L'architettura software è stata realizzata utilizzando il linguaggio di programmazione C [Kernighan 88], le chiamate di sistema fornite dal sistema operativo Unix [Bach 88] e le primitive di comunicazione del modello *socket* [Stevens 90] per la comunicazione in ambiente cliente-servente.

2.2 Il cliente Noopolis

L'architettura software del cliente Noopolis è composta dai moduli Gestore Interfaccia Utente Noopolis (GIUN), Gestore Interrogazione Noopolis (GIN) e Gestore Comunicazione Cliente Servente Noopolis (GCCSN) (vedi fig. 13).

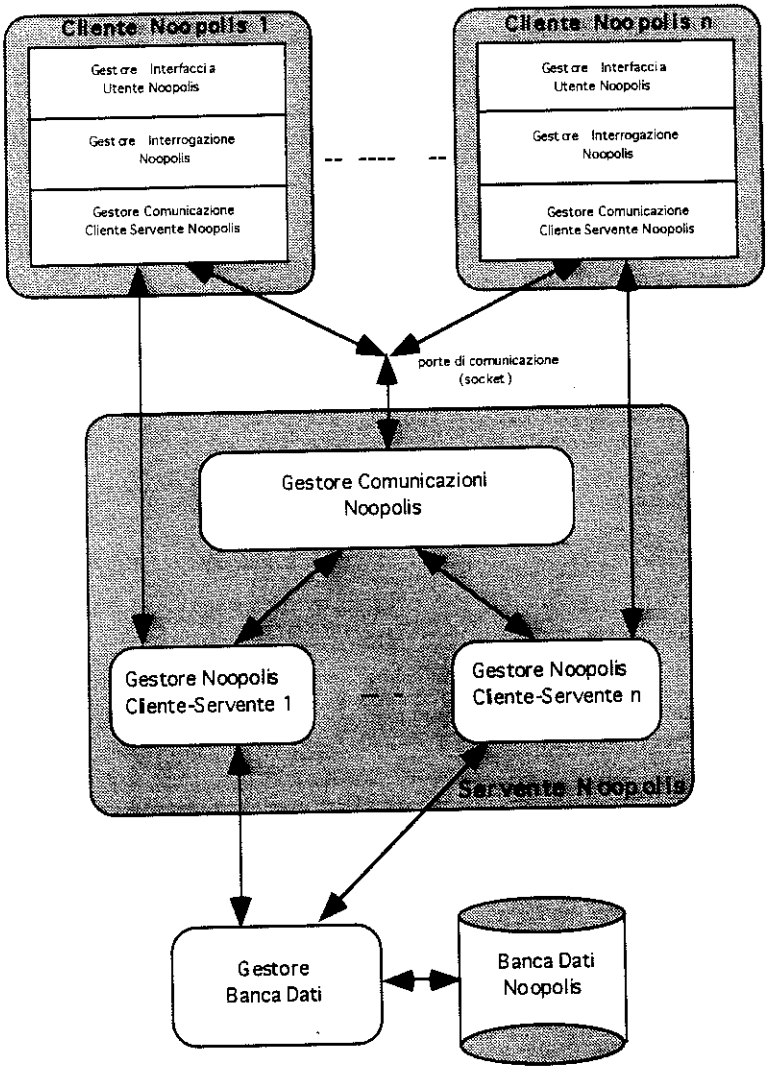


fig. 13 - Architettura di Noopolis con enfatizzazione del cliente

Le richieste che un cliente Noopolis può inviare al servente Noopolis sono:

- l'inizio di una sessione o dialogo;
- il termine con successo di una sessione (*quit*);
- il termine con fallimento di una sessione (*abort*);
- la richiesta di una interrogazione;
- la richiesta di lettura di un documento selezionato tramite la precedente interrogazione;

Gestore Interfaccia Utente Noopolis

Il modulo Gestore Interfaccia Utente Noopolis (GIUN) gestisce un'interfaccia, utilizzabile anche da terminale video tipo VT100, che consente di inoltrare l'interrogazione al modulo successivo e successivamente di visualizzare il risultato all'utente.

Lo schema di funzionamento del modulo GIUN è il seguente:

```
1: GIUN:
2:     begin
3:         <visualizzazione menù principale della
              interrogazione, in cui è possibile selezionare il
              menù relativo alla lingua preferita o un menù
              di aiuto sulle modalità di effettuare
              l'interrogazione>;
4:         switch (choice_type) {
5:             case(LANG):
                   <relativamente alla lingua selezionata, viene
                   visualizzato il successivo menù, in cui è
                   possibile effettuare un'interrogazione in
                   una delle tre possibili BD (Enti
                   Erogatori, Borse di Studio Valide, Borse di
                   Studio Scadute);
6:                 <visualizzazione dei successivi menù, con
                   possibilità di inserzione dei valori
                   relativamente a cui si è interessati>;
7:                 <visualizzazione del menù relativo alla
                   conferma della ricerca, con i valori inseriti
                   precedentemente>;
8:                 switch (query_choice) {
```

```
9:                  case(YES):
                       <composizione del messaggio,
                       contenente i valori inseriti, da
                       trasmettere al modulo GIN, il quale lo
                       elaborerà in un messaggio di inizio
                       dialogo con il servente Noopolis ed
                       in un successivo messaggio di
                       ricerca>;
10:                    <ricezione dell'elenco dei documenti
                       risultanti dalla interrogazione
                       precedentemente effettuata ed
                       visualizzazione di tale elenco,
                       nell'ordine e formato opportuno>;
11:                    <composizione e trasmissione al
                       modulo GIN del messaggio
                       contenente l'identificatore di
                       documento selezionato, nel caso
                       venga richiesta, da parte dell'utente,
                       la visualizzazione di uno, o
                       successivamente, più documenti
                       risultanti dall'interrogazione
                       precedente>.
12:                 case(NO):
                       <liberazione delle strutture
                       contenenti i valori inseriti dall'utente
                       e ritorno al menù precedente, in cui
                       è possibile riinserire nuovi valori per
                       l'interrogazione>;
13:                 case(QUIT):
                       <uscita dal cliente>.
14:         } end;
15:            case(HELP):
                  <visualizzazione del menù, creato per
                  aiutare l'utente nella formulazione
                  dell'interrogazione. All'uscita da tale menù
                  rivisualizzazione del menù contenente lo
                  stesso>;
16:            case(QUIT):
                  <uscita dal cliente, senza aver effttuato la
                  ricerca>;

17:         } end;
18:      end.
```

Gestore Interrogazione Noopolis

Il modulo Gestore Interrogazione Noopolis (GIN), ricevuta la richiesta e le informazioni necessarie, costruisce opportunamente (secondo lo standard dei protocolli *Z39.50* e *WAIS*) il messaggio da inoltrare al modulo Gestore Comunicazione Cliente Servente Noopolis (GCCSN), il quale, attivato il dialogo con il servente ed inoltrata la richiesta, comunicherà l'esito relativo alla richiesta, dopo aver ottenuta la risposta.

Lo schema di funzionamento del modulo GIN è il seguente:

```
1:  GIN:
2:      begin
3:          <ricezione dei dati, inseriti dall'utente, dal modulo
            GIUN>
4:          switch (request_type) {
5:              case(INIT):
                    <elaborazione del messaggio di richiesta
                    di inizializzazione (inizio dialogo con il
                    servente) da trasmettere al modulo
                    GCCSN, affinché attivi una connessione
                    con il servente Noopolis>;

6:              case(SEARCH):
                    <elaborazione di un messaggio di ricerca,
                    secondo le specifiche dei protocolli
                    Z39.50 e WAIS, da trasmettere, tramite il
                    modulo GCCSN, al servente Noopolis>;
7:                  <rielaborazione del messaggio di risposta
                    ricevuto dal modulo GCCSN e successivo
                    invio al modulo GIUN nel formato noto al
                    cliente Noopolis>;

8:              case(GET_DOC):
                    <elaborazione del messaggio di richiesta
                    tramite l'identificatore di documento
                    trasmesso dal modulo GIUN e successivo
                    invio al servente Noopolis, tramite il
                    modulo GCCSN>;
9:                  <rielaborazione del messaggio di risposta
                    ottenuto dal modulo GCCSN e successiva
                    trasmissione del documento al modulo
                    GIUN, che lo visualizzerà
                    opportunamente>;
```

```
10:          case(QUIT):
                  <invio del messaggio di richiesta di
                  terminazione con successo della sessione
                  al modulo GCCSN, il quale interromperà il
                  dialogo instaurato con il servente
                  Noopolis>;

11:          case(ABORT):
                  <invio del messaggio di richiesta di
                  terminazione con fallimento della
                  sessione al modulo GCCSN, il quale
                  interromperà il dialogo instaurato con il
                  servente Noopolis>;

12:       } end;
13:    end.
```

Gestore Comunicazione Cliente Servente Noopolis

Il modulo Gestore Comunicazione Cliente Servente Noopolis (GCCSN), ricevuta dal modulo GIN la richiesta di un inizio sessione, effettua la connessione con il servente, e da quel momento in poi fino alla ricezione del messaggio di terminazione con successo o fallimento da parte del modulo GIN, gestisce l'invio e la ricezione dei messaggi fra il cliente ed il servente. La comunicazione fra il modulo GCCSN nel cliente ed il servente Noopolis avviene sempre tramite il modello dei *socket*. Il tipo di *socket* usato è il circuito virtuale, perche' garantisce trasmissioni sequenziali, affidabili e non duplicate.

Lo schema di funzionamento di tale modulo è il seguente:

```
1:  GCCSN:
2:     begin
3:        sd = socket(domain,type,protocol);
4:           {crea un descrittore di socket sd};
5:        connect(sd,addr,addrlen);
6:           {viene attivata una connessione con un socket
              remoto per dialogare con il servente
              Noopolis};
7:           <riceve dal modulo GIN la richiesta di inizio
              sessione da inoltrare al servente Noopolis>;
8:        while (type_req != (quit or abort))  {
9:           <riceve dal modulo GIN la richiesta da inoltrare
              al servente Noopolis>;
10:          send(sd,msg,len,flag);
11:             {inoltra la richiesta al servente};
```

```
12:             recv(sd,msg,len,flag);
13:                 {riceve la risposta dal servente relativa alla
                    richiesta inoltrata};
14:                 <comunica il risultato ottenuto al modulo
                    Gestore Interrogazioni Noopolis>;
15:         }
16:         close(sd);
17:         shutdown(..);
18:     end.
```

2.3 Il servente Noopolis

Il servente svolge compiti di schedulazione delle richieste di accesso ai valori (documenti) della BD Noopolis. Il servente ed i clienti comunicano utilizzando il modello di comunicazione dei *socket*.

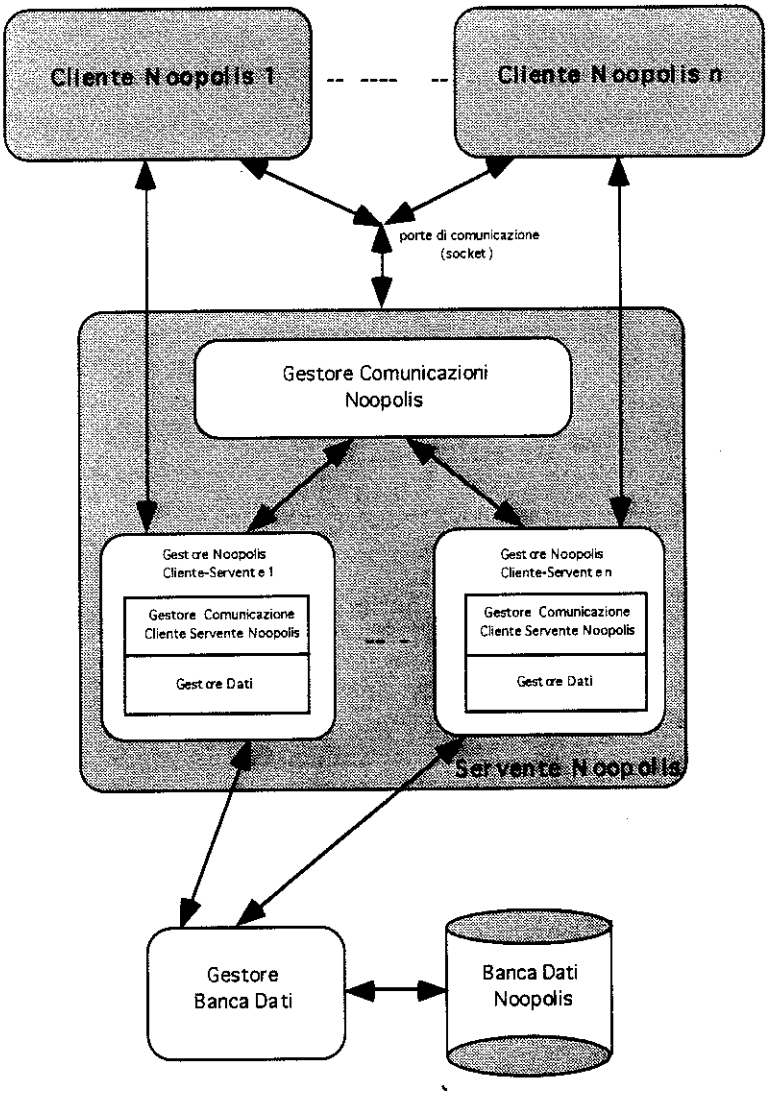


fig. 14 - Architettura di Noopolis con enfatizzazione del servente

L' architettura software del servente Noopolis (vedi fig. 14) è composta da:

1) il processo Gestore Comunicazioni Noopolis (GCN). Tale processo, oltre ad essere in attesa di connessioni dai processi clienti, ha il compito di creare un processo Gestore Noopolis Cliente Servente (GNCS) al momento di una connessione da parte di un cliente. I processi GNCS sono eseguiti concorrentemente al processo padre GCN;

2) i processi Gestore Noopolis Cliente Servente (GNCS). Ogni processo GNCS funge da intermediario tra il Gestore della Banca Dati ed il cliente a lui connesso. Il compito principale di ciascun processo è gestire ogni richiesta pervenuta dal cliente, modificandola opportunamente. L'esistenza del processo GNCS è legata alla durata della sessione lanciata dal cliente connesso (ovvero durerà fin quando il cliente non effettuerà una richiesta di terminazione con successo o con fallimento).
Ogni processo GNCS è composto dai moduli Gestore Comunicazione Cliente Servente Noopolis (GCCSN), Gestore Dati (GD).

I messaggi che il servente può inviare ad un cliente sono:

- la risposta alla comunicazione di un inizio di sessione o dialogo;
- la risposta alla comunicazione del termine con successo di una sessione (*quit*);
- la risposta alla comunicazione del termine con fallimento di una sessione (*abort*);
- la risposta con l'opportuno risultato ad una richiesta di ricerca;
- la risposta con l'opportuno documento in caso di richiesta di lettura di un documento selezionato tramite la precedente interrogazione.

Gestore Comunicazioni Noopolis

Il processo Gestore Comunicazioni Noopolis (GCN) è un processo che gestisce le comunicazioni tra i clienti ed il servente Noopolis, in modo tale che ogni cliente possa comunicare con un processo nel servente esclusivamente a lui dedicato (GNCS). La comunicazione tra i processi GCN, GNCS ed i vari clienti avviene tramite il modello di comunicazione dei *socket*.

Le funzionalità principali del GCN sono:

- stabilire un punto finale di un canale di comunicazione (*socket*) restituendo il descrittore associato;
- creare, per ogni connessione effettuata da parte di un cliente, un processo GNCS per la comunicazione con quel cliente ed un nuovo descrittore di *socket* con le stesse proprietà del descrittore restituito la prima volta, per la ricezione e l'invio delle richieste dello specifico cliente.

Lo schema di funzionamento del GCN è il seguente:

```
1:  GCN:
2:     begin
3:        sd = socket(domain,type,protocol);
4:        <crea descrittore di socket sd>;
5:        bind(sd,addr,addrlen);
6:           {associa un nome al descrittore di socket sd}
7:        listen(sd,5);
8:           {crea una coda e specifica la massima
              lunghezza per memorizzare le richieste di
              connessione in arrivo};
9:        repeat
10:          new_sd = accept(sd, addr,addrlen);
11:          {qualche cliente vuole comunicare};
12:          <attiva un processo GNCS per la gestione
              delle richieste del singolo cliente>;
13:          close(new_sd);
14:          {chiude il descrittore di socket new_sd, che
              sarà utilizzato dal processo GNCS creato
              per la comunicazione con il cliente};
15:       until forever;
16:    end.
```

Ogni processo GNCS viene creato dal GCN ogni qualvolta un cliente richiede un servizio sulla BD.

Gestore Comunicazione Cliente Servente Noopolis

Il modulo Gestore Comunicazione Cliente Servente Noopolis (GCCSN) funge da intermediario tra il modulo Gestore Dati ed il cliente a lui connesso.

Le principali funzionalità di tale modulo sono:

- chiudere il descrittore di *socket* su cui il padre GCN riceve le connessioni, in modo tale da non interferire con il suo traffico;
- attendere dal processo padre GCN il valore relativo al nuovo descrittore di *socket* associatogli. Il valore del nuovo descrittore di *socket* viene utilizzato per individuare il punto finale del canale di comunicazione attraverso il quale ricevere le richieste ed inviare le risposte al cliente connesso;
- attendere di volta in volta la richiesta inviata dal cliente associato;
- comunicare il messaggio ricevuto al modulo GD;
- ricevere dal modulo GD ed inviare il messaggio di risposta al cliente connesso.

Lo schema di funzionamento del GCCSN è il seguente:

```
1:  GCCSN(n):
2:      begin
3:      repeat
4:      close(sd);
5:          {chiude il descrittore di socket sd su cui il
             padre GCN riceve le connessioni da parte
             dei clienti};
6:      while <non arriva alcun messaggio sul socket n>
            do      Skip;
7:        recv(new_sd,msg,len,flag);
8:          {riceve dal cliente la richiesta da
             eseguire};
9:          <inoltra il messagio al modulo gestore
             informazioni>;
10:         <riceve dal modulo gestore informazioni il
             messaggio di risposta>;
11:       send(new_sd,msg,len,flag);
12:         {invia al cliente connesso la risposta
             relativa};
13:       until <arriva una richiesta per un quit od un
                 abort>;
14:       close(new_sd);
15:       end
```

Gestore Dati

Il modulo Gestore Dati (GD) si occupa della gestione dei dati fra il cliente ed il Gestore della Banca Dati. Le principali funzionalità del modulo GD sono:

- accettare dal modulo GCCSN l'interrogazione formulata dal cliente da trasmettere nel formato opportuno al Gestore della Banca Dati.
  In caso di ricerca documenti, data una frase in linguaggio pseudonaturale eliminare le parole non significative specificate in un file apposito (file di *stoplist*) ed eventualmente espandere le parole contenenti caratteri "*wildcards*" (?,*) e successivamente interagire con il Gestore della Banca Dati per comunicargli le informazioni da ricercare ed ottenere l'elenco dei risultati.
  In caso di caricamento di un documento, noto il riferimento al documento, caricare in memoria il relativo contenuto;
- ricevere dal Gestore della Banca Dati il messaggio di risposta ed inviare al cliente il messaggio opportunamente modificato.

Lo schema di funzionamento del modulo GD è il seguente:

```
1:  GD:
2:      begin
3:          switch (request_type) {

4:              case(SEARCH):
                  <rielaborazione del messaggio di richiesta
                    inviato dal modulo GCCSN.>
5:                <eliminare dalla frase in linguaggio
                    pseudonaturale le parole non
                    significative specificate nel file di
                    stoplist ed  eventualmente espandere le
                    parole contenenti caratteri "wildcards"
                    (?,*)>;
6:                <Inviare al Gestore della Banca Dati la
                    richiesta nel formato opportuno>.
7:                <rielaborazione del messaggio di risposta
                    ricevuto dal Gestore della Banca Dati e
                    successivo invio al modulo GCCSN>;
```

```
8:          case(GET_DOC):
             <rielaborazione del messaggio di richiesta
               inviato dal cliente Noopolis e successivo
               invio al Gestore della Banca Dati nel
               formato opportuno>.
9:          <caricare in memoria il documento, noto il
               riferimento al documento>;
10:         <Successivo invio del documento al GCCSN
            nel formato noto al cliente Noopolis>;
11:         case(QUIT):
            <invio del messaggio di richiesta di
               terminazione con successo della sessione
               al modulo GCNN>;
12:         case(ABORT):
            <invio del messaggio di richiesta di
             terminazione con fallimento della sessione
             al modulo GCCSN>;
13:     } end;
14: end.
```

2.4 Il Gestore della Banca Dati

Il Gestore della Banca Dati effettua la ricerca dei documenti nella BD ed espleta funzioni di mantenimento ed aggiornamento della stessa. Esso costituisce il motore che realizza le funzioni basilari della gestione dei documenti; realizza le funzioni primarie di indicizzazione e ricerca. Il Gestore della Banca Dati crea un indice di parole presenti nei documenti, cosicché possano essere ricercate più velocemente. Esso conosce varie forme di dati: testi normali, vari formattatori di testo (ad esempio LaTex), formato della posta elettronica, etc.

Per l'indicizzazione il meccanismo delle liste invertite, utilizzate anche per il recupero delle informazioni.

3. Schema funzionale del sistema distribuito Noopolis

In questo paragrafo viene descritto lo schema funzionale dei componenti del sistema distribuito Noopolis. In particolare in fig. 15 viene descritto il flusso delle operazioni eseguite dal servente Noopolis, in seguito ad una richiesta da parte di un cliente.

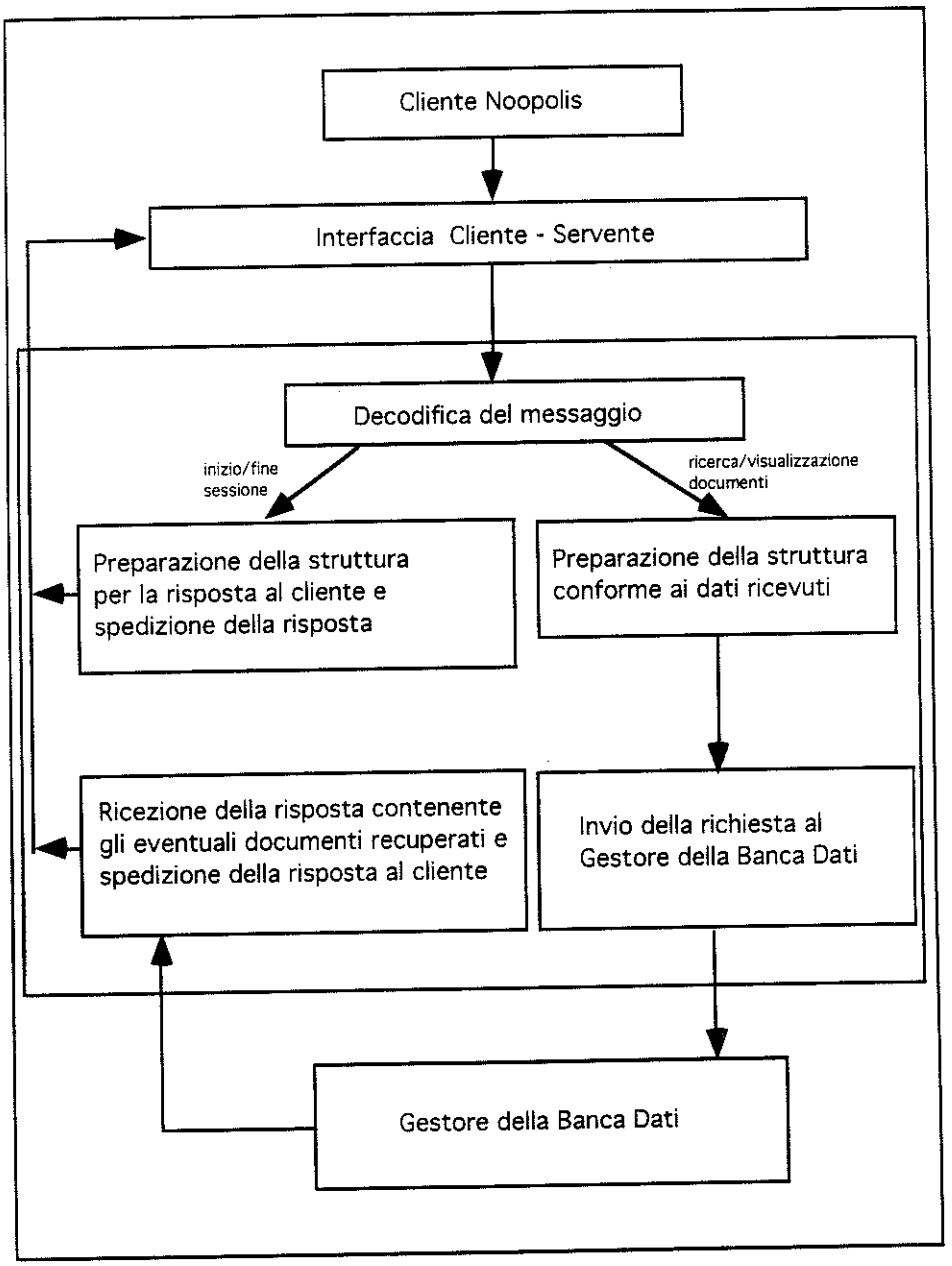


fig. 15 - Fasi del servente Noopolis

Servente:

```
1:  begin
2:      sd = socket(domain,type,protocol);
3:          {crea un descrittore di socket, ovvero un punto
            di accesso al servizio di un dato tipo};
4:          bind(sd,sockaddr,len);
5:          {associa un nome ascii al socket sd creato};
6:          listen(sd,queuelen);
7:          {crea una coda per memorizzare le richieste di
            connessione in arrivo};
8:          repeat    {
9:          new_sd = accept(sd,addr,addrlen);
10:         {il servente attende una connessione da un
            cliente e rimuove la richiesta, non appena essa
            è stata inserita nella coda};
11:         if ((child_proc = fork()) == 0)  {
12:             {la primitiva di sistema fork crea un nuovo
                processo, che possiede le stesse strutture
                dati del padre. Il processo figlio ha tuttavia
                un identificatore diverso da quello del
                padre. La primitiva fork restituisce 0 al
                processo figlio e ritorna l'identificatore
                del processo figlio al processo padre};
13:             close(sd);
14:                 {il processo figlio chiude il socket su
                    cui il padre riceve le connessioni,
                    per non intralciare con il suo
                    traffico};
15:             serve_client(new_sd,buffer);
16:                 {tale funzione gestisce la ricezione
                    delle richieste, l'elaborazione e
                    l'invio delle risposte al cliente};
17:             close(new_sd);
18:                 {il processo figlio chiude il
                    proprioente e' terminata};
19:             exit(0);
20:                 {il processo figlio termina
                    l'esecuzione, poichè la sua vita è
                    legata alla durata della sessione
                    lanciata dal cliente};
21:             };
22:         else      {
23:             close(new_sd);
24:                 {il processo padre chiude il nuovo
                    socket creato, per non interagire
                    con il traffico del figlio};
25:             }
26:         } until forever;
27:     end.
```

```
serve_client(new_sd,buffer):

1:  begin
2:      while (type_req != (quit or abort))  {
3:          switch(type_req)          {
4:              case(init_session):
5:                  handle_init(.....);
6:                      {il processo figlio, tramite la
                        primitiva recv riceve le
                        informazioni necessarie per
                        attivare la sessione (il nome
                        della BD da interrogare,
                        la massima dimensione
                        dei messaggi da trasmettere, il
                        massimo numero di documenti
                            da ricevere, etc.};
7:              case(search):
8:              interpret_buffer(new_sd,buffer,db,..);
9:                  {tramite la primitiva recv e' stata
                    letta una richiesta di search. La
                    risposta da inviare al cliente, tramite
                    la primitiva send, comprende:
                    (a) il numero dei documenti
                    risultanti,
                    (b) per ogni documento recuperato:
                    la misura della rilevanza del
                    documento rispetto alla
                    interrogazione che lo ha
                    prodotto come risultato (peso),la
                    dimensione, l'identificatore ed il
                    titolo del documento}.
10:             case(view):
11:         interpret_buffer(new_sd,buffer,db,doc_id);
12:                 {la richiesta ricevuta tramite la
                    primitiva recv è la lettura del testo di
                    un documento ,precedentemente
                    recuperato e con identificatore
                    doc_id. La risposta, comunicata
                    tramite la primitiva send,
                    comprende il testo del documento
                    selezionato}.
13:             case(quit):
14:             case(abort):
15:                 close(new_sd);
16:                     {il servente chiude la
                        connessione con il cliente};
17:             }   end;
18:         };
19: end.
```

Cliente:

```
1:   begin
2:        sd = socket(domain,type,protocol);
3:        connect(sd,addr,addrlen);
4:             {il cliente avvia una connessione remota};
5:        while (type_req != (quit or abort))   {
6:             switch (type_req)   {
7:                  case (init_session):
8:                       init_connection(buffer,db);
9:                            {il cliente stipula tutti gli accordi
                              necessari per effettuare il
                              dialogo, ad esempio: il nome
                              della BD da interrogare,
                              la massima dimensione dei
                              messaggi da trasmettere, il
                              massimo numero di documenti
                              da ricevere, etc. Lo scambio dei
                              messaggi avverrà tramite
                              le primitive send e recv};
10:                 case (search):
11:                      <invio, tramite la primitiva send,
                         della richiesta al servente Noopolis.
                         La richiesta comprende: la frase di
                         ricercare, la BD da interrogare, etc.>;
12:                      <ricezione, tramite la primitiva recv,
                         dell'elenco dei descrittori dei
                         documenti risultanti, nel caso in cui
                         la ricerca abbia avuto esito positivo>;
13:                 case(view):
14:                      <invio, tramite la primitiva send,
                         della la lettura del testo di un
                         documento,precedentemente
                         recuperato e con identificatore
                         noto>;
15:                      <ricezione, tramite la primitiva recv,
                         del testo del documento>;
16:                 case(quit):
17:                 case(abort):
18:                      close(sd);
19:                      {il cliente chiude la comunicazione
                         con il servente};
20:                 }    end;
21:            };
22:       end.
```

4. Il Protocollo Z39.50

Al fine di completare la descrizione del sistema Noopolis, è opportuno descrivere le specifiche del protocollo Z39.50 e del protocollo WAIS, sui quali esso si basa.

Il protocollo Z39.50 è uno degli insiemi di standard OSI, *Open System Interconnection*, prodotti per facilitare l'interconnessione aperta di sistemi di *computer*. L'obiettivo del protocollo è attualmente quello di facilitare l'interconnessione fra utenti interessati ad accedere a basi di dati o BD remote, permettendo l'utilizzo degli stessi formati di messaggi e delle stesse regole per lo scambio di messaggi.

Il protocollo di cui si parlerà di seguito è la versione 2 dello standard ANSI Z39.50 (1988). Esso è lo standard su cui è basato il protocollo WAIS.

Questo standard definisce un protocollo in un livello applicativo del modello di riferimento, concernente in particolare il recupero delle informazioni memorizzate nelle basi di dati o BD locali e remote; più precisamente lo standard descrive il servizio applicativo e specifica il protocollo applicativo per il recupero delle informazioni, nell'interconnessione di sistemi aperti.

Il servizio applicativo per il recupero di informazioni è descritto in termini di servizi offerti nell'applicazione. La descrizione non specifica né limita l'implementazione in un sistema di *computer*. Lo scopo della descrizione del servizio è definire le funzioni che il protocollo deve supportare.

Le specifiche del protocollo includono la definizione delle informazioni di controllo del protocollo, le regole per lo scambio di tali informazioni ed i requisiti necessari per effettuare l'implementazione di questo protocollo. Più precisamente lo standard specifica la comunicazione orientata alla connessione, la comunicazione fra applicazioni, ma non specifica lo scambio di informazioni con i terminali o altri mezzi fisici.

Modello

L'obiettivo dello standard Z39.50 è facilitare l'interconnessione aperta di utenti di basi di dati o BD con i gestori delle stesse.

Occorre distinguere tra l'insieme degli standard OSI, l'hardware e l'implementazione del software di un sistema che utilizza i protocolli specificati in questi standard. Le modalità tramite le quali le basi di dati o le BD sono implementate differiscono considerevolmente; sistemi diversi hanno differenti stili per descrivere la memorizzazione dei dati ed i mezzi attraverso cui possono essere acceduti. Spesso viene utilizzato un modello astratto comune per la descrizione di basi di dati o BD, anche se poi un particolare sistema può effettuare una propria implementazione.

Il termine base di dati e BD utilizzato nello standard Z39.50 si riferisce ad una collezione di documenti. L'unità di informazione per il recupero di informazioni da una base di dati è un documento. Tutti i documenti hanno una strutturazione comune, contengono un insieme comune di dati ed un insieme comune di punti di accesso.

Un punto di accesso alla base di dati o alla BD, in questo standard, è una chiave unica o non unica che può essere specificata sia singolarmente che in combinazione con altri punti di accesso in una ricerca.

Un'interrogazione può essere effettuata ad una base di dati o BD, specificando i valori da confrontare con i punti di accesso della stessa. Il sottinsieme di elementi (documenti) restituiti dalla ricerca è detto insieme risultato. Un insieme risultato può essere esso stesso utilizzato in un'interrrogazione successiva per restituire un nuovo insieme risultato.

Servizio per il recupero delle informazioni

La definizione del servizio descrive un'attività fra due applicazioni su *computer* distinti: un'applicazione mittente, l'origine, ed un'applicazione destinataria, la destinazione. La destinazione è associata ad una o più basi di dati o BD. La comunicazione tra l'origine e la destinazione avviene attraverso un'associazione fra le applicazioni. Un'associazione è esplicitamente stabilita dall'origine e può essere esplicitamente terminata sia dall'origine che dalla destinazione, o implicitamente terminata da un fallimento della comunicazione o a causa di eventi esterni.

I ruoli dell'origine e della destinazione possono non essere interscambiabili in un'associazione. Un'associazione può non essere ripristinata, cosicché non viene mantenuta alcuna informazione sullo stato una volta che viene rilasciata un'associazione.

Il servizio per il recupero delle informazioni è composto dall'elemento del servizio per il controllo dell'associazione, il quale fornisce una gestione dell'associazione, ed uno o più servizi per la specifica applicazione, come il servizio applicativo per il recupero di informazioni.

Esistono tre distinte fasi durante la vita di un'associazione fra applicazioni: stabilimento dell'associazione, trasferimento delle informazioni e terminazione.

L'elemento del servizio per il controllo dell'associazione fornisce tutti i servizi richiesti durante le fasi di stabilimento e terminazione dell'associazione, inclusa la selezione di un contesto dell'applicazione che specifica, fra le varie cose, l'insieme degli elementi del servizio validi durante la fase di trasferimento delle informazioni.

4.1 Definizione dei tipi del protocollo

Lo standard Z39.50 specifica le definizioni necessarie al livello applicativo e definisce le specifiche del protocollo per il recupero delle informazioni.

Sintassi astratta del protocollo per il recupero delle informazioni

Il protocollo applicativo per il recupero delle informazioni definisce le specifiche e le procedure che regolano il trasferimento delle informazioni tra le applicazioni per il recupero delle informazioni.

Le specifiche del protocollo includono la definizione dell'informazione utilizzata per il controllo del protocollo, le regole per lo scambio di informazioni e per il controllo delle richieste. Esso specifica la comunicazione orientata alla connessione, la comunicazione fra applicazioni, ma non specifica lo scambio di informazioni con terminali o altri mezzi fisici.

Una unità di informazione, passata tra due applicazioni è chiamata APDU, *Application Protocol Data Unit*.

Il protocollo Z39.50 definisce le strutture per ogni unità base del protocollo applicativo, APDU, e supporta le funzioni che gestiscono tali strutture (funzioni per la creazione, lettura, aggiornamento e distruzione).

Le unità dati del protocollo per il recupero delle informazioni sono tipi di dato complessi. La sintassi di trasferimento di questi tipi di dati è negoziata da colui che fornisce il servizio per il recupero delle informazioni.

Il protocollo WAIS definisce varie strutture che possono essere trasformate nelle unità APDU e fornisce anche le definizioni delle funzioni per la creazione, lettura, aggiornamento e distruzione, associate al protocollo Z39.50. E` responsabilità dell'applicazione creare, inviare, ricevere e distruggere gli oggetti del protocollo Z39.50 e del protocollo WAIS.

Definizione dei tipi primitivi del protocollo

Il protocollo Z39.50 fornisce il supporto per le funzioni che manipolano i tipi di dato APDU e le funzioni di utilità per la manipolazione dei dati e le funzioni utilizzate dal livello applicativo.

I tipi primitivi utilizzati dal protocollo Z39.50 sono:

a) *any*. Tale tipo è definito come una struttura contenente un campo <dimensione> ed un <blocco di caratteri> (sequenza di caratteri ascii) di lunghezza pari al valore dell'attributo dimensione.

```
typedef struct any {
        unsigned long  size;
        char           bytes;
        } any;
```

b) *ascii*. Il tipo è definito come una stringa alfanumerica, una sequenza di lettere e/o numeri;

c) *bit map*. Il tipo è definito come il tipo *any*, contenente un campo <dimensione > ed un <blocco di *bit*>;

d) *data tag*. Tale tipo è definito come un <unsigned int>, ovvero tipo intero senza segno del linguaggio C;

e) *pdu type*. E' definito come il tipo <enum> del linguaggio C;

f) *binary integer*. Il tipo è definito come il tipo <long> del linguaggio C.

Specifiche delle strutture APDU

Le definizioni in questa sezione specificano la sintassi astratta degli elementi (strutture) per ogni unità dati, APDU, del protocollo.

```
typedef struct InitAPDU {
        pdu_type        PDUType;
        boolean         willSearch,willPresent,willDelete;
        boolean         supportAccessControl,
        boolean         supportResourceControl;
        long            PreferredMessageSize;
        long            MaximumRecordSize;
        char*           IDAuthentication;
        char*           ImplementationID;
        char*           ImplementationName;
        char*           ImplementationVersion;
        any*            ReferenceID;
        void*           UserInformationField;
        } InitAPDU;
```

```
typedef struct InitResponseAPDU {
        pdu_type        PDUType;
        boolean         Result;
        boolean         willSearch,willPresent,willDelete;
        boolean         supportAccessControl,
        boolean         supportResourceControl;
        long            PreferredMessageSize;
        long            MaximumRecordSize;
        char*           IDAuthentication;
        char*           ImplementationID;
        char*           ImplementationName;
        char*           ImplementationVersion;
        any*            ReferenceID;
        void*           UserInformationField;
        } InitResponseAPDU;
```

```
typedef struct SearchAPDU {
    pdu_type        PDUType;
    long            SmallSetUpperBound;
    long            LargeSetLowerBound;
    long            MediumSetPresentNumber;
    boolean         ReplaceIndicator;
    char*           ResultSetName;
    char**          DatabaseNames;
    char*           QueryType;
    char**          ElmentSetNames;
    any*            ReferenceID;
    void*           Query;
    } SearchAPDU;


typedef struct SearchResponseAPDU {
    pdu_type        PDUType;
    long            SearchStatus;
    long            ResultCount;
    long            NumberOfRecordsReturned;
    long            NextResultSetPosition;
    long            ResultSetStatus;
    long            PresentStatus;
    any*            ReferenceID;
    void*           DatabaseDiagnosticRecords;
    } SearchResponseAPDU;


typedef struct PresentAPDU {
    pdu_type        PDUType;
    long            NumberOfRecordsRequested;
    long            ResultSetStartPosition;
    char*           ResultSetID;
    char*           ElementSetNames;
    any*            ReferenceID;
    void*           PresentInfo;
    } PresentAPDU;


typedef struct PresentResponseAPDU {
    pdu_type        PDUType;
    boolean         PresentStatus;
    long            NumberOfRecordsReturned;
    long            NextResultSetPosition;
    any*            ReferenceID;
    void*           DatabaseDiagnosticRecords;
    } PresentResponseAPDU;
```

4.2 Le funzioni del protocollo

Il protocollo Z39.50 fornisce un servizio di comunicazione orientato alla connessione (*connection-oriented*).

Un mittente Z39.50 stabilisce le associazioni con il destinatario, impegnato nell'attivitè Z39.50. Una singola associazione fra applicazioni puè essere utilizzata per inviare in più ricerche una serie elementi di tipi di dato APDU del protocollo Z39.50. Un singolo sistema può essere simultaneamente impegnato in più associazioni applicative con sistemi remoti multipli.

Il protocollo assume che i servizi richiesti siano:

1) Rilascio ordinato dell'associazione. Entrambe le parti sono d'accordo nel rilascio e nel caso non siano stati persi i dati durante la trasmissione.
2) Terminazione con fallimento dell'associazione (*abort*). Sia il mittente che il destinatario, in qualsiasi momento, possono terminare esplicitamente l'associazione immediatamente ed incondizionatamente. I dati nella trasmissione possono essere persi.

Le funzioni del protocollo Z39.50 possono essere divise in quattro categorie:

a) funzioni che operano su dati non predefiniti. Esse sono utilizzabili per gli attributi che forniscono informazioni per l'utente;
b) funzioni che operano su dati predefiniti. Tali funzioni sono usate per supportare le funzioni del gruppo (1). Esse non possono essere utilizzate per le informazioni di utente;
c) funzioni relative alla versione del protocollo. Esse forniscono la definizione per i campi opzionali della versione del protocollo;
d) funzioni di utilità.
Le funzioni di utilità gestiscono i tipi di dati del nucleo del protocollo Z39.50. Tali funzioni sono usate internanamente dalle funzioni di libreria dello Z39.50 e possono essere usate per costruire le strutture relative alle informazioni di utente.

a) Funzioni che operano su dati non predefiniti

Le funzioni che operano sui dati non predefiniti sono elencate di seguito.

```
diagnosticRecord* makeDiag ((boolean surrogate,char*
        code,char* addInfo));
void freeDiag ((diagnosticRecord* diag));
char* readDiag ((diagnosticRecord** diag,char* buffer));
char* writeDiag ((diagnosticRecord* diag,char* buffer,long*
      len));

any* makeAny ((unsigned long size,char* data));
void freeAny ((any* a));
any* duplicateAny ((any* a));
char* writeAny ((any* a,data_tag tag,char* buffer,long*
len));
char* readAny ((any** anAny,char* buffer));
unsigned long writtenAnySize ((data_tag tag,any* a));

any* stringToAny ((char* s));
char* anyToString ((any* a));
unsigned long writtenStringSize ((data_tag tag,char* s));

any* longToAny ((long Num));
long anyToLong ((any* a));

char* writeString ((char* s,data_tag tag,char* buffer,long*
len));
char* readString ((char** s,char* buffer));

bit_map* makeBitMap ((unsigned long numBits,...));
void freeBitMap ((bit_map* bm));
boolean bitAtPos ((long pos,bit_map* bm));
char* writeBitMap ((bit_map* bm,data_tag tag,char*
buffer,long* len));
char* readBitMap ((bit_map** bm,char* buffer));

char* writeNum ((long num,data_tag tag,char* buffer,long*
len));
char* readNum ((long* num,char* buffer));
unsigned long  writtenNumSize ((data_tag tag,long num)).
```

b) Funzioni che operano su dati predefiniti

Le funzioni che operano sui dati predefiniti sono elencate di seguito.

```
char* writeCompressedInteger ((unsigned long num,char*
buf,long* len));
char* readCompressedInteger ((unsigned long *num,char*
buf));
char * writeCompressedIntWithPadding ((unsigned long
num, unsigned long size, char * buffer, long * len));
unsigned long writtenCompressedIntSize ((unsigned long
num));

char* writeTag ((data_tag tag,char* buf,long* len));
char* readTag ((data_tag* tag,char* buf));
data_tag peekTag ((char* buf));
unsigned long writtenTagSize ((data_tag tag));

char* writeByte ((unsigned long byte,char* buf,long* len));
char* readByte ((unsigned char* byte,char* buf));

char* writeBoolean ((boolean flag,char* buf,long* len));
char* readBoolean ((boolean* flag,char* buf));

char* writePDUType ((pdu_type pduType,char* buf,long*
len));
char* readPDUType ((pdu_type* pduType,char* buf));
pdu_type peekPDUType ((char* buf));

char* writeBinaryInteger ((long num,unsigned long size,
char* buf, long* len));
char* readBinaryInteger ((long* num,unsigned long
size,char* buf));

unsigned long writtenCompressedBinIntSize ((long
num)).
```

c) Funzioni relative alla versione del protocollo

Vengono di seguito elencate le funzioni relative alle informazioni sulla versione del protocollo.

```
char* writeProtocolVersion ((char* buf,long* len));
char* defaultImplementationID ((void));
char* defaultImplementationName ((void)).
```

d) Funzioni di utilità del protocollo

Vengono di seguito elencate le funzioni di utilità del protocollo.

```
InitAPDU* makeInitAPDU ((boolean search,boolean
      present,boolean deleteIt, boolean
      accessControl,boolean resourceControl,
      long prefMsgSize,long maxMsgSize,
      char* auth,char* id,char* name, char* version,
      any* refID,void* userInfo));
void freeInitAPDU ((InitAPDU* init));
char* writeInitAPDU ((InitAPDU* init,char* buffer,long*
      len));
char* readInitAPDU ((InitAPDU** init,char* buffer));

InitResponseAPDU* makeInitResponseAPDU ((boolean
      result, boolean search,boolean present,boolean
      deleteIt, boolean accessControl,boolean
      resourceControl, long prefMsgSize,long
      maxMsgSize,
char* auth,char* id,char* name, char* version,
      any* refID,void* userInfo));
void freeInitResponseAPDU ((InitResponseAPDU* init));
char* writeInitResponseAPDU ((InitResponseAPDU*
      init,char*buffer,long* len));
char* readInitResponseAPDU ((InitResponseAPDU**
      init,char* buffer));

InitResponseAPDU* replyToInitAPDU ((InitAPDU*
      init,boolean result,void* userInfo));

SearchAPDU* makeSearchAPDU ((long small,long large,
      long medium, boolean replace,char* name,char**
      databases, char* type,char** elements,any*
      refID,void* queryInfo));
void freeSearchAPDU ((SearchAPDU* query));
char* writeSearchAPDU ((SearchAPDU* query,char*
      buffer,long* len));
char* readSearchAPDU ((SearchAPDU** query,char*
      buffer));

SearchResponseAPDU* makeSearchResponseAPDU
      ((long result,long count, long recordsReturned,long
      nextPos, long resultStatus,long presentStatus,
      any* refID,void* records));
void freeSearchResponseAPDU
      ((SearchResponseAPDU*queryResponse));
```

```
char*
        writeSearchResponseAPDU((SearchResponseAPDU*
        queryResponse,
char* buffer,long* len));
char* readSearchResponseAPDU
        ((SearchResponseAPDU** queryResponse,char*
        buffer));

PresentAPDU* makePresentAPDU _((long recsReq, long
        startPos,
char* resultID,any* refID,void* info));
void freePresentAPDU ((PresentAPDU* present));
char* writePresentAPDU ((PresentAPDU* present,char*
        buffer,long* len));
char* readPresentAPDU ((PresentAPDU** present,char*
        buffer));

PresentResponseAPDU* makePresentResponseAPDU
        ((boolean status,long recsRet,
        long nextPos,any* refID,
         void* records));
void freePresentResponseAPDU
        ((PresentResponseAPDU* present));
char* writePresentResponseAPDU
        ((PresentResponseAPDU* present, char*
        buffer,long* len));
char* readPresentResponseAPDU
        ((PresentResponseAPDU**present,char* buffer));
```

4.3 Integrazione dei protocolli Z39.50 e WAIS

Il protocollo WAIS supporta le strutture relative alle informazioni di utente, utilizzando gli elementi APDU del protocollo Z39.50. Ogni elemento APDU di Z39.50 ha un riferimento ad un attributo che contiene le informazioni di utente e possiede le funzioni di lettura ed aggiornamento di tali informazioni.

Le funzioni principali, relative all'integrazione sono:

```
char* writeInitInfo ((InitAPDU* init,char* buffer,long*
        len));
char* readInitInfo ((void** info,char* buffer));

char* writeInitResponseInfo ((InitResponseAPDU*
        init,char* buffer,long* len));
char* readInitResponseInfo ((void** info,char* buffer));
```

```
char* writeSearchInfo ((SearchAPDU* query,char*
      buffer,long* len));
char* readSearchInfo ((void** info,char* buffer));

char* writeSearchResponseInfo ((SearchResponseAPDU*
      query,char* buffer,long* len));
char* readSearchResponseInfo ((void** info,char* buffer));

char* writePresentInfo ((PresentAPDU* present,char*
      buffer,long* len));
char* readPresentInfo ((void** info,char* buffer));

char* writePresentResponseInfo
      ((PresentResponseAPDU* present, char*
      buffer,long* len));
char* readPresentResponseInfo ((void** info,char*
      buffer));
```

Le funzioni di aggiornamento vengono invocate dopo l'aggiornamento di ogni elemento standard APDU e solo se l'attributo relativo all'informazione utente è non nullo. Le funzioni di lettura sono invocate dopo aver letto un elemento standard APDU. Esse restituiscono un valore nullo se non esiste l'informazione di utente associata, altrimenti esse utilizzeranno le funzioni di utilità del protocollo Z39.50 per ricostruire l'informazione di utente iniziale.

5. Il Protocollo WAIS

Il protocollo WAIS è basato sul protocollo ANSI Z39.50.
Lo scopo di questa interfaccia è stabilire un protocollo per un livello applicativo per interrogazioni a basi di dati e BD remote e per il recupero delle informazioni. L'implementazione iniziale forniva un protocollo per interrogare la base di dati *DOWQUEST*, realizzata *dal Dow Jones News Retrieval*. L'obiettivo futuro è stato di fornire una sofisticata ed espandibile interfaccia utente per basi di dati e BD.

Il protocollo è basato sulle definizioni del servizio per il recupero delle informazioni e sulle specifiche del protocollo Z39.50. La successiva versione sarà basata sullo Z39.50, ma sarà scritta in ASN.1.

5.1 Specifiche del protocollo

Le estensioni del protocollo WAIS supportano principalmente le interrogazioni "*Relevance Feedback*". Una interrogazione "*Relevance Feedback*" consente di ricercare elementi, documenti, simili a quelli già trovati. Essa consente di prendere una parte del documento e di far sì che WAIS ne estragga le parole chiavi da usare nelle ricerche future. Una interrogazione di tipo "*Relevance Feedback*", viene supportata da un'interrogazione di tipo 3 (*Type-3*) (Tale tipo di interrogazione non è stata usata per la nostra realizzazione).

Per il recupero degli elementi, dei documenti, è stata usata l'interrogazione di tipo 1 (*Type-1*). Nel caso di recupero dei documenti, il protocollo richiede che il sistema destinatario restituisca un identificatore di documento associato ad ogni elemento presente nell'insieme risultato. Questo identificatore può essere utilizzato dal mittente per specificare i documenti da visualizzare.

La sintassi della interrogazione di tipo 3, che supporta la "*Relevance Feedback*", non fa parte delle specifiche del protocollo Z39.50. Un'interrogazione *Type-3* contiene fra i vari attributi: l'identificatore di documento scelto, il massimo numero di documenti che si vuole ricevere, riferimento iniziale e finale alla parte di testo selezionato, etc.

Il protocollo WAIS è distribuito tramite due librerie (la libreria base Z39.50 e la libreria del protocollo WAIS). Il livello Z39.50 definisce le strutture per ogni unità base APDU del protocollo e supporta le funzioni di APDU, funzioni che manipolano tali strutture.

Il protocollo WAIS può essere strutturato in due parti.

La prima parte definisce le strutture e le funzioni per gestire le strutture WAIS che sono direttamente trasformate nelle strutture APDU del protocollo Z39.50. Quindi il protocollo WAIS definisce varie strutture che possono essere inglobate nelle unitè APDU e fornisce le definizioni delle funzioni agganciate al protocollo Z39.50 per creare, leggere, aggiornare e distruggere tali strutture.

La seconda parte definisce le strutture che vengono usate dal protocollo WAIS per trasmettere specifici elementi fra il mittente ed il destinatario.

5.2 Specifiche delle strutture del protocollo

```
typedef struct WAISInitResponse {
        long        ChunkCode;
        long        ChunkIDLength;
        char*       ChunkMarker;
        char*       HighlightMarker;
        char*       DeHighlightMarker;
        char*       NewlineCharacters;
        } WAISInitResponse;


typedef struct WAISSearch {
        char*       SeedWords;
        DocObj**    Docs;
        char**      TextList;
        long        DateFactor;
        char*       BeginDateRange;
        char*       EndDateRange;
        long        MaxDocumentsRetrieved;
        } WAISSearch;


typedef struct WAISDocumentHeader {
        any*        DocumentID;
        long        VersionNumber;
        long        Score;
        long        BestMatch;
        long        DocumentLength;
        long        Lines;
        char**      Types;
        char*       Source;
        char*       Date;
        char*       Headline;
        char*       OriginCity;
        } WAISDocumentHeader;
```

```
typedef struct WAISDocumentShortHeader {
        any*        DocumentID;
        long        VersionNumber;
        long        Score;
        long        BestMatch;
        long        DocumentLength;
        long        Lines;
        } WAISDocumentShortHeader;

typedef struct WAISDocumentLongHeader {
        any*        DocumentID;
        long        VersionNumber;
        long        Score;
        long        BestMatch;
        long        DocumentLength;
        long        Lines;
        char**      Types;
        char*       Source;
        char*       Date;
        char*       Headline;
        char*       OriginCity;
        char*       StockCodes;
        char*       CompanyCodes;
        char*       IndustryCodes;
        } WAISDocumentLongHeader;


typedef struct WAISDocumentText {
        any*        DocumentID;
        long        VersionNumber;
        any*        DocumentText;
        } WAISDocumentText;


typedef struct WAISDocumentHeadlines {
        any*        DocumentID;
        long        VersionNumber;
        char*       Source;
        char*       Date;
        char*       Headline;
        char*       OriginCity;
        } WAISDocumentHeadlines;
```

```
typedef struct WAISDocumentCodes {
        any*        DocumentID;
        long        VersionNumber;
        char*       StockCodes;
        char*       CompanyCodes;
        char*       IndustryCodes;
        } WAISDocumentCodes;


typedef struct WAISSearchResponse {
        char*                           SeedWordsUsed;
        WAISDocumentHeader**            DocHeaders;
        WAISDocumentShortHeader**       ShortHeaders;
        WAISDocumentLongHeader**        LongHeaders;
        WAISDocumentText**              Text;
        WAISDocumentHeadlines**         Headlines;
        WAISDocumentCodes**             Codes;
        diagnosticRecord**              Diagnostics;
        } WAISSearchResponse;


typedef struct DocObj {
        any*        DocumentID;
        char*                           Type;
        long        ChunkCode;
        union {
                    long   Pos;
                    any*   ID;
                    } ChunkStart;
        union {
                    long   Pos;
                    any*   ID;
                    } ChunkEnd;
        } DocObj;
```

5.3 Funzioni del protocollo

```
WAISInitResponse* makeWAISInitResponse ((long
        chunkCode,long chunkIDLen,
char* chunkMarker,char* highlightMarker,
char* deHighlightMarker,char* newLineChars));

void freeWAISInitResponse ((WAISInitResponse* init));
```

```
WAISSearch* makeWAISSearch ((char*
      seedWords,DocObj** docs,char** textList,
      long dateFactor,char* beginDateRange,
      char* endDateRange, long maxDocsRetrieved));
void freeWAISSearch ((WAISSearch* query));
WAISSearchResponse* makeWAISSearchResponse((char**
      seedWordsUsed,
      WAISDocumentHeader** docHeaders,
      WAISDocumentShortHeader** shortHeaders,
      WAISDocumentLongHeader** longHeaders,
      WAISDocumentText** text,
      WAISDocumentHeadlines** headlines,
      WAISDocumentCodes** codes,
            diagnosticRecord** diagnostics));
void freeWAISSearchResponse ((WAISSearchResponse*
      response));
void CSTFreeWAISInitResponse ((WAISInitResponse*
      init));
void CSTFreeWAISSearch ((WAISSearch* query));
void CSTFreeWAISSearchResponse
      ((WAISSearchResponse* response));

WAISDocumentHeader* makeWAISDocumentHeader
      ((any* aDocID,long versionNumber,
            long score,long bestMatch,long docLen,
            long lines,char** types,char* source,
            char* date,char* headline,char* originCity));
void freeWAISDocumentHeader ((WAISDocumentHeader*
      header));
char* writeWAISDocumentHeader
      ((WAISDocumentHeader* header,
            char* buffer,long* len));
 char* readWAISDocumentHeader
      ((WAISDocumentHeader** header,char* buffer));

WAISDocumentShortHeader*
      makeWAISDocumentShortHeader ((any* aDocID,
            long versionNumber, long score,
            long bestMatch,long docLen, long lines));
void freeWAISDocumentShortHeader
      ((WAISDocumentShortHeader* header));
char* writeWAISDocumentShortHeader
      ((WAISDocumentShortHeader* header,
            char* buffer,long* len));
char* readWAISDocumentShortHeader
      ((WAISDocumentShortHeader** header,
            char* buffer));
```

```
        WAISDocumentLongHeader*
makeWAISDocumentLongHeader ((
            any* aDocID,long versionNumber,
            long score,long bestMatch,long docLen,
            long lines,char** types,char* source,
            char* date, char* headline,char* originCity,
            char* stockCodes,char* companyCodes,
            char* industryCodes));
        void freeWAISDocumentLongHeader
            ((WAISDocumentLongHeader* header));
        char* writeWAISDocumentLongHeader
            ((WAISDocumentLongHeader* header,
                char* buffer,long* len));
        char* readWAISDocumentLongHeader
            ((WAISDocumentLongHeader** header,
                char* buffer));

        WAISDocumentText* makeWAISDocumentText ((any*
            aDocID,
            long versionNumber, any* documentText));
        void freeWAISDocumentText ((WAISDocumentText*
            docText));

        char* writeWAISDocumentText ((WAISDocumentText*
            docText,char* buffer,long* len));
        char* readWAISDocumentText ((WAISDocumentText**
            docText,char* buffer));

        WAISDocumentHeadlines* makeWAISDocumentHeadlines
            (( any* aDocID,long versionNumber,
                char* source,char* date,char* headline,
                char* originCity));
        void freeWAISDocumentHeadlines
            ((WAISDocumentHeadlines* docHeadline));
        char* writeWAISDocumentHeadlines
            ((WAISDocumentHeadlines* docHeadline,
                char* buffer,long* len));
        char* readWAISDocumentHeadlines
            ((WAISDocumentHeadlines** docHeadline,
                char* buffer));

        WAISDocumentCodes* makeWAISDocumentCodes ((any*
            aDocID,long versionNumber,
                char* stockCodes,char* companyCodes,
                char* industryCodes));
        void freeWAISDocumentCodes ((WAISDocumentCodes*
            docCodes));
        char* writeWAISDocumentCodes ((WAISDocumentCodes*
            docCodes, char* buffer,long* len));
```

```
char* readWAISDocumentCodes
        ((WAISDocumentCodes** docCodes,char* buffer));

any* makeWAISTextQuery ((DocObj** docs));
DocObj** readWAISTextQuery ((any* terms));
```

Le funzioni successive lavorano sui documenti e parti di questi.

```
DocObj* makeDocObjUsingWholeDocument ((any*
        aDocID,char* type));
DocObj* makeDocObjUsingBytes ((any* aDocID,char*
        type,long start,long end));
DocObj* makeDocObjUsingLines ((any* aDocID,char*
        type,long start,long end));
DocObj* makeDocObjUsingParagraphs ((any*
        aDocID,char* type,any* start,any* end));
void freeDocObj ((DocObj* doc));
```

6. Conclusioni

Nel presente rapporto è stato presentato il sistema distribuito Noopolis descrivendone le modalità di accesso, le funzionalità e la strutturazione software.

Il sistema distribuito Noopolis presenta le seguenti caratteristiche:

- consente interrogazioni concorrenti all BD Noopolis, la quale può essere locale o remota per i clienti che desiderano accederla.
  L'utente effettua le ricerche in maniera del tutto indipendente dalla località dei dati.

- offre la possibilità di poter accedere ai dati tramite due distinte interfacce utente. Ad uno dei due tipi di interfaccia utente è stato dato particolare enfasi, perchè realizzata nell'ambito del progetto.

Appendice A

I parametri <Nominativo Ente Erogatore> e <Paese Ente Erogatore> possono assumere, rispettivamente, uno soltanto dei seguenti valori:

UPPSALA University	Svezia
Ambasciata Reale di Svezia	Svezia
Ambasciata d'Austria	Austria
University of Vienna	Austria
Educational Institute of Scotland	Regno Unito
LEEDS Polytechnic	Inghilterra

Appendice B

I valori che può assumere il campo <Nazione> sono:

ogni	per indicare che devono essere ricercate le Borse di Studio per studenti di ogni nazionalità.
straniera	per indicare che la ricerca deve essere effettuata per le Borse di Studio riservate a studenti stranieri.
italiana	per selezionare le Borse di Studio riservate a studenti italiani.

Nel campo <Materia> possono essere specificati fino a quattro valori, di cui solo alcuni sono riportati qui:

AGRARIA
BIBLIOGRAFIA
CARDIOLOGIA
DANZA
ECONOMIA
FARMACIA
GEOCHIMICA
IDIOMI
MAGISTERO
OCEANOGRAFIA

Appendice C

Il parametro <Sede> può assumere non più di quattro dei seguenti valori:

AFGHANISTAN
ALBANIA
ALGERIA
ALTO VOLTA
ANGOLA
ANTARTIDE
ARABIA SAUDITA
AUSTRALIA
AUSTRIA
BAHAMAS
BAHREIN
BANGADESH
BARBADOS
BELGIO
BENIN
BERMUDA
BIELORUSSIA
BIRMANIA
BOLIVIA
BOTSWANA
BRASILE
BULGARIA
BURUNDI
CAMBOGIA
CAMERUN
CANADA
CIAD
CILE
CINA
CIPRO
COLOMBIA
COMORES
CONGO
COSTA DI AVORIO
COSTA RICA
DANIMARCA
ECUADOR
EGITTO
EL SALVADOR
EMIRATI ARABI UNITI
LIBIA
LIECHTENSTEIN
LITUANIA
LUSSEMBURGO
MADAGASCAR
MALAWI
MALAYSIA
MALTA
MAROCCO
MARTINICA
MAURITANIA
MAURITIUS
MESSICO
MONACO
MONGOLIA
MOZAMBICO
NEPAL
NICARAGUA
NIGER
NIGERIA
NORVEGIA
NUOVA ZELANDA
OVUNQUE
PAESI BASSI
PAESI CEE
PAESI CONSIGLIO D'EUROPA
PAESI NATO
PAKISTAN
PANAMA
PAPAU E NUOVA GUINEA
PARAGUAY
PERU
POLONIA
PORTORICO
PORTOGALLO
PSV
REGNO UNITO
REPUBBLICA CECA
REPUBBLICA CENTRAFRICANA
REPUBBLICA DEMOCRATICA DI COREA

ERITREA
ESTERO
ETIOPIA
EX REPUBBL YUGOSLAVIA DI MACEDONIA
FEDERAZIONE RUSSA
FIJI
FILIPPINE
FINLANDIA
FRANCIA
GABON
GAMBIA
GERMANIA
GHANA
GIAMAICA
GIAPPONE
GIBILTERRA
GIORDANIA
GRECIA
GROENLANDIA
GUATEMALA
GUINEA
GUYANA
HAITI
HONDURAS
HONG KONG
INDIA
INDONESIA
IRAN
IRAQ
IRLANDA
ISLANDA
ISOLE SALOMONE
ISRAELE
ISTITUZIONI INTERNAZIONALI
ITALIA
KENIA
KUWAIT
LAOS
LIBANO
LIBERIA
REPUBBLICA DI BOSNIA ERZEGOVINA
REPUBBLICA DI COREA
REPUBBLICA DOMENICANA
REPUBBLICA SLOVACCA
ROMANIA
SAN MARINO
SANTA SEDE
SENEGAL
SIERRA LEONE
SINGAPORE
SIRIA
SLOVENIA
SOMALIA
SPAGNA
SRI LANKA
SUDAFRICA
SUDAN
SURINAME
SVEZIA
SVIZZERA
SWAZILAND
TAIWAN
TANZANIA
THAILANDIA
TOGO
TRINIDAD E TOBAGO
TUNISIA
TURCHIA
UCRAINA
UGANDA
UNGHERIA
URUGUAY
USA
VENEZUELA
VIETNAM
YEMEN
YEMEN REPUBBLICA DEMOCRATRICA
ZAIRE
ZAMBIA
ZIMBAWE

Il parametro <Tipo Borsa> può assumere soltanto uno dei seguenti valori:

borsa di studio
borsa di ricerca
premio
premio per tesi di laurea
corso di formazione
corso di specializzazione
corso di perfezionamento
master
altri corsi
contributi e prestiti
corso gratuito
corso a pagamento
stage
dottorato di ricerca
concorso
posto per ricercatore
posto per tecnico laureato
posto per funzionario tecnico
posto per collaboratore tecnico
altri

Il parametro <Ente Erogatore> può assumere soltanto uno dei seguenti valori:

UPPSALA University	Svezia
Ambasciata Reale di Svezia	Svezia
Ambasciata d'Austria	Austria
University of Vienna	Austria
Educational Institute of Scotland	Regno Unito
LEEDS Polytechnic	Inghilterra

Il parametro <Tipo Fonte> può assumere soltanto uno dei seguenti valori:

C.N.R.	(Consiglio Nazionale delle Ricerche)
DIRETTE	(Fonti Dirette)
G.U.	(Gazzetta Ufficiale)
GUIDE UNIV.	(Guide Universitarie)
INDIRETTE	(Fonti Indirette)
M.A.E.	(Ministero degli Affari Esteri)
M.A.E.-COOP./SVILUPPO	(M.A.E Cooperazione allo Sviluppo)
UNESCO	(Organizzazione delle Nazioni Unite per l'Educazione, la Scienza e la Cultura)

Il parametro <Scadenza> può assumere uno solo dei seguenti valori:

maggiore

minore

eguale

Bibliografia

[Albano 85]
A. Albano e R. Orsini, "Basi di Dati", *Boringhieri*, Torino, 1985.

[Bach 88]
M. J. Bach, "The design of the UNIX operating system", *Prentice-Hall*, 1988.

[Comer 90]
D. E. Comer, "Internet With TCP/IP", *Prentice-Hall*, 1990.

[Kernighan 88]
B. W. Kernighan, D. M. Ritchie, "Linguaggio C", *Prentice-Hall*, 1987

[Krol 94]
E. Krol, "The Whole Internet Users Guide and Catalog", *Prentice-Hall*, 1994

[Martin 94]
J. Martin, K. K. Chapman, J. Leben, "LAN", *Prentice Halll*, 1994.

[Stevens 90]
W. R. Stevens, "UNIX Network Programming", *Prentice Halll*, 1990.

[Tanenbaum 91]
A. S. Tanenbaum, "Computer Networks", *Prentice-Hall*, 1991.

[Zanetti 94]
G. Zanetti, "LINUX", *Libreria Progetto*, Padova, 1994.