POLITECNICO DI MILANO
FACOLTÀ DI INGEGNERIA

Corso di laurea in ingegneria informatica

OPENAAC:
SISTEMA INFORMATIVO, BASATO SU ARCHITETTURA CLIENT/SERVER, DI AUSILIO ALLA COMUNICAZIONE AUMENTATIVA ALTERNATIVA

Tesi di laurea di: William GIANNELLI
Matricola 647440

Relatore: Prof. Licia Sbattella
Correlatore: Ing. Matteo Matteucci

Anno accademico 2004/05

SOMMARIO

Scopo di questa tesi è proporre un sistema software aperto e flessibile in grado di supportare un qualsiasi linguaggio di Comunicazione Aumentativa Alternativa (CAA) e di interoperare al meglio con funzionalità messe a disposizione da applicazioni esterne al programma allo scopo di migliorare la qualità della comunicazione alternativa e aumentativa e la semplicità di utilizzo da parte dell'utente.
Il presente lavoro prende spunto da Bliss2003, sistema che implementa un comunicatore software per uno specifico linguaggio di AAC, estendendone le funzionalità e aggiungendo un supporto multilingua e multilinguaggio.
L'aspetto innovativo del progetto è l'attenzione prestata alla necessità di avere applicazioni AAC in grado di potersi adattare alle esigenze, presenti e future, degli utenti fruitori facendo in modo che sia il sistema ad adattarsi all'uomo e non viceversa.
Questo intento è ottenuto grazie alla struttura *client/server* dell'applicazione che permette, e permetterà in futuro, di personalizzare le caratteristiche del programma a seconda dell'utente e della piattaforma che lo ospita grazie a un efficace meccanismo di plug-ins.

INDICE

1

INTRODUZIONE

Comunicare ed esprimersi sono due diritti universali alla base di qualsiasi rapporto sociale, di fatto questa è la principale ragione che porta l'uomo a concentrare i propri sforzi nello sviluppo di applicazioni e architetture a supporto di chi, per svariati motivi, si trova ad avere difficoltà di varia natura nell'avvalersi di tale diritto.
Il progetto al centro di questo lavoro di tesi ha lo scopo di implementare un'applicazione collocabile nell'ambito dei sistemi rivolti verso l'area della pratica clinica denominata *Comunicazione Aumentativa Alternativa* (Augumentative Alternative Communication AAC). Tale pratica cerca di indirizzare le disabilità verbali verso diverse abilità aventi lo scopo comunicativo, espressivo e relazionale. Essa fa uso di modalità di comunicazione non verbali, infatti, sfruttando *linguaggi alternativi* opera nell'intento di potenziare le risorse comunicative ancora sussistenti nella persona disabile.

Il sistema oggetto del presente lavoro di tesi, demominato *OpenAAC*, prende spunto dall'applicazione Bliss2003 [1] (implementata anch'essa presso il Politecnico di Milano), enfatizzandone gli aspetti che consentono la personalizzazione nei confronti degli utenti fruitori. Lo scopo principale di questa tesi è quindi quello di realizzare un software che renda possibile l'apertura dell'applicazione ad altri linguaggi alternativi oltre al Bliss, nonchè una concreta apertura del sistema rendendolo portabile verso i dispositivi palmari, cellulari o altro.
Lo sviluppo di un'applicazione come OpenAAC e delle applicazioni che lo hanno preceduto, come il Bliss2003, ha un importante ruolo a supporto di utenti con *disabilità verbali e cognitive*; utenti impossibilitati o in difficoltà nell'uso dei consueti canali di comunicazione scritto o orale. Spesso l'utente fruitore di tali applicazioni, ha difficoltà nel comunicare, ma anche in alcuni casi nel comporre correttamente frasi tramite *alfabeti*, *grammatiche* e *sintassi comuni*; per sopperire a tali disabilità la storia ci ha consegnato vari linguaggi alternativi, tra cui i linguaggi *Bliss [2]* e *PCS [3]*. L'utilità dei linguaggi alternativi e dei programmi che li supportano può essere chiarita con un esempio, potremmo infatti provare a fare il processo mentale opposto a quello di *C.K. Blitz [4]* (inventore del linguaggio Bliss). Immaginiamo di appartenere a una grande comunità che, come caratteristica principale ha quella di essere molto vasta e quindi ricca di diversità, tra le quali quella della lingua parlata. In questo modo è possibile comprendere facilmente la necessità di uniformare in qualche modo la comunicazione per potersi far comprendere in modo semplice da chi appartiene alla nostra stessa comunità, nonchè di permettere alla nostra comunità di essere compresa da chi vive un'altra realtà. La soluzione, spunto per la creazione del linguaggio Bliss, trovata dalla comunità cinese è l'adozione di ideogrammi aventi un significato univoco in tutta la Cina ma la cui fonetica differisce da regione a regione.

Un ulteriore esempio lo si può avere osservando il linguaggio matematico i cui ideogrammi o più semplicemente simboli sono internazionali e hanno un'interpretazione univoca in tutto il mondo, pur avendo una fonetica differente a seconda della lingua; per esempio la fonetica associata al simbolo di somma + in Francia è differente da quella associata in Italia, pur rappresentando nei due paesi lo stesso significato.

Linguaggi simbolici come Bliss e PCS non sono altro che un insieme di significati rappresentati in modo figurativo e univoco, che hanno l'importate funzione di porre in comunicazione utenti disabili appartenenti alla stessa comunità nonchè utenti diversamente abili e utenti comunemente abili. In queste due direzioni della comunicazione sono racchiuse le due principale funzioni di programmi di supporto alla comunicazione come OpenAAC.

Per comprendere a pieno l'importanza di un sistema informativo per il supporto alla comunicazione per disabili verbali o cognitivi, basta pensare alla quantità di informazioni che sono oggetto di scambio nella vita quotidiana di una persona. In ogni momento ognuno di noi è fonte e terminale d'informazioni che viaggiano in messaggi composti, seguendo regole di composizione apprese negl'anni su un unico canale informativo sotto varie forme, principalmente voce e scrittura. Provando a eliminare questo bagaglio di conoscenze, dal comporre una frase in modo corretto all'esternarla, possiamo immaginare lo stato di incomunicabilità in cui ci si può trovare avvolti. Restando ben presente la capacità di creare pensieri e la volontà di esternare sensazioni ci si trova impossibilitati nell'essere compresi; un sito [5] recita in una delle sue parti *"non riuscire a parlare non vuol dire non avere niente da dire"***,** frase che da sola evidenzia il grosso e frequente equivoco di cui le persone con disabilità verbale o cognitiva sono vittime.

Il fine primario che si ha nel voler implementare strumenti quali OpenAAC è quindi quello di voler mettere a disposizione della persona con disabilità strumenti che gli permettano di esternare i propri pensieri, le proprie esigenze e le proprie sensazioni senza nessuna barriera comunicativa o espressivo/relazionale e creando così un anello di congiunzione tra la mente e il mondo esterno, in modo da creare una diversa abilità. Questo anello di congiunzione però è da vedere non come rimedio all'impossibilità verbale, bensì come via alternativa, come arricchimento personale e diversità di conoscenza, più che come una mancanza.

Introduzione alle scelte progettuali

Il nucleo dell'applicativo ha il compito di fornire il servizio base di creazione di linee di discussione tramite la semplice selezione di simboli e di fornire ospitalità agli applicativi aggiuntivi.

Un piccolo insieme di *plug-in* di certo sviluppo, vista la loro importanza, sono la *selezione di tabelle utente*, la *selezione dei dizionari di lingua*, la s*elezione dei dizionari di linguaggio*, la *funzione di scansione automatica* e il *supporto alla sintetizzazione vocale*.

Durante la progettazione di OpenAAC è stata prestata attenzione alla struttura dell'applicativo, essa è composta da un nucleo attorno cui *ruotano* differenti *plug-in* i quali hanno lo scopo di fornire servizi opzionali in ausilio all'utente. Per fare ciò essi si integrano con il nucleo dell'applicazione, e interagendo con esso ampliano le funzionalità del sistema.

La struttura *client/server* di OpenAAC permette di porre le basi per lo sviluppo futuro di plug-in che amplino le capacità espressive di questo sistema informativo di ausilio alla comunicazione.
Il requisito base dell'applicazione è unico e basilare, esso consiste nella selezione di simboli da un dizionario, qualsiasi esso sia. Tutto ciò che può essere di ausilio è opzionale. Al centro dell'applicazione è stato quindi posto il nucleo, insieme di funzioni base indispensabili per la comunicazione, e attorno ad esso tutto ciò che in qualche modo fa da automatismo o da personalizzazione dell'applicativo.

Argomenti trattati
Nel *capitolo 2* viene contestualizzato il lavoro di tesi svolto e vengono illustrate le problematiche esistenti in modo da permettere di comprendere facilmente le scelte progettuali effettuate. Nel *capitolo 3*, vengono illustrate tali scelte progettuali, l'implementazione e l'utilizzo di OpenAAC. Nel *capitolo 4* viene illustrata, con un maggior grado di dettaglio, l'architettura dell'applicazione. Nel *capitolo 5*, sono spiegati i *plug-in* implementati e infine nel *capitolo 6*, vengono esposti gli sviluppi futuri dell'applicazione e le conclusioni a cui questo lavoro di tesi ha portato.

2

STATO DELL'ARTE

2.1 Introduzione

E' fondamentale, o se non altro doveroso, effettuare una panoramica sul contesto applicativo in cui OpenAAC trova spazio, in secondo luogo, è altrettanto importante, discutere le soluzioni informatiche presenti per poi poter finalmente introdurre quali soluzioni questa tesi propone nel mondo dei sistemi di AAC. *In altre parole è fondamentale conoscere il passato per comprendere il presente e poter guardare al futuro.*

Durante la fase di progettazione di un sistema qualsiasi, specie nel caso di un'applicazione informatica, risulta fondamentale tenere a mente lo stato dell'arte in cui l'applicazione si deve andare a inserire. In particolar modo tale esigenza si manifesta nello sviluppo di un sistema *client/server* come OpenAAC, in un ambito che, come vedremo, raramente se non mai è stato interessato da sistemi con tale architettura.

In generale l'interesse verso lo stato dell'arte contemporaneo alla progettazione di un nuovo applicativo è importante al fine di poter apprendere, dai sistemi e dalle problematiche esistenti, il contesto in cui l'applicazione dovrà agire in modo da realizzare meglio le funzionalità e i servizi che devono comporre l'applicazione.

Come il progettista di automobili deve conoscere al meglio la tipologia di utenti che usufruirà della macchina che intende progettare e la tipologia di situazioni che l'autovettura dovrà affrontare, anche l'ingegnere del software, che intende progettare un'applicazione per una determinata tipologia di utenti, deve conoscere le problematiche con cui tale applicazione deve confrontarsi, al fine di evitare di fare un applicazione che al posto d'essere d'ausilio diventi d'intralcio.

Per poter conoscere, almeno in minima parte, il contesto in cui OpenAAC deve operare, sono state effettuate alcune ricerche, principalmente tramite *internet* e biblioteche. Tali ricerche sono risultate di estremo interesse e hanno riguardato principalmente tre ambiti: *persone con disabilità verbali e/o cognitive*, *storia dell'AAC* e *sistemi informatici di AAC*.

Con grande rammarico, notevoli difficoltà si sono presentate nella ricerca di informazioni su supporto cartaceo, scarseggia infatti la letteratura a riguardo in Italia. Essa spesso riguarda unicamente la pedagogia e l'educazione. Per contro vi è disponibilità dei dizionari dei linguaggi alternativi quali Bliss e PCS, acquistabili [6] anche in formato cartaceo; tale reperibilità diviene più difficoltosa se si desiderano dizionari arricchiti o strutturati in un formato informatizzato, ad esempio in modalità *database*.

2.2 Persone con disabilità verbali e/o cognitive

Creek, un giovane con paralisi cerebrale, scrive:
"Se volete sapere come ci si sente quando si è impossibilitati a comunicare, andate ad una riunione e fingete di non poter parlare. Usate le mani ma non carta e matita, perché queste generalmente non possono venir usate da persone con gravi disabilità fisiche, impossibilitate a esprimersi verbalmente. Sarete circondati da persone che parlano: che parlano davanti a voi, dietro di voi, intorno a voi, sotto di voi, attraverso voi e anche per voi, ma mai con voi. Voi verrete ignorati finché vi sentirete come un elemento dell'arredamento". [7]
Il fine di riportare questa citazione è quello di porre l'accento su uno dei principali problemi legati alla disabilità: la *ghettizzazione* che il disabile subisce trovandosi al centro di un equivoco di fondo, la superficiale confusione che la società compie nel confondere l'impossibilità di esternare la propria persona, tramite i comuni canali di comunicazione, e l'essere una persona.
Per un animale sociale quale è l'uomo, ritrovarsi a essere ignorati fino a sentirsi come un elemento dell'arredamento è il peggior torto che l'uomo può fare all'uomo.

Condizioni di disabilità
Le condizioni di disabilità che comportano un'impossibilità a più livelli nella comunicazione possono essere molteplici, principalmente di natura congenita, acquisita, evolutiva o temporanea. Di natura congenita sono ad esempio la Paralisi Cerebrale Infantile (P.C.I.) e l'autismo, sindromi genetiche tra cui ad esempio la sindrome di Angelman; di natura acquisita sono ad esempio l'ictus cerebrale, il trauma cranico e l'afasia; di natura evolutiva sono la sclerosi laterale amiotrofica, la sclerosi multipla, il morbo di Parkinson e l'AIDS cerebrale e infine di natura temporanea è il trauma chirurgico (intubazione, tracheotomia).
Una delle condizioni di disabilità tra quelle citate è l'afasia, a testimonianza dell'importanza delle strategie di AAC come approccio a tale problema vengono riportati stralci di un'articolo [8] del settembre 2000.

"Nuovi orientamenti nell'approccio al problema afasia - Oggi è possibile compensare le carenze comunicative delle persone disabili verbali mediante strategie di AAC (Augmentative Alternative Comunication), un nuovo orientamento clinico-riabilitativo-educativo, che considera il problema da un punto di vista globale: relazionale, cognitivo, funzionale. La comunicazione è il prodotto di un insieme di fattori che, integrandosi tra di loro, danno luogo ad un sistema articolato. Per consentire un ripristino comunicativo l'AAC (Augmentative Alternative Comunication), si avvale di tecniche e strumenti che possono potenziare, facilitare o sostituire il linguaggio verbale [...]"

"[...] L'acronimo AAC (Augmentative Alternative Comunication) è stato coniato negli Stati Uniti nel 1983 con la costituzione dell'ISAAC (International Society Augmentative Alternative Communication), un'associazione internazionale, nata per volontà di un gruppo multidisciplinare, che riconosce ad ogni individuo il diritto di comunicare anche in situazioni di grave impedimento verbale. Gli scopi dell'ISAAC sono quelli di informare e diffondere le attuali conoscenze di questo settore (strumenti, codici, metodologie, strategie specifiche). Promuovere studi e ricerche nel campo della disabilità verbale.

Creare una cultura della comunicazione umana con un'accezione più ampia del linguaggio verbale, che comprenda modalità complementari, ma anche alternative alla parola: tutto ciò che consenta, faciliti o potenzi le intenzionalità espressive di un individuo, ivi compresi i residui vocali, lo sguardo, la mimica, i gesti, la postura, i codici alternativi, gli strumenti tecnologici e qualunque ausilio possa essere utile allo scopo. Storicamente l'AAC nasce all'inizio degli Anni 70 per dare possibilità espressive ai bambini cerebrolesi, anartrici e con problemi motori. Infatti, le tabelle di figure, allora in uso, con cui essi potevano esprimere bisogni semplici e primari, non erano sufficienti alle loro esigenze comunicative, emozionali e cognitive. Ciò era possibile solo con l'acquisizione della letto/scrittura. Uno fra i primi codici utilizzati a questo scopo è stato un linguaggio semantografico: i Simboli Bliss. I risultati ottenuti mediante l'utilizzo di questo linguaggio hanno incentivato gli operatori coinvolti nella ricerca e nello studio per la creazione di nuovi linguaggi e per la loro applicazione nel settore tecnologico. Dai primi lavori sui bambini, l'AAC ha oggi trovato applicazione in una popolazione sempre più vasta, comprendente persone con differenti patologie ed età. I sistemi di AAC costituiscono quindi una risposta ai bisogni comunicativi di molti disabili verbali. Anche in assenza della parola e di abilità di letto/scrittura è possibile esprimere i propri pensieri, desideri, emozioni, mediante l'utilizzo di codici alternativi simbolici. [...]"

Parlando di strategie di AAC è importante ribadire che tali strategie non hanno come fine ultimo di sostituire i comuni canali di comunicazione, quali quello verbale, esse infatti in presenza di anche minime attività vocali cercano di potenziarle al fine di poterle integrare con le altre vie di comunicazione possibili. Un importante linguaggio alternativo è di certo il Bliss:

"[...] Simboli Bliss: una scrittura ideografica - Il ripristino della funzione comunicativa può essere indotto mediante diversi linguaggi alternativi non verbali (gestuali) o rappresentati da segni grafici (disegni, figure, simboli). I simboli Bliss sono segni grafici basati sul significato e non sulla fonetica, derivati dalla scrittura ideografica cinese nella quale i significati sono facilmente decodificabili da tutti i cinesi indipendentemente dai loro diversi dialetti parlati. Questa scrittura semantografica è stato ideata nel 1941 dall'ingegnere chimico austriaco C.K. Bliss, con l'obiettivo di creare un linguaggio universale, che consentisse una comunicazione superando le barriere linguistiche. I simboli Bliss sono strutturati in un vocabolario di significati rappresentati con segni grafici (38 simboli + 11 caratteri lineari). La struttura organizzativa del linguaggio considera: cose/persone (1) che compiono azioni (2) in uno spazio (3), in un tempo (4) e in un certo modo (5). Questi 5 elementi (cose, azioni, spazio, tempo, valutazione soggettiva) costituiscono la base per una prima organizzazione dei significati (simboli) al fine di dare senso compiuto ad un messaggio [...]"

2.3 Augmentative Alternative Communication (AAC)

L'Augmentative Alternative Communication (AAC) è una strategia di supporto a persone con varie disabilità nell'ambito della comunicazione, dell'espressione, della relazione o dell'apprendimento. Il fine principale dell'AAC è quello di mettere in condizione persone con tali disabilità di poter ricoprire nel modo più completo possibile un ruolo sociale tipico del genere umano, fornendo degli strumenti alternativi che permettano di interagire, imparare, seguire un percorso educativo e quant'altro con meno barriere e difficoltà.

L’AAC è qualcosa di più di una strategia, nel corso degli anni essa è stata supportata nei suoi intenti da tecniche e tecnologie in grado di poter sfruttare e offrire in modo più semplice e immediato quell’insieme di conoscenze maturate negl’anni in cui tale strategia è stata impiegata.
Il termine AAC, che in italiano potremmo tradurre in CAA, Comunicazione Aumentativa Alternativa, indica modalità di comunicazione atte ad ampliare la comunicazione verbale e non come erroneamente si può pensare modalità atte a sostituire il modo comune di comunicare.
L’AAC, nasce ufficialmente in Nord America nel 1983 con la creazione della International Society of Augmentative and Alternative Communication (I.S.A.A.C.) [9].
I primi a comprenderne le potenzialità sono stati i paesi del Nord America (USA e Canada) e i paesi anglosassoni e del Nord Europa, che già negli anni settanta in via sperimentale, tramite attività di ricerca e riabilitazione, ne facevano uso per facilitare la comunicazione di utenti disabili tramite l’uso di linguaggi grafici, linguaggi gestuali e ausilii frutto degli albori della tecnologia elettronica. Il vantaggio temporale, dovuto appunto alla sperimentazione nei decenni passati, ha portato l’AAC, in questi paesi, a costituire una componente fondamentale e indiscussa dell'intervento riabilitativo. Lo stesso non si può dire dell’Italia che nel campo dell’AAC soffre di un ritardo causato da un interesse verso questa strategia riabilitativa successivo rispetto ai paesi precursori.
Questo fattore non è risultato, e si spera non risulterà, un ostacolo allo sviluppo di migliori tecnologie inerenti l’AAC, grandi esempi sono certamente il Bliss2000 [10] e il successivo Bliss2003, implementati presso il Politecnico di Milano, nonchè gli altri software di AAC proposti da società estere e italiane.
Un ruolo importante nella diffusione dell’uso di strategie riabilitative, proprie dell’AAC, lo hanno ricoperto gli elaboratori elettronici, i quali hanno goduto e godono tuttora di un importante considerazione nella risoluzione dei problemi di comunicazione, apprendimento e integrazione sociale del grave disabile, in particolar modo nei disabili motori. I motivi di tale importanza nel campo riabilititavo sono legati principalmente alla difficoltà di persone con disabilità nell’usare linguaggi grafici pittografici per la selezione di un simbolo per volta in tabelle fisiche di grandissime dimensioni, nonchè alla necessità di costruire tabelle personalizzate e di espandare i dizionari grafici introducendo nuovi simboli.
A scapito dell’importanza dell’uso dei computer, sono presenti ancora in Italia notevoli vuoti a riguardo della cultura dell’AAC, cioè di tutto quell’insieme di teorie e metodologie di analisi cha fanno, o quanto meno dovrebbero fare, da fondamenta all’uso di tali strategie.
Come dire che per ora in Italia ci sono i mezzi tecnici ma scarseggiano i testi e le pubblicazioni a riguardo, notevoli sono infatti le difficoltà che si incontrano nella ricerca di testi specifici sull’AAC e su argomenti ad esso correlati in lingua italiana. In molti casi sono maggiori le risposte date dalle rete di quelle date dalla carta, se si pensa che si sta parlando di strategie che hanno visto la luce dai venti ai trent’anni fa, si comprende bene il paradosso.

2.4 Sistemi informatici di AAC

In questa sezione saranno presentati brevemente alcuni dei principali software di comunicazione attualmente in commercio. Le descrizioni qui riportate sono tratte da siti internet delle società che li commercializzano, al fine di riportare integralmente e in maniera corretta le specifiche dei software citati.

Comunica con simboli PCS prodotto da Easy Labs [11]

Comunica con simboli PCS è un software di comunicazione con uscita vocale.
Include la libreria completa dei simboli PCS. Attualmente vi sono 4800 simboli, a colori ed in bianco/nero, tradotti in italiano e con ordinamento per categorie e alfabetico.
Permette di creare tabelle di comunicazione personalizzate, da usare dinamicamente sul computer, con possibilità di uscita in voce digitalizzata.
In breve queste sono le potenzialità del prodotto della Easy Labs:

- costruire delle tabelle di comunicazione attraverso la creazione di celle, modificabili per quantità, dimensioni e disposizione (facilità di modifica delle celle)
- inserire in ogni cella un simbolo PCS (Libreria PCS), un'immagine o una fotografia scannerizzata, un disegno, e di associare un testo
- associare a ogni cella un file audio personalizzato
- avere la predizione del contenuto di una cella/tabella (Anteprima audio) con un messaggio personalizzato
- definire una struttura ad albero tra le varie tabelle di uno stesso file
- avere illimitate possibilità di aggiunta di tabelle collegabili tra loro, ed anche del numero di celle contenute nelle tabelle. L'unica limitazione potrebbe arrivare, dalle caratteristiche del computer in uso.
- utilizzare diversi dispositivi di accesso, quali Tastiera Mouse Sensori
- utilizzare diverse modalità di accesso, quali selezione diretta e indiretta: scansione manuale ed automatica con la possibilità di definire i tempi di scansione e d'intervallo tra i due input successivi
- offrire un rinforzo visivo all'utente attraverso l'uso di una barra frasica, nella quale viene riportato ogni simbolo selezionato, e la possibilità di stamparne il contenuto.

Speaking Dynamically prodotto da Mayer-Johnson [12]

Speaking Dynamically è un programma per costruire sistemi di comunicazione personalizzati, da utilizzare sul personal computer, da tavolo o portatile. Speaking Dynamically è dedicato particolarmente a quegli utenti che impiegano simbologie grafiche come mezzo alternativo di comunicazione e può essere usato con mouse (o emulatori), touch screen o a scansione tramite sensori.
ùNelle tabelle sono integrabili simboli PCS (non forniti col programma) e qualunque altra immagine proveniente da Cd, Internet, ripresa con scanner o con macchina fotografica digitale.
Ad ogni simbolo-immagine è possibile associare:
• un messaggio ripetuto dalla sintesi vocale
• oppure un messaggio pre-registrato
Il messaggio composto dall’utente può essere anche essere scritto in una apposita zona del monitor e poi stampato.

Il programma permette di realizzare sistemi articolati di comunicazione attraverso i quali l'utente può comporre frasi e messaggi facendo uso di più tabelle di simboli collegate fra loro. Vediamo alcuni esempi.
Dalla tabella principale per la scelta delle attività si passa a specifiche tabelle secondarie. Quando l'utente seleziona, ad esempio, il simbolo di "GIOCARE" si apre la tabella secondaria che permette di scegliere il tipo di gioco (giocare ai pompieri, con le macchinine, coi Pokemon, con l'album dei dinosauri, coi Lego, alla guerra).
In questo esempio, la tabella principale presenta sei categorie dei simboli contenuti nel sistema di comunicazione (persone, tempo, azioni, luoghi, oggetti, aggettivi). Quando l'utente seleziona la categoria "TEMPO" accede ad una tabella coi simboli di "Ieri", "Oggi", "Domani", "Il mese scorso", "Il mese prossimo", "Nel fine settimana", "Il giorno del compleanno", etc.
La tabella principale può contenere non solo i collegamenti del sistema di comunicazione, ma anche icone per accedere ad altro materiale presente sul computer: ad esempio, un album di fotografie, un programma di gioco, etc.
Le tabelle possono essere costruite in modo completamente personalizzato o possono essere ricavate adattando gli oltre 90 esempi pronti.
Speaking Dynamically può anche essere usato con tabelle alfabetiche per comporre messaggi in forma scritta.

Optimist II prodotto da ZYGO Industries Inc [13]
E' un ausilio dedicato alla comunicazione aumentativa e alternativa (AAC) basato su un personal computer portatile di dimensioni e peso molto contenuti.
Grazie alla possibilità di utilizzare la sintesi vocale e il grande schermo LCD sensibile al tocco, Optimist può essere usato nella vita di tutti i giorni come supporto alla comunicazione verbale.
Le sue dimensioni ridotte, l'alimentazione a batteria completamente autonoma e i diversi dispositivi di fissaggio disponibili rendono Optimist II un utile supporto alla comunicazione durante la vita di tutti i giorni, specialmente fuori dalla propria casa, per tutti gli utilizzatori la cui capacità di scambio verbale risulti compromessa.
La possibilità di utilizzare sensori e interruttori di vario genere lo rende utilizzabile anche in presenza di limitazioni più o meno gravi delle capacità motorie.

PowPow prodotto da Tribal Voice
Questo software, il cui nome e simboli iconici traggono origine dalla cultura degli indiani d'America, appartiene alla famiglia di quelli che oggi si definiscono programmi di instant messaging, che permettono cioè di scambiare immediatamente messaggi di testo, flussi di informazione vocale, immagini e icone con altri utenti collegati in quel momento in rete.
Anche se oramai i software di questo genere sono molto famosi (come ad esempio Aol Messenger, ICQ, Yahoo! Messenger) PowWow è stato uno fra i primi programmi di comunicazione aumentata ad essere realizzato e offerto agli utenti di Internet già diversi anni fa, quando l'unica forma di comunicazione in 'tempo reale' era rappresentata dal servizio di chat testuale.
PowWow permette infatti, sin dalle sue prime versioni, di contattare il server centrale di registrazione e quello che contiene l'anagrafica degli utenti (la buddy list, letteralmente l'elenco degli amiconi) per permettere a tutti di partecipare a conferenze più o meno

affollate scambiando testi, suoni, immagini, indirizzi di risorse (Url), semplici disegni grafici creati sul momento.
L'architettura del software permette sia la comunicazione in un gruppo, ovvero la partecipazione ad una community nel gergo di PowWow, sia il dialogo uno a uno fra due utenti, sempre per mezzo dei server centrali di comunicazione che rendono possibile lo scambio di informazioni tra i vari client.

Vantage prodotto da Prentrom [15]
Vantage è un dispositivo portatile di supporto alla comunicazione. Si basa su un'interfaccia a icone e utilizza un sistema di sintesi vocale proprietario per la pronuncia del testo generato.
Il software del dispositivo utilizza metodi di predizione della parola per facilitare e rendere più veloce la composizione delle frasi.

I software citati sono solo alcuni dei molteplici presenti sul mercato, essi hanno svariate caratteristiche comuni, come la selezione di simboli da tabelle grafiche, o la sintesi vocale; alcuni sono puramente software, quindi da utilizzare con un pc che risponda ai requisiti minimi, mentre altri sono ospitati da sistemi progettati *ad-hoc* (es. *Vantage*). Ognuno di essi però presenta delle caratteristiche peculiari tali da fornire importanti spunti per lo sviluppo, sia del nucleo dell'OpenAAC che, in particolar modo, per lo sviluppo dei servizi aggiuntivi (*plug-in*) di corredo all'OpenAAC stesso.
Tali servizi riguardano la predizione dei possibili simboli oggetto di selezione, la personalizzazione delle tabelle e dei simboli o ancora un servizio di chat tramite linguaggi grafici quali Bliss, PCS o quant'altro. L'intento perseguito è quello di creare un'applicazione in grado di fornire molteplici servizi, ponendo l'applicazione in condizione da potersi adattare alle molteplici esigenze dei suoi utenti, senza che essi debbano adattare le proprie esigenze ai diversi applicativi esistenti.

2.5 Il predecessore di OpenAAC: Bliss2003

L'applicazione da cui è partita la progettazione di OpenAAC è Bliss2003, software oggetto di tesi di due studenti del Politecnico di Milano nell'anno accademico 2002/2003. Bliss2003 in unica applicazione permette all'utente di compiere molteplici funzioni, dalla semplice selezione di simboli alla riproduzione vocale delle selezioni.
L'uso di Bliss2003 è vincolato strutturalmente, tali vincoli costringono a una selezione degli utenti fruitori del servizio, uno di questi vincoli è l'utilizzo di Bliss come unico linguaggio simbolico. Tale scelta costringe utenti aventi differente bagaglio culturale ad utilizzare altri supporti di ausilio, inoltre mette a disposizione un numero limitato di lingue per la traduzione dei significati e dei menù dell'applicazione, costringendo l'impiego dell'applicativo solo in determinate aree geografiche, o ancora dispone di un numero di funzioni limitato la cui modifica o espansione comporta interventi sul blocco monolitico dell'applicazione.
Nei sistemi di AAC quali, il Bliss2003 e il suo precedessore il Bliss2000, strutture di tipo client/server sono assenti, come assente è la prerogativa di un sistema distribuito atto a integrare diverse funzioni aventi la finalità di migliorare la qualità del supporto alla comunicazione e di conferire al sistema elasticità tale, da potersi adattare facilmente alla tipologia di utente utilizzatore.

2.6 OpenAAC e il futuro

La presentazione delle applicazioni AAC in commercio effettuata nel *paragrafo 2.4* ha lo scopo di mettere in risalto gli obiettivi dello sviluppo di OpenAAC, nonchè le sue potenzialità.
Il punto focale di ogni applicazione AAC, come si è potuto vedere, è la comunicazione, tutto ciò che è di ausilio alla comunicazione deve convergere inevitabilmente verso questo punto.
Le applicazioni attualmente disponibili convergono secondo vari passaggi verso tale punto, spesso seguendo percorsi comuni. Nessuna di esse però fornisce un insieme completo di percorsi e tanto meno da la possibilità di integrare nuovi percorsi, facendo così una selezione a valle degli utenti fruitori. Le soluzioni adottate nel corso della progettazione dell'OpenAAC permettono all'applicazione di non porre tali vincoli d'utilizzo, e nel caso in cui un vincolo di questo genere si venisse a creare per una determinata categoria di utenti, l'applicazione è aperta ad essere adattata alle particolari esigenze degli utenti.

Aspetti innovativi introdotti da questo lavoro di tesi

L'aspetto innovativo che questo progetto vuole introdurre, in applicazioni d'ausilio alla comunicazione, è quello di mettere in condizione il sistema di poter ospitare in qualsiasi momento una nuova funzionalità, creata *ad hoc* per risolvere una determinata problematica che l'applicativo così come è sviluppato in un dato momento non permette di risolvere. Probabilmente unendo tutti gli applicativi di AAC finora creati in un ipotetico *macro applicativo* otterremmo l'applicazione AAC *ideale*, risulta chiaro che questo non può avvenire però così semplicemente. OpenAAC parte proprio con questa prerogativa, vuol porre le basi per la creazione di un applicativo in grado di poter accogliere le varie forme di ausilio alla comunicazione, che fanno uso della selezione di simboli grafici da un dizionario personalizzabile.

Tali esigenze richiedono quindi che l'applicazione sia *flessibile*, aspetto assente nella quasi totalità delle applicazioni esistenti.
Lo sviluppo dell'OpenAAC si può paragonare al centro di più circonferenze concentriche che con il passare del tempo si andranno tracciando permettendo lo sviluppo dell'applicazione AAC in ogni direzione. Ciò che ha portato allo sviluppo di OpenAAC tramite un architettura *client/server* è la volontà di creare un'applicazione che prosegua il suo cammino verso l'ampliamento delle sue funzionalità nel futuro, ma non seguendo un percorso lineare, bensì espandendosi dal centro verso l'esterno in ogni direzione, in modo da poter integrare in sé diverse funzionalità. Lo studio che seguiterà l'attuale implementazione di OpenAAC riguarderà la ricerca delle particolari situazioni in cui l'uomo si trova a doversi adattare in maniera forte alla macchina per poterne sfruttare le potenzialità. Tale studio avrà lo scopo di *ribaltare* tali situazioni implementando funzionalità opzionali al programma che risolvano tali problematiche. Queste particolari situazioni possono riguardare disabilità paricolari, o particolari combinazioni di disabilità che per qualche motivo rendono vane alcune funzionalità dell'applicazione, come per esempio l'uso di particolari periferiche di input che si adattino alle caratteristiche dell'utente fruitore, o l'uso di particolari linguaggi grafici, noti all'utente o che in qualche misura gli siano di particolare ausilio rispetto a quelli *standard*, anche se di standard spesso in AAC non è possibile parlare.

3

PROGETTAZIONE, IMPLEMENTAZIONE E UTILIZZO DI OPENAAC

3.1 Introduzione

In questo capitolo sarà presentata la struttura dell'applicazione OpenAAC e del suo nucleo, inteso come insieme dei servizi base attorno cui ruotano tutti gli ulteriori servizi di AAC dell'applicazione. OpenAAC in modalità base, cioè senza alcun plug-in connesso, sfrutta unicamente il nucleo dell'applicazione che è in grado di funzionare autonomamente fornendo l'insieme di servizi base richiesti a un'applicazione di AAC.
Il nucleo dell'applicazione implementa tutti i servizi base del *software*, quali l'*interfaccia grafica*, il *caricamento dei dizionari grafici*, il *motore di navigazione delle tabelle* e *la selezione dei simboli*.
Il numero limitato di servizi offerti dal nucleo fa parte delle scelte effettuate in fase di progettazione dell'applicativo, esso infatti deve avere due caratteristiche principali, essere essenziale e aperto verso l'esterno.
L'*essenzialità* è dettata dalla volontà di poter rendere un oggetto personalizzabile; per questo si devono porre il minor numero di vincoli in modo da poter avere il massimo della flessibilità e adattabilità verso il mondo esterno.
L'*apertura* verso l'esterno è la caratteristica che permette di mettere a frutto la precedente scelta strutturale, tale apertura infatti rende possibile l'arricchimento dell'applicazione tramite l'interfacciamento di servizi esterni, utili, ma non indispensabili, come i servizi appartenenti al nucleo.
Oggetto di discussione in questo capitolo è la struttura del nucleo dell'applicazione, nonchè le scelte progettuali che hanno portato alla sua implementazione.

3.2 Tecnologie adottate per la progettazione e l'implementazione

Le tecnologie che hanno reso possibile lo sviluppo e l'implementazione di OpenAAC sono: un *tool* di *design* UML (MagicDraw UML [16]) e l'*editor* della Borland per il linguaggio C++ (Borland C++ Builder [17]); per la stesura della tesi e la sua presentazione sono stati usati Microsoft Word [18], Microsoft PowerPoint e Acrobat Writer [19].

Design

Tramite il tool di design UML è stato possibile realizzare diagrammi a vari livelli di dettaglio riguardanti la struttura e l'implementazione dell'applicazione con il fine di presentare, in modo astratto, concetti spesso prettamente tecnici dell'informatica e dell'ingegneria del software.

La scelta del linguaggio *semi-formale* UML è stata quasi forzata dalla posizione di standard di fatto che tale linguaggio descrittivo ha saputo conquistarsi negl'anni, quale mezzo di descrizione di architetture, funzionalità, nonchè per l'esposizione di casi d'uso delle applicazioni.

Il ruolo di tale tool di design risulta di fondamentale importanza nella fase di produzione della documentazione di corredo alla progettazione del software. Tale documentazione costituisce uno degli anelli fondamentali nell'ingegneria del software, utile per presentare quanto è stato prodotto sia in maniera approfondita dal punto di vista tecnico sia in maniera superficiale dal punto di vista puramente esplicativo.

La documentazione prodotta tramite diagrammi e schemi deve essenzialmente esporre gli aspetti fondamentali di un progetto in modo da far comprendere al meglio come e cosa fa una determinata applicazione, su quale struttura si erge, quali architetture sono state adottate nonchè coordinare un eventuale lavoro di squadra contemporaneo o futuro.

Nel caso dell'OpenAAC la documentazione prodotta ha l'importate scopo di fare da memoria storica di quanto è stato prodotto a vantaggio di chi nel futuro dovesse lavorare per ampliare il progetto OpenAAC, con l'integrazione di nuove applicazioni satellite o effettuando migliorie al nucleo dell'applicazione.

Implementazione

Per l'implementazione dell'applicazione è stato adottato il linguaggio C++, tramite l'uso dell'editor Borland C++ Builder; al fine di ottenere un'applicazione il più portabile possibile sono state utilizzate librerie di oggetti standard. La scelta di tale linguaggio di programmazione è stata conseguente a una serie di valutazioni che hanno coinvolto vari linguaggi di programmazione, quali Java, ideale per l'obiettivo della portabilità e .Net della Microsoft, anch'esso ideale sulla carta per la portabilità e l'efficienza, tali valutazioni però hanno portato alla conclusione di effettuare una scelta di robustezza rivolta all'utilizzo del C++ e di libreria per lo più standard lasciando la portabilità alla fase di compilazione dell'applicazione.

3.3 Processo d'ingegnerizzazione di OpenAAC

Come più volte ribadito il progetto OpenAAC ha come punto di riferimento il Bliss2003, fonte d'ispirazione principalmente nella ricerca di migliorie nell'ambito dell'interfaccia grafica e della navigabilità delle tabelle.
Nell'intento di apportare miglioramenti nella visualizzazione a video delle informazioni, l'impostazione grafica data a OpenAAC presenta un numero limitato di colori (per lo più standard) e un numero limitato di sezioni (due nella modalità *base*), la visualizzazione delle sezioni è affidata alla selezione della cartella corrispondente.
Il fine perseguito nella creazione dell'interfaccia grafica è stato quello della semplicità d'utilizzo da parte di utenti per la maggior parte dei casi non esperti (persone con disabilità e terapisti).

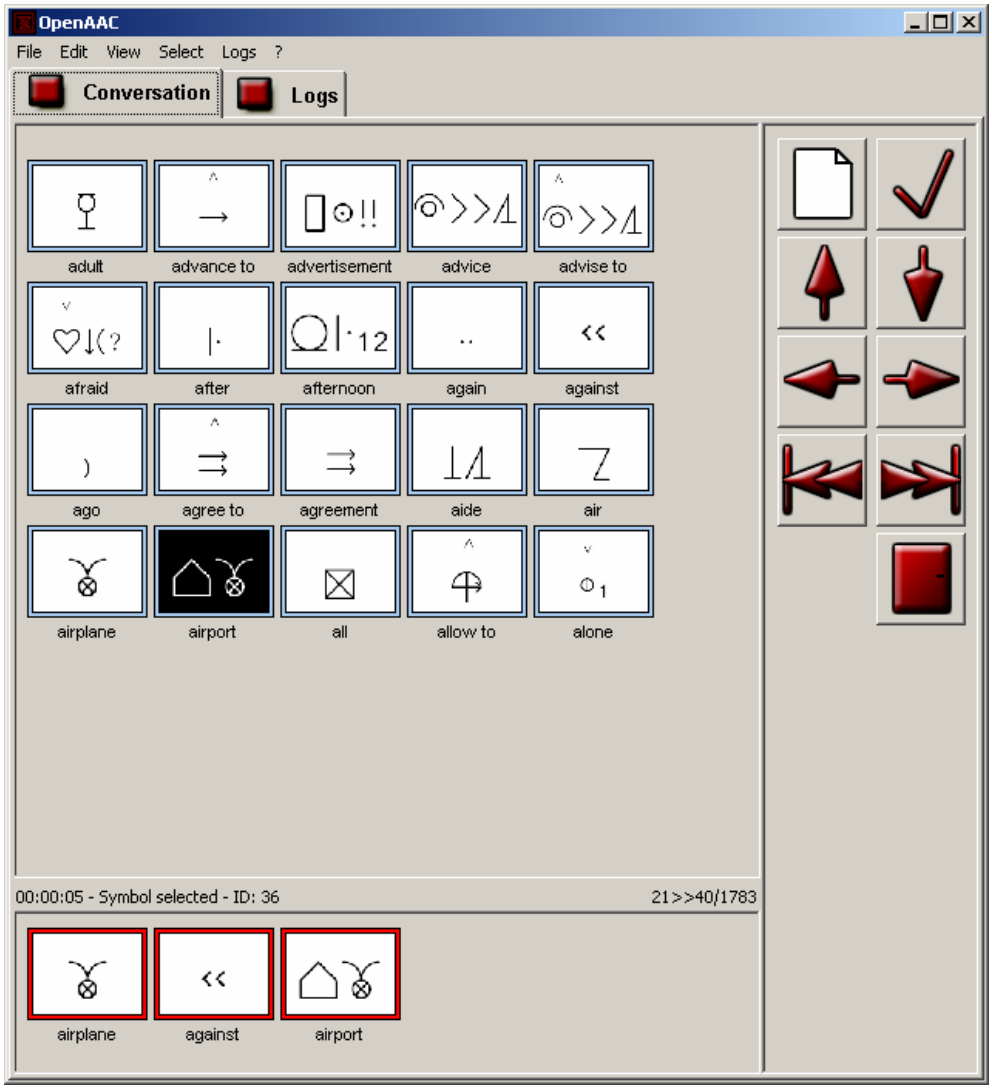

fig. 3.3.1 – OpenAAC in modalità *base*

Quanto ha portato alla creazione di OpenAAC, più che un processo di reingegnerizzazione, è stato un processo di ingegnerizzazione con un occhio a quanto è stato fatto nel passato, con l'intento di migliorare gli aspetti comuni tra il vecchio e il nuovo. La scelta di *abbandonare*, se così si può dire, l'idea di una vera e propria reigegnerizzazione del Bliss2003 è stata effettuata nelle prime fasi di analisi dell'applicazione esistente e dell'applicazione che si intendeva implementare, questo a seguito di un evidente diversità di struttura tra il vecchio e il nuovo che rendeva *impossibile ristrutturare* ciò che già esisteva senza sconvolgerne completamente la struttura.
A seguito di queste conclusioni si è deciso di partire nuovamente dalle fondamenta adottando una metodologia che nell'ingegneria del software è detta *bottom up.* Tale metodologia prevede di partire a progettare le soluzioni a un problema dal dettaglio, cercando di comprendere quali siano le singole implementazioni che soddisfano la specifica di progetto.
La tecnica del *bottom up,* è stata applicata unicamente alla fase di progettazione, infatti nella prima fase, quella di analisi dei requisiti e di stesura della specifica è stato considerato più efficace un differente approccio detto *top down.* Tale approccio prevede di partire da una vista generale per arrivare in ultima analisi ad analizzare le singole problematiche.
Il Bliss2003 in questo procedimento è risultato di notevole aiuto in quanto ha permesso di avere immediatamente un vista generale di quali siano le specifiche di un programma di ausilio alla comunicazione. Tramite tale vista è stato possibile porre l'accento sulle aspettative di un programma di questo genere, nonchè di comprendere quali fossero le problematiche esistenti allo stato dell'arte.
La struttura data a OpenAAC rispecchia la metodologia con cui è stato implementato, adottando un approccio di tipo *bottom up* all'implementazione di un applicativo, esso si va componendo per passi successivi, in cui ogni singola funzione viene creata in modo indipendente o semi-indipendente da ciò che è stato già implementato. L'applicazione finale risulta quindi essere un mosaico, le cui tessere sono le singole funzionalità. In questa ottica è facile comprendere come per estendere tale mosaico basti aggiungere delle tessere, proprio in questo modo OpenAAC permette di estendere le sue funzionalità; la sua architettura, come vedremo, permette di ospitare e integrare le nuove funzioni implementate.
La prima funzione implementata durante le fasi di progettazione è la selezione dei simboli da un dizionario bliss, tale selezione avviene tramite una tabella contenente i simboli del dizionario. La progettazione è proseguita iterando sempre la stessa domanda: *Quale funzionalità implementare ora?*
Tale domanda ha avuto come risposta l'insieme di funzioni che costituiscono il nucleo di e i *plug-in* dell'applicazione.
La *fig. 3.3.2* schematizza tale processo d'ingegnerizzazione, mostrando alcune delle funzioni implementate. A tal proposito è importante tener presente che ognuna delle singole funzioni illustrate in figura è il risultato di un procedimento analogo, esse sono infatti l'unione di singole funzioni create al fine di fornire i servizi caratterizzanti la macro funzione da esse composta.

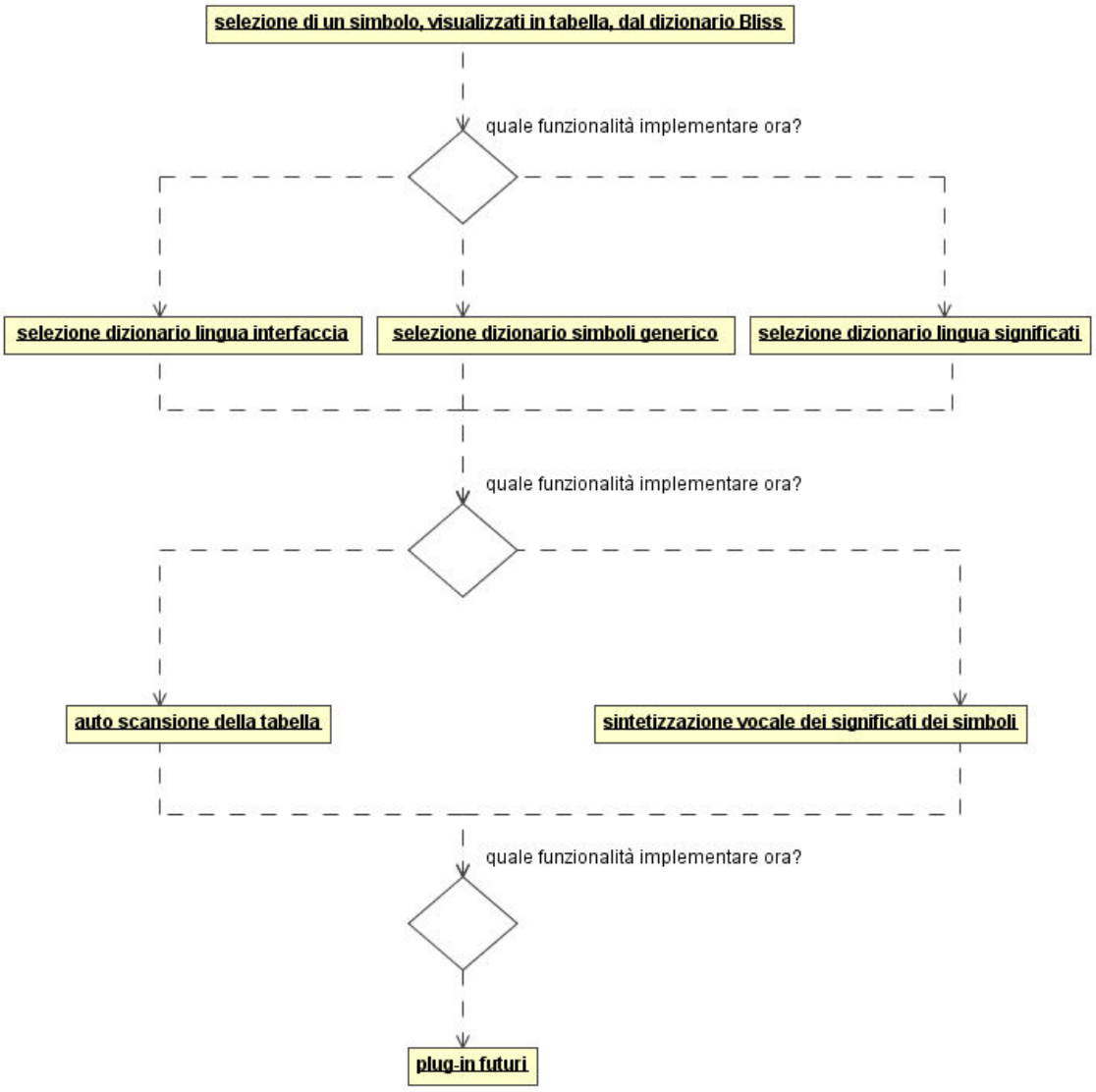

fig. 3.3.2 – procedimento bottom-up

Un esempio a tale proposito sono i servizi di seguito elencati, essi costituiscono il nucleo dell'applicativo attorno a cui ruotano tutti i servizi di ausilio alla comunicazione. Tali servizi sono stati creati in modo indipendente l'uno dall'altro e poi fatti collaborare in modo da ottenere il risultato auspicato.

I servizi caratterizzanti del nucleo sono:

1. *Visualizzazione di contenuti informativi tramite linguaggi grafici, in un formato accessibile cioè comprensibile e interpretabile dall'utente con disabilità verbale o cognitiva*
2. *Visualizzazione dei significati dei simboli grafici in linguaggio comune, interpretabile quindi anche da chi non è in grado di comprendere un linguaggio grafico*
3. *Navigabilità delle tabelle grafiche tramite comuni periferiche di input, quali tastiera o mouse*

4. *Componibilità di frasi tramite la selezione dei simboli grafici, resa possibile dalla navibilità delle tabelle stesse*
5. *Creazione di due registri di log con il fine di memorizzare le sessioni e le frasi composte. Tali registri in futuro potranno fornire all'applicazioni importanti informazioni sull'utente, tali da permettere all'applicativo di auto apprendere il suo comportamento*

Come vedremo tale architettura fa capo a una struttura nota nell'ambito informatico, argomento di discussione nel *capitolo 4*.

3.4 Struttura e implementazione

Con l'ausilio della *fig. 3.4.1*, in questo paragrafo viene illustrata la struttura del progetto OpenAAC e le classi che lo compongono, la cui implementazione ha portato alla creazione del nucleo dell'applicazione e dei *plug-in*.

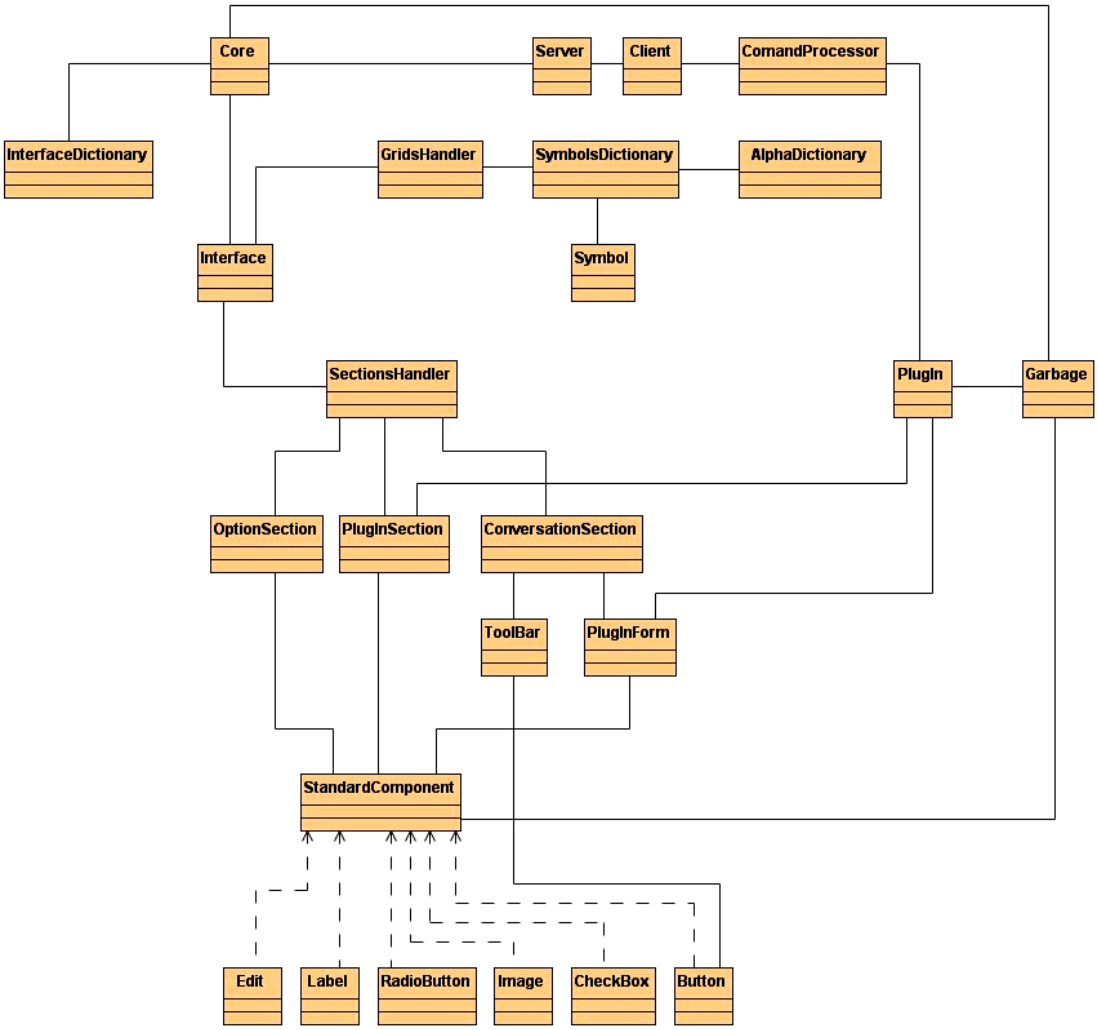

fig. 3.4.1 – *class diagram* del nucleo di OpenAAC

L'applicazione è composta da un nucleo denominato *Core* che ha il compito di istanziare e coordinare tutte le operazioni all'interno del programma. Il nucleo istanzia direttamente: l'interfaccia grafica *Interface*, i dizionari delle voci d'interfaccia *InterafaceDictionary*, i dizionari dei simboli grafici *SymbolsDictionary*, un gestore dei componenti grafici creati dinamicamente, *Garbage*, e il *Server*.
L'interfaccia grafica, come è facile intendere, ha il ruolo di *front-end* tra l'utente e il motore applicativo. L'oggetto *Interface* istanzia un gestore di sezioni denominato *SectionsHandler* che ha il compito di istanziare e gestire le sezioni costituenti l'applicativo, tramite esso è infatti possibile selezionare le sezioni d'interesse, esso mette in condizione il nucleo di creare dinamicamente una nuova sezione su richiesta dei *plug-in*. Nella modalità *base* l'applicativo, per le motivazioni già note, è diviso in sole due sezione, quella di conversazione, *ConversationSection,* e quella di log, *LogsSection.* Il numero di sezioni che il programma può ospitare non ha limiti in modalità distribuita, cioè connettendo *plug-in* di vario genere, infatti il nucleo mette a disposizione delle *applicazioni satellite* lo spazio per la creazione di al più una propria sezione, che avrà il titolo scelto dal *plug-in*, tale tipologia di sezioni è denominato *PlugInSection*.
La *ConversationSection* è composta da componenti standard, *StandardComponent,* da una barra degli strumenti, *ToolBar,* e da eventuali *Form*. Tramite i componenti standard vengono visualizzati i simboli, disposti in una tabella gestita dal componente *GridsHandler,* e il cursore di navigazione. Tramite la *ToolBar* vengono rese disponibili all'utente richiami veloci alle funzioni del programma altrimenti richiamabili tramite il *menu* delle funzioni. La *ToolBar*, come si evince dalla *fig. 3.4.1,* è composta unicamente da bottoni, *Button*, che nella modalità *base* corrispondono alle funzioni base del nucleo, mentre nella modalità distribuita possono richiamare anche funzioni implementate esternamente a esso dato che ogni *plug-in* può creare un numero a sua scelta di bottoni nella *ToolBar* per mettere a disposizione dell'utente le proprie funzionalità. Nella creazione dei bottoni, a differenza delle sezioni, i *plug-in* non hanno vincoli di massimo, essi stanno solo al buon senso di chi implementa il *plug-in*.
La *ConversationSection*, in modalità *client/server*, può contenere delle *form*, assenti nella modalità *base,* esse vengono create unicamente dai *plug-in*, i quali ne possono istanziare al più una. All'interno delle *form*, lo sviluppatore del *plug-in*, può far istanziare al nucleo l'ambiente che desidera, con il solo vincolo di utilizzare *StandardComponent*.
La *LogsSection*, come anticipato contiene i due log dell'applicazione, essi hanno l'onere di registrare le operazioni compiute dall'applicazione e dall'utente, più precisamente uno contiene tutti gli eventi dell'applicazione, mentre l'altro i significati della frase creata dall'utente tramite la selezione dei simboli.
A riguardo della *PlugInSection* vale quanto detto per le *form* istanziabili dal *plug-in* nella *ConversationSection*, hanno infatti le stesse caratteristiche e gli stessi vincoli. L'uso di *PlugInSection* è il miglior modo per integrare l'interfaccia di un generico *plug-in* con quella di OpenAAC.
Sostenendo una continuità logica vediamo ora la funzionalità del dizionario d'interfaccia *InterfaceDictionary*: esso ha lo scopo di rendere disponibile l'utilizzo dell'applicativo a persone di diverse nazionalità, tramite questo dizionario OpenAAC permette di visualizzare i menù e i log applicativi in diverse lingue, nella modalità *base* la lingua d'interfaccia è l'inglese.

La comunicazione, obiettivo primario

Per proseguire nella descrizione della struttura dell'applicazione è doveroso intrudurre un concetto che sta alla base del concepimento dell'architettura di OpenAAC. Volendo riassumere in una frase il ruolo del nucleo nell'applicazione e il rapporto che esso ha con i *plug-in* possiamo dire che: "*Il nucleo esprime un potenziale, cioè la possibilità di compiere un'azione, al plug-in è assegnato il compito di sfruttare il potenziale seguendo un proprio fine*"; nel caso dell'interfaccia grafica, il potenziale consiste nel poter creare componenti in modalità remota, nel caso del dizionario d'interfaccia permettere al *plug-in* specifico di aggiungere dizionari interfaccia all'applicazione e quant'altro possa in qualche modo automatizzare o personalizzare le funzionalità del nucleo di OpenAAC.

In maniera del tutto simile all'*InterfaceDictionary*, il *SymbolsDitionary* raccoglie un insieme di simboli appartenenti ad un linguaggio grafico, nella modalità *base* il linguaggio adottatto è unicamente il *Bliss*, mettendoli a disposizione del nucleo e dei plug-in. Il potenziale espresso dal *SymbolDictionary* è forse quello di maggiore importanza tra tutti quelli esprimibili dal nucleo del OpenAAC, esso infatti rende possibile l'utilizzo dei più svariati linguaggi grafici esistenti permettendo di andare oltre l'utilizzo del Bliss o comunque di un unico linguaggio. Il potenziale espresso da questa caratteristica di OpenAAC si esprime grazie all'implementazione di uno specifico *plug-in* che permette di utilizzare un qualsiasi linguaggio grafico, quale ad esempio PCS, PicSIMS o Rebus.

Ad ogni *SymbolsDictionary* inoltre possono essere assiociati uno o più dizionari di significati *AlphaDictionary*, ad esempio nella modalità *base* ai simboli Bliss è associato il dizionario di lingua inglese con il fine di comprendere il significato dei simboli anche a chi non conosce i linguaggi grafici.

Il gestore dei componenti creati dinamicamente, denominato *Garbage*, così presentato può sembrare un oggetto oscuro, esso non è altro che un componente che indicizza i componenti creati dinamicamente nel corso dell'interazione del programma con l'utente e ne gestisce lo smaltimento in maniera ottimale. Nella modalità *base* il programma fa uso dell'oggetto *Garbage* in due contesti, nello smaltimento dei simboli visualizzati a schermo, reso necessario, ad esempio in seguito al cambiamento di pagina nella navigazione del dizionario e nello smaltimento della linea di discussione, per esempio nell'inizio di una nuova sessione. Per smaltimento si intende la cancellazione dalla memoria, e conseguentemente dallo schermo dell'oggetto *collezionato* dal garbage. La sua funzione è quindi quella di indicizzare tutti gli elementi dinamici presenti nell'interfaccia e permettere di eliminarli quando richiesto. Se questo componente non fosse presente, l'applicazione farebbe un uso scorretto della memoria, *colmandola* di istanze di oggetti non più visualizzati o utilizzati.

Il ruolo del *Garbage* ha una notevole importanza anche nella modalità distribuita, esso infatti viene istanziato da ogni *plug-in* che si integra con l'applicazione per gestire i componenti remoti creati e il loro smaltimento necessario ad esempio alla chiusura di un *plug-in* o in altri contesti definiti in fase di programmazione del *plug-in* stesso.

L'utilizzo in questa modalità esemplifica meglio quanto esposto in via teorica; immaginando solo per un attimo di non adottare un componente simile, ogni volta che un *plug-in* connesso all'applicazione viene chiuso, in memoria resterebbero i componenti da esso istanziati. L'uso dei *Garbage* ha quindi il compito di rendere efficiente l'uso della

memoria, al fine di non creare sprechi e ridondanze, utili nell'ottica di trasportare l'applicativo su cellulari, palmari, piccoli terminali o dispositivi ad-hoc.
Il ruolo ricoperto dal *Server* è il più incisivo nell'aprire il sistema di ausilio alla comunicazione verso il mondo esterno. Esso permette di aprire canali d'interfacciamento, facendo da tramite per lo scambio d'informazioni e comandi tra il nucleo e le applicazioni satellite. Una discussione approfondita sulla struttura client/server è oggetto del *capitolo 4*, in questo di capitolo l'interesse è maggiormente rivolto al suo ruolo all'interno del nucleo.
Il nucleo possiede uno e un solo *server*, il quale può relazionarsi con una molteplicità di client. Ad ogni client, al momento della sua connessione con il server, viene assegnato uno e un solo processore di comandi, *ComandProcessor*, che ha il compito di interpretare i comandi in ingresso utilizzando il relativo protocollo. Tramite i comandi del protocollo *server* il generico *plug-in* è in grado di istanziare da remoto componenti dell'interfaccia grafica, tali componenti sono gestiti dalla classe *PlugIn*, tali istanze devono chiaramente rispettare i vincoli esposti nella discussione dell'interfaccia grafica.
Come vedremo i comandi a disposizione dei *plug-in* per integrarsi con il nucleo, non riguardano unicamente la creazione di componenti remoti, ma anche l'utilizzo di funzioni proprie del nucleo e la richiesta d'informazioni proprie dell'applicazione, il tutto tramite il *ComandProcessor*.

Sequence diagram
Dopo questa vista *panoramica* sulla struttura del nucleo dell'applicazione vengono ora descritte in dettaglio alcune sequenze di operazioni svolte dal nucleo. Tali descrizioni sono supportate da diagrammi UML, principalmente *sequence diagram*, utili nello schematizzare una sequenza di operazioni. Questa seconda fase permette di comprendere a fondo il ruolo e il comportamento dei componenti introdotti nella fase precedente.
La *fig. 3.4.2* schematizza il caricamento del dizionario di simboli grafici *SymbolsDictionary*, il caricamento del dizionario interfaccia *IntefaceDictionary* e dei significati *AlphaDictionary*, la visualizzazione della prima pagina del dizionario dei simboli. In pratica corrisponde al *bootstrap* dell'applicazione.
Il *core* come primo passo inizializza l'applicazione per permettere il corretto funzionamento del *software*, come secondo passo *lancia* il caricamento dell'interfaccia che richiede la creazione dei componenti *standard* impostati staticamente in fase di progettazione. Una volta termina questa terza fase, *interface* richiede le voci di menù contenute in un file di testo e caricate in memoria tramite *interface dictionary*, che restituisce tali voci in riferimento alla lingua impostata di *default* (inglese). Caricate le voci di menù l'applicazione richiede i simboli del linguaggio alternativo, impostato di *default* (Bliss), all'*interface dictionary*, che passa i riferimenti fisici ai file .jpg raffiguranti i simboli all'oggetto *gridhandler*, che dopo aver richiesto e ottenuto i significati dei simboli in suo possesso all'*alphadictionary*, li associa univocamente ai simboli e li *restituisce* all'*interface* che si fa carico di visualizzarli a schermo. Nel diagramma di *fig. 3.4.2*, sono state sottointese alcune fasi, quali ad esempio il caricamento della *ToolBar*, delle sezioni e dei due componenti *Garbage*. Le prime due fasi sono state trascurate per economia di descrizione, al fine di non appesantire la schematizzazione, esse non sono di fatto passaggi fondamentali al fine di comprendere la struttura e il comportamento dell'applicazione, mentre la terza fase, quella che interessa il caricamento dei due Garbage, non è stata schematizza in quanto è oggetto della *fig. 3.4.3*

che descrive l'istanziamento e il funzionamento di un di essi, più precisamente del *Garbage* inerente la gestione dei simboli selezionati e inseriti nella *linea di discussione*.

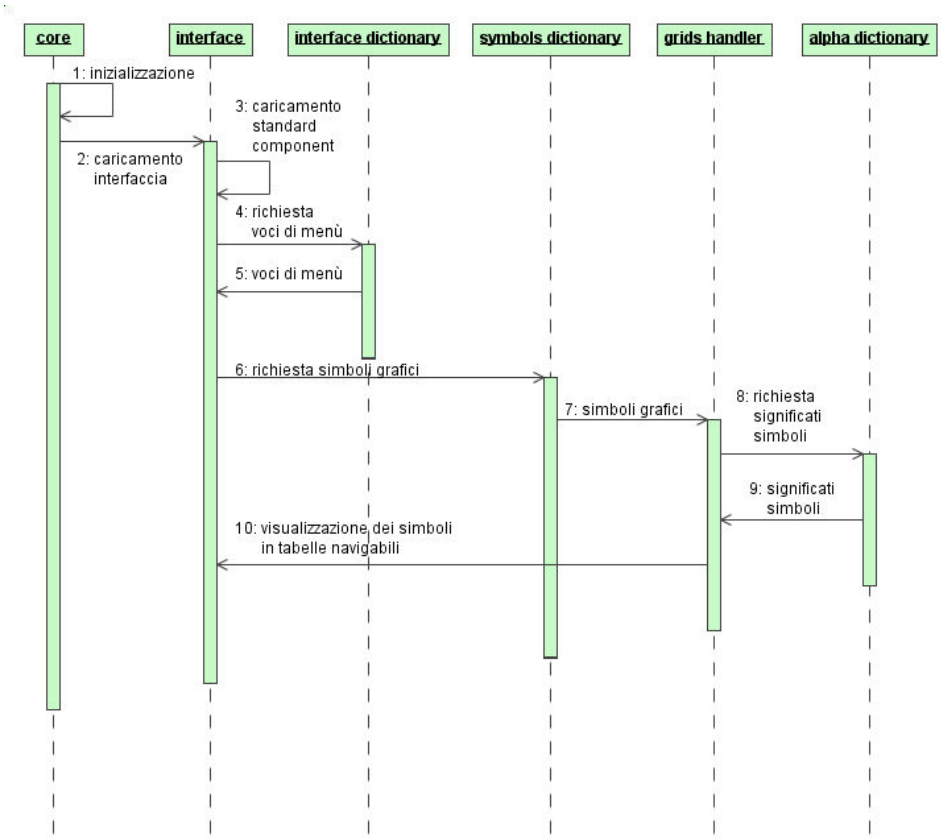

fig. 3.4.2 – *sequence diagram* delle fasi di *bootstrap* dell'applicazione

La *fig. 3.4.3* è un *sequence diagram* rapprensentante la sequenza di operazioni interessate nella selezione di un simbolo e nella successiva richiesta d'inizio di una nuova sessione. Il diagramma focalizza la sua attenzione sul comportamento del *Garbage*, in questo caso del *discussionGarbage.* Il primo passo descritto dal *sequence diagram* riguarda la richiesta di selezione di un simbolo, che l'*interface*, fa al *core* dell'applicazione (per esempio a seguito di una richiesta di un utente). Tale richiesta viene *girata* dal *core* al *gridshandler* che *ordina:* di aggiungere il simbolo richiesto al *discussion garbage*, in quanto tale oggetto è stato creato dinamicamente e come tale dovrà essere *eliminato* nel caso in cui venga richiesto dall'applicazione e all'*interface* di visualizzare il simbolo nella *discussion line.* Queste prime fasi sono state descritte al fine di poter far comprendere l'utilità dell'oggetto *garbage.*
Nella *fase 5* l'*interface* richiede al *core* di iniziare una nuova sessione (per es. su richiesta dell'utente o di un *plug-in*), ricevuta tale richiesta il *core*, oltre a effettuare tutte le procedure necessarie a tale fine, richiede di cancellare dalla *memoria* tutte le istanze a oggetti creati dinamicamente, come per esempio il simbolo creato a seguito della sua selezione nelle fasi precedenti.
Come si può vedere dalla figura a seguito di una richiesta d'inizio sessione gli elementi creati dinamicamente durante la sessione vengono cancellati dalla memoria e conseguentemente cancellati dall'interfaccia. Il diagramma come detto è incentrato sul

discussionGarbage, ma il procedimento è del tutto identico al *selectionGarbage* inerente alla visualizzazione a schermo della tabella con i simboli.

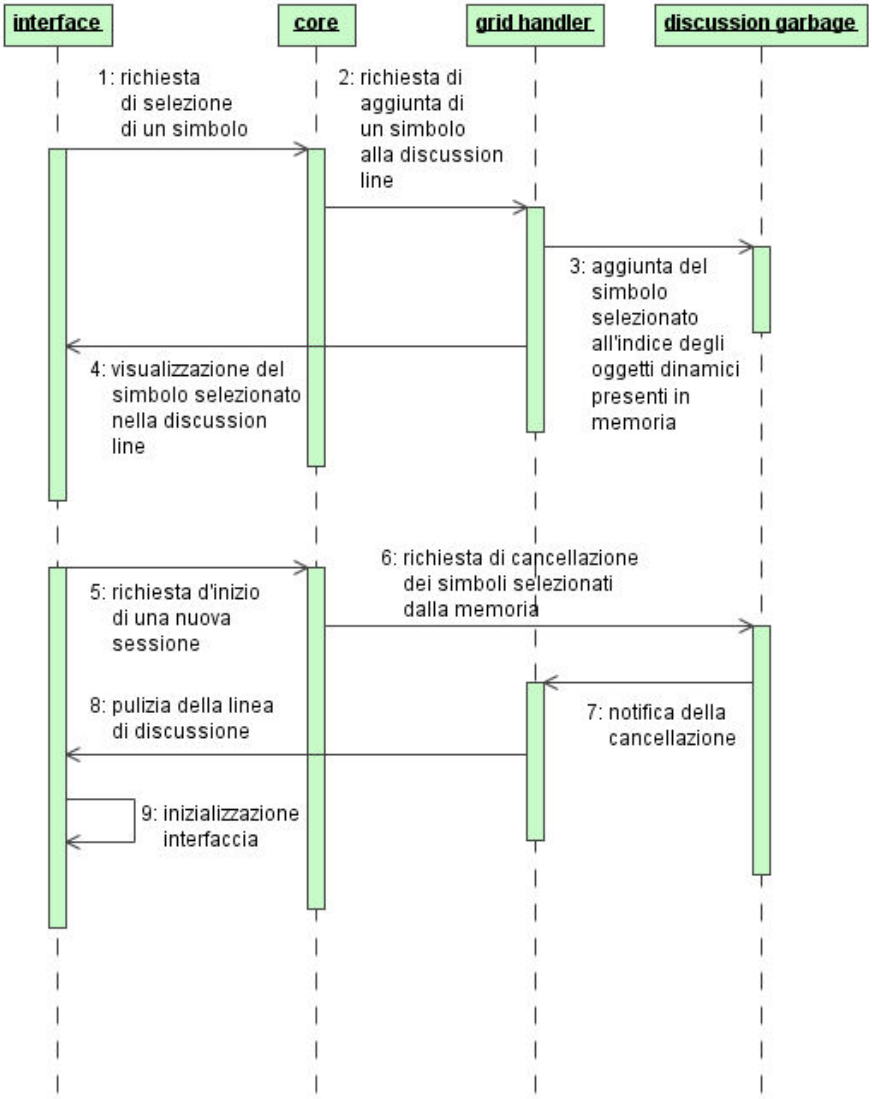

fig. 3.4.3 – *sequence diagram* della selezione di un simbolo e di una successiva richiesta d'inizio sessione

La *fig. 3.4.4* illustra tramite un *sequence diagram* l'interazione tra nucleo e server nelle fasi di creazione di un generico componente remoto.
In questo *sequence diagram* l'enfasi è posta sulla relazione tra *core* e *server*, con l'intento inoltre di contestualizzare il ruolo del *comand processor*, il quale intepreta i comandi che il *server* riceve e colloquia con il *core* al fine di compiere l'operazione richiesta. Il diagramma di figura riguarda chiaramente l'esecuzione di un comando in cui l'intepretazione è andata a buon fine, se per un qualsivoglia motivo il comando non fosse stato riconosciuto, l'operazione sarebbe stata abortita e sarebbe stato restituito un messaggio d'errore al *server*. Queste situazioni saranno trattate nel capitolo seguente, dove inoltre verrà esemplificato con maggior dettaglio il ruolo d'*interfaccia* con il mondo esterno ricoperto dal *server*.

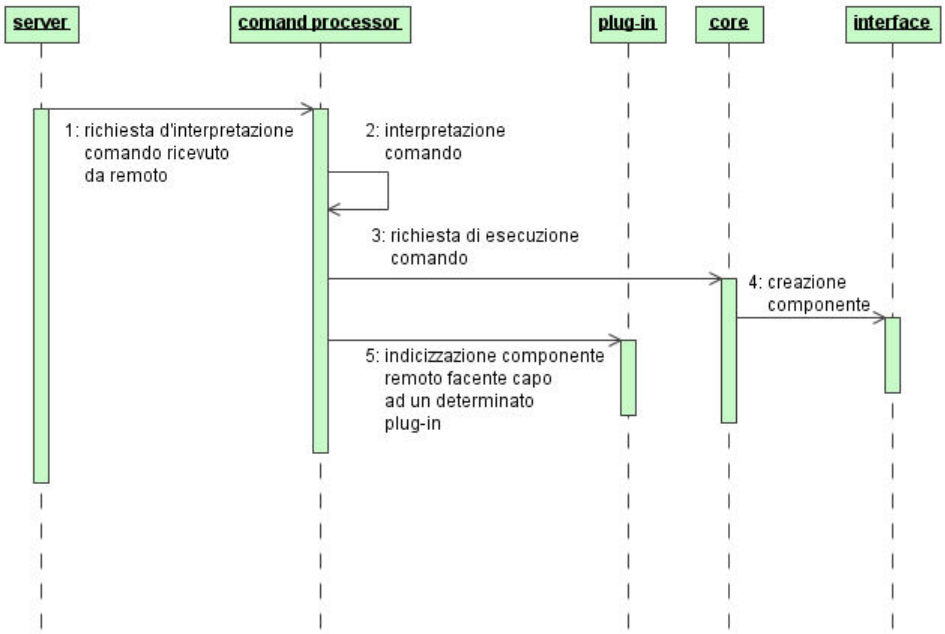

fig. 3.4.4 – *sequence diagram* inerente alla creazione di un componente remoto

3.5 Utilizzo in modalità *base*

Quanto qui descritto si riferisce all'utilizzo dell'applicazione in modalità *base*, quindi senza alcun plug-in connesso, facendo quindi uso unicamente dei servizi messi a disposizione dal nucleo.
In questa modalità d'utilizzo un utente, a conoscenza del linguaggio Bliss, è in condizione di comunicare selezionando i simboli di suo interesse da un dizionario di circa 1300 significati. L'utente quindi è in condizione di comunicare anche con chi non conosce il linguaggio Bliss ma conosce la lingua inglese.
I simboli sono visualizzati a video in tabelle, ogni tabella costituisce una pagina del dizionario. L'utente, tramite l'uso della tastiera o del mouse può navigare la tabella o, cambiando le pagine, l'intero dizionario. La navigazione della tabella è supportata da un cursore video che evidenzia il simbolo selezionabile.
La *fig. 3.5.1* mostra la sezione *conversazione* in cui è possibile appunto navigare le tabelle e quindi l'intero dizionario. In figura è possibile vedere la disposizione dei simboli all'interno dell'interfaccia grafica, il cursore di colore nero, posizionato sul simbolo *airport*, la *linea di discussione*, in cui sono presenti tre simboli precedentemente selezionati, nonchè la *ToolBar* per la selezione delle funzioni di navigazione e selezione.
L'utilizzo del OpenAAC è di fatto molto semplice, nella modalità *base*, l'applicazione visualizza all'apertura i primi simboli del dizionario Bliss in suo possesso e posiziona il cursore sul primo simbolo in alto a sinistra. A questo punto, tramite le frecce di una comune tastiera o tramite il proprio mouse, l'utente è in grado di navigare la tabella dei simboli e, tramite i tasti di *pag-up* e *pag-down*, selezionare la tabella da navigare. Ogni tabella corrisponde ad una porzione di dizionario ordinato.
Tramite la navigazione della tabella come detto è possibile posizionarsi su un simbolo di interesse e inserirlo nella *linea di discussione*, posta sotto la griglia di selezione, semplicemente premendo invio o tramite il click del mouse.
L'interfaccia grafica dell'OpenAAC è composta unicamente da due sezioni, quella principale di selezione dei simboli e visualizzazione delle selezioni, e quella secondaria in cui sono presenti due aree di log nelle quali vengono registrati rispettivamente i comandi eseguiti e la frase composta tramite la selezione dei simboli.
La selezione della sezione avviene in maniera del tutto intuitiva, basta infatti *cliccare* sul nome della sezione. (vedi *fig. 3.5.1)*
Nella sezione di composizione, nell'area destra dello schermo, vi è una barra di comando *ToolBar* contenente i bottoni corrispondenti a tutte le funzioni base dell'OpenAAC, raffigurate da icone significative e da *hint*, messaggi, che ne visualizzano il significato al passaggio del puntatore del mouse.

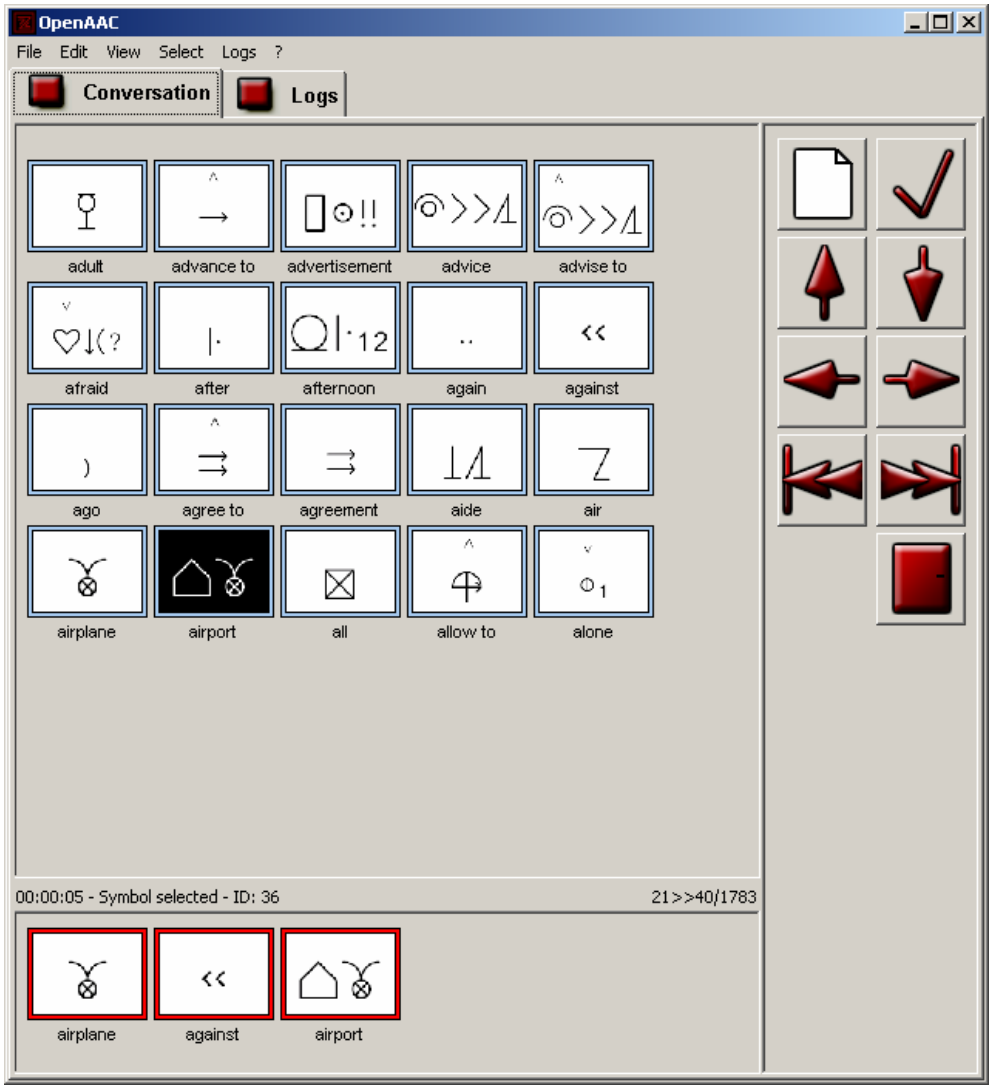

fig. 3.5.1 – *screen shot* della sezione di conversazione di OpenAAC

Risulta evidente che eseguire le operazioni sopra descritte per alcuni utenti con particolari disabilità può risultare difficoltoso, a questo proposito invito a considerare tali operazioni come punto di partenza per molteplici funzioni alternative che permetteranno a ogni utente l'utilizzo dei servizi basilari. Tali operazioni non sono in alcun modo vincolanti per l'utilizzo dell'applicazione, anzi il loro intento è diametralmente opposto, vogliono infatti porsi come interfaccia tra il mondo esterno e l'elaboratore. Esse infatti rappresentano cosa l'utente deve poter fare, come questo viene fatto in questa fase èdi secondaria importanza, infatti il nucleo dell'applicazione, tramite il server, si pone in *ascolto* in attesa che qualcuno gli *dica cosa fare*. Le modalità d'interfacciamento particolare con l'utente spetta quindi alle applicazioni satellite che, sviluppate per rispondere a determinate e specifiche esigenze, *comanderanno* il nucleo dell'applicazione. Questo permetterà di avvicinare maggiormente la persona con disabilità verbali o cognitive e il sistema d'ausilio alla comunicazione, con l'intento di far adattare quest'ultimo alle situazioni e non viceversa.
L'OpenAAC ha come punto di arrivo quello di mettere a disposizione la struttura atta a supportare le molteplici strade ausiliare che portano alla comunicazione, per questo motivo al centro vi è la comunicazione e attorno ad essa satellitano i mezzi che portano al centro, tali mezzi sono per l'appunto i plug-in.

La *fig. 3.5.2* mostra l'altra sezione presente in modalità *base*, tale sezione come anticipato contiene due aree di testo in cui l'applicazione salva rispettivamente le operazioni compiute, complete di *time stamp*, e il significato legato ai simboli selezionati.
In figura si può vedere nella sezione *Application's log* che dopo l'avvio del programma l'utente ha richiesto la visualizzazione della pagina successiva (*time stamp 00:00:03*) ha selezionato 5 simboli differenti, ha cambiato tre volte pagina, e ha selezionato un sesto simbolo (*time stamp 00:01:39*) il cui *ID* univoco è *89*. Il *time stamp* è utile dato che indica il tempo in cui un'azione viene compiuta rispetto all'inizio della sessione, per esempio la selezione del simbolo con *ID 26* è avvenuta dopo un minuto e trenta secondi dall'avvio della sessione. L'*Application's log* oggetto di *fig. 3.5.2* mostra l'esecuzione di più sessioni che hanno comportato la selezione di più simboli.
Il contenuto della sezione denominata *Discussion's log* riporta il significato dei simboli selezionati nella sessione attiva, in questo caso i significati si riferiscono rispettivamente ai simboli con *ID 35, 29* e *36*.
L'importanza della presenza di due aree di *log* come quelle appena descritte è legata alla necessità di *registrare* l'utilizzo dell'applicazione da parte dell'utente. In questo modo è possibile, in prima analisi, utilizzare tali registrazioni al fine di testare l'applicazione, e, in seconda analisi, poter in futuro permettere all'applicazione di utilizzare tali *log* per dare maggiore supporto all'utente che con il tempo l'applicazione avrà imparato a conoscere.

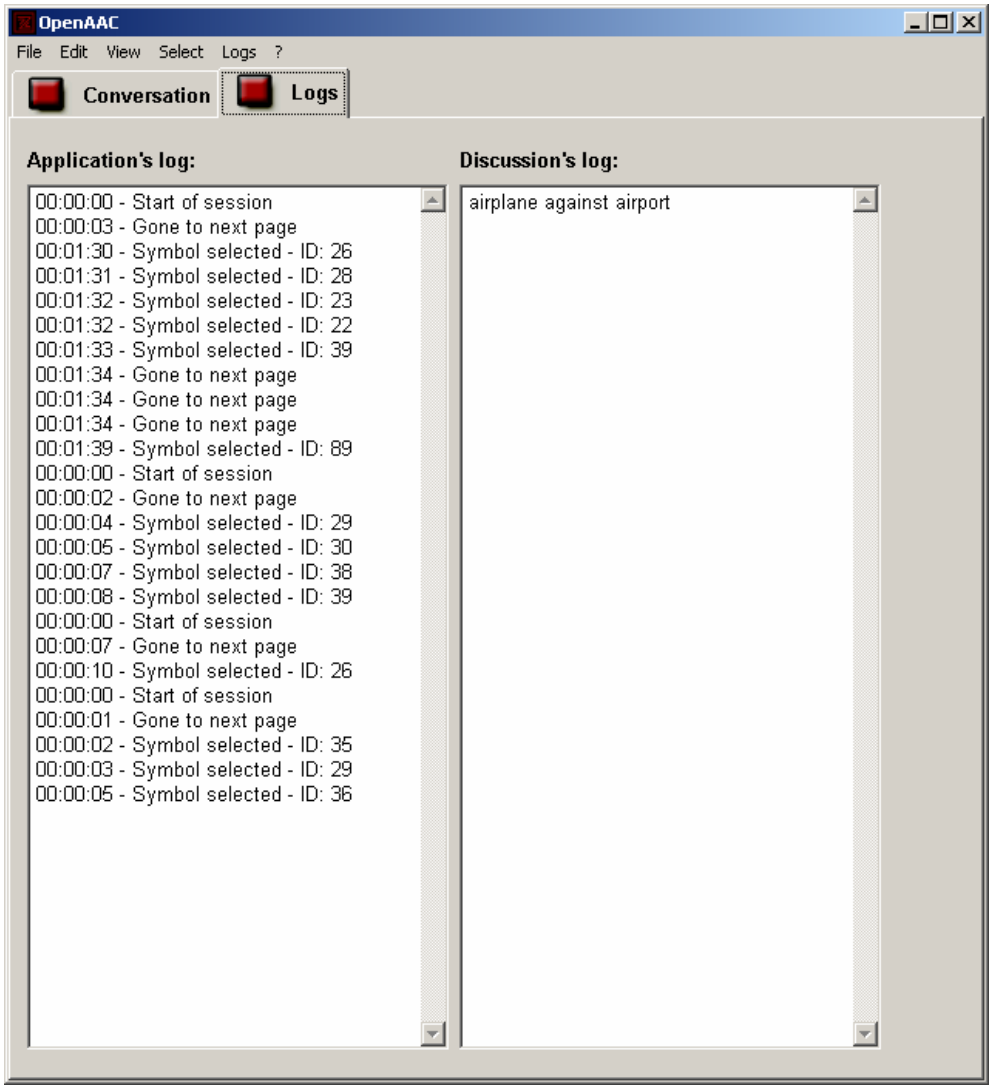

fig. 3.5.2 - sezione logs

4

ARCHITETTURA CLIENT/SERVER: OPENAAC COME SISTEMA DISTRIBUITO

4.1 Introduzione

La principale prerogativa, alla base dello sviluppo di OpenAAC, è di progettare un sistema aperto in grado di mettere a disposizione le funzioni base del proprio nucleo ad applicazioni satellite che ne ampliano la potenzialità.
Durante la fase di analisi dei requisiti dell'applicazione, la struttura client/server è risultata essere la struttura più adatta per conferire a OpenAAC tale caratteristica.
Come detto il nucleo dell'OpenAAC contiene solo poche e indispensabili funzioni al suo interno, funzioni però che hanno un forte potenziale se sfruttate da applicazioni di supporto che ne automatizzino e personalizzino l'utilizzo a seconda del singolo scopo che tali *plug-in* dovranno raggiungere.
Nei capitoli precedenti è stato accennato uno dei ruoli ricoperti dal nucleo dell'applicazione, più esattamente quello di porsi in *ascolto* in attesa che un generico *plug-in* richieda di integrarsi con l'applicazione e di interagire con essa.
Tale comportamento, come vedremo in questo capitolo, è alla base dell'architettura *client/server*. Nella prima parte di questo capitolo viene illustrata a *grandi linee* l'architettura di un generico sistema di tipo *client/server*, esso verrà inoltre esemplificato al fine di far comprendere la diffusione di tale architettura, la sua utilità e le sue caratteristiche principali.
Nella seconda parte viene illustrato come tale architettura caratterizza OpenAAC, come l'applicazione è strutturata e in quale modo essa sfrutta questa struttura distribuita. Per fare ciò vengono presentati nuovamente dei diagrammi UML utilizzati al fine di schemattizzare in maniera efficace le operazioni compiute dall'applicazione in *modalità distribuita*.
In questo capitolo sono inoltre presentati i protocolli definiti *ad hoc* durante le fasi di progettazione e implementazione dell'applicazione, essi sono due, uno utilizzato dal *server* e intepretato dai *client* e uno utilizzato dai *client* e interpretato dal *server*. L'utilità di tali protocolli risulta facilmente comprensibile, essi hanno lo scopo di porre in comunicazione *client* e *server* in maniera univoca permettendo alle due tipologie di attori di potere colloquiare e cooperare al meglio; maggior dettaglio sulle loro funzioni è dato di seguito.

Quelle che finora abbiamo chiamato applicazioni satellite o *plug-in*, in questo capitolo vanno sotto il nome di *client*, per porre l'accento sull'architettura in questione.

4.2 Caratteristiche della struttura client/server

Il termine *client/server* si riferisce normalmente all'architettura di un *software* divisa in due parti ben distinte, costituite da due programmi fra loro indipendenti: un *programma server*, che fornisce il servizio di base, e un *programma client*, il quale accede all'applicazione lato *server* per usufruire del servizio base.
Tipicamente l'*applicazione client* è principalmente composta da una *interfaccia utente* funge da *front-end* tra l'utente e il servizio *server*. Tra i compiti del *client*, spesso, ma non sempre, vi è quello di verificare la correttezza dei dati inseriti, tale *validazione* può avvenire anche a lato *server*. L'applicazione *client* e quella *server* comunicano tramite uno o più protocolli a più livelli in modo da potersi scambiare in modo univoco *informazioni*, principalmente sottoforma di *semplici messaggi*.

Il successo di tale architettura è legato a vari aspetti dell'ingegneria del *software*. Alcuni dei principali vantaggi derivano dal fatto che *client* e *server* hanno funzioni radicalmente diverse e sono in larga misura indipendenti fra loro essi sono *legati* unicamente dall'insieme di messaggi che possono scambiarsi.

Questo è un piccolo insieme di caratteristiche tipiche dell'architettura *client/server*:

- l'architettura si presta ad applicazioni in cui è richiesto un solo *server* che fornisce servizio a molti *client*
- l'architettura si presta ad essere utilizzata in ambiente distribuito (*es. reti*)
- la parte *client* e la parte *server* di una applicazione possono essere sviluppate separatamente in modo semplice da gruppi di lavoro diversi, e in tempi diversi
- le due parti possono essere ottimizzate, ciascuna per il suo compito, indipendentemente
- è possibile sostituire i *client* o i *server* con programmi di tipo diverso o versioni successive, purché accettino lo stesso *protocollo di messaggi*. Ad esempio un utente può scegliere fra diverse interfaccie utente quella che preferisce o quella che meglio si adatta alle sue esigenze

La *fig. 4.2.1* mostra il comportamento di un generico *server* e di tre generici *client A, B e C*. La figura intende schematizzare la fase di ascolto del *server*, *listening (1, 4 e 8)*, il quale resta in attesa di richieste di connessione, la fase di richiesta di connessione da parte del *client*, *connection request (2, 5 e 9)*, la fase di concessione della connessione da parte del *server*, *connectioTo (3, 6 e 10)*, e infine la fase di disconnessione del *client*, *disconnect (7)*.
Tutto ciò che avviene tra la fase di avvenuta connessione e la fase di disconnessione è una sessione di dialogo tra le due *applicazioni*, la cui durata è identificata dal rettangolo di colore verde, che per esempio nel caso del *server* dura per tutta la durata di vita dell'applicazione dato che esso deve essere sempre pronto ad accogliere nuove richieste di connessione, nonchè deve essere disponibile a dialogare con i *client* connessi.
La figura inoltre mette in evidenza come un *server* possa accogliere le richieste di molteplici *client*, questa caratteristica conferisce ulteriore importanza all'architettura che permette di realizzare sistemi distribuiti e *espandibili*.

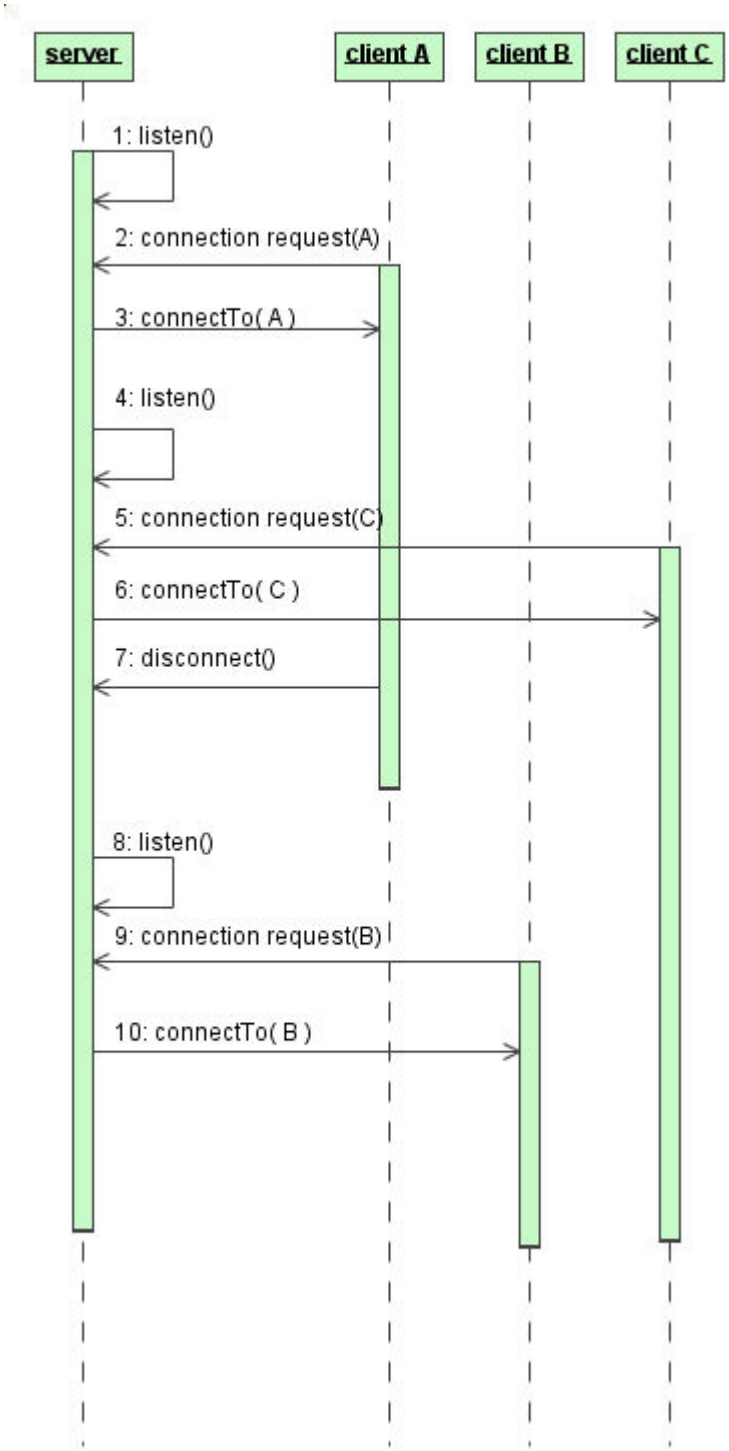

fig. 4.2.1 – esempio di connessione *client/server*

Alcuni tipici esempi di servizi implementati con architetture *client/server* sono:

- i servizi di posta elettronica forniti dai provider Internet: il provider fornisce un *server* che ha il solo compito di trasmettere e ricevere messaggi e mantenerli

memorizzati fino a che un *client* non li richiede. Ogni utente del provider è provvisto di un *client* (Netscape, Eudora, Pine, MS-Exchange, ecc.) che gli mostra l'elenco dei messaggi, gli consente di leggerli, cancellarli, archiviarli, scriverne di nuovi, ecc. Ogni *client dialoga* con il *server* secondo un protocollo detto *POP3*

- i database di tipo *SQL server*. In questo caso il *server* si occupa della gestione di un *database*, scrittura e lettura dei *record*, ricerche, mantenimento degli indici, e altro. Esso riceve comandi in un linguaggio standard detto *SQL* attraverso collegamenti di rete. I *client* da un lato ricevono le richieste di operazioni dagli utenti e le traducono in linguaggio *SQL* per trasmetterle al *server*, dall'altro ricevono le risposte dal *server* e le presentano nel formato opportuno all'utente.
- i *printer server*; in questo caso il *server* fornisce un servizio di stampa e i *client* non sono di norma utenti *umani* direttamente, ma *driver* di stampa su *computer remoti* non dotati di stampante che sono in grado di attivare una richiesta di stampa sul *server*

Il ruolo del *server* è atto a fornire servizi, informazioni o architetture a chi, sotto autorizzazione del *server* stesso, ne necessitasse.
Il ruolo del *client* è per contro ricoperto da tutti i soggetti che si trovano a sfruttare i servizi, le informazioni o le architetture tipiche di un *server*, utilizzando tali servizi o presentando e/o elaborando le informazioni fornitegli dal *server*.
Fondamentale in una struttura così composta è l'interazione, spesso intensa, tra i vari attori; scambio bidirezionale, quindi sia da *client* a *server* sia viceversa, nonchè intraruolo, cioè tra *client* e *client* o *server* e *server*. In una struttura *client/server* infatti nulla vieta a un *client* di svolgere anche il ruolo di *server* per altri *client* o viceversa.

L'esempio più diretto e famoso che si possa riportare parlando di struttura *client/server* è quello del *world wide web*, in cui ogni singolo utente che si connette ad un sito per trarne informazioni ricopre il ruolo di *client* mentre il sito ricopre quello di *server*, o ancora quello delle applicazioni *p2p* che tramite la rete di *internet* permette ad ogni utente di ricercare informazioni, quindi agire da *client*, nonchè di agire da *server* ponendosi come fonte d'informazioni.
In quest'ottica risulta meno difficoltoso comprendere la struttura data all'applicazione in cui il nucleo ricopre principalmente il ruolo di *server*, proprietario del portafoglio d'informazioni dell'applicazione, mentre i *plug-in* ricoprono il ruolo di *client*, avendo il compito di colloquiare con il nucleo dell'applicazione, richiedere le informazioni di cui necessitano e pilotare il nucleo di conseguenza.

4.3 Ruolo del *server* in OpenAAC

Il *server* dell'applicazione è implementato come detto dal nucleo, esso si pone in ascolto all'avvio dell'OpenAAC in modo da poter asincronamente accettare l'integrazione dei *plug-in*.
Il *server*, una volta effettuata la connessione con un generico *plug-in* attiva una comunicazione in ingresso e in uscita con esso, ricevendo e interpretando i comandi *ad-hoc* del *plug-in* e spedendo comandi, anch'essi *ad-hoc* al *plug-in*.
Maggiore attenzione nel caso del *server* va prestata alla spedizione che a seconda dell'informazione oggetto del messaggio può avvenire in modalità *broadcast* (es. evento dell'applicazione) oppure in modalità *p2p* (es. evento oggetto integrato appartenente ad un *plug-in* specifico). I comandi *ad-hoc* sono *instruzioni* costituenti due protocolli (uno *server* e uno *client*) definiti durante la fase di progettazione della struttura client/server dell'OpenAAC. I protocolli, con le relative specifiche dei comandi, sono riportati nei *paragrafi 4.4 e 4.6*.
La struttura *client/server* descritta in questo paragrafo raffigura in maniera abbastanza chiara il concetto di unione per il raggiungimento di un obiettivo; l'*obiettivo* è la comunicazione posta al centro, *server*, l'*unione* è costituita dall'insieme di ausili, *client* che satellitano intorno al nucleo dell'applicazione fornendo all'utente un insieme di ausili atti a migliorare la qualità della comunicazione.
Risulta quindi chiaro, che con il tempo, tramite lo sviluppo di ulteriori plug-in e il perfezionamento delle strutture esistenti, si potrà dare sempre più maggior dettaglio alla tipologia di ausilii disponibili, nonchè conferire maggiore grado di personalizzazione dell'applicazione nei confronti dello specifico utente fruitore.

Il *server* di OpenAAC è implementato da due oggetti, rispettivamente l'oggetto *Cserver*, che ha l'onere di gestire le connessioni con i *client*, nonchè la spedizione e la ricezione dei comandi e l'oggetto *CComandProcessor* che interpreta i comandi ricevuti dai *client* connessi all'applicazione.
Tramite l'oggetto *CServer* OpenAAC si pone in *ascolto* e accetta le richieste di connessione provenienti dai *plug-in*; una volta instaurata la connessione, l'oggetto registra il *client* al suo interno in modo da poter restituire a ogni *client* solo le informazioni che gli riguardano.
Come vedremo in seguito, infatti, i vari *client* potranno creare all'interno dell'interfaccia dell'applicazione vari oggetti di loro proprietà, per fare ciò però il *server*, deve tenerne traccia. Solo in questo modo il *server* può spedire ai *client* proprietari le notifiche degli eventi generati dai propri oggetti sul *server*.
Come già anticipato il *server* comunica con i *client* in due modalità differenti, una che potremmo definire *p2p* (*peer-to-peer*) come nel precedente esempio e una denominata *broadcast*, per esempio, per notificare eventi propri del nucleo dell'applicazione.
Il *costruttore* dell'oggetto *CServer* richiede come parametro un oggetto di tipo *TServerSocket* oggetto standard delle librerie del Borland C++ Builder. Tale oggetto è quello che fisicamente gestisce la connessione ai *client* e che nel caso dell'OpenAAC è dichiarato nel nucleo dell'applicazione.
La *struct SClient* identifica il tipo che permette la memorizzazione, tramite lista dei *client* connessi, assegnando ad ognuno un *ID* che identifichi univocamente i singoli *plug-in*

connessi e un *processore di comandi* le cui funzionalità verranno presentate nel prosieguo del capitolo.
La *connessione* è affidata a tre procedure che gestiscono tre eventi distinti del *SocketServer*, cioè la connessione e la disconnessione del *client* e la lettura dei comandi spediti dai *client* connessi, oltre a una quarta procedura che gestisce la spedizione di messaggi (eventi o informazioni) in modalità *broadcast*. La modalità *p2p* come vedremo è gestita dai singoli processori di comando relativi ai singoli *plug-in*.
Tramite l'oggetto di tipo *CComandProcessor* l'applicazione è in grado di interpretare i comandi spediti dai *plug-in*, comandi appartenenti al protocollo *server*.
Tramite questi comandi (le cui sintassi e significati verranno presentate nel *paragrafo 4.4*) permettono ai *plug-in* di interagire e integrarsi con l'applicazione; i *plug-in* potranno infatti integrarsi con l'interfaccia principale creando al suo interno oggetti standard quali *form*, bottoni, etichette e altro, nonchè interagire con essa avendo la notifica degli eventi generati dagli oggetti di sua proprietà e dal nucleo dell'applicazione. I *plug-in* possono manipolare gli oggetti appartenenti al nucleo di OpenAAC.
Le tipologie di comandi, alla luce delle due distinte esigenze dei *client* quali integrazione e interazione, sono due, una inerente alla creazione e modifica di oggetti all'interno dell'interfaccia grafica e una inerente ai comandi atti a relazionare il *plug-in* con l'applicazione, con i suoi oggetti e le sue informazioni; entrambe le modalità fanno capo allo stesso protocollo.
Le due tipologie di comandi sono filtrate dal processore di comandi e indirizzate a due differenti procedure per essere interpretati e eseguite; le due procedure in questione sono rispettivamente *InterfaceProcess* e *ApplicationProcess*.
Il processore di comando, univoco per ogni *plug-in* connesso all'applicazione, tiene traccia degli oggetti creati dal *plug-in* a cui è associato, in modo da poter notificare gli eventi, effettuare modifiche sugli oggetti creati e effettuare l'eliminazione degli oggetti in caso di disconnessione del *plug-in* stesso.
In fase di progettazioni scelte obbligate hanno portato a dover limitare il numero di oggetti integrabili da remoto, restrizioni che, in caso di esigenze future, potranno essere ampliate. Le restrizioni definite non riguardano unicamente la tipologia di oggetti integrabili ma anche, nel caso di *form* e *TabSheet* (sezioni), la quantità, infatti ogni *plug-in* potrà creare al più una *form* e al più un *TabSheet*.
Per *TabSheet* si intende più ad alto livello una sezione dell'interfaccia dell'applicazione, esso ad esempio ha due *TabSheet* uno contenente la sezione *Conversazione* e uno alla sezione *Logs*.
Ad ogni oggetto creato dal *plug-in* viene associato, ad uno o più eventi tipici di quell'oggetto, un evento che permette la sua notifica secondo i termini propri del protocollo *client*. Ogni oggetto creato fa capo alla *form* del *plug-in*, tranne la *form* del *plug-in* stesso (che fa capo al *TabSheet Conversazione*) e l'oggetto *TabSheet* che fa capo al gestore dei *TabSheet* stessi.
L'uso dell'oggetto di tipo *CGarbage* ha la funzione di registrare i componenti creati in modo dinamico in modo da poterli eliminare quando non sono più utili.
Come detto l'oggetto di tipo *CComandProcessor* ha il compito di spedire messaggi in modalità *p2p* al singolo *plug-in*, i messaggi spediti, inerenti a eventi o a informazioni richieste dal *plug-in* stesso, sono spediti utilizzando il protocollo *client* come del resto i comandi spediti in modalità *broadcast*.

La discussione dell'oggetto *CComandProcessor* non si può esaurire in questo paragrafo, esso infatti ha un doppio ruolo fondamentale, quello di interprete del *client* e quello di *ambasciatore* del *server*, il primo d'interesse di questo paragrafo mentre il secondo inerente ai comandi che dovrà intepretare il *client*, comandi racchiusi nel protocollo *client*, comandi a cui ogni *client* farà corrispondere le funzioni specifiche per cui è stato progettato.
Come vedremo la struttura riguardante la gestione dei protocolli è simmetrica tra il *client* e il *server*, pur trattando contenuti differenti, con significati differenti.

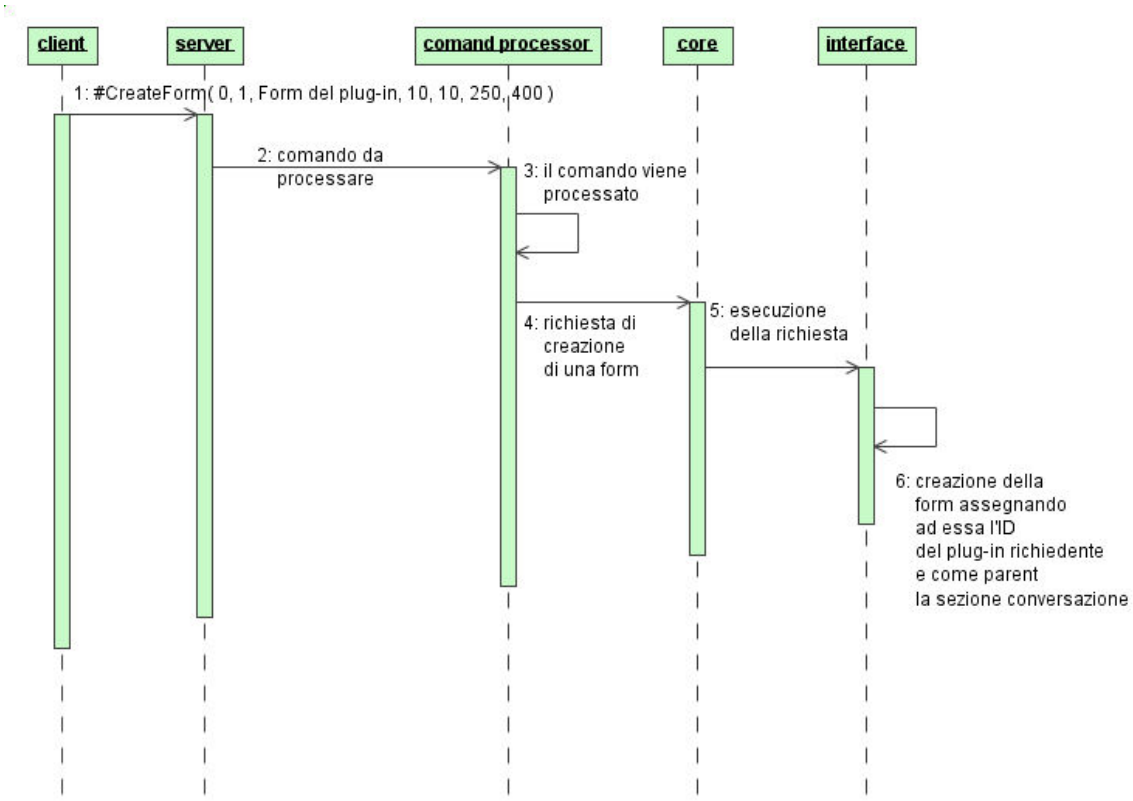

fig. 4.3.1 – *sequence diagram* della creazione da remoto di una *form* all'interno della sezione conversazione dell'applicazione

La *fig. 4.3.1* rappresenta la sequenza di azioni compiute dall'applicazione a seguito di una richiesta di creazione di un componente remoto da parte del *client*. In questo caso la richiesta riguarda la creazione di una *form* all'interno della sezione di *conversazione*. Il significato dei singoli parametri riguardanti il comando appartenente al protocollo è oggetto di discussione dei paragrafi seguenti.

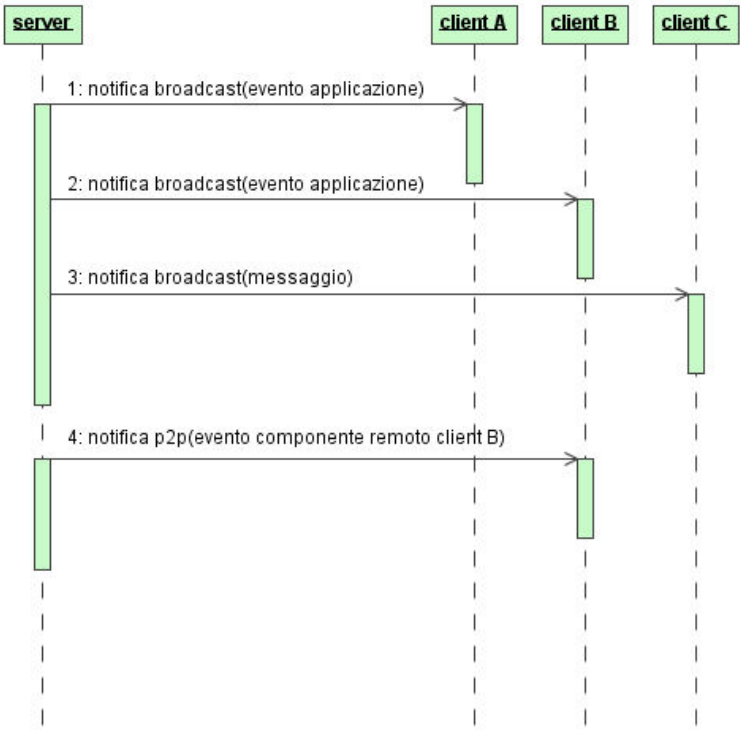

fig. 4.3.2 – *sequence diagram* che illustra le due differenti modalità di notifica, *broadcast* e p2p

La *fig. 4.3.2* mostra le due differenti modalità di notifica, le operazioni *1*, *2* e *3* indicano la notifica di un evento applicazione, per esempio l'inizio di una nuova sessione, evento che deve necessariamente essere notificato a tutti i *plug-in* connessi all'applicazione; mentre l'operazione *4* riguarda la notifica di un evento generato da un componente creato in precedenza dal *client (plug-in) B.* Tale evento può essere per esempio la pressione di un bottone contenuto in una *form* creata dal *plug-in* al momento della connessione.
Tramite la prima modalità di notifica i *client* connessi vengono informati che qualcosa è cambiato nello stato dell'applicazione e se l'evento è di loro interesse possono agire di conseguenza, la seconda modalità invece rigurda più da vicino le funzioni del *plug-in*, tramite tali notifiche viene infatti comandato il *plug-in,* proprio come se questi eventi fossero stati generati dall'applicazione *client.*

4.4 Protocollo *server*

Il seguente protocollo indica i comandi per la creazione e gestione di componenti interfaccia da remoto *(da client a server)*

Lista comandi per l'integrazioni di componenti nell'interfaccia di OpenAAC, tali comandi vengono spediti dal *plug-in* al *server*:

- Comando per la richiesta di creazione di una form *plug-in*
  #CreateForm(Owner, ID, caption, left, top, width, height)
- Comando per la richiesta di creazione di un bottone
  #CreateButton(Owner, ID, caption, left, top)
- Comando per la richiesta di creazione di un etichetta
  #CreateLabel(Owner, ID, caption, left, top)
- Comando per la richiesta di creazione di una casella di inserimento testo
  #CreateEdit(Owner, ID, caption, left, top)
- Comando per la richiesta di creazione di una sezione
  #CreateSection(Owner, ID, caption)
- Comando per la richiesta di creazione di un selettore a spunta
  #CreateCheckBox(Owner, ID, caption, left, top)
- Comando per la richiesta di creazione di un selettore di alternativa (Radio button)
  #CreateRadioButton(Owner, ID, caption, left, top)
- Comando per la richiesta di Visualizzazione di un'immagine o di un simbolo
  #CreateImage(Owner, ID, filename, left, top, width, height, color)

Il parametro Owner serve per specificare il proprietario del componente che si sta creando; per *default* è *PlugInForm*.
Il parametro *ID* serve a un generico *client* per identificare univocamente un componente remoto da lui generato, in modo da sapere quale componente ha generato un evento. Dal lato *server* ad ogni componente saranno associati uno o più eventi, che spediscono al client l'*ID* del componente e l'*ID* dell'evento verificatosi, per permettere al *client* di identificare l'evento e chi l'ha generato.

Elenco *Owner* (proprietario):

- *Form* del principale — *Owner* = 0
- *Form* del *plug-in* — *Owner* = 1
- *TabSheet* del *plug-in* — *Owner* = 2
- *ToolBar* della sezione Conversazione — *Owner* = 3
- Gestore delle sezioni — *Owner* = 4
- Gruppo di Radio Button 1 — *Owner* = 5
- Gruppo di Radio Button 2 — *Owner* = 6
- Gruppo di Radio Button 3 — *Owner* = 7
- Gruppo di Radio Button 4 — *Owner* = 8

(gli ID componenti inerenti i RadioButton di un gruppo devono essere consecutivi e ordinati; i gruppi radio button fanno capo alla sezione plug-in che deve essere necessariamente creata per il loro utilizzo)

Tipologia degli eventi dei componenti integrati notificati ai plug-in proprietari:

- Oggetti di tipo *Form*: *OnShow*
- Oggetti di tipo *Button*: *OnClick*
- Oggetti di tipo *Label*: *OnClick*
- Oggetti di tipo *Edit* *OnChange*
- Oggetti di tipo *TabSheet*: *OnShow*
- Oggetti di tipo *CheckBox*: *OnClick*
- Oggetti di tipo *RadioButton*: *OnClick*
- Oggetti di tipo *Image*: *OnClick*

Ogni tipologia di evento notificato è identificato univocamente da un ID, seconda la seguente tabella:

- *OnShow* *ID = 0*
- *OnClick* *ID = 1*
- *OnChange* *ID = 2*

Lista comandi per l'interazione con l'applicazione, il prefisso *§* indica che il comando riguarda una richiesta d'*interazione* con l'applicazione, mentre il prefisso # indica una richiesta d'*integrazione* di un componente:

- Richiesta di avviare una nuova sessione
§NewSession()
- Richiesta di ridimensionamento form
§SetFormProperties(caption, width, height, left, top)
- Richiesta di ridimensionamento griglia visualizzazione simboli
§SetGridDimension(columns, rows)
- Richiesta di ridimensionamento del simbolo grafico
§SetSymbolsDimension(width, height)
- Richiesta di spostamento del cursore di selezione
§MoveCursor([+/-] columns, [+/-] rows)
- Richiesta di selezione di un simbolo
§SelectSymbol()
- Richiesta di cambiamento di pagina dizionario
§ChangeGridPage([+/-] NumberOfPage)
- Richiesta di cambiamento dizionario interfaccia
§ChangeInterfaceDictionary(ID, path)
- Richiesta di cambiamento dizionari linguaggio simbolico
§ChangeSymbolsDictionary(ID, path)
- Richiesta di cambiamento dizionario dei significati
§ChangeAlphaDictionary(ID, path)
- Richiesta d'informazioni sullo stato della *form*
§**GetFormStatus()**
- Richiesta ID dei dizionari lingua, linguaggio grafico, lingua significati
§GetDictionariesID()

- Richiesta Path dei dizionari lingua, linguaggio grafico, lingua significati
  §GetDictionariesPath()
- Richiesta d'informazioni su dove poter posizionare una *form* di proprietà del *plug-in*
  §GetLocationForForm()
- Richiesta d'informazioni su dove poter posizionare un componente nella *ToolBar*
  §GetLocationForTBComponent()
- Richiesta d'informazioni sullo stato della conversazione
  §GetConversationStatus()
- Richiesta d'informazioni sulle dimensioni della griglia
  §GetGridDimension()
- Richiesta d'informazione sulle dimensioni del simbolo
  §GetSymbolsDimension()
- Seleziona la sezione di conversazione
  §SetFocusOnConversationSection()
- Seleziona la sezione *plug-in*
  §SetFocusOnPlugInSection()

4.5 Ruolo del *client* in OpenAAC

Client è ogni singolo *plug-in* che si connette e si integra con OpenAAC al fine di svolgere una o più funzioni specifiche; spesso essi agiscono compiendo azioni a lui delegate dall'uomo, come per esempio navigare tabelle o leggere il significato dei simboli evidenziati.
L'implementazione di tali *plug-in* ha quindi il principale scopo di creare degli ausilii, non invadenti, che siano di supporto alla comunicazione cercando di facilitare la riabilitazione di strutture comunicative presenti, nonchè l'utilizzo di vie di comunicazione alternative a quelle di uso comune.
La comunicazione con il *server*, come abbiamo visto segue due direzioni, una in ingresso, tramite i comandi del protocollo *server* e una in uscita con i comandi del protocollo *client*.
La funzione principale dei comandi intepretabili dal lato *client* è quella di permettere al server di comunicare con il *client* stesso, fornendo principalmente informazioni legate allo stato dell'applicazione, notifiche degli eventi dei componenti remoti (modalità *p2p*) e notifica degli eventi applicazione (modalità *broadcast*), nonchè fornire un *feedback* ai comandi spediti dal *client* al *server*.
E' importante precisare che i due protocolli *ad-hoc* inerenti il *client* e il *server*, hanno il solo scopo di regolamentare i contenuti dei messaggi, essi non interferiscono in alcun modo con le modalità di trasmissione, connessione e quant'altro, tutto questo è gestito dal componente standard di tipo *TSocket* presente nelle librerie del *Borland C++ Builder*.
Per permettere l'interpretazione del protocollo *client* è stato implementato un oggetto che permette di processare i messaggi in ingresso (simile a quello presente sul *server*) a disposizione di chi implementa i *plug-in* in modo da dover semplicemente utilizzare le informazioni senza doverle interpretare, tale oggetto prende il nome di *ClientMessageProcessor* e restituisce al plug-in le informazioni formattate.

4.6 Protocollo *client*

Ricezione messaggi spediti in modalità *p2p*:
Feedback messaggio d'errore esecuzione comando remoto:

- File non trovato
  0~0
- Simbolo non trovato
  0~1
- Impossibile creare nuova form plug-in (max 1)
  0~2
- Impossibile creare componente, form plug-in non presente
  0~3
- Impossibile creare nuova sezione plug-in (max 1)
  0~4
- Impossibile creare componente, sezione plug-in non presente
  0~5
- Impossibile creare componente comando o owner errati
  0~6
- Componente remoto non trovato (ID inesistente)
  0~7

Feedback richiesta informazioni:

- Informazioni Form applicazione
  1~0(Caption, width, height, left, top)
- Informazioni ID dizionari lingua, linguaggio grafico, lingua significati
  1~1(InterfaceID, SymbolsID, AlphaID)
- Informazioni path dizionari lingua, linguaggio grafico, lingua significati
  1~2(InterfacePath, SymbolsPath, AlphaPath)
- Informazioni locazione PlugInForm
  1~3(left, top)
- Informazioni locazione componente in ToolBar
  1~4(width, height, left, top)
- Informazioni sulla linea di discussione
  1~5(text)
- Informazioni sulla dimensione della griglia
  1~6(columns, rows)
- Informazioni sulla linea di discussione
  1~7(width, height)
- Informazioni sulla posizione del cursore
  1~8(column, row)

Notifica evento componente remoto:

- Notifica evento OnCreate:
  2~0(ComponentID)
- Notifica evento OnClick
  2~1(ComponentID)
- Notifica evento OnChange
  2~2(ComponentID, text)

Ricezione messaggi spediti in modalità *broadcast*:
Notifica eventi applicazione:

- Nuova sessione
  3~0
- Ridimensionamento form
  3~1(width, height, left, top)
- Ridimensionamento griglia
  3~2(columns, rows)
- Ridimensionamento simbolo
  3~3(width, height)
- Spostamento cursore
  3~4(rel_column, rel_row, SymbolID, SymbolsName)
- Simbolo evidenziato
  3~5(column, row, SymbolID, SymbolsName)
- Simbolo selezionato
  3~6(SymbolID, SymbolsName)
- Cambiamento pagina
  3~7(NumberOfPages)
- Cambiamento dizionario interfaccia

3~8(CommonLangID)

- Cambiamento dizionari linguaggio simbolico

3~9(SymbolsLangID)

- Cambiamento dizionario dei significati

3~10(CommonLangID)

5

I *PLUG-IN* DI OPENAAC

5.1 Introduzione

In questo capitolo sono presentati i principali *plug-in* implementati attualmente per OpenAAC. E' bene precisare che l'intenzione di questo capitolo è quella di presentare alcune delle possibili direzioni verso cui l'ampliamento delle funzioni di OpenAAC possono portare quest'applicazione e il suo ausilio alla comunicazione, nonchè per ribardire il concetto contenuto anche nel titolo di questa tesi, e cioè la creazione di un sistema di ausilio alla comunicazione.

5.2 Dictionary Expander (DE)

Questo *plug-in* è stato implementato con lo scopo di *estendere* i dizionari applicazione, esso infatti permette di aggiungere dizionari interfaccia, *InterfaceDictionary*, dizionari grafici, *SymbolsDictionary*, e i correlati dizionari dei rispettivi significati *AlphaDictionary*. *InterfaceDicionary* è il dizionario d'interfaccia, che permette all'applicazione di essere *multilingue*, come anche *AlphaDicionary* che permette di associare a ogni simbolo il significato in diverse lingue. *SymbolsDictionary* permette di utilizzare differenti linguaggi grafici.
Il nucleo dell'applicazione indicizza le lingue e i linguaggi in modo da assegnare a ognuna di esse un identificatore univoco, questo permette a tutti i *plug-in*, attuali e futuri di poter conoscere quale dizionario è attualmente utilizzato. Nel caso in cui venga aggiunto un dizionario facente capo a una lingua o linguaggio non indicizzato esso viene aggiunto alla tabella dei dizionari.
L'importanza di questo *plug-in* è notevole dato che esso permette a OpenAAC di supportare le lingue di qualsiasi paese, nonchè di supportare linguaggi simbolici qualsiasi, dal *Bliss* al *PCS*, passando per tutti i linguaggi definiti e definibili simbolicamente.
Il plug-in si integra graficamente con l'OpenAAC creando una sezione *ad-hoc* per la selezione dei dizionari.
Selezionando un dizionario interfaccia qualsiasi il *plug-in* modificherà in tempo reale le voci di menù dell'applicazione, nonchè della sezione plug-in, lo stesso avverrà selezionando un dizionario di simboli, in tempo reale i simboli visualizzati nella griglia saranno quelli dei dizionario prescelto (tale cambiamento rende necessario l'inizio di una nuova sessione). Nulla d'altro è d'aggiungere alla selezione di un nuovo dizionario dei significati, anche questo cambiamento avviene in modo analogo.
Il supporto per più dizionari interfaccia, per le voci inerenti a *plug-in* aggiuntivi è carico degli stessi, essi infatti conoscendo l'*ID* del dizionario in uso potranno, caricare il loro

dizionario interfaccia inerente alla lingua selezionata; nel caso in cui non disponessero di tale dizionario dovranno impostare quello di default dell'applicazione (Inglese).
La struttura *client/server* ricopre una notevole importanza in quanto oltre a permettere al nucleo e al *plug-in* di comunicare, permette di notificare quali cambiamenti un *plug-in* genera nella presentazione delle informazioni, in modo da tenere tutti i *plug-in* connessi costantemente informati, in questo modo, nel caso specifico del *plug-in* ora descritto, tutti i *plug-in* connessi sanno quale dizionario è stato selezionato in modo da potersi adattare, se lo ritengono opportuno, alle nuove impostazioni.
Nella *fig. 5.2.1* è illustrata la sezione DE in cui è possibile selezionare i dizionari d'interesse, tale sezione può essere ampliata semplicemente mettendo mano ai sorgenti del *plug-in* e non a quelli dell'intero progetto.

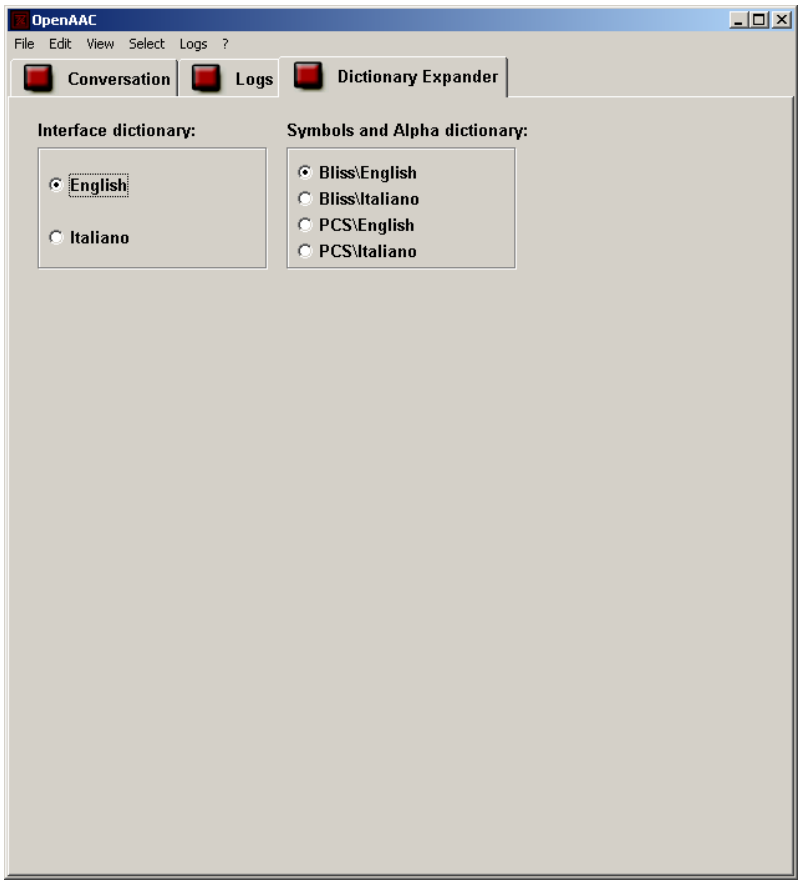

fig. 5.2.1 – sezione DE (Dictionary Expander)

Il *plug-in* dictionary expander interagisce con il nucleo, come tutti i *plug-in*, tramite i comandi del protocollo. Oltre alla richiesta d'integrazione dei suoi componenti d'interfaccia, il *plug-in* interagisce con il nucleo per richiedere di modificare i dizionari in uso, passando come parametro all'applicazione l'*ID* del dizionario da adottare, e il percorso dove andare a recuperare i contenuti del dizionario. Tali parametri sono stati esposti nel capitolo precedente dove è stato illustrato il protocollo *client.* In questo modo il *plug-in* comanda l'applicazione principale, che si fa carico di notificare a tutti i *plug-in* connessi il cambiamento di dizionario, d'interfaccia o di simboli, che sia, in modo da permettere a tutti i *plug-in* connessi di essere a conoscenza del cambiamento.

E’ carico di ogni singolo *plug-in* modificare il suo comportamento a seconda del dizionario selezionato, per esempio, se cambia il dizionario interfaccia, spetta al singolo *plug-in* modificare le proprie voci d’interfaccia di conseguenza.
Questo *plug-in* è di notevole importanza, in quanto elimina uno dei principali vincoli esistenti nella quasi totalità di applicazioni di AAC che spesso permettono di utilizzare un solo linguaggio grafico.
La *fig. 5.2.2* mostra l’utilizzo di OpenAAC con i simboli PCS a cui sono associati i significati in italiano.

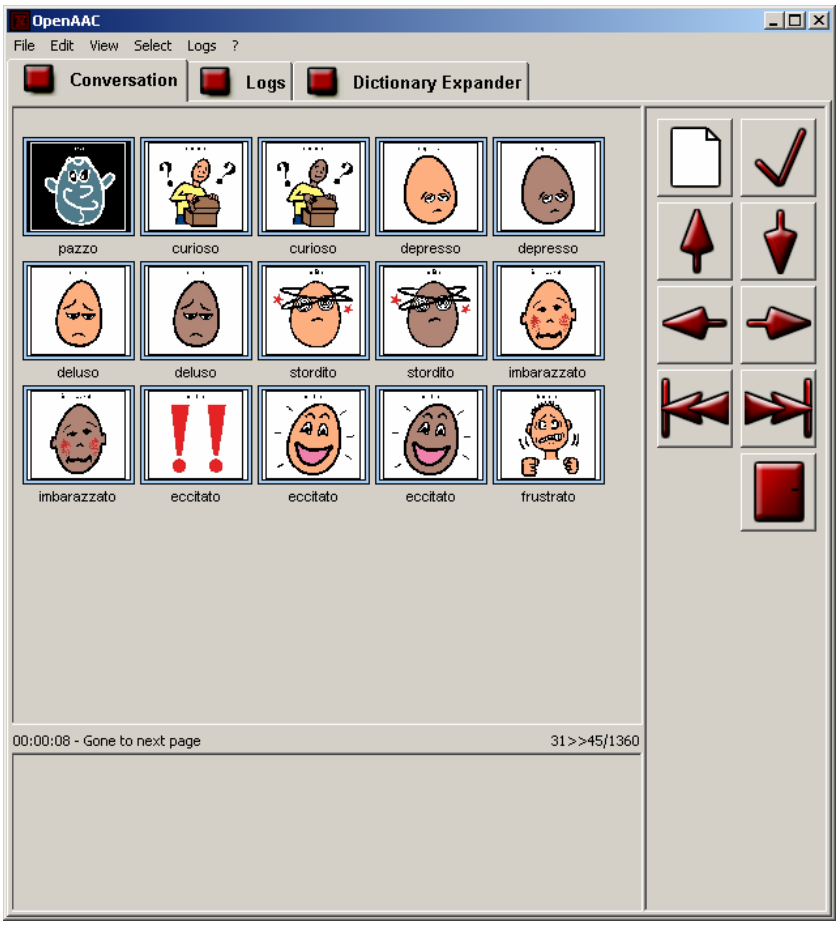

fig. 5.2.2 – utilizzo di OpenAAC con il linguaggio PCS e i significati in italiano

5.3 Synth Voice Tool (SVT)

Il *plug-in Synth Voice Tool* è stato implementato per fornire un ausilio vocale all'utilizzo dell'applicazione, esso infatti sintetizza vocalmente il significato dei simboli.
Il suo utilizzo risulta di notevole aiuto, specialmente nel caso di utenti ipovedenti che tramite questo strumento sono in grado di avere un ritorno vocale sia dei simboli evidenziati che di quelli selezionati, potendo così avere un riscontro sul simbolo selezionato. Tale ausilio visualizza inoltre il simbolo evidenziato o selezionato, in una finestra a parte con dimensioni maggiori, in una finestra a parte (*fig. 5.3.2*).
Il significato dei simboli viene prelevato dal *plug-in* e con il supporto di un applicazione esterna di sintesi vocale (es. *TextAloud – fig. 5.3.3*) lanciata in *background* viene sintetizzata e pronunciata dall'elaboratore.
La *fig. 5.3.1* mostra il *plug-in* integrato con l'applicazione, nella *ToolBar* alla destra dello schermo si può notare il bottone di avvio e arresto della funzione. La *fig. 5.3.3* mostra l'applicazione attualmente utilizzata per la sintetizzazione vocale, TextAloud Mp3 che tra le varie funzioni permette di salvare in formato mp3 la sintetizzazione effettuata.
Questo *plug-in* interagisce in vari modi con il nucleo dell'applicazione, esso ogni volta che viene notificata la selezione di un simbolo o il movimento del cursore su un particolare simbolo richiede il significato del simbolo e lo passa all'applicazione esterna di sintetizzazione vocale. Inoltre visualizza il simbolo selezionato o evidenziato, ingrandito, nella *form* creata alla sua connessione nella sezione conversazione, questo al fine di visualizzare al meglio il simbolo d'interesse, in ausilio a utenti *ipovedenti (fig. 5.3.2).*

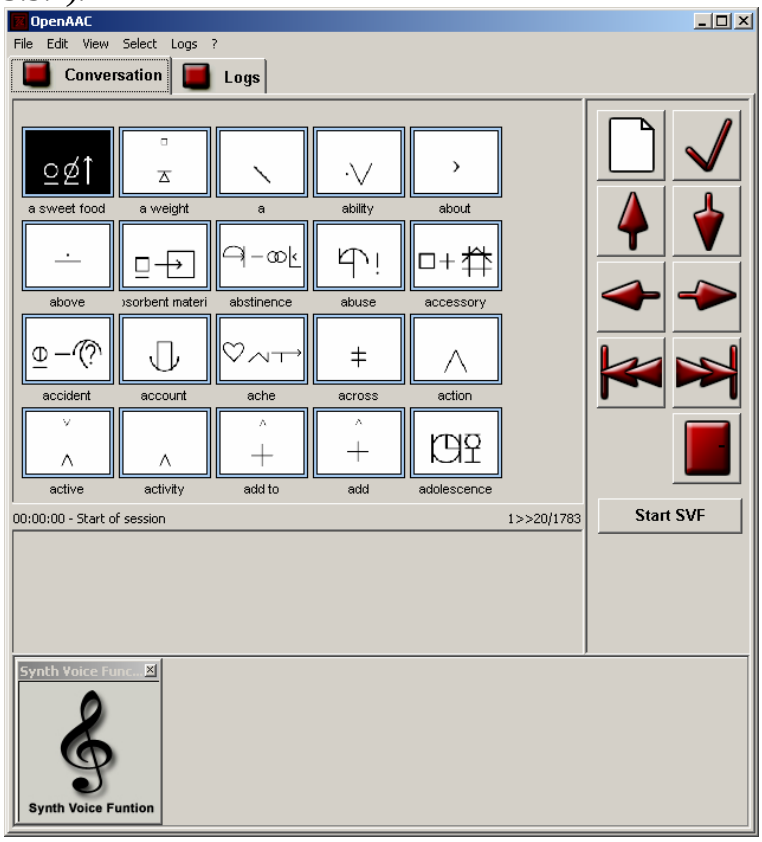

fig. 5.3.1 – Plug-in SVF (Synth Voice Function) integrato

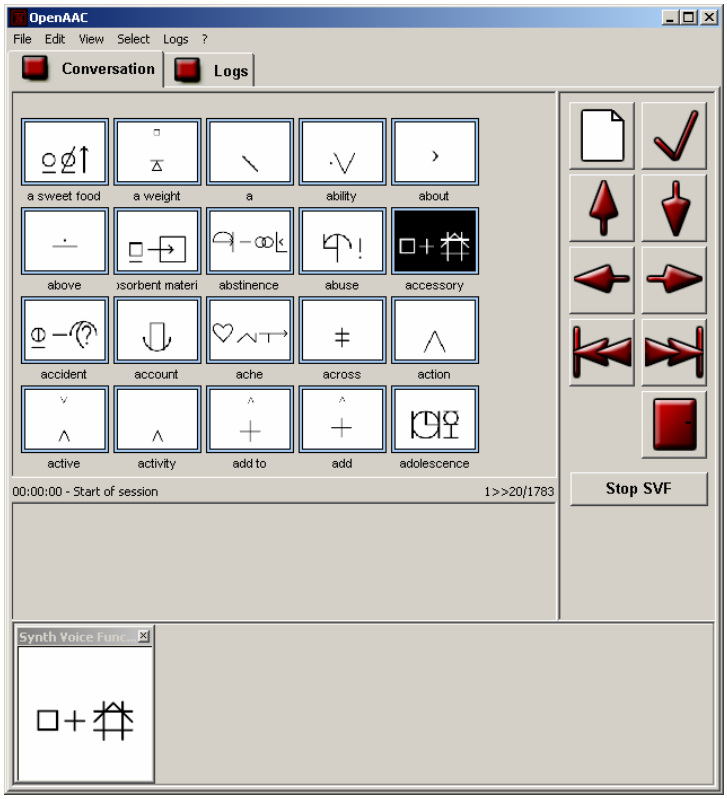

fig. 5.3.2 – Plug-in SVF (Synth Voice Function) integrato e in funzione

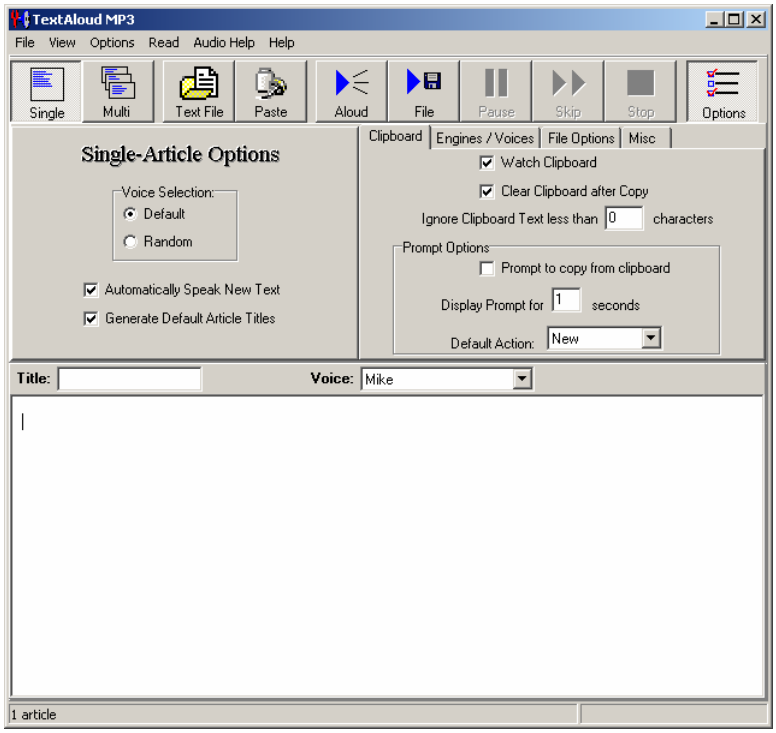

fig. 5.3.3 – TextAloud Mp3 (software di sintetizzazione utilizzato in *background*)

5.4 AutoScan Function (ASF)

La funzione di *auto scansione* è il ruolo che questo *plug-in* ricopre, esso infatti permette di navigare la tabella in modo automatico seguendo determinate logiche di scansione selezionabili dal pannello di controllo posto nella sezione creata *ad-hoc* per tale *plug-in* nella sua fase di integrazione con l'applicazione (*fig. 5.4.3*).
Tramite questa sezione l'utente può decidere la tipologia di *auto scansione* e il tempo di *idle* della scansione.
L'*auto scansione* delega il compito di muovere il cursore lungo la tabella di selezione presente nella sezione di *Conversazione* dell'applicazione al computer, in modo da lascaire all'utente l'unico compito di selezionare il simbolo desiderato dando un input (es. *click del mouse*) nel momento in cui tale simbolo è evidenziato dal cursore stesso, *click* da effettuare direttamente sul simbolo o tramite l'utlizzo dell'area contenuta nella finestra apposita area contraddistinta da un cursore di grosse dimensioni (*fig. 5.4.2*).
Le tipologie di scansione posso essere varie, *lineare* (da sinistra a destra, dall'alto al basso), *riga/colonna* modalità che richiede due conferme dato che prima il cursore scorre lungo la prima colonna permettendo di selezionare la riga che contiene il simbolo desiderato e poi scorre lungo le colonne della riga selezionata per permette di selezionare il simbolo desiderato, *per blocchi* (anch'essa richiede due selezioni) in cui vengono visualizzate le pagine e alla conferma viene scansionata la pagina desiderata in modalità *lineare*, e infine per *categoria*, modalità che muove il cursore solo su simboli appartenenti ad una determinata categoria (questa funzione non è al momento disponibile in quanto i dizionari in possesso non sono *categorizzati*).
Quest'ultima modalità è forse la modalità che esprime il maggiore potenziale, in quanto in una futura implementazione sarà supportato in tale compito da un predittore che su varie basi logiche permetterà di fornire all'utente un numero limitato di simboli tra cui scegliere, tali simboli saranno quelli che con maggiore probabilità corrisponderanno al simbolo desiderato dall'utente stesso.
Questo *plug-in* interagisce con l'applicazione unicamente tramite due tipologie di comandi del protocollo *client*: quelli interenti il movimento del cursore e quelli legati alla selezione del simbolo. Tramite queste due tipologie di comandi il *plug-in* è in grado di simulare il comportamento umano, e combinando tra di essi i comandi permette di scegliere la tipologia di scansione, a seconda delle esigenze dell'utente.

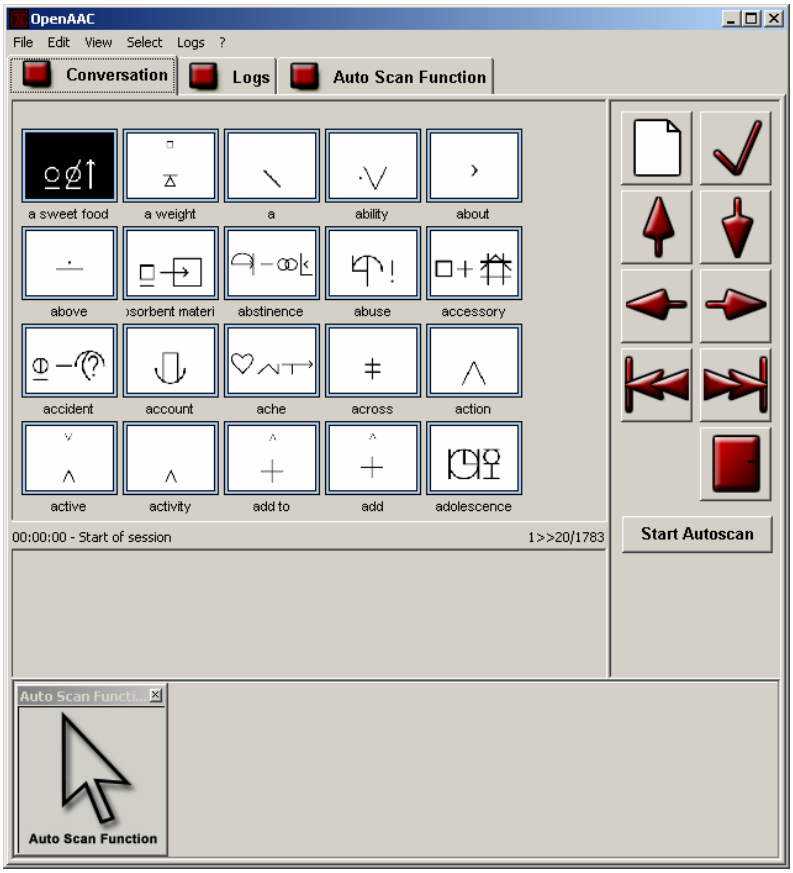

fig. 5.4.1 – Plug-in ASF (AutoScan Function) integrato

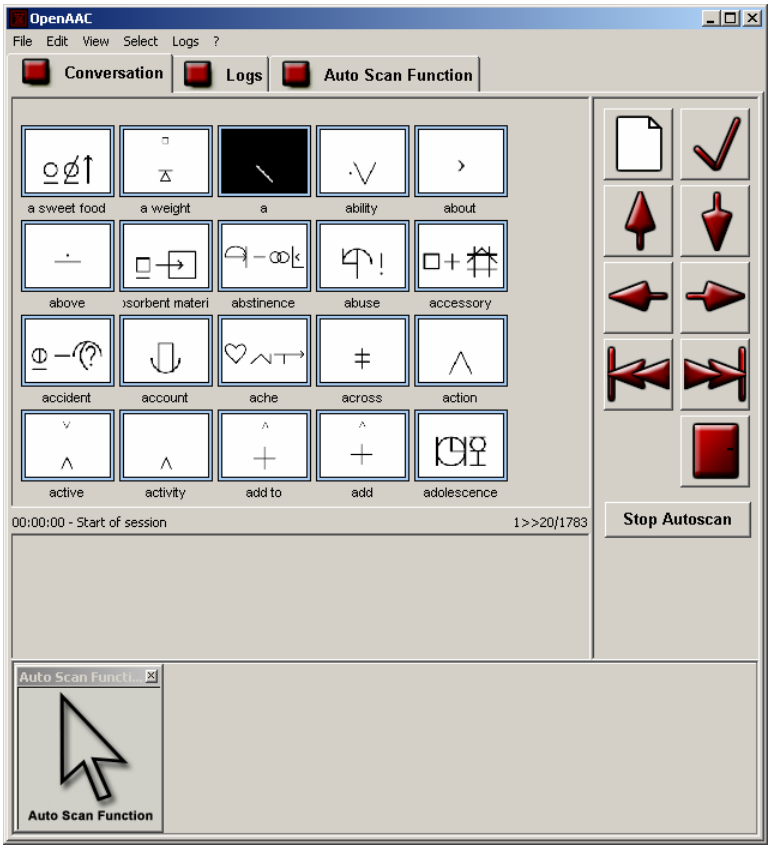

fig. 5.4.2 – Plug-in ASF (AutoScan Function) in funzione

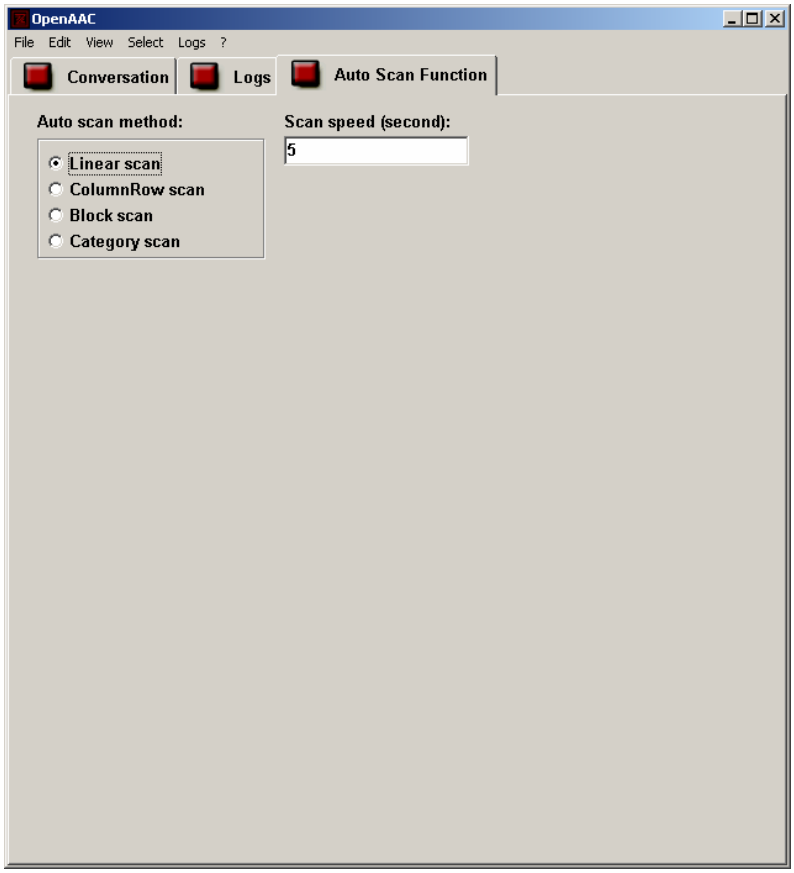

fig. 5.4.3 – Sezione opzioni del Plug-in ASF (AutoScan Function)

6

SVILUPPI FUTURI E CONCLUSIONI

6.1 Sviluppi futuri

In questa sezione saranno introdotti i plug-in di futura implementazione, qui di seguito saranno esposte le specifiche che tali *plug-in* dovranno rispettare:

Modulo per la descrizione di tabelle utente personalizzate
Tramite questo *plug-in* l'utente potrà crearsi delle tabelle risultanti dalla selezione di particolari simboli di suo interesse, al fine di poter personalizzare il dizionario da cui selezionare i simboli.
Tale *plug-in* ricopre una notevole importanza in quanto permette di filtrare i simboli selezionabili in modo da alleggerire la tabella di selezione da tutti quei simboli che non destano l'interesse del singolo utente.
Tale personalizzazione potrà essere effettuata in due modalità, manuale o automatica. La prima modalità cosiste nella selezione diretta dei simboli d'interesse, mentre la seconda modalità agisce in *background* durante l'utilizzo dell'applicazione e esclude nel tempo i simboli mai interessati da una selezione. Quest'ultima modalità, come è chiaro, richiede tempo in quanto si basa su un *autoapprendimento* conseguente all'utilizzo dell'applicazione stessa.

Modulo per la descrizione di nuovi linguaggi grafici
Questo *plug-in* rientra tra i quelli di personalizzazione, esso infatti permetterà di creare nuovi linguaggi grafici tramite l'importazione di simboli in formato *jpeg* e tramite l'importazione dei corrispondenti significati in formato *alfanumerico* (file di testo). Questo *plug-in*, come quello precedentemente illustrato, lavorerà in collaborazione con il *plug-in DE (Dictionary Expander)* tramite il quale sarà possibile selezionare i dizionari e le tabelle utente personalizzate.

Modulo per l'aggiunta di simboli grafici
Tramite l'utilizzo di questo *plug-in* sarà possibile realizzare nuovi simboli grafici, tramite l'utilizzo di una tavolozza o tramite l'importazione di immagini. Esso si troverà a lavorare in collaborazione del precedente *plug-in*, che permetterà ai simboli creati di essere raggruppati in dizionari personalizzati.

Modulo di analisi grammaticale della frase composta
Il modulo di analisi grammaticale della frase composta avrà il compito di analizzare e ricomporre la frase creata tramite la selezione dei simboli grafici al fine di dare una costruzione grammaticale quanto più corretta possibile della frase composta.

Tale struttura grammaticale è spesso assente in frasi composte tramite la selezione dei simboli a causa della natura stessa dei linguaggi grafici, i quali esprimono concetti univoci e non contestualizzati.
Un esempio pratico del lavoro svolto da questo *plug-in* può essere quello di coniugare nella maniera più corretta i verbi, aggiungere coniugazioni e preposizioni e quant'altro legato alla correttezza grammaticale di una frase.

Predittore di sequenza
Il principale compito di questo *plug-in* sarà quello di fornire un forte ausilio nella composizione delle frasi, suggerendo una serie di possibili simboli che l'utente può voler selezionare a seguito delle selezioni precedenti. Tali suggerimenti seguiranno più logiche, da quelle grammaticali, per esempio dopo il soggetto un verbo, dopo un verbo un aggettivo (La pasta -> è -> buona) o logiche di memoria storica, per esempio un determinato utente dopo la parola "Io" spesso pone il verbo "volere", secondo questa logica, se l'utente seleziona la parola "Io" la prima che il *plug-in* suggerirà sarà "volere".
I possibili suggerimenti saranno circa cinque, e nel caso in cui nessuna dei suggerimenti fosse il simbolo pensato l'utente è libero di navigare le tabelle alla ricerca di tale simbolo.
Il *plug-in* che svolgerà la funzione di predittore di sequenza avrà quindi il compito di autoeducarsi alla ricerca dei simboli corretti da suggerire. Tali conoscenze potranno quindi aumentare nel tempo unicamente con l'utilizzo dell'applicazione stessa.

6.2 Conclusioni

Lo sviluppo di OpenAAC è partito dall'idea di creare un sistema informatico di ausilio alla comunicazione verbale, avendo come utente finale persone con disabilità verbali o cognitive. Partendo da questi due presupposti la progettazione è stata supportata da due analisi procedute parallelamente.
Una ha interessato prettamente lo studio di un'architettura in grado di conferire all'applicazione il *titolo* di *sistema informativo con architettettura client/server*, quindi ha avuto come oggetto la progettazione di un software in grado di *colloquiare con applicazioni esterne*.
L'altra ha interessato l'aspetto prettamente legato allo *studio* della tipologia di ausilio che l'*Augumentative Alternative Comunication* fornisce alle persone con disabilità verbali, ha riguardato quindi i linguaggi alternativi esistenti, quali *Bliss* e *Pcs*, e come essi vengono usati al fine riabilitativo.

L'unione di queste due *strade* ha quindi portato alla realizzazione di un software che vuole in qualche modo porre le basi far concepire in modo diverso agli sviluppatori di software lo sviluppo di applicazioni di AAC, ponendo maggiormente l'attenzione sulla necessità di creare applicativi di ausilio in grado di poter essere personalizzati alle esigenze dell'utente.

BIBLIOGRAFIA

[1]D. Danzi, G. Gatti. *Bliss2003: Un sistema di Assistive Technology in aiuto ai disabili verbali*. Tesi di Laurea a.a. 2002-2003

R. Bissi, M. Cannao, G. Cossu, M. L. Gava, G. Moretti, M. Somalvico, and G. Stella. *La comunicazione alternativa. Sistemi comunicativi nelle disabilità verbali.* Franco Angeli, Milano, 1991

C. Bitelli, E. Hoogerwerf, and A. Lisley. Bridge, *Ausili tecnologici contro l'esclusione sociale*. Technical report, AIAS, Bologna, ONLUS 2002

[2] [4]C. K. Bliss *Semantography.* Semantography Blisssymbolic communication, Sidney, 1996

[16]G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modelling Language User Guide.* Addison-Wesley, 1999

E. Calvino, L. Carrubba, I. Grazzani, and M. Groppo. *Psicologia dello sviluppo del linguaggio e della comunicazione.* ISU Università Cattolica, Milano, 1995

R. Campi, M. Fini, S. Lorini, and L. Rossi. *Informatica e handicap.* Il contributo dell'informatica all'uomo per il superamento dell'handicap. Etas, Milano, 1990

A. Defilo. *Parlare non basta.* EDT, 1992

M. L. Gava. *...e se manca la parola, quale comunicazione e quale linguaggio? AAC: una risposta nell'ambito delle disabilità verbali.* Riabilitazione oggi, 1999

M. Martinoni and G. Scascighini. *Pedagogia specializzata e informatica.* Edition SZH/SPC, Biel, Svizzera, 1997

J. McShane. *Lo sviluppo cognitivo.* Il mulino, Bologna, 1994

[10]M.L. Gava, M. Somalvico – *Bliss 2000* – Sviluppato presso il Politecnico di Milano

[6] [12]Sito della *Mayer-Johnson* società che detiene di diritti per il linguaggio *PCS*
http://www.mayer-johnson.com/

[11]Sito *Easy labs* informatica e tecnologie comunicative per disabili
http://www.easylabs.it

[13]Sito *Zygo Inc.* società statunitense che produce ausili informatici alla comunicazione
http://www.zygo-usa.com

[15]Sito *Prentrom* società che produce e distribuisce ausili alla comunicazione
http://www.prentrom.com

[7]Osservatorio regionale sulla disabilità (Regione Piemonte)
http://www.alihandicap.org

[5]Sito d'informazione e sensibilizzazione sulle problematiche dell'afasia
http://www.aphasiaforum.com/

Sito di commercializzazione ausili per disabili
http://www.leonardoausili.com

[9]Sito dell'ente creatore dell'AAC
http://www.isaac-online.org/en/home.html

Mediateca, software e disabilità
http://www.areato.org

Sito della Borland proprietaria di *Borland C++ Builder*
http://www.borland.com

Sito di Microsoft Italia proprietaria di *Microsoft Word*
http://www.microsoft.it

Sito dell'Adobe Systems proprietaria di *Adobe Writer*
http://www.adobe.com

[8]Articolo del settembre 2000, autori Mauro Marin e Lara Lazzaretti dell'Associazione Incremento Comunicazione Alternativa (AICA)

Appendice A

STRALCI DI CODICE

Garbage

```
class CGarbage {
   private:
      struct GarbageItem {
         TObject *obj;
         int ID;
         GarbageItem *next;
      };

      GarbageItem *items;
      GarbageItem *head;

   public:
      CGarbage();
      void addItem(TObject* obj);
      void addItem(TObject* obj, int ID);
      TObject* getItem(int ID);
      void clearGarbage();
};
```

InterfaceDictionary

```
class CLanguageSource {
   private:
   struct InterfaceItem {
      AnsiString ID;
      AnsiString text;
      InterfaceItem *next;
   };
   InterfaceItem *headInterfaceItems;
   InterfaceItem *interfaceItem;
   InterfaceItem *endInterfaceItems;

   void addItem(AnsiString ID, AnsiString text);

   public:
```

```
        CLanguageSource(AnsiString pathLangSource);
        AnsiString getText(int ID);
};
```

SymbolsDictionary e AlphaDictionary

```
class CSymbols {
    private:
        struct TSymbol {
            AnsiString ID_Symbol;
            AnsiString Name_Symbol;
            TSymbol *Next;
        };

        TSymbol *GridPointer;
        TSymbol *FirstSymbol;
        CSymbols *Next;

    public:
        CSymbols();
        AnsiString GetID();
        AnsiString GetName();
        AnsiString GetName(int ID);
        bool NextSymbol();
        void PointToFirstElement();
        bool OffsetTab(int Offset);
        void AddItemToGrid(AnsiString ID, AnsiString Name);
};
```

Core e Interface

```
class TMainForm : public TForm
{
__published:   // IDE-managed Components
    TPageControl *PageControl;
    TTabSheet *ConversationTabSheet;
    TTabSheet *OptionsTabSheet;
    TMainMenu *MainMenu;
    TMenuItem *File;
    TMenuItem *Exit;
    TMenuItem *N1;
    TMenuItem *About;
    TPanel *FuntionButtonPanel;
    TPanel *SelectionSymbolsPanel;
    TPanel *DiscussionPanel;
    TPanel *MainConversationPanel;
    TSpeedButton *SelectionButton;
```

```
  TSpeedButton *LeftButton;
  TSpeedButton *RightButton;
  TSpeedButton *UpButton;
  TSpeedButton *DownButton;
  TSpeedButton *ExitButton;
  TShape *SelectImageShape;
  TSpeedButton *PreviousPage;
  TSpeedButton *NextPage;
  TTimer *AutoScan;
  TLabel *LanguageLabel;
  TBevel *BevelAutoScan;
  TLabel *AutoScanLabel;
  TCheckBox *EnableAS;
  TRadioButton *LowSpeedScan;
  TRadioButton *AvSpeedScan;
  TRadioButton *HiSpeedScan;
  TPanel *SpeedScan;
  TRadioButton *Linear;
  TRadioButton *RowColumn;
  TRadioButton *Categories;
  TRadioButton *Blocks;
  TPanel *KindOfScan;
     TPanel *GridSizePane;
     TLabel *GridSizeLabel;
     TRadioButton *LowRes;
     TRadioButton *MedRes;
     TRadioButton *HiRes;
  TScrollBox *ScrollBoxDiscussion;
  TSpeedButton *BNewSession;
     TMemo *LogMemo;
     TLabel *LogLabel;
     TMemo *DiscussionMemo;
     TLabel *DiscussionLabel;
  TTimer *SessionTime;
  TTimer *SplitTimer;
  TImageList *TabImageList;
  TScrollBox *ScrollBoxSelection;
  TMenuItem *MSelect;
  TMenuItem *Up1;
  TMenuItem *Down1;
  TMenuItem *Left1;
  TMenuItem *Right1;
TMenuItem *Previouspage1;
  TMenuItem *Nextpage1;
  TMenuItem *Selectsymbol1;
  TMenuItem *Edit2;
  TMenuItem *View1;
```

```
    TMenuItem *Option1;
    TMenuItem *MGrid1;
    TMenuItem *MGrid2;
    TMenuItem *MGrid3;
    TMenuItem *MNewSession;
    TMenuItem *Language1;
    TMenuItem *Italiano1;
    TMenuItem *English1;
    TMenuItem *Nofunctionsavaible1;
    TStatusBar *SelectionStatusBar;
    TComboBox *InterfaceLanguageComboBox;
    TPanel *ToolBar;
    TScrollBox *ToolBarScrollBox;
    TServerSocket *ServerSocket;

    void __fastcall ExitClick(TObject *Sender);
    void __fastcall SelectSymbolClick(TObject *Sender, TMouseButton Btn, TShiftState
ShiftState, int x, int y);
    void __fastcall FormResize(TObject *Sender);
    void __fastcall ShowGridButtonClick(TObject *Sender);
    void __fastcall SelectImageShapeMouseDown(TObject *Sender,
        TMouseButton Button, TShiftState Shift, int X, int Y);
    void __fastcall ExitButtonClick(TObject *Sender);
    void __fastcall RightButtonClick(TObject *Sender);
    void __fastcall DownButtonClick(TObject *Sender);
    void __fastcall LeftButtonClick(TObject *Sender);
    void __fastcall UpButtonClick(TObject *Sender);
    void __fastcall SelectionButtonClick(TObject *Sender);
    //0 mostra la pagina attuale
    //1 mostra la pagina successiva
    //2 mostra la pagina precedente
    void __fastcall NextPageClick(TObject *Sender);
    void __fastcall PreviousPageClick(TObject *Sender);
    void __fastcall AutoScanTimer(TObject *Sender);
    void __fastcall EnableASClick(TObject *Sender);
    void __fastcall LowSpeedScanClick(TObject *Sender);
    void __fastcall AvSpeedScanClick(TObject *Sender);
    void __fastcall HiSpeedScanClick(TObject *Sender);
    void __fastcall AutoScanBtnClick(TObject *Sender);
    void __fastcall RowColumnClick(TObject *Sender);
    void __fastcall MedResClick(TObject *Sender);
    void __fastcall LowResClick(TObject *Sender);
    void __fastcall HiResClick(TObject *Sender);
    void __fastcall ConversationTabSheetShow(TObject *Sender);
    void __fastcall BNewSessionClick(TObject *Sender);
    void __fastcall SessionTimeTimer(TObject *Sender);
    void __fastcall LinearClick(TObject *Sender);
```

```
    void __fastcall CategoriesClick(TObject *Sender);
    void __fastcall BlocksClick(TObject *Sender);
    void __fastcall SplitTimerTimer(TObject *Sender);
    void __fastcall FormShow(TObject *Sender);
    void __fastcall Selectsymbol1Click(TObject *Sender);
    void __fastcall Up1Click(TObject *Sender);
    void __fastcall Down1Click(TObject *Sender);
    void __fastcall Left1Click(TObject *Sender);
    void __fastcall Right1Click(TObject *Sender);
    void __fastcall Previouspage1Click(TObject *Sender);
    void __fastcall Nextpage1Click(TObject *Sender);
    void __fastcall MGrid1Click(TObject *Sender);
    void __fastcall MGrid2Click(TObject *Sender);
    void __fastcall MGrid3Click(TObject *Sender);
    void __fastcall MNewSessionClick(TObject *Sender);
    void __fastcall Option1Click(TObject *Sender);
    void __fastcall FormKeyDown(TObject *Sender, WORD &Key,
        TShiftState Shift);
    void __fastcall FormKeyUp(TObject *Sender, WORD &Key,
        TShiftState Shift);
    void __fastcall InterfaceLanguageComboBoxChange(TObject *Sender);
    void __fastcall English1Click(TObject *Sender);
    void __fastcall Italiano1Click(TObject *Sender);
    void __fastcall ScanSpeedComboBoxChange(TObject *Sender);
    void __fastcall ServerSocketClientRead(TObject *Sender,
        TCustomWinSocket *Socket);
    void __fastcall ServerSocketAccept(TObject *Sender,
        TCustomWinSocket *Socket);
    void __fastcall ServerSocketClientDisconnect(TObject *Sender,
        TCustomWinSocket *Socket);
    void __fastcall FormCreate(TObject *Sender);
private:        // User declarations
    int widthFirstGridItem;
    int heightFirstGridItem;
    int leftFirstGridItem;
    int topFirstGridItem;
    int xGap;
    int yGap;

    int tick;
    int tTick;
    bool scanRow;

    int columns;
    int rows;
    int offsetSymbols;
```

```
int LowResCol;
int MedResCol;
int HiResCol;

bool KeyPressed;

CSymbols *Dictionary;
CSymbols *HeadDictionary;

CSymbols *discussionLine;
CSymbols *headDiscussionLine;
CGarbage *selectionSymbolsGarbage;
CGarbage *discussionSymbolsGarbage;

CLanguageSource *languageInterface;
CLanguageSource *languageLog;

int numberOfSymbols;

struct SessionClock {
        int sec;
        int min;
        int hour;

        AnsiString time;
    };

struct ImageCollection {
        int x;
        int y;
        int ID;
        AnsiString Name;

        ImageCollection *next;
    };

int discussionIndex;

CAAC25Server *AAC25Server;

SessionClock *Clock;

ImageCollection *imageCollection;
ImageCollection *headImageCollection;

struct Language {
    AnsiString Name;
```

```
        AnsiString interfacePath;
        AnsiString logPath;
        AnsiString dictionaryPath;
        AnsiString symbolsPath;
    };

    Language *programLanguage;

    Language *italiano;
    Language *english;
    Language *application;

    AnsiString findImageTag();
    void LogEvent(AnsiString event);              //event_notification
    void GetSymbolsInfo();                       //event_notification
    void SetAutoScan();
    void ShowGrid(int showNextOrPreviousPage);
    void ShowDiscussionLine();
    void ResizePanels();                         //event_notification
    void InitInterface();
    void InitGrid(int Columns, int Rows, int Left, int Top, int xGapGridItem, int
yGapGripItem);
    void InitGridCells(int widthGridItem, int heightGridItem);
    void SetInterfaceItems();
    void ShowInterface();                        //event_notification
    void NewSession();                           //event_notification
    void SetItemCaption(CLanguageSource *LanguageInteface, TObject *Sender);
    void CreateSymbol(int WIDTH, int HEIGHT, int LEFT, int TOP, int index, AnsiString
ID, AnsiString Name, TWinControl *parent);
    void ShowSymbol(AnsiString ID, AnsiString Name, int index, int columns,
TWinControl *parent);   //event_notification
    void ResizeGrid();
    void SelectImage(TObject* Sender);
    void AddInDiscussion();
    void MoveSelectionRight();
    void MoveSelectionLeft();
    void MoveSelectionUp();
    void MoveSelectionDown();
    void MoveToPreviousPage();
    void MoveToNextPage();
    void ConfirmSelection();
    void AutoScanTimerTick();
    void TimerTick();

public:          // User declarations
    __fastcall TMainForm(TComponent* Owner);
    //funzioni definite per l'esigenza di remotizzare
```

```
    //informazioni ed eventi
    AnsiString GetFormProperties();
    void SetFormProperties(AnsiString Caption, int Width, int Height, int Left, int Top);
};
```

4.7 Stralci di codice – Server

Header del tipo oggetto COpenAACServer (OpenAACServer.h):

(Server)

```
#include "ComandProcessor.h"

class COpenAACServer {
private:
    TServerSocket *ServerSocket;

    struct SClient {
        CComandProcessor *ComandProcessor;
        int ID_CONNECTION;
        SClient *next;
    };

    SClient *Clients;
    SClient *HeadClients;

public:            // User declarations
    COpenAACServer(TServerSocket* ServerSocket);
    void SendEventBroadcast(AnsiString msg);

    void __fastcall ServerSocketAccept(TObject *Sender,
          TCustomWinSocket *Socket);
    void __fastcall ServerSocketClientRead(TObject *Sender,
          TCustomWinSocket *Socket);
    void __fastcall ServerSocketClientDisconnect(TObject *Sender,
          TCustomWinSocket *Socket);
};
```

ComandProcessor e PlugIn

```
#include "Garbage.h"

class CComandProcessor {
    private:
        TForm *Form;
        TForm *PlugInForm;
        TTabSheet *PlugInTabSheet;

        TWinControl *parent;
```

```
    TCustomWinSocket *Connection;
    int ID_CONNECTION;
    int CountComponents;

    CGarbage* Components;

    void InterfaceProcess(AnsiString command, AnsiString parameter);
    void ApplicationProcess(AnsiString command, AnsiString parameter);

  public:
    CComandProcessor(int id_component, TCustomWinSocket *Socket);
    void Process(AnsiString parameters);
    TForm* GetPlugInForm();

    void __fastcall SendComponentEvent(TObject *Sender);
    void SendApplicationStatus(AnsiString msg);
    void SendErrorMessage(AnsiString msg);
    void CreateSymbol(int ID_COMPONENT, int WIDTH, int HEIGHT, int LEFT, int
TOP, AnsiString FilePath, AnsiString Name, TWinControl *parent, int Color);

    void DestroyComponents();
};
```

Case estratto dal file ComandProcessor.cpp, per la selezione del proprietario dell’oggetto remoto da intergrare:

```
[...]
case 0: OWNER = StrToInt(Value);
        switch (OWNER) {
          case 0: //parent = MainForm;
                  parent = MainForm->ConversationTabSheet;
            break;
          case 1: //controllo d'esistenza
                  if (PlugInForm != NULL)
                    parent = PlugInForm;
                  else
                    //ID_error = 0 Parent non esistente
                    SendErrorMessage("0~");
            break;
          case 2: //controllo d'esistenza
                  parent = PlugInTabSheet;
            break;
          case 3: parent = MainForm->ToolBar;
            break;
          case 4: parent = MainForm->OptionsTabSheet;
            break;
```

```
            case 5: parent = MainForm->PageControl;
                break;
        }
        break;
[...]
```

ClientComandProcessor

```
class CClientComandProcessor {
private:
    TClientSocket *ClientSocket;

    void ProcessMessage(AnsiString Message);

    AnsiString TempMessage;

public:
    CClientComandProcessor(TClientSocket *ClientSocket);

    struct TMessageFromServer {
        //tramite questa struct è possibile recuperare il contenuto
        //dei messaggi ricevuti dal server, e la loro tipologia
        int ID_Type_Message;
        int ID_SubType_Message;

        //contiene gli eventuali parametri contenuti nel messaggio
        //sono separati da virgole
        AnsiString TextOfMessage;

        bool read;

        TMessageFromServer *NextMessage;
    };

    TMessageFromServer *MessageFromServer;
    TMessageFromServer *HeadMessageFromServer;

    void SendToServer(AnsiString Message);
    void ReceiveFromServer(AnsiString Message);

    //da client a server interfaccia
    void CreateForm(int OwnerID, int ID_Component, AnsiString Caption, int Left, int
Top, int Width, int Height);
    void CreateButton(int OwnerID, int ID_Component, AnsiString Caption, int Left, int
Top);
    void CreateLabel(int OwnerID, int ID_Component, AnsiString Caption, int Left, int
Top);
```

```
    void CreateEdit(int OwnerID, int ID_Component, AnsiString Caption, int Left, int
Top);
    void CreateSection(int OwnerID, int ID_Component, AnsiString Caption);
    void CreateCheckBox(int OwnerID, int ID_Component, AnsiString Caption, int Left,
int Top);
    void CreateRadioButton(int OwnerID, int ID_Component, AnsiString Caption, int
Left, int Top);
    void CreateImage(int OwnerID, int ID_Component, AnsiString path, int Left, int Top,
int Width, int Height, int Color);

    //da client a server
    void NewSessionFromRemote();
    void SetFormProperties(AnsiString Caption, int Width, int Height, int Left, int Top);
    void SetGridDimension(int Columns, int Rows);
    void SetSymbolsDimension(int Width, int Height);
    void MoveCursor(int Rel_Columns, int Rel_Rows);
    void SelectSymbol();
    void ChangeGridPage(int NumberOfPage);
    void ChangeInterfaceDictionary(int ID, AnsiString Path);
    void ChangeSymbolsDictionary(int ID, AnsiString Path);
    void ChangeAlphaDictionary(int ID, AnsiString Path);
    void GetFormStatus();
    void GetDictionariesID();
    void GetDictionariesPath();
    void GetLocationForForm();
    void GetLocationForTBComponent();
    void GetConversationStatus();
    void GetGridDimension();
    void GetSymbolsDimension();
    void GetCursorPosition();
    void SetFocusOnConversationSection();
    void SetFOcusOnPlugInSection();
};
```