UNIVERSITÀ DI PISA

Facoltà di Scienze, Matematiche, Fisiche e Naturali

Corso di laurea in Tecnologie Informatiche

Progettazione e sviluppo di interfacce multimodali composte da grafica e voce per piattaforme desktop e mobile basata su modelli

Candidato:

Marco Manca

Relatore:

Prof. Fabio Paternò

Anno accademico 2009/2010

A Teresa

Sommario

L'obiettivo di questa tesi è la definizione di un linguggio, basato su XML, che descriva in maniera logica le interfacce multimodali integrando la modalità grafica con quella vocale. Viene proposto un metodo per la trasformazione da interfaccia logica a interfaccia multimodale reale, utilizzando template XSLT che producono codice XHTML+Voice.

Abstract

This thesis aims to define a XML-based language for the logical description of multimodal interfaces composed by graphical and vocal modality. It proposes a method to transform such logical description in actual multimodal interfaces by XSLT templates which generate XHTML+Voice code.

Indice

Elenco delle figure

Capitolo 1

Introduzione

La grande maggioranza delle interfacce utente (UI[1]) nelle attuali applicazioni è di tipo grafico (GUI[2]); dal momento che le GUI restringono l'interazione uomo macchina (HCI[3]) alla sola interazione visuale, non permettono di sfruttare le svariate modalità di interazione possibili. Inoltre le GUI come le abbiamo sempre viste non sono adatte per alcuni utenti che presentano delle difficoltà a battere sulla tastiera o difficoltà visive, oppure in alcune circostanze in cui le mani sono occupate in altri compiti (es. Guidare).

Un'altra motivazione che ha spinto la ricerca di nuove modalità di interazione è la diffusione massiccia dei dispositivi mobili, i quali permettono di accedere ai contenuti in tutti i luoghi, ma sono affetti da problemi dovuti alla dimensione ridotta del display e della tastiera; basta provare a immaginare come sarebbe più agevole compilare un form su un sito web attraverso dei comandi vocali.

Per superare i problemi legati alle interfacce grafiche si ha bisogno di un nuovo paradigma di interfacce utente: le interfacce multimodali (MMUI[4]) nelle quali vengono considerate simultaneamente molteplici metodologie di interazione tra uomo e macchina. Queste metodologie includono la tastiera, touch

[1]User Interface
[2]Graphical User Interface
[3]Human Coumputer Interaction
[4]Multimodal User Interface

screen, riconoscimento della scrittura manuale, sintesi e riconoscimento vocale, riconoscimento dei gesti e dei movimenti. In questa tesi ci focalizzeremo sulla sintesi e riconoscimento vocale e sulla interazione con le componenti grafiche.
L'evoluzione tecnologica sta rendendo la tecnologia multimodale disponibile per il mercato di massa con un'affidabilità sempre maggiore. [24].

Le interfacce multimodali stanno conoscendo un'ampia espansione grazie all'aumento dell'accuratezza nella percezione dei sistemi di input (voice, handwriting e visual recognition) e grazie all'aumento dell'ubiquità delle piattaforme (mobile phones, PDA, laptop, etc).

Obiettivi

Il mio lavoro di tesi si colloca all'interno del progetto europeo SERVFACE[5] [1] in corso d'opera presso il laboratorio HIIS[6] dell'ISTI[7] al CNR[8]. Più specificatamente mi sono occupato dell'evoluzione della sezione del progetto chiamata MARIA[9] [27].

Gli obiettivi di questa tesi sono:

1. Progettare e realizzare, a partire da un modello di interfaccia astratta preesistente, un linguaggio basato su XML[10] in grado di descrivere logicamente interfacce multimodali composte da grafica e voce.

2. Implementare una trasformazione XSLT[11] in grado di prendere in input una suddetta descrizione logica e generare la relativa interfaccia multimodale (in codice XHTML+VOICE).

3. Estendere il tool MARIAE[12] fornendogli il supporto per la realizzazione di descrizioni logiche di interfacce multimodali secondo il linguaggio realizzato.

4. Mostrare i risultati ottenuti proponendo un caso di studio complesso

[5] Service Annotations for User Interface Composition
[6] Human Interfaces in Information Systems
[7] Istituto di Scienza e Tecnologie dell'Informazione
[8] Consiglio Nazionale della Ricerca
[9] Model b-Ased descRiption of Interactive Application
[10] Extensible Markup Language
[11] eXtensible Stylesheet Language Transformations
[12] MARIA authoring Environment

Contenuto

Il capitolo due affronta la storia delle interfacce multimodali attraverso la letteratura. Successivamente viene fatta una panoramica dello stato dell'arte dei vari linguaggi e framework che supportano le interfacce multimodali. Inoltre viene data una definizione formale delle CARE properties utilizzate per descrivere com combinare la modalità grafica con quella vocale.

Nel capitolo tre verrà introdotto il contesto in cui si colloca la tesi, ovvero verranno presentati il linguaggio preesistente (MARIA) ed il tool di authoring (MARIAE).

Il quarto capitolo presenta la progettazione e le caratteristiche del linguaggio per la descrizione delle interfacce multimodali dando particolare rilievo alle CARE properties e ai valori che queste possono assumere in ogni elemento del linguaggio.

Nel capitolo seguente viene effettuata una panoramica del linguaggio XHTML+Voice trattando argomenti come i costrutti di dialogo, le grammatiche, l'output, la gestione degli eventi e il setting delle proprietà.

Il capitolo sei è composto da una breve rassegna sull'utilizzo dell'XSLT: dai template ai principali costrutti di supporto; per poi concentrarsi sul processo di trasformazione da interfaccia logica a codice XHTML+Voice.

Lo scopo del capitolo sette è mostrare come dalla definizione del linguaggio si è arrivati alla sua integrazione nell'authoring environment MARIAE.

Il capitolo otto presenta un'applicazione pratica mettendo in evidenza come effettivamente si possa utilizzare il tool MARIAE per generare interfacce multimodali e mostrando il risultato della generazione dell'interfaccia multimodale finale.

Il capitolo nove riassume il lavoro di tesi, analizzando i risultati ottenuti e proponendo nuove idee per futuri sviluppi.

Capitolo 2

Le interfacce multimodali

2.1 Cosa sono?

Le interfacce mutimodali sono delle interfacce in cui due o più modalità di input, come vocale, tastiera, mouse, touch, gesti della mano, lo sguardo degli occhi, movimenti della testa e del corpo, vengono processati in maniera coordinata con gli output di sistemi multimediali. [22]

Prima di analizzare lo stato dell'arte attuale è bene tornare indietro nel tempo e ripercorrere il cammino che ci ha portato sino ai giorni nostri.

La comunicazione faccia a faccia è sempre stata e continua ad essere multimodale perché ha interessato tutti i modi in cui è possibile comunicare: parole, gesti, immagini, tatto, odori. Nella comunicazione oltre alla voce ha grande importanza l'immagine, non solo per comprendere la situazione in cui le informazioni vocali si inseriscono, ma anche per valutare la reazione emotiva che la persona con la quale si comunica esprime attraverso il viso e il corpo. La comunicazione ha distanza è nata unimodale e solo recentemente si è evoluta, in quanto era basata solo sulla scrittura o sui disegni. Anche i mezzi di comunicazione moderni, come il telefono e il telegrafo, inizialmente erano unimodali. La prima rivoluzione si è avuta solo con la nascita del cinema e della televisione coi quali si inizia a comunicare simultaneamente in più modi con strumenti differenti coniugando audio, video e testo; per arrivare

ai giorni nostri in cui si prevede di aggiungere agli strumenti descritti il tatto e l'olfatto.

Le interfacce multimodali rappresentano una nuova direzione per l'informatica e un cambiamento di paradigma rispetto alle interfacce convenzionali WIMP[1] [22], nelle quali l'interazione con la macchina avveniva solo attraverso il mouse e lo schermo grafico.

Sin dall'apparizione del sistema Put that there [6] sviluppato da Bolt, nel quale oggetti grafici venivano creati e spostati in uno schermo utilizzando il riconoscimento vocale e il puntamento con le dita, sono emerse una varietà di sistemi multimodali e questi sono stati visti come un'importante area di ricerca nel campo dell'HCI.

Il crescente interesse verso le interfacce multimodali è guidato dall'obiettivo di supportare in maniera più trasparente, flessibile, efficiente e con maggiore espressività l'interazione uomo-macchina [22]. L'utente deve avere facilità e naturalezza nell'utilizzo di questo tipo di interfacce; tutto questo è ottenuto lasciando all'utente la possibilità di scelta del tipo di input preferito e favorendo così un grande numero di utenti e task. Per esempio, nel caso di utenti disabili o in caso di utilizzo dei sistemi in situazioni avverse (es. Ambienti rumorosi, impossibilità di utilizzare le mani, durante la guida) alcuni task non potevano essere eseguiti a causa del fatto che una singola modalità di input non era sufficiente.

I sistemi che supportano input multimodali mirano a fornire agli utenti strumenti migliori per controllare visualizzazioni sofisticate e le potenzialità multimediali degli output che sono ormai integrati in molti sistemi. Al contrario, gli input tramite tastiera e mouse sono relativamente limitati, specialmente quando si tratta di interagire con ambienti virtuali. Inoltre vari studi hanno dimostrato che nuovi e complessi task sono risolti più velocemente con l'uso di modalità differenti, rispetto a interfacce GUI [9], ad esempio in [20] si è dimostrato un incremento dell'efficienza del 20 – 40 % usando sistemi vocali

[1] Windows Icons Menus Pointers

in confronto a interfacce che utilizzano tecnologie differenti, come input da tastiera. Nonostante l'interazione vocale possa sembrare molto promettente, potenziali problemi tecnici (es. Rumori di fondo, microfoni scadenti, etc) con le interfacce vocali potrebbero irritare l'utente e ridurre l'efficienza [28], questo è il motivo per cui le interfacce vocali sono spesso combinate con altre modalità, per bilanciare la debolezza di quest'ultime [10]. Specialmente le GUI sono combinate con la modalità vocale.

2.2 Stato dell'arte

Lo sviluppo di interfacce multimodali attualmente è ancora difficoltoso a causa della mancanza di ambienti di authoring sviluppati per questo scopo; in questa sezione verranno presentati vari linguaggi per la descrizione di interfacce con una particolare attenzione verso quelle multimodali.

Il tool Damask [18] introduce il concetto di layers per supportare lo sviluppo di interfacce per più dispositivi (desktop, mobile, voice). Lo sviluppatore può specificare gli elementi che appartengono a tutte le versioni dell'interfaccia e gli elementi propri di un dispositivo. Questo approccio è utile per lo sviluppo separato dell'interfaccia che riguarda una sola modalità alla volta ma non introduce un supporto adatto a indicare come comporre diverse modalità insieme.
XFormsMM [14] è un tentativo di estendere XForms in modo da ricavare entrambe le interfacce grafiche e vocali. L'idea base è specificare i controlli astratti con gli elementi XForms e utilizzare *aural* e *visual* CSS[2] rispettivamente per il rendering vocale e grafico. Il problema è che i CSS aural possiedono limitate capacità in termini di interazione vocale e la soluzione proposta per lavorare richiede un ambiente ad hoc.
In Teresa [23] è affrontata la possibilità di descrivere interfacce multimodali, ma le trasformazioni sono integrate nell'codice per l'implementazione del tool

[2]Cascading Style Sheet

e le descrizioni logiche non sono in grado di descrivere le interazioni tipiche del Web2.0 e l'accesso ai Web services.
Obrenovic et al. [21] hanno esaminato l'uso di modelli concettuali espressi in UML in modo da ricavare interfacce grafiche per desktop e mobile o interfacce vocali. UML è uno standard per l'ingegneria del software utilizzato per descrivere le funzionalità interne di un'applicazione, per cui è inadatto a catturare e descrivere le caratteristiche specifiche di un'interfaccia utente e le interazioni tra gli elementi che la compongono.
XISL[3] [16] è un linguaggio di markup basato su XML per la definizione di sistemi web di interazione multimodale. XISL permette la descrizione della sincronizzazione degli input/output multimodali e il controllo del flusso e della transizione tra i dialoghi. Il linguaggio permette la separazione dei contenuti (definiti in file HTML) dall'interazione (definita in documenti XISL). Il problema introdotto da questo linguaggio è che ha bisogno di un suo interprete e un suo sistema di esecuzione ad hoc.
Il framework OpenInterface [2] fornisce una piattaforma open source per il design e lo sviluppo di applicazioni multimodali permettendo di esplorare differenti modalità di interazione. Gli oggetti base manipolati dalla piattaforma sono chiamati componenti e forniscono un insieme di servizi/funzioni che variano da driver dei dispositivi di input, moduli di rete, interfacce grafiche, etc. Per manipolare un componente la piattaforma richiede la descrizione dell'interfaccia del componente che può essere specificata in OICDL[4]. L'ambiente grafico del framework è chiamato OIDE[5] permette di assemblare componenti in modo da specificare una pipeline che definisce l'interazione multimodale. Nella figura 2.1 viene mostrato un semplice esempio di una pipeline che combina comandi vocali con il puntamento tramite gesti. Questo framework è molto interessante perchè permette di integrare facilmente diverse modalità di input all'interno di un'interfaccia, il problema è che l'interfaccia sviluppata ha bisogno di un ambiente ad hoc per l'esecuzione.

[3] eXtensible Interaction Scenario Language
[4] OpenInterface Component Description Language
[5] Openinterface Interaction Development Environment

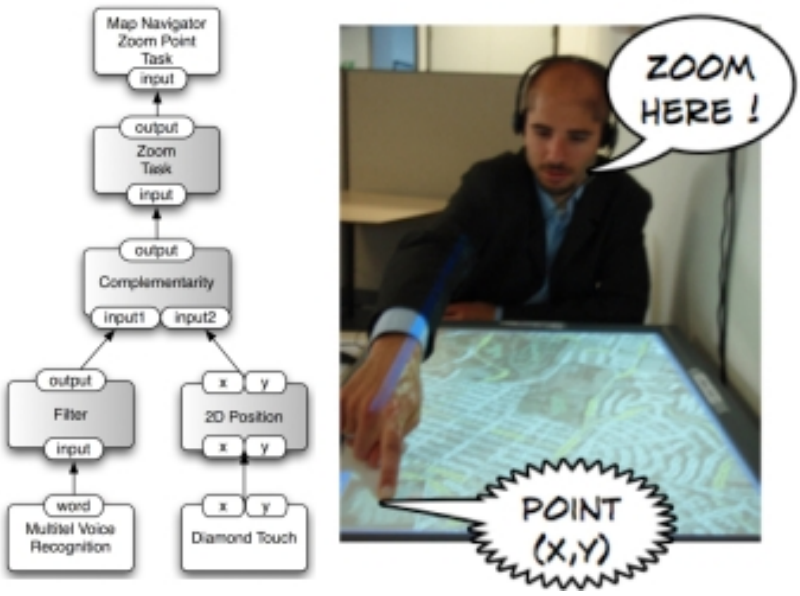

Figura 2.1: OpenInterface pipeline

In [17] [19] si definiscono i requisiti identificati dal W3C per l'interazione multimodale.
Nella figura 2.2 vengono mostrati i componenti base che costituiscono il W3C multimodal interaction framework:

1. Human user: inserisce l'input nel sistema e osserva e ascolta le informazioni presentate dal sistema;

2. Input component: contiene multiple modalità di input come audio, testo sintetizzato, scrittura a mano e da tastiera, mouse, etc;

3. Output component: supporta una o più modalità di output come audio, testo sintetizzato, testo, immagini e animazioni.

4. Integration manager: è il componente che coordina i dati e gestisce il flusso di esecuzione tra le varie modalità di input e output;

5. Session component: fornisce un'interfaccia all'integration manager per la gestione dello stato e la persistenza delle sessioni nelle applicazioni multimodali;

6. System and Environment component: permette all'integration manager di conoscere e rispondere ai cambiamenti delle proprietà dei devices, delle preferenze dell'utente e delle condizioni ambientali (es. quale modalità l'utente vuole utilizzare, risoluzione del display, etc)

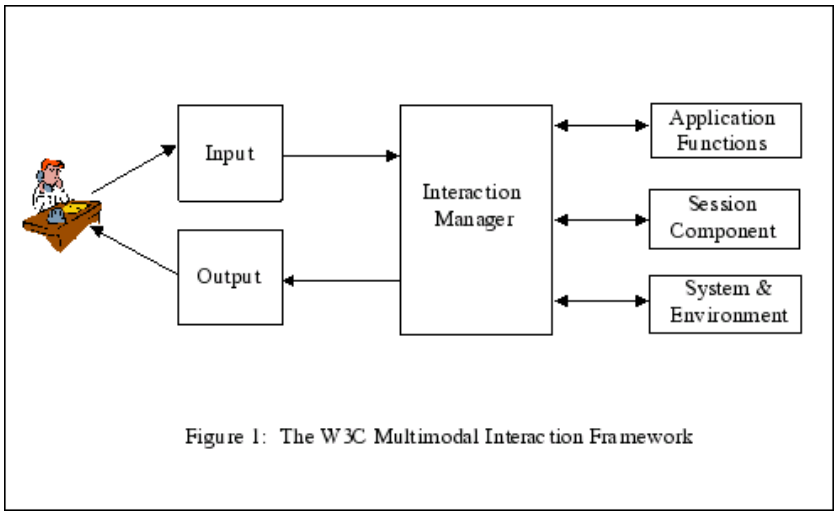

Figura 2.2: Componenti base del w3c multimodal framework

L'interazione multimodale estende le interfacce web permettendo multiple modalità di interazione e dando la possibilità agli utenti di scegliere di utilizzare la propria voce o altri dispositivi di input.

Nella figura 2.3 sono presentati i moduli che compongono il componente che gestisce l'input:

1. Recognition component: cattura l'input dall'utente e lo trasforma in una forma più pratica (useful) per essere processata in seguito.

2. Interpretation component: identifica il significato o la semantica intesa dall'utente, ad esempio parole come 'yes', 'ok', 'I agree' possono essere interpretate come 'yes';

3. Integration component: combina l'output proveniente dai vari interpretation component, in questa fase possono presentarsi delle inconsistenze

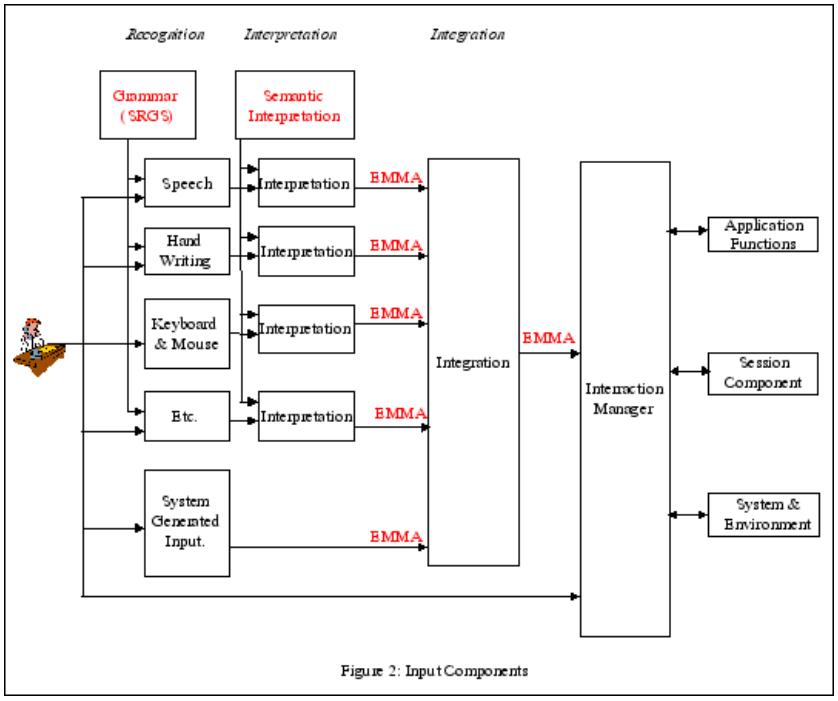

Figura 2.3: W3C multimodal framework input component

ad esempio nel caso in cui un utente dice 'si' (interazione vocale) ma clicca 'no'.

L'input può essere classificato come sequenziale, simultaneo o composto:

- sequenziale: è l'input ricevuto in una singola modalità che può però cambiare nel corso del tempo;
- simultaneo: è l'input ricevuto da multiple modalità e viene trattato separatamente;
- composto: è l'input ricevuto da multiple modalità nello stesso momento e viene trattato come se fosse un unico input.

Nella figura 2.4 sono presentati i moduli che compongono il componente che gestisce l'output:

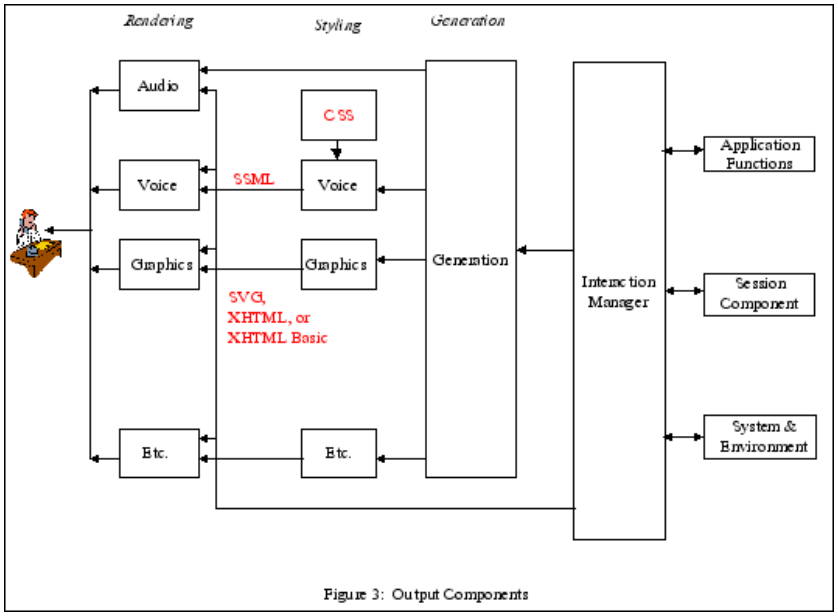

Figura 2.4: W3C multimodal framework input component

1. Generation component: determina quali modalità di output verranno utilizzate per presentare le informazioni dall'interaction manager all'utente;

2. Styling component: aggiunge informazioni su come l'output verrà presentato all'utente; per esempio lo styling component per un display specifica come gli oggetti grafici vengono disposti nel canvas, mentre per quanto riguarda l'audio specifica pause e modulazioni della voce nel testo che verrà poi renderizzato da un sintetizzatore vocale.

3. Rendering component: converte le informazioni provenienti dallo styling component in un formato facilmente comprensibile dall'utente.

Ogni modalità di output possiede sia i componenti di styling e di rendering. Lo style component per la voce costruisce stringhe di testo contenenti SSML[6] tag che descrivono come le parole devono essere pronunciate, questi poi verranno convertiti in voce dal rendering component vocale. Lo style component

[6]Speech Synthesis Markup Language

per la grafica crea XHTML[7] o SVG[8] tag che descrivono come gli oggetti grafici verranno visualizzati. Si può utilizzare SMIL[9] per coordinare i vari output multimediali.

La possibilità per l'utente di specificare la modalità o il device per un'interazione in una particolare situazione permette di migliorare l'accessibilità, l'affidabilità dell'interfaccia utente, specialmente per le applicazioni destinate a dispositivi mobili. L'utente interagisce con l'applicazione nel contesto di una sessione (intervallo di tempo in cui un'applicazione e il suo contesto sono associate a un utente in maniera persistente) usando una o più modalità; all'interno di una sessione l'utente può sospendere e riprendere l'interazione con l'applicazione utilizzando la stessa modalità oppure può cambiare modalità.

2.3 CARE properties

Durante il mio lavoro di tesi mi sono concentrato sulla progettazione e la realizzazione, a partire da un modello di interfaccia astratta preesistente, di un linguaggio basato su XML in grado di descrivere logicamente interfacce multimodali composte dall'unione della modalità grafica con quella vocale.

Durante la progettazione è stato necessario indicare un modo per combinare la modalità grafica e la modalità vocale e per questo motivo sono state considerate le proprietà CARE[10] [11]. Le proprietà CARE sono state interpretate nel modo seguente:

- Complementarity: La parte considerata dell'interfaccia è considerata come parzialmente supportata da una modalità e parzialmente dall'altra;
- Assignment: La parte considerata dell'interfaccia è supportata da una modalità assegnata (graphic assignment o vocal assignment);

[7] eXtensible HyperText Markup Language
[8] Scalable Vector Graphics
[9] Synchronized Multimedia Integration Language
[10] Complementarity, Assignment, Redundancy, Equivalence

- Redundancy: La parte considerata dell'interfaccia è supportata da entrambe le modalità;
- Equivalence: La parte considerata dell'interfaccia è supportata da una modalità o dall'altra.

In [11] viene data una definizione formale delle CARE properties che dipende dalle nozioni di stato (state), obiettivo (goal), modalità (modality) e relazioni temporali (temporal relationship).
Uno *stato* è un insieme di proprietà che possono essere misurate in un momento preciso per caratterizzare una situazione. Un *obiettivo* è lo stato che un agente intende raggiungere. Un *agente* è un entità capace di eseguire delle azioni (es. un utente o un sistema). Una *modalità* è un metodo che un agente utilizza per raggiungere l'obiettivo. Una *relazione temporale* caratterizza l'uso, lungo un periodo di tempo, di un insieme di modalità. L'utilizzo di queste modalità può avvenire simultaneamente o in sequenza all'interno di una finestra temporale.

Basandosi su questi parametri è possibile fornire una definizione formale delle CARE properties:
Equivalenza (E): Le modalità di un insieme M sono *equivalenti* per raggiungere lo *stato* $s^{'}$ a partire dallo *stato* s, se è necessaria e sufficiente una qualunque di queste modalità. Si assume che M contenga almeno 2 modalità:

$$Equivalence(s, M, s^{'}) \Leftrightarrow (Card(M) > 1) \wedge (\forall m \in MReach(s, m, s^{'}))$$

Assegnamento (A): La modalità m è detta *assegnata* se per raggiungere uno *stato* $s^{'}$ a partire da uno *stato* s non vengo utilizzate altre modalità:

$$Assignment(s, m, s^{'}) \Leftrightarrow Reach(s, m, s^{'}) \wedge (\forall m^{'} \in MReach(s, m^{'}, s^{'})) \Rightarrow m^{'} = m$$

La differenza del caso dell'equivalenza, l'assegnamento esprime l'assenza di una scelta per raggiungere lo stato obiettivo.

Ridondanza (R): Le modalità di un insieme M sono utilizzate in modo *ridondante* se per raggiungere uno *stato* $s^{'}$ a partire da uno *stato* s hanno lo stesso potere espressivo (sono equivalenti) e vengono tutte utilizzate all'interno della stessa finestra temporale (tw):

$$Redundancy(s, M, s^{'}, tw) \Leftrightarrow Equivalence(s, M, s^{'}) \wedge (Sequantial(M, tw) \vee Parallel(M, tw))$$

Complementarietà (C): Le modalità di un insieme M sono usate in maniera *complementare* se per raggiungere lo *stato* $s^{'}$ a partire dallo *stato* s devono essere usate tutte, cioè se nessuna presa singolarmente è in grado di raggiungere lo stato obiettivo.

$$Complementarity(s, M, s^{'}, tw)$$

$$\Leftrightarrow (Card(M) > 1) \wedge (Duration(tw) \neq \propto) \wedge$$

$$(\forall M^{'} \in PM(M^{'} \neq M \Rightarrow \neg REACH(s, M^{'}, s^{'}))) \wedge$$

$$REACH(s, M, s^{'}) \wedge (Sequential(M, tw) \vee Parallel(M, tw))$$

Capitolo 3

MARIA

MARIA è, per sua definizione, un linguaggio dichiarativo universale per applicazioni Service-Oriented in ambienti ubiquitari. Esiste poi ¨MARIA Authoring Environment˝(MARIAE), un tool che permette di utilizzare il linguaggio per descrivere interfacce a vari livelli di astrazione e di trasformare tali descrizioni in effettive interfacce concrete.

Le prossime sezioni, basate su [27], forniscono una panoramica di base.

3.1 Il linguaggio

MARIA nasce in concomitanza con la grande diffusione, in questi ultimi anni, di applicazioni basate su web services e con la massiccia presenza sul mercato di dispositivi mobili dotati dei più disparati sensori (es. multitouch dell'Apple ™iPhone).

Il linguaggio trova impiego sia lato progettazione, fornendo supporto per la realizzazione di web services ed applicativi web, che a runtime come parte del processo di migrazione di un interfaccia in ambienti ubiquitari.
Uno degli obiettivi perseguiti è quello di fornire allo sviluppatore un mezzo pratico per poter realizzare interfacce, disinteressandosi degli aspetti tecnici legati al linguaggio di programmazione con cui verrà implementata l'interfaccia e delle peculiarità dei singoli dispositivi. Infatti, sviluppare le interfacce

utente separatamente per ogni dispositivo target aumenta il tempo di sviluppo, il costo per il mantenimento della consistenza tra le varie piattaforme, complica i problemi di gestione della configurazione e impegna le risorse disponibili che dovrebbero dedicarsi all'usabilità [26]. La soluzione a questo problema è utilizzare una descrizione astratta dell'interfaccia che sia indipendente dalla piattaforma in modo che lo sviluppo possa concentrarsi sulle attività logiche che devono essere supportate e le relazioni tra loro.

Dal momento che l'accesso ad applicazioni remote avviene soprattutto attraverso web services che sono definiti prima delle applicazioni che ne faranno uso, l'interesse si concentra sul riutilizzo delle funzionalità fornite da quest'ultimi, sul disegno e sviluppo dei SFE[1] (che generalmente non sono inclusi) la cui interfaccia si adatti automaticamente al dispositivo target.

Lo sviluppo di MARIA riprende dalla precedente esperienza del progetto TERESA [23]. L'ampio utilizzo in diversi settori ha permesso di raccogliere una gran quantità di suggerimenti relativi ad usabilità e funzionalità. Tali suggerimenti possono essere così riassunti:

- Massima flessibilità : lo sviluppatore deve avere il controllo totale dell'interfaccia in ogni fase della progettazione;
- Gestione di modelli di dialogo e navigazione complessi;
- Supporto a tecnologie che permettano di cambiare il contenuto di un'interfaccia in maniera asincrona rispetto all'interazione con l'utente, ad esempio attraverso AJAX[2];
- Inclusione di un modello dei dati (assente in TERESA);
- Leggibilità e consistenza: la specifica delle interfacce astratte e concrete in Teresa era troppo verbosa a causa di molte ridondanze, per questo motivo è stata introdotta una *cross-reference* tra gli XSD[3], l'interfaccia

[1]Service Front End
[2]Asynchronous Javascript And XML
[3]XML Schema Definition

concreta estende quella astratta aggiungendo gli elementi relativi al dispositivo considerato;

- Supporto per la creazione di applicazioni front-end a partire da funzionalità preesistenti in Web Services.

- Le trasformazioni per generare l'implementazione finale non devono essere codificate nel codice (come avveniva in Teresa) ma devono essere specificate esternamente per permettere una personalizzazione senza cambiare l'implementazione del tool, che richiede uno sforzo considerevole.

Nella prossima sezione analizzeremo più nel dettaglio le caratteristiche di questo nuovo linguaggio.

3.1.1 Caratteristiche

L'approccio modulare di MARIA (fig. 3.1) riprende quello di TERESA: al livello più alto troviamo la descrizione dell'interfaccia astratta (AUI[4]), mentre nel livello sottostante le descrizioni delle varie interfacce concrete, dipendenti dalle piattaforme, che raffinano l'astratta considerando le risorse disponibili. L'ultimo livello infine è dedicato ai linguaggi di programmazione coi quali verranno implementate effettivamente le interfacce.
In MARIA è stata introdotta una descrizione astratta del modello dati dell'interfaccia. Tale descrizione è necessaria per la corretta rappresentazione dei tipi di dati e dei loro valori gestiti dall'interfaccia utente. Infatti, tramite la definizione di un modello di dati, gli interattori (che rappresentano gli elementi dell'interfaccia), che compongono un'interfaccia astratta (o concreta), possono essere associati ad un tipo di dato specifico o a un elemento di un tipo definito nel modello dati astratto (o rispettivamente concreto). Il mo-

[4]Abstract User Interface

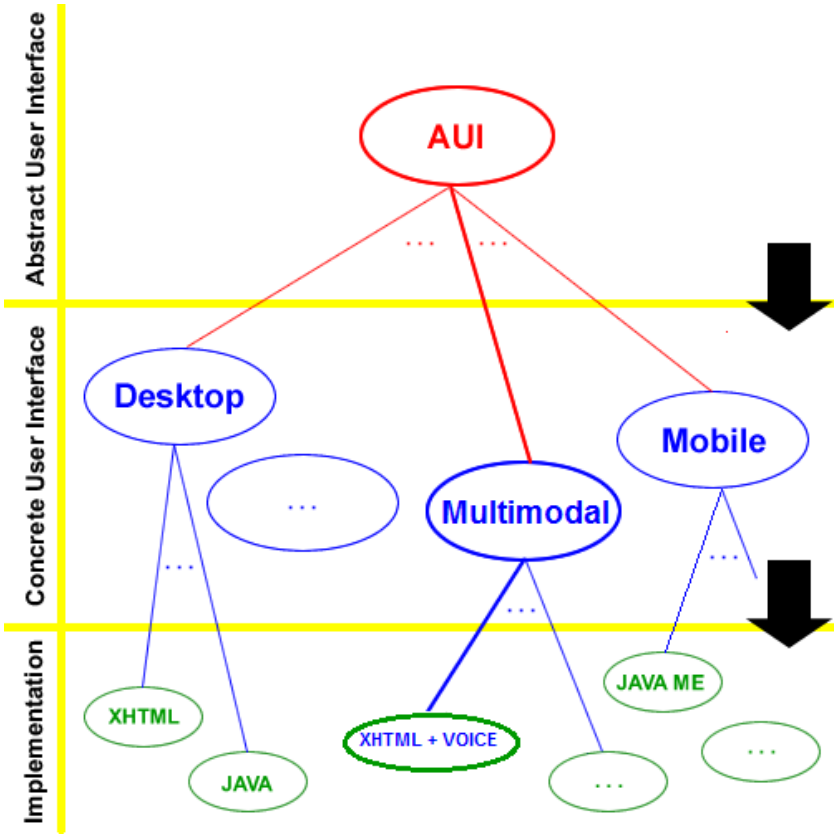

Figura 3.1: Approccio modulare in MARIA

dello dei dati utilizzato è descritto in modo astratto con XSD versione 1.0 di cui troviamo una esaustiva descrizione in [13], [30] e [5] [5].

Il linguaggio TERESA è definito tramite DTD[6]. La scelta di abbandonare questa strada è stata motivata da molteplici ragioni. A fronte di una maggiore complessità gli XSD consentono la definizione di tipi di dato. Inoltre i DTD utilizzano una sintassi differente da quella XML, mentre l'XSD è XML. [7]

[5] *L' "XML Schema Working Group" sta lavorando al completamento di XML Schema 1.1. L'obiettivo è quello di correggere i bug della versione precedente e di apportare quante più migliorie possibili senza perdere però la retro-compatibilità [29].*

[6] Document Type Definition

[7] *L'XML è un sottoinsieme dello SGML[8]. Il suo scopo è di permettere che generici SGML siano, serviti, ricevuti e processati sul Web nel modo che è attualmente possibile per l'HTML[9]. XML è stato progettato per facilità di implementazione e per interoperabilità con SGML e HTML. [7]*

L'introduzione del modello dei dati permette un maggiore controllo sulle operazioni e la loro ammissibilità e permette inoltre:

- la correlazione tra i vari valori degli elementi di un interfaccia;
- la rappresentazione condizionale delle varie parti di un interfaccia;
- la specifica del formato dei valori di input;
- la generazione dell'applicazione a partire dalla descrizione dell'interfaccia.

Un altro punto focale in MARIA è il modello ad eventi che permette di specificare, a differenti livelli di astrazione, come l'interfaccia utente risponde agli eventi lanciati dall'utente. Sono previsti 2 tipi di eventi:

- *Property Change Events*: vanno a modificare lo stato delle proprietà dell'interfaccia
- *Activation Events*: generati quando si cerca di attivare una funzionalità (es. Web Services, accesso ad un database). In questo caso l'evento può sia cambiare una proprietà che attivare uno script.

Una caratteristica molto interessante di MARIA è il supporto per tecnologie come AJAX. Ogni elemento dell'interfaccia astratta è fornito di un attributo *continuous update*: se attivo allora gli attributi dell'elemento verranno aggiornati di continuo. Sarà naturalmente responsabilità del livello implementativo realizzare l'effettivo update.

Infine citiamo la possibilità di cambiare dinamicamente i contenuti ed il comportamento di una UI grazie all'uso di connessioni condizionali tra presentazioni.

Vedremo nella prossima sezione una panoramica più dettagliata dell'interfaccia astratta.

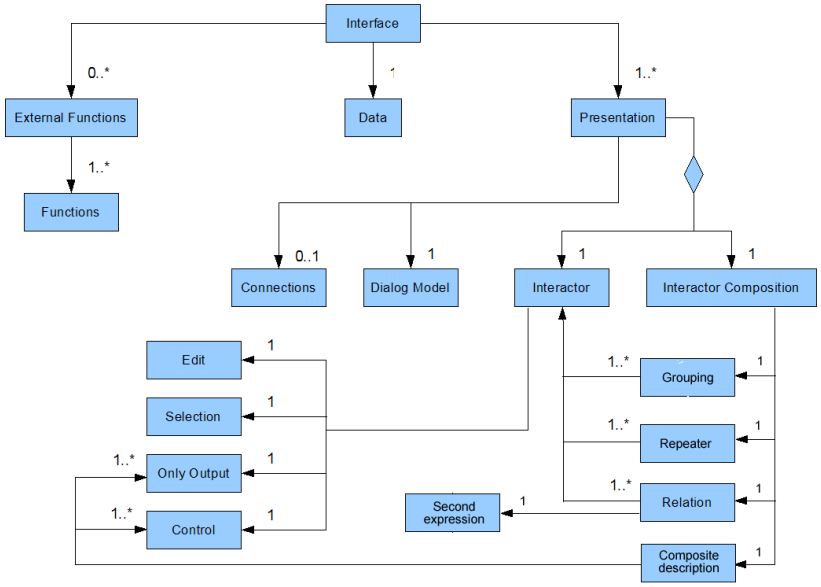

Figura 3.2: Abstract User Interface (Simplified)

3.1.2 Abstract User Interface (AUI)

L'Abstract User Interface è il livello più alto di astrazione e nasce dalla necessità di dare ai progettisti un metodo (indipendente dalla piattaforma) per concentrarsi sulle scelte principali senza dover pensare ai dettagli e alle specifiche del particolare ambiente considerato. Uno schema ad alto livello dell'AUI è rappresentato in figura 3.2.

Dalla figura si nota che un'interfaccia è composta da un modello *dati*, da una o più *funzioni esterne* e da una o più *Presentation*. Una *Presentation* dal punto di vista delle interfacce grafiche è riconducibile al concetto di vista, ad esempio nel modo del web potrebbe essere una pagina html; è caratterizzata da un nome, da zero o più *connection* (collegamenti con altre presentation), da un insieme di *interactor* e di *interactor composition*. Le presentation sono associate a un *dialog model* che fornisce le informazioni sugli eventi che possono innescati un certo momento. Il comportamento dinamico degli eventi, e

degli handler associati, è specificato attraverso gli operatori temporali CTT[10] (es. concorrenza, scelte mutue esclusive o sequenzialità). In [25] è possibile trovare una descrizione dei CTT. Gli *interactor composition* hanno la funzione di mettere insieme gli *interactor* tra loro e si dividono in *grouping*, *relation*, *repeater* e *composite description*. Il *grouping* permette di raggruppare tra loro 2 o più *interactor*, attraverso l'attributo *ordering* è possibile specificare un certo ordinamento tra gli interattori, mentre con l'attributo *hierarchy* è possibile stabilire una gerarchia tra gli interattori. L'*interactor composition* `relation` viene usato quando un certo numero di elementi dell'interfaccia astratta sono in relazione con un altro elemento. L'elemento *composite description* raggruppa al suo interno solo elementi di output o di navigazione; la si può immaginare come una pagina contenente del testo, immagini e link.

Gli elementi dell'interfaccia si distinguono in *interactor* che supportano l'interazione con l'utente (*interaction group*) e quelli che hanno il fine di fornire un output all'utente (*only output group*). L'*interaction group* è composto da elementi di tipo *selection* che permettono di effettuare una scelta tra un insieme di elementi; *edit* utilizzati per la modifica di un campo; *control* il cui fine è attivare una funzionalità (*activator*) o permettere la navigazione tra le presentation (*navigator*). Gli interattori *only output* si dividono in *description*, *object*, *feedback* e *alarm* a seconda del tipo di output che forniscono all'utente.

Il raffinamento dell'interfaccia definita secondo questo modello astratto può essere fatto solo fissando un dispositivo di destinazione e conoscendo dunque ulteriori caratteristiche dipendenti dalla piattaforma target. Andiamo ora a vedere come può essere realizzata una interfaccia utente concreta.

3.1.3 Concrete User Interface (CUI)

Per realizzare la CUI[11] occorre porsi ad un livello intermedio nel quale si è a conoscenza del tipo di dispositivo cui è dedicata l'interfaccia ma non

[10] ConcurTaskTrees

[11] Concrete User Interface

si fanno ipotesi sul linguaggio finale di implementazione. La CUI, in altre parole, è dipendente dalla piattaforma ma indipendente dal linguaggio di implementazione.
Il procedimento di realizzazione consiste in un raffinamento della AUI con l'integrazione di nuovi tipi propri del dispositivo.

In figura 3.3 vediamo schematizzato il tipo `single_choice` per l'interfaccia astratta.

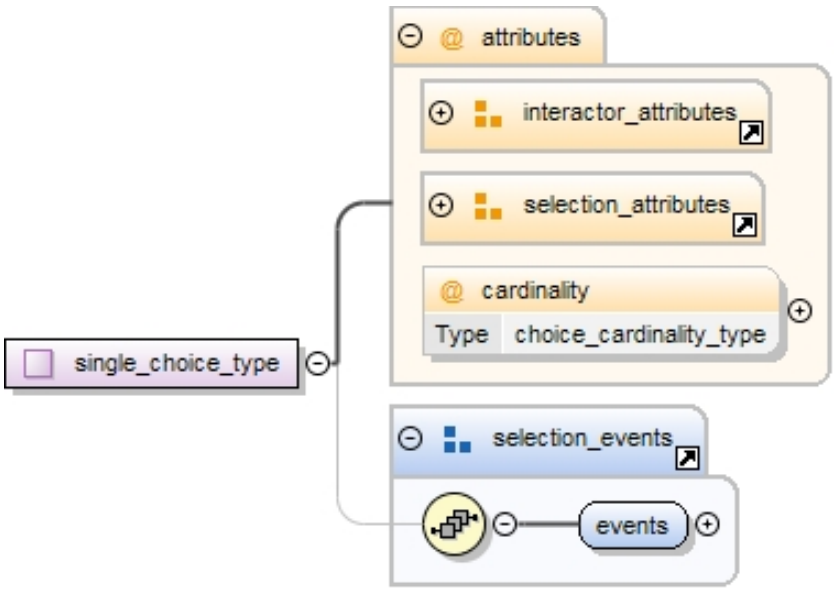

Figura 3.3: AUI single_choice type

Un passo di concretizzazione verso un interfaccia desktop porta ad un'estensione di questa struttura. Vediamo infatti in figura 3.4 una possibile espansione con i seguenti nuovi elementi:

- *radio_button*: permette di scegliere un elemento da un insieme predefinito.
- *list_box*: gli elementi vengono presentati in una casella di testo multilinea;
- *drop_down_list*: simile alla list box ma quando non è selezionata mostra solo un elemento;

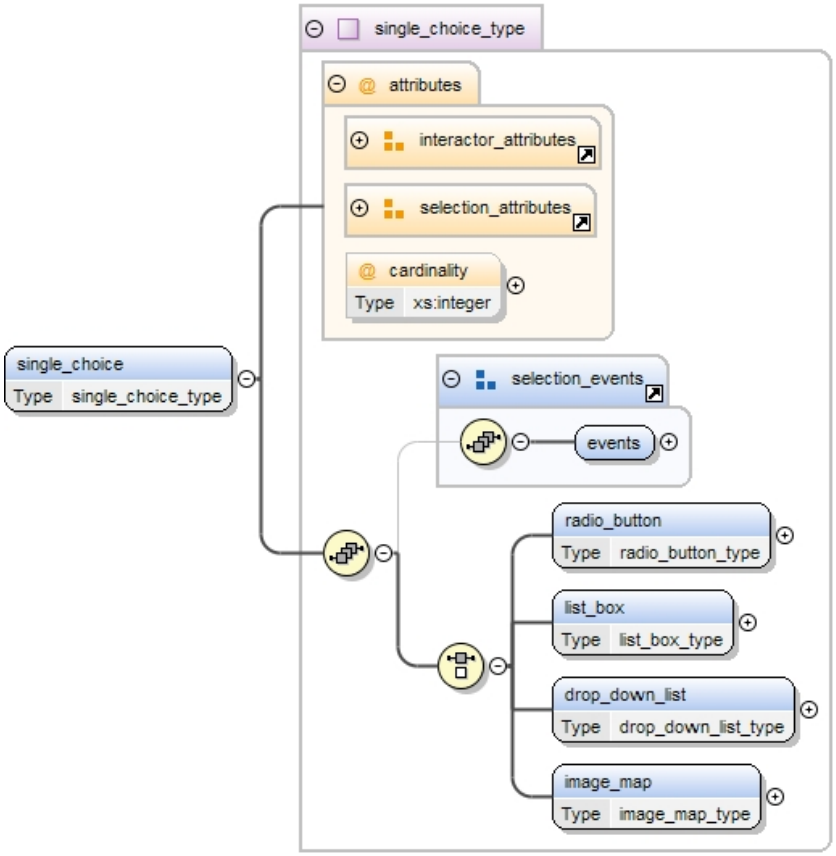

Figura 3.4: Desktop CUI single_choice type

- *image_map*: immagine con aree selezionabili.

E' interessante notare come l'interfaccia concreta venga specializzata a seconda della piattaforma. In MARIA esistono due varianti della piattaforma mobile, quella di tipo *mobile large* descrive le caratteristiche dei dispositivi mobili con schermi più ampi, come smart phones e PDA[12] e la variante *mobile small* per i dispositivi con display ridotti come cellulari. Riprendendo l'esempio precedente relativo all'interattore *single choice* si può vedere nella figura 3.5 come la sua specializzazione per la piattaforma *mobile small* non predenti l'elemento *list box*, questo perché i list box sono degli interattori

[12]Personal Digital Assistant

troppo costosi dal punto di vista dello spazio occupato nei dispositivi con capacità limitata.

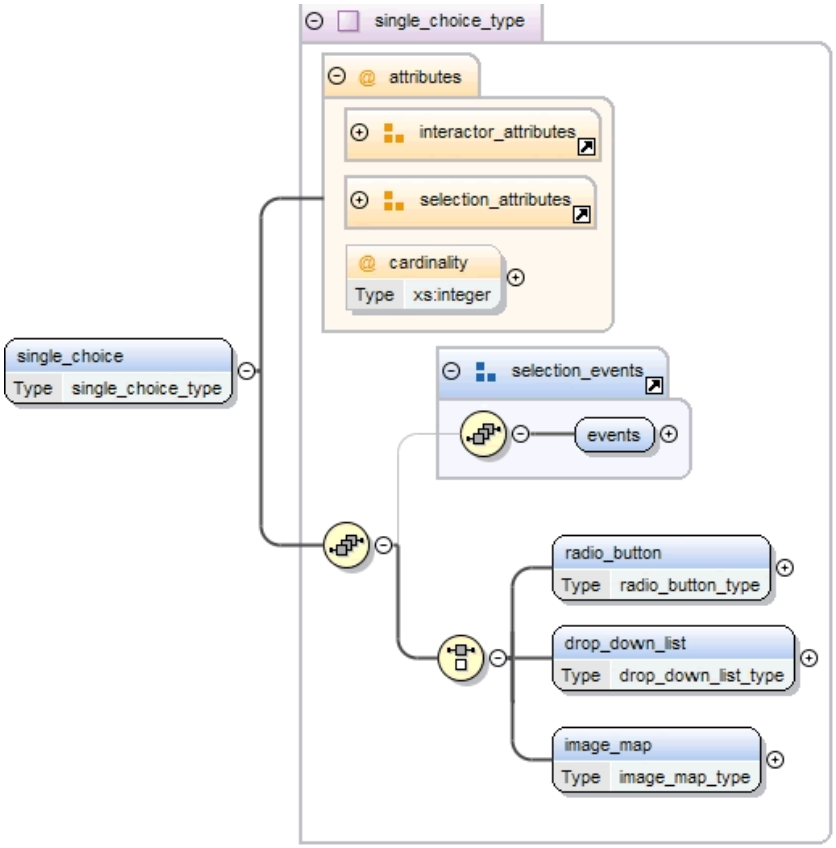

Figura 3.5: Mobile small CUI single_choice type

3.2 L'Authoring Environment

Il tool fornisce un ambiente per la progettazione e lo sviluppo di applicazioni basate su Web Services e consente l'associazione dei file WSDL[13] con l'interfaccia utente astratta. E' inoltre possibile effettuare una serie di trasformazioni tramite un processo semi automatico che porta alla generazione effettiva dell'interfaccia.

[13]Web Services Description Language

Le trasformazioni possono essere bidirezionali ma il passaggio da un astrazione di più basso livello ad una di più alto potrebbe causare perdita di informazione. Le trasformazioni tra i vari livelli sono definite esternamente all'applicazione, in questo modo le loro modifiche non si ripercuotono sull'implementazione del tool. Questo ambiente di sviluppo permette approcci variegati, non solo l'approccio *top-down* tipico delle applicazioni *Model-Based* a livello di progettazione, ma anche *bottom-up* e *mixed*.

L'approccio *top-down* consiste essenzialmente in una scomposizione di un sistema globale raffinandolo in sotto-sistemi. Di fatti, poiché l'approccio top-down mira a ridefinire l'intero sistema, è particolarmente efficace quando il design inizia da zero, in modo che lo sviluppatore abbia un immagine globale del sistema che verrà progettato e ridefinito in maniera graduale. Invece, se il progettista vuole includere pezzi di software già esistenti, come servizi in SOA[14], questo richiede necessariamente che si adotti un approccio *bottom-up* in modo da sfruttare le funzionalità preesistenti e identificare le relazioni tra di loro per comporle ad un livello più alto. Tuttavia, l'opzione migliore sembra che sia una soluzione *ibrida* nella quale è usato un mix tra l'approccio bottom-up e top-down. Inizialmente è previsto un approccio bottom-up in modo da analizzare i web services che forniscono le funzionalità utili per le nuove applicazioni da sviluppare. Questo implica un'analisi delle operazioni e dei tipi di dati collegati ai parametri di input e di output, in modo da associarli agli opportuni interattori astratti. Il passaggio successivo serve a definire le relazioni tra gli elementi, per fare questo vengono utilizzati i CTT per descrivere l'applicazione interattiva e come si assume verranno eseguiti i task. Una volta ottenuto il modello dei task è possibile generare i vari descrittori di interfaccia con un approccio top-down e raffinandoli con MARIA Authoring Environment sino all'implementazione finale.

[14]Service Oriented Architeture

3.3 Le trasformazioni

Il motore di trasformazione è implementato usando un generatore XSLT che prende in input il foglio di stile contenente le regole di trasformazione e la CUI, a partire da queste genera il documento nel linguaggio di destinazione. Ne vediamo una rappresentazione ad alto livello in figura 3.6.

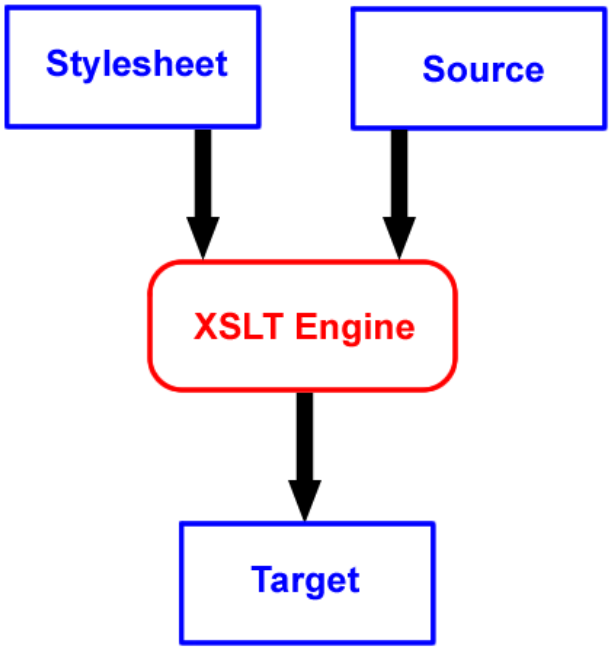

Figura 3.6: XSLT Transformation Engine

3.4 Metodologie di progettazione

Dal punto di vista della progettazione di applicazioni basate su Web Services, MARIAE sta trovando spazio in tre segmenti:

- Supporto all'uso di *web annotation* nei file WSDL che descrivono i web services

- Supporto per la composizione di più Web Services in una singola applicazione
- Supporto *model-based* per la creazione di interfacce

3.4.1 UI Annotation per Web Services

I Web Services sono utilizzati per l'accesso a funzionalità di applicazioni remote; essi sono descritti attraverso il linguaggio WSDL che altro non è che un linguaggio in formato XML che specifica come interagire con un determinato servizio. Un documento WSDL contiene:

- *cosa* può essere utilizzato (le operazioni messe a disposizione dal servizio);
- *come* utilizzare un web service (protocollo di comunicazione, formato dei messaggi in input e in output);
- *dove* corrisponde all'endpoint del servizio (URI).

Un documento WSDL dunque non contiene informazioni riguardo all'interfaccia per accedere al servizio, per questo motivo in MARIAE si considerano le annotazioni che forniscono suggerimenti per la costruzione dell'interfaccia utente per accedere ai web services.
Nella figura 3.7 è mostrato un semplice esempio della definizione di un servizio per la gestione di account clienti che è composto da 2 operazioni: una per creare un nuovo cliente e l'altra per recuperare le informazioni di un cliente. Le annotazioni forniscono alcuni indizi per come il servizio debba essere reso dal punto di vista grafico, alcune informazioni riguardano il tipo di dati manipolati dal servizio (espressioni regolari per la validazione, label di default per i campi), mentre altre informazioni riguardano l'interfaccia utente per accedere alle operazioni (group, input validation, relazioni tra i parametri delle operazioni e i tipi di dato).

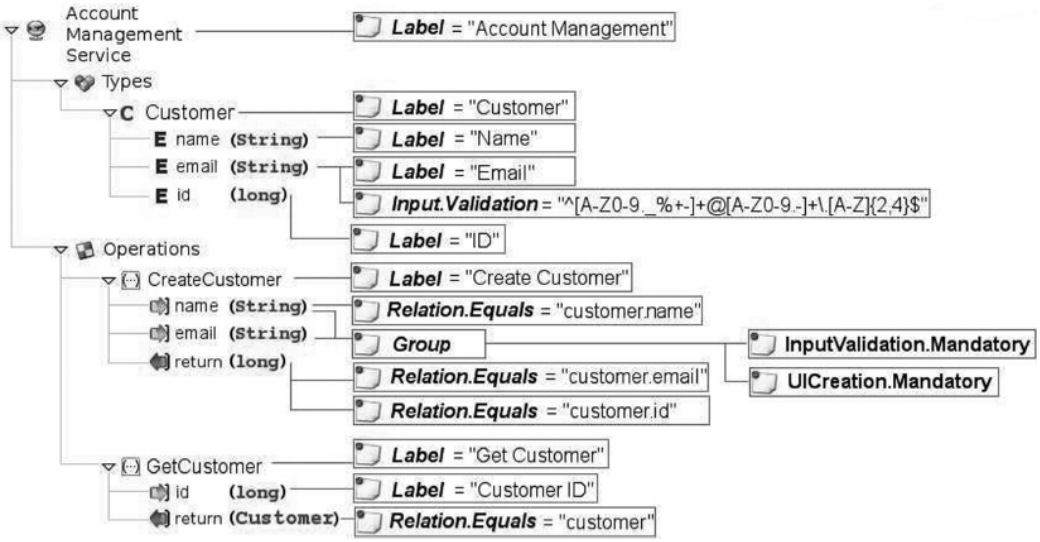

Figura 3.7: Esempio di annotazioni per un webservice

3.5 Graphical CUI

In questa sezione vengono introdotti i concetti fondamentali per la descrizione delle interfacce grafiche concrete facendo riferimento alla piattaforma desktop.

3.5.1 Presentation

Il concetto di *presentation* dal punto di vista di un'interfaccia grafica è riconducibile al concetto di vista; nel mondo web equivale a una pagina HTML. La presentation definita nella AUI è specializzata con l'elemento *presentation settings* nel quale vengono settate alcune proprietà dell'interfaccia come il titolo, il colore o l'immagine di sfondo e le impostazioni relative al font.

3.5.2 Events

Gli eventi grafici sono generati da mouse e tastiera; gli eventi generati dall'interazione col sono: mouse enter, mouse move, mouse leave, mouse click, mouse over e mouse double click. Per quanto riguarda la tastiera gli eventi

generati sono: key down, key up e key press. A questo tipo di eventi è possibile aggiungere uno o più handler che permettono la gestione eseguendo uno script oppure cambiando la proprietà di un interattore.

3.5.3 Connections

Una connessione indica quale sarà la prossima presentazione attiva quando una data interazione avrà luogo; gli interattori che causano un passaggio da una connessione ad un'altra sono i *navigator* e gli *activator* che saranno descritti in seguito nelle sezioni 3.5.8. Esistono diversi tipi di connessioni:

- *elementary connection*: si tratta di una connessione semplice con una presentation target statica;
- *complex connection*: si utilizza quando operatori booleani compongono diverse connessioni;
- *conditional connection*: utilizzate quando vengono associate a una connessione specifiche condizioni (es.quando si passa a una determinata presentation a seconda del valore di un interatore).

In generale le connessioni sono caratterizzate da 2 attributi fondamentali:

- *interactor id*: l'identificativo dell'interatore che causa il passaggio da una presentation all'altra (può essere un navigator o activator);
- *presentation name*: nome della presentation target;

3.5.4 Grouping

Come spiegato in 3.1.2 uno degli elementi che permettono la composizione degli interattori è il raggruppamento (*grouping*); dal punto di vista grafico viene reso con una tabella, un div, un elenco di elementi ordinati secondo un certo ordine o gerarchia. Trattandosi di un raffinamento dell'interfaccia astratta per una piattaforma desktop l'elemento grouping possiede gli attributi tipici delle pagine html che riguardano la posizione (position, margin,

padding, float), le dimensioni (height e width) e i bordi (border color, border style, border width); inoltre è possibile specificare le proprietà riguardanti lo sfondo (background color e background image) e il font del raggruppamento. Il grouping è composto dall'interactor express group che permette una scelta tra una lista di interattori e una lista di operatori di composizione; inoltre possiede un elemento grid che permette di visualizzare gli interattori usando un layout a griglia composta da righe e colonne. Il grouping contiene un elemento properties nel quale è possibile settare il colore o l'immagine di sfondo e altre proprietà che saranno valide solo all'interno del raggruppamento.

3.5.5 Interactor: Only Output

Gli interattori di tipo *only output* si occupano esclusivamente di fornire un output all'utente; a seconda del tipo di output si può avere un elemento di tipo description o alarm. L'elemento *description* graficamente è reso attraverso un testo, un'immagine, una tabella o un video; ognuno di questi al suo interno possiede gli attributi per il posizionamento, le dimensioni e i bordi. Un elemento *alarm* può essere reso con un box contenente il messaggio di allarme o un file audio.

3.5.6 Interactor: Selection

L'interattore di selezione permette all'utente di operare una scelta tra un dato insieme di elementi; esistono due tipi di elementi selection:

- **scelta singola** *single choice*: l'utente può selezionare un solo elemento della lista
- **scelta multipla** *multiple choice*: l'utente può scegliere più elementi contemporaneamente.

3.5.6.1 Single choice

Quest'elemento è composto da una serie di *choice element* che vanno a specificare la lista delle possibili scelte che l'utente può operare; inoltre è possibile specificare un elemento della lista che venga selezionato di default. La AUI viene raffinata concretizzando i vari modi in cui l'utente può operare la scelta; parallelamente all'html la CUI prevede una specializzazione tramite **radio button**, **list box**, **drop down list** e **image map**. Come tutti gli interattori, anche i single choice presentano gli attributi di posizionamento, di dimensione e gli attributi riguardanti i bordi; attraverso l'elemento *Label* è possibile specificare un'etichetta.

3.5.6.2 Multiple choice

Come dei single choice anche per i multiple choice è possibile specificare la lista delle scelte, con la differenza che in questo caso è possibile elencare la lista degli elementi selezionati di default. Graficamente i multiple choice vengono implementati tramite **check box** o **list box**; entrambi presentano gli attributi per il posizionamento, bordi e dimensione, oltre che la definizione dell'etichetta (Label) e degli eventi grafici (mouse e keyboard events).

3.5.7 Interactor: Edit

Gli interattori di tipo **edit** permettono di ricevere l'input dell'utente; a differenza degli interattori *selection* l'utente ha una maggiore libertà sull'input da inserire in quanto non viene fornito a priori un elenco di scelte possibili. Tramite gli interattori edit è possibile raccogliere 2 tipi di input:

1. testuale attraverso l'elemento **text edit**;
2. numerico attraverso l'elemento **numerical edit**.

3.5.7.1 Text edit

Gli elementi di tipo *text edit* vengono utilizzati per permettere l'inserzione di testo; dal punto di vista grafico sono implementati attraverso elementi di tipo **text field** e **text area** che hanno una corrispondenza diretta con i tag html *input type='text'* e *textarea*. Sia text field che text area presentano gli attributi e gli elementi tipici degli interattori quali attributi di posizionamento, bordi, dimensioni e gli elementi per la definizione dell'etichetta (Label) e degli eventi grafici (mouse e keyboard events); inoltre è possibile inserire il valore corrente dell'interattore, la lunghezza massima dell'input e segnalare il fatto che sia o no un campo contente una password in modo che il contenuto venga visualizzato con degli asterischi invece che col testo in chiaro.

3.5.7.2 Numerical edit

Questo elemento presenta forti analogie con l'elemento *text edit* (sez. 3.5.7.1), la differenza sta nel fatto che in questo caso ci si aspetta che l'input sia di tipo numerico. Esistono due specializzazioni degli elementi numerical edit:

- **numerical edit full**: permette l'inserzione di qualsiasi numero;
- **numerical edit in range**: permette l'inserzione di un numero all'interno di un certo range specificato dagli attributi *min* e *max*.

L'elemento numerical edit viene specializzato attraverso un **text field**, uno **spin box** o **track bar** (solo per il numerical edit in range), come gli altri interattori questi elementi possiedono gli attributi per il posizionamento, per la definizione dei bordi e della dimensione; sono presenti anche gli elementi per la definizione dell'etichetta (Label) e degli eventi grafici (mouse e keyboard events).

3.5.8 Interactor: Control

Gli interattori di controllo hanno due compiti principali:

1. attivare funzionalità (*activator*);

2. permettere la navigazione verso altre presentation (*navigator*).

L'activator e il navigator vengono specializzati attraverso un elemento *button*, *text link*, *image link*, *image map* e *mail to* (quest'ultimo solo per l'activator); come gli altri interattori questi elementi possiedono gli attributi per il posizionamento, per la definizione dei bordi e della dimensione; sono presenti anche gli elementi per la definizione dell'etichetta (Label) e degli eventi grafici (mouse e keyboard events).

3.6 Vocal CUI

In questa sezione vengono introdotti i concetti fondamentali per la descrizione delle interfacce vocali concrete.

3.6.1 Presentation

Una presentation è specializzata per le interfacce vocali specificando le proprietà di *sintetizzazione* e le proprietà del *riconoscitore vocale*.

Le proprietà di sintetizzazione riguardano tutte quelle proprietà che caratterizzano una voce sintetizzata come volume, età del parlante, genere, enfasi, pitch e rate. Tra le proprietà di sintetizzazione troviamo l'elemento *bargein* col quale è possibile specificare il comportamento dell'interprete in caso di tentativo di interruzione della riproduzione da parte dell'utente; l'attributo *active* se settato a true sta a significare che l'utente può interrompere la riproduzione in qualsiasi momento. Le proprietà del riconoscitore vocale riguardano tutti quei valori che contraddistinguono una piattaforma di riconoscimento: livello di confidenza, sensibilità ai rumori esterni, durata massima dell'input vocale, etc. L'attributo timeout ha una particolare rilevanza in quanto indica il periodo di tempo che la piattaforma attenderà un input dell'utente prima di sollevare l'evento *no input*.

3.6.2 Events

Gli *eventi vocali* sono eventi legati all'input vocale dell'utente. Nel caso in cui la piattaforma non rilevi alcun input dell'utente entro un tempo pari al valore dell'attributo timeout settato in precedenza viene sollevato l'evento **no input**. L'evento *no match* viene sollevato nel caso in cui l'input dell'utente non corrisponda agli input ammissibili. Nel caso l'utente pronunci la parola di aiuto predefinita per la piattaforma viene sollevato un evento di tipo *help*. Ogni evento vocale permette di impostare 2 attributi: *message* nel quale si specifica il messaggio che deve essere sintetizzato in seguito alla cattura dell'evento; *reprompt* che se settato a true richiede alla piattaforma di sintetizzare nuovamente l'ultimo messaggio.

3.6.3 Grouping

Dal punto di vista vocale è necessario specificare un metodo per indicare all'utente l'inizio o la fine di un raggruppamento; soluzioni proposte sono 4 e il progettista dell'interfaccia è libero di utilizzare la soluzione (o la combinazione di soluzioni) che reputa migliore per le sue esigenze:

1. **inserimento di suoni** (insert sound): la riproduzione di un suono all'inizio e alla fine di un raggruppamento; perché l'utente non venga distratto o confuso il suono deve essere breve, chiaro e preferibilmente dovrebbe essere riprodotto lo stesso suono all'inizio e alla fine;

2. **inserimento di pause** (insert break): le pause possono essere una soluzione meno invasiva rispetto alla riproduzione di un suono, a patto che non siano né troppo brevi perché in questo caso risulterebbero inutili né troppo lunghe perché altrimenti l'eccessiva attesa potrebbe irritare l'utente;

3. **inserimento di parole chiave** (insert keyword): un raggruppamento potrebbe essere identificato dalla sintetizzazione di parole chiave (es. GROUPING START - GROUPING END);

4. **cambio delle proprietà di sintesi** (change synthesis properties): la soluzione meno invasiva prevede il cambio delle proprietà della voce sintetizzata in modo che il contenuto del raggruppamento sia sintetizzato con una voce dalle caratteristiche differenti da quelle precedentemente sintetizzate.

3.6.4 Interactor: Only Output

Gli elementi *only output* dal punto di vista vocale vengono trattati nello stesso modo in quando l'output vocale può essere essenzialmente di 3 tipi:

1. *text to speech*: testo sintetizzato dalla piattaforma;
2. *messaggio vocale registrato*: file audio contenente un messaggio preregistrato;
3. *suono*: file audio contenente un suono.

Ai 3 tipi di output vocale corrispondono 3 diversi elementi della CUI vocale:

- **Vocal**: per il testo da sintetizzare;
- **Prerecorded message**: per i messaggi preregistrati;
- **Sound**: per l'output sonoro.

Vocal

Un elemento di tipo *vocal* è utilizzato per sintetizzare un testo; è implementato attraverso due elementi in mutua esclusione: *content* nel quale si specifica il testo che deve essere sintetizzato e *text path*: contenente il path di un file che col testo da sintetizzare. L'attributo **counter** dell'elemento vocal è un valore intero che serve ad implementare la tecnica del *tapered prompting*.

Il tapered prompting è una tecnica di programmazione vocale il cui scopo è incrementare l'esperienza dell'utente; il principio su cui si fonda è che un

utente man mano che utilizza l'applicazione diventa più familiare con essa e dunque i messaggi verso l'utente dovrebbero essere sempre più concisi. La situazione opposta si ha quando l'utente richiede assistenza o commette errori nel fornire un input, in questo caso i messaggi devono essere più precisi. Il funzionamento del tapered prompting è implementato in questo modo: quando la piattaforma vocale incontra degli elementi di tipo vocal riproduce inizialmente quello il cui valore dell'attribuito count è uguale a uno, una volta che il prompt è riprodotto il contatore è incrementato in modo che in caso debba essere riprodotto nuovamente il prompt venga selezionato quello col valore dell'attributo count maggiore o uguale al valore del contatore.

L'elemento **properties** permette di settare tutte le proprietà tipiche della sintetizzazione vocale quali volume, tono, enfasi, sesso della voce, etc.

L'elemento di tipo vocal è un componente chiave delle interfacce vocali, in questo paragrafo è stato mostrato come specializzi gli elementi only output, ma viene utilizzato per fornire un prompt all'utente nel caso degli elementi *selection*, *edit* e *control*.

3.6.4.1 Prerecorded message e sound

L'elemento di tipo *prerecorded message* viene utilizzato per sintetizzare file audio contenenti un messaggio vocale mentre l'elemento *sound* serve per la riproduzione di suoni. La distinzione è solamente a livello logico, è solo il contenuto del file riprodotto a essere differente; a livello concreto sono equivalenti. Prerecorded message e sound si compongono di 3 elementi: *Path* che indica è il percorso del file audio; *Alternative content* che rappresenta un testo da sintetizzare nel caso in cui il file specificato nel path non sia disponibile e *Properties*: permette di settare le proprietà di riproduzione del file audio (enfasi, pitch, rate, volume).

3.6.5 Interactor: Selection

Come indicato nella AUI l'interattore selection si divide in *single choice* e *multiple choice*; dal punto di vista vocale questi due elementi vengono trattati allo stesso modo attraverso un elemento di tipo *vocal selection* che permette alla piattaforma vocale di collezionare l'input dell'utente fornendo un numero determinato di alternative; la procedura parte dalla piattaforma vocale che richiederà all'utente l'inserzione di un input, questo avviene attraverso un elemento **question** che è un elemento di tipo vocal descritto nella sezione 3.6.4. E' possibile inserire quanti elementi question si vogliono, infatti non è stato definito un limite al numero di domande da porre all'utente, in modo da implementare la tecnica del *tapered prompting* descritta in precedenza (sez.3.6.4); lo sviluppatore ha il compito di gestire il tapered prompting impostando il valore dell'attributo *count* dei vari elementi question.

3.6.6 Interactor: Edit

Gli interattori Edit possono essere di tipo text edit (input testuale) e numerical edit (input numerico).

3.6.6.1 Text edit

L'elemento **vocal textual input** contiene tutti gli elementi necessari per fornire un prompt all'utente attraverso una sequenza di elementi **request** di tipo *vocal* e l'elemento **events** per la gestione degli eventi vocali. La differenza con gli elementi di tipo selection sta nel fatto che l'utente può inserire il testo che vuole senza doverlo per forza sceglierlo tra un insieme di scelte; per permettere al riconoscitore vocale di trasformare l'input vocale in testo è presente un elemento **grammar** che composto da:

- **grammatica esterna** (external grammar): nel quale è possibile specificare il percorso (*src*) di un file contenente una grammatica e il tipo (*type*) della grammatica.

- **grammatica interna** (internal grammar): nel quale è possibile specificare i possibili valori del testo che la grammatica riconosce.

Nel caso di *vocal selection* (sez. 3.6.4) non è previsto l'uso di grammatiche perché le alternative in input siano relativamente poche ed è possibile generare in maniera automatica la grammatica a partire dalle scelte disponibili.

3.6.6.2 Numerical edit

Esistono due specializzazioni degli elementi numerical edit: **numerical edit full** che permette l'inserzione di qualsiasi numero e **numerical edit in range**che consente l'inserzione di un numero all'interno di un certo range specificato dagli attributi *min* e *max*.
La definizione vocale dell'elemento numerical edit è specificata attraverso l'elemento **vocal numerical input** la cui struttura è analoga all'elemento *vocal textual input*; i 2 casi si differenziano per il fatto che nell'elemento *vocal numerical input* oltre a permettere l'inserimento di una grammatica esterna e interna, si permette di specificare tra gli attributi il tipo di numero che ci si aspetta che l'utente inserisca. I tipi di numeri definibili nell'attributo **built in grammar** sono:

- **date**: l'utente può inserire una data specificando il mese, il giorno e l'anno;
- **digits**: l'utente può inserire valori interi come singole cifre (da 0 a 9), il valore inviato sarà una stringa di una o più cifre;
- **currency**: l'utente può inserire numeri che riguardano la valuta americana in dollari e centesimi;
- **number**: l'utente può inserire numeri positivi da 0 a 999999.
- **phone**: l'utente può inserire un numero telefonico;

- **time**: l'utente può inserire l'orario usando ore e minuti sia nel formato 12 che 24 ore. Il valore inviato sarà nel formato *hhmmx*, dove x rappresenta AM o PM.

3.6.7 Interactor: Control

Gli interattori di controllo si dividono in activator e navigator.
L'activator è implementato con 2 elementi differenti:

- **command**: per l'esecuzioni di script;
- **submit**: per sottomettere al server i dati raccolti.

L'elemento *command* esprime il comando inteso come chiamata di funzione. L'elemento *submit* specializza l'activator definito nella AUI per sottomettere un form. L'unico attributo presente è **label** col quale si specifica la parola che l'utente deve dire per sottomettere il form.
Come gli altri interattori vocali anche questi due elementi presentano un elemento **request** di tipo vocal per presentare il prompt all'utente.
Il navigator viene specializzato con un elemento di tipo link che contiene un elemento request di tipo vocal col quale si fornisce l'input all'utente; tra gli attributi è possibile specificare la sequenza DTMF o la stringa da pronunciare per attivare il link.

Capitolo 4

Multimodal CUI grafica e voce

Il linguaggio proposto è un raffinamento della AUI, una definizione più ad alto livello per la descrizione delle interfacce multimodali. Dal momento che le interfacce da me considerate sono di tipo multimodale composte da grafica e voce, la AUI sarà estesa in modo da aggiungere gli elementi concreti riguardanti la modalità grafica, vocale e la loro composizione (attraverso gli attributi che specificano le proprietà CARE).

4.1 CARE properties

Le proprietà CARE introdotte nella sezione 2.3 sono molto importanti per descrivere la maniera in cui la parte grafica dell'interfaccia convive con quella vocale. In modo da fornire un ambiente flessibile ho dato la possibilità agli sviluppatori di applicare queste proprietà nei vari aspetti che caratterizzano la descrizione logica: operatori di raggruppamento, interattori ed elementi only output. Inoltre, per dare la possibilità di controllare la multimodalità a un livello di granularità più fine gli elementi di interazione sono strutturati in diversi stadi:

1. *Prompt*: rappresenta l'output che indica quando l'interfaccia è pronta a ricevere l'input dell'utente;

2. *Input*: rappresenta in che modo l'utente può fornire l'input;

3. *Feedback*: rappresenta il modo in cui il sistema risponde dopo l'input dell'utente.

Negli elementi composition e only output è presente uno stadio denominato *output* che rappresenta come il sistema fornisce l'output all'utente. Facendo riferimento al caso di interfacce multimodali composte da grafica e voce le proprietà CARE sono state applicate nel modo seguente:

- Complementarity: La parte considerata dell'interfaccia è considerata come parzialmente supportata da una modalità e parzialmente dall'altra;
- Assignment: La parte considerata dell'interfaccia è supportata da una modalità assegnata (graphic assignment o vocal assignment);
- Redundancy: La parte considerata dell'interfaccia è supportata da entrambe le modalità;
- Equivalence: La parte considerata dell'interfaccia è supportata da una modalità o dall'altra.

Nella pratica non tutte le proprietà CARE possono essere applicate in tutti gli stadi dell'interazione. In particolare, l'equivalenza può essere applicata solo nel caso dell'input quando due modalità sono disponibili e una o l'altra possono essere usate in maniera indifferente per inserire l'input. Al contrario, la ridondanza può essere applicata al prompt e al feedback ma non all'input, poiché questo significherebbe che lo stesso input sia fornito attraverso diverse modalità contemporaneamente e questo non è ne utile ne efficiente. La complementarietà può essere applicata in tutti i livelli, tuttavia per quanto riguarda l'input assume un significato solo nel caso di input strutturati, ad esempio nel caso del sistema Put that there [6] l'input viene dato in maniera complementare da voce e gesti; considerando input di tipo atomico che richiedono semplici azioni (es. selezione di un bottone) difficilmente queste possono essere eseguite attraverso un input complementare. Logicamente l'assegnamento può essere applicato a tutti gli stadi senza problemi.

In questo modo si permette allo sviluppatore di decidere che supporto multimodale fornire per ogni stadio.
Nella tabella 4.1 è mostrato come le CARE properties sono proposte dall'authoring environment in modo da ottenere la generazione di interfacce multimodali grafiche e vocali per entrambe le piattaforme desktop e mobile, ma l'approccio presentato può essere valido anche per altri tipi di modalità.

I valori possibili sono stati scelti tenendo conto delle caratteristiche della piattaforma target; in particolare la piattaforma desktop presenta un'ampia disponibilità di risorse grafiche rispetto alla piattaforma mobile.
Nel caso degli *operatori di composizione* l'attributo **output** rappresenta la modalità in cui è possibile indicare l'inizio e la fine di un raggruppamento; nelle interfacce desktop e mobile i valori che può assumere sono *Assegnamento grafico* o *Ridondanza*, inoltre nella piattaforma mobile è previsto anche *l'assegnamento solo vocale*.
Nella piattaforma desktop l'attributo **output** degli elementi *only output* prende i valori: *Assegnamento grafico*, *Ridondanza* o *Complementarietà*; nel primo caso possiamo immaginare un elemento di solo testo, nel secondo un testo che viene mostrato nella sua forma scritta sia sintetizzato dalla piattaforma vocale. La complementarietà viene considerata perché alcuni elementi non possono essere resi nella stessa maniera dal punto di vista grafico e vocale, ad esempio un'immagine è difficile renderla in maniera equivalente graficamente e vocalmente, al massimo è possibile inserire una descrizione vocale che la descriva in parte e in questo casi si avrebbe complementarietà.
I valori dell'attributo output disponibili per la piattaforma mobile sono differenti a causa del fatto le risorse grafiche sono ridotte e dunque si incoraggia l'uso della voce permettendo anche output solo vocali (Vocal Assignment).
Gli elementi di interazione della piattaforma desktop permettono di indicare attraverso l'attributo *input* la modalità di inserimento dei dati, in questo caso il concetto di equivalenza indica che è possibile inserire l'input attraverso la voce o graficamente in maniera indifferente. Il prompt e il feedback possono essere forniti solo in modo solo grafico o in maniera ridondante con entrambe

Element type	Interaction phase	CARE property for desktop	CARE property for mobile
Composition Operator			
Grouping Relation	Output	**Graphical Assignment** Redundancy	Vocal Assignment Graphical Assignment **Redundancy**
Only Output Interactor			
Description,Object, Feedback, Alarm, Table	Output	**Graphical Assignment** Redundancy Complementarity	Vocal Assignment Graphical Assignment **Redundancy** Complementarity
Interaction Interactor			
Single/multiple selection Text Edit Numerical Edit	Input	Graphical Assignment **Equivalence**	Graphical Assignment **Equivalence** Vocal Assignment
	Prompt	**Graphical Assignment** Redundancy	Graphical Assignment **Redundancy** Vocal Assignment
	Feedback	**Graphical Assignment** Redundancy	Graphical Assignment **Redundancy** Vocal Assignment
Activator	Input	Graphical Assignment **Equivalence**	Graphical Assignment **Equivalence** Vocal Assignment
	Prompt	**Graphical Assignment** Redundancy	Graphical Assignment **Redundancy** Vocal Assignment
	Feedback	**Graphical Assignment** Redundancy	Graphical Assignment **Redundancy**
Navigator	Input	Graphical Assignment **Equivalence**	Graphical Assignment **Equivalence** Vocal Assignment
	Prompt	**Graphical Assignment** Redundancy	Graphical Assignment **Redundancy** Vocal Assignment
	Feedback	Vocal Assignment **None**	Vocal Assignment **None**

Figura 4.1: CARE properties per piattaforme desktop e mobile

le modalità. Vale la pena puntualizzare il significato dell'attributo feedback per quanto riguarda l'interattore navigator che permette di spostarsi da una presentation a un'altra; in genere quest'elemento non presenta un feedback immediato perché il feedback è rappresentato dal caricamento della nuova presentation, tuttavia è possibile avere un feedback vocale attraverso un testo sintetizzato, prima del caricamento della nuova pagina, che indica che il cambio di presentation è stato eseguito.
I valori delle CARE properties per gli interattori della piattaforma mobile assumono valori differenti rispetto a quelli della piattaforma desktop, il motivo è da ricercare nel tentativo di riduzione dello spazio occupato dagli elementi grafici in schermi tipicamente piccoli; perciò negli attributi l'input e prompt è stato aggiunta la possibilità di avere anche un *Assegnamento Vocale* oltre agli altri due valori previsti nella piattaforma desktop.
Per agevolare lo sviluppatore l'ambiente fornisce, per ogni elemento e per ogni stadio, dei valori di default per le CARE properties, questi saranno i valori applicati se lo sviluppatore non effettua cambiamenti. Nella tabella 4.1 i valori suggeriti sono indicati in grassetto.
Nel caso della piattaforma *multimodal desktop*, nella quale non si hanno problemi di risorse grafiche, di default gli operatori di composizione sono supportati solo graficamente; gli *interaction elements* sono strutturati in modo che il prompt sia grafico (assegnamento), l'input possa essere assunto in modo grafico o vocale (equivalenza) e il *feedback* sia fornito graficamente (assegnamento); nel caso dei navigator di default non è previsto alcun feedback. Per quanto riguarda gli elementi *only output* di default sono presentati solo graficamente.

Nella piattaforma *multimodal mobile*, nella quale le risorse grafiche sono ridotte, i valori di default per gli operatori di composizione comprendono sia la parte grafica che quella vocale (ridondanza); negli *interactor elements* sia per il prompt che per l'input e il feedback le modalità suggerite sono entrambe (equivalenza per l'input e ridondanza per il prompt e il feedback). Nel caso degli *only output elements* le due modalità interagiscono in maniera

ridondante in modo da ridurre le risorse utilizzate.

Per garantire una maggiore flessibilità nella gestione dell'interazione tra la parte grafica e quella vocale viene messo a disposizione dello sviluppatore un pannello (fig. 4.2) nel quale è possibile modificare, per ogni elemento coinvolto nell'interazione, i valori degli attributi CARE; in questo modo si rende customizzabile la progettazione delle interfacce multimodali.

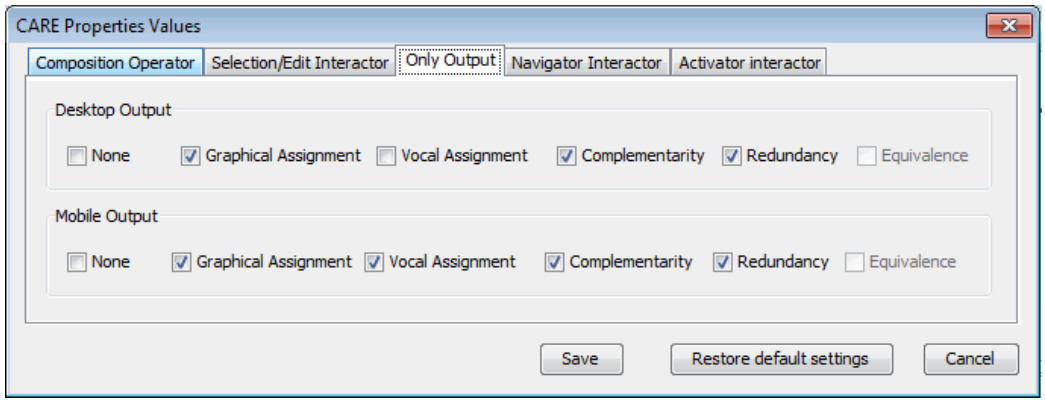

Figura 4.2: Pannello per la personalizzazione delle proprietà CARE

4.2 Progettazione linguaggio logico

Il linguaggio concreto per comporre le modalità grafica e voce è basato su due linguaggi (uno per la modalità grafica e uno per quella vocale) definiti precedentemente in maniera separata e aggiunge la possibilità di specificare come comporli attraverso le proprietà CARE.
In modo da comprendere meglio come funziona viene preso un esempio per ogni tipo di operatore: operatore di composizione (grouping), operatore only output (description) e operatore di interazione (text edit).

4.2.1 Composition interactor

Nella figura 4.3 viene mostrato l'elemento grouping relativo alla piattaforma desktop, si nota l'elemento *interactor express group*, definito nella AUI

e quindi presente in tutte le interfacce concrete, che rappresenta una scelta tra una lista di interattori o un insieme di operatori di composizione; l'elemento grid è specifico della piattaforma desktop e permette di visualizzare gli elementi del grouping usando un layout a griglia (righe e colonne).

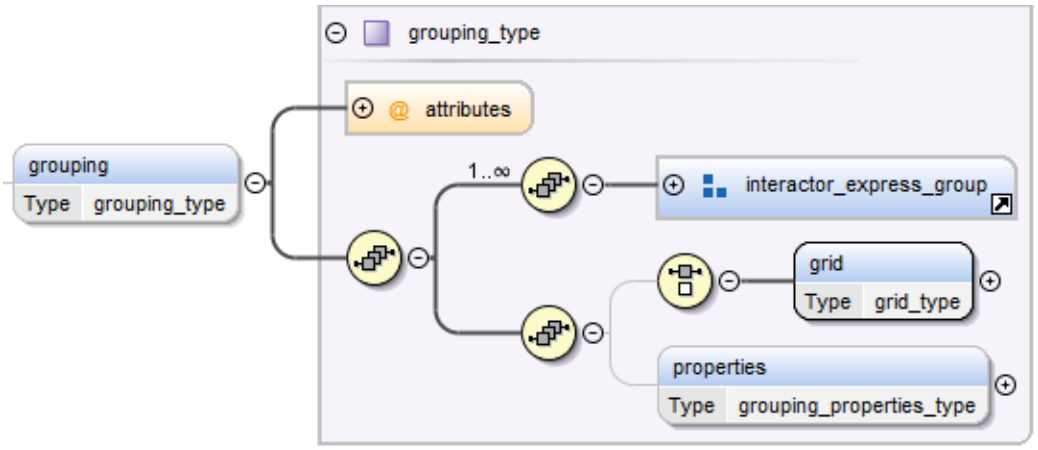

Figura 4.3: Grouping per piattaforma desktop

La figura 4.4 rappresenta la definizione vocale dell'elemento grouping: l'elemento interactor express group è sempre presente mentre si notano i nuovi elementi *grouping start* e *grouping end* utilizzati per esprimere vocalmente rispettivamente l'inizio e la fine di un raggruppamento tramite l'inserimento di un suono, una pausa, una parola chiave o cambiando le proprietà di sintesi (sez.3.6.3).
La definizione dell'elemento grouping per le interfacce multimodali (fig. 4.5) include la definizione del grouping di entrambe le modalità e attraverso l'attributo output indica come comporle.

4.2.2 Interactor: Edit

Nella figura 4.6 è mostrata la definizione dell'elemento text edit per l'interfaccia concreta desktop: si può notare come l'elemento sia caratterizzato dagli elementi tipici per l'inserimento di un testo nelle interfacce grafiche: text field e text area.

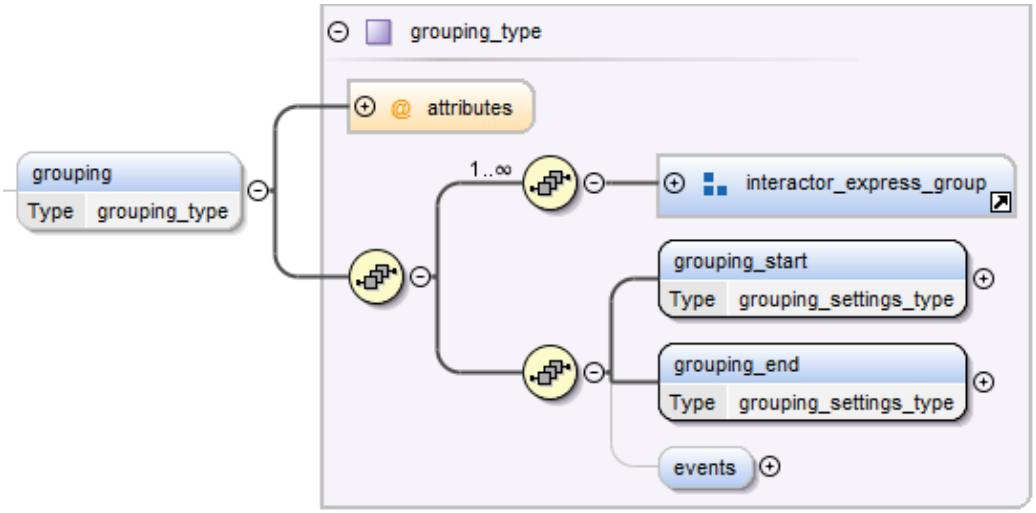

Figura 4.4: Grouping per piattaforma vocale

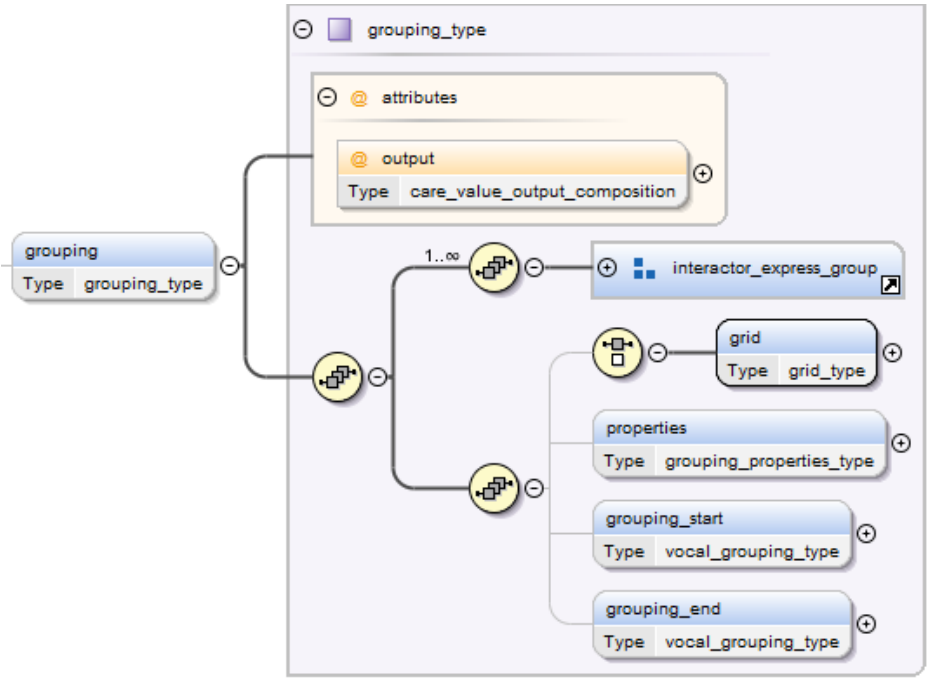

Figura 4.5: Grouping per piattaforma multimodale

L'interfaccia vocale (fig. 4.7) è specializzata attraverso l'elemento vocal textual input che permette l'inserimento di un prompt e di una grammatica per il riconoscimento di un input vocale.

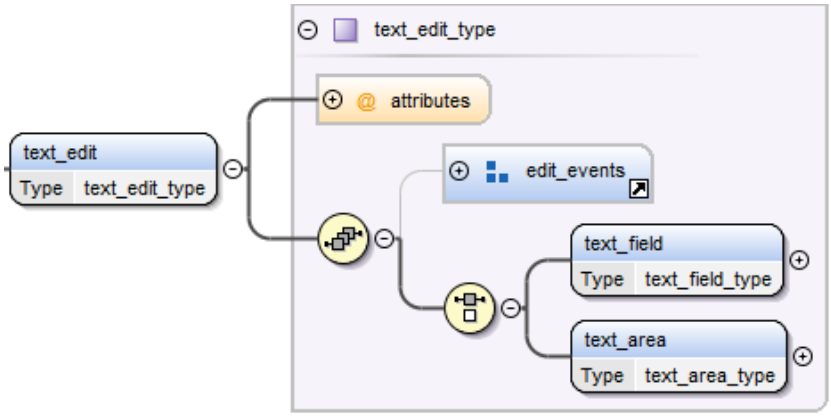

Figura 4.6: Text edit per piattaforma desktop

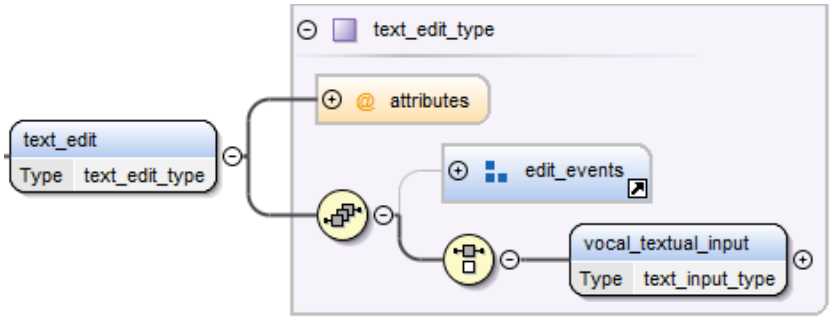

Figura 4.7: Text edit per interfaccia vocale

La descrizione multimodale dell'elemento text edit è composta dall'unione delle definizioni della piattaforma desktop e di quella vocale e degli attributi *input*, *prompt* e *feedback* che assumono i valori CARE (fig 4.8).
Per permettere l'inserimento di un feedback vocale da dare all'utente in seguito all'inserimento di un input, nella definizione vocale del text edit per l'interfaccia multimodale è stato aggiunto l'elemento feedback (di tipo Vocal) che consente l'inserimento di un testo da sintetizzare (fig. 4.9).
Alla definizione degli elementi vocal dell'interfaccia vocale (sez. 3.6.4) è stato aggiunto l'attributo **timeout** permette di impostare l'intervallo di tempo durante il quale la piattaforma attende un input dell'utente prima di sollevare l'evento *no input* (sez. 3.6.2).

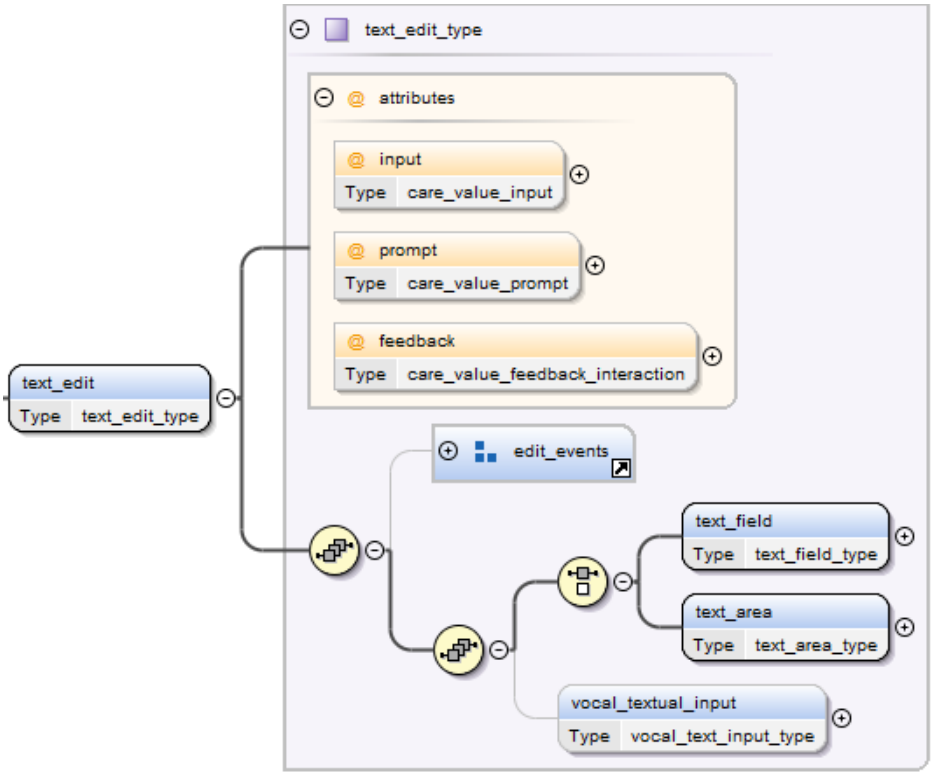

Figura 4.8: Text edit per interfaccia multimodale

4.2.3 Interactor: Only Output

Gli interattori *only output* sono composti dagli elementi *description*, *object*, *feedback* e *alarm*; il più interessante di questi è l'elemento description che è specializzato dagli elementi text, image, table, audio e video. Chiaramente dal punto di vista vocale l'elemento description, come gli altri interattori only output, non possiede queste distinzioni puramente grafiche ed è composto solo da un elemento *speech* che consente di sintetizzare un testo, dall'elemento *prerecorded message* che permette di riprodurre un file audio contenente un messaggio registrato o da un elemento *sound* per riprodurre un file audio contenente un suono (sez.4.2.3).

Un elemento *table* (fig.4.10) è costituito graficamente da una *label*, da un elemento *head*, *body* e *foot*; l'head costituisce l'intestazione della tabella ed è composto da un insieme di celle, il body rappresenta il corpo ed è costituito

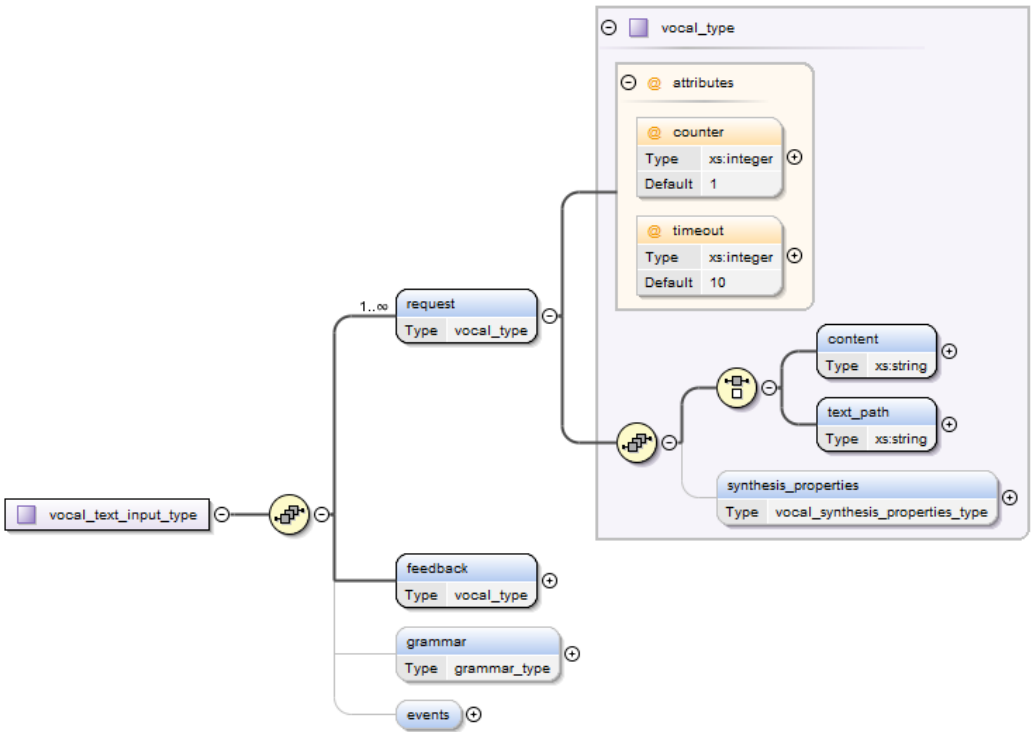

Figura 4.9: Vocal Text Edit

da righe e celle; l'elemento foot rappresenta il footer ed è composto da un insieme di celle.

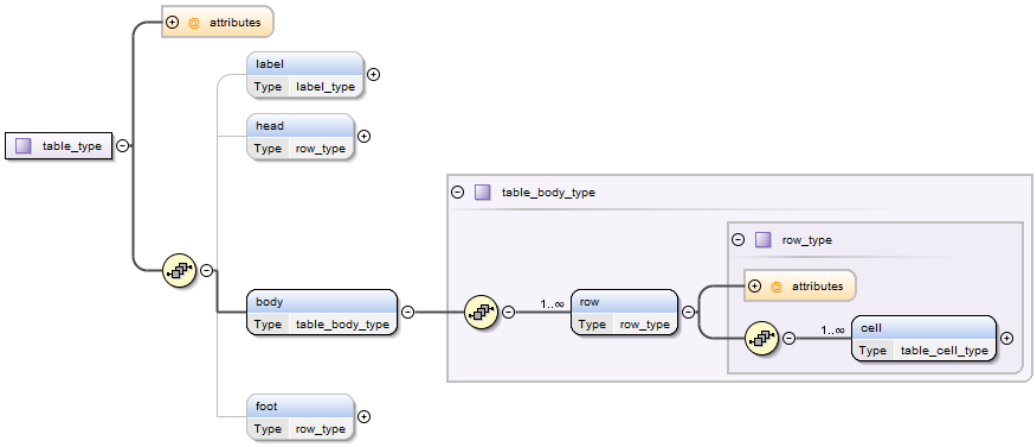

Figura 4.10: L'elemento table per la piattaforma desktop

Nella CUI vocale non era presente una descrizione dell'elemento table; nel

caso della CUI multimodale ho ripreso la definizione grafica della tabella e all'elemento cella ho aggiunto un elemento *vocal cell* (fig.4.11) che permette l'inserimento di un testo da sintetizzare o di un suono.

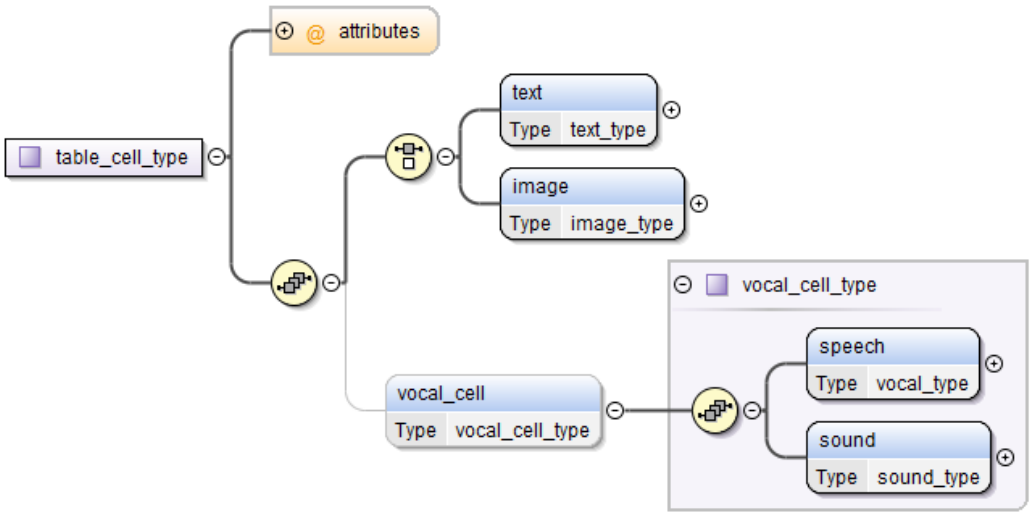

Figura 4.11: L'elemento table cell per la piattaforma multimodale

4.3 Differenze tra desktop e mobile

Le differenze tra lo schema desktop e lo schema mobile dal punto di vista grafico sono minime:

- Attributo *device type* di interface per specificare il tipo di device;
- Attributo *orientation* della presentation che indica l'orientamento del dispositivo;
- Gli eventi del mouse sono solo *mouse move*, *mouse click* e *mouse double click*.

Capitolo 5

XHTML + Voice

5.1 Panoramica

XHTML + Voice [3] è un linguaggio basato su XML utilizzato per sviluppare interfacce utente multimodali composte da grafica e voce; in genere viene abbreviato con **X+V** e d'ora in avanti per riferirmi al XHTML + VOICE utilizzerò la sigla X+V. Attualmente il linguaggio X+V è supportato dal browser Opera.

X+V combina il linguaggio XHTML per la parte grafica e un sottoinsieme di VoiceXML per la parte vocale; entrambi sono standard del W3C. XHTML è un linguaggio di markup che associa proprietà del XML con le caratteristiche dell'HTML. Il VoiceXML è anche esso un linguaggio di markup standardizzato dal W3C utilizzato per la creazione di interfacce vocali. Le caratteristiche principali del VoiceXML riguardano la creazione di dialoghi audio attraverso il supporto al sistema text to speech [1] e alla riproduzione di audio pre-registrati; il VoiceXML permette di ricevere input vocali tramite ASR[2] e DTMF[3] e di registrare l'input dell'utente. Uno dei vantaggi di basare X+V

[1]Nei sistemi **Text To Speech** l'audio è interamente generato dalla macchina, il procedimento consiste nel passare un testo a un sintetizzatore vocale che lo trasforma in parlato.

[2]Automatic speech recognition

[3]Dual Tone Multi-frequency

su VoiceXML è l'esistenza di una vasta comunità di sviluppatori che porta con se un buon supporto, frammenti di codice esemplicativi e tool. Avendo a disposizione la parte di codice XHTML e quella VoiceXML si ha tutto quello che serve per lo sviluppo di un'applicazione web multimodale, l'unica cosa di cui si ha bisogno è un modo per dire al multimodal browser come correlare una porzione del codice voice con un elemento grafico e quando avviare ogni dialogo vocale (perché lo speech engine può avere solo un dialogo attivo alla volta): X+V utilizza gli eventi XML standard, attraverso gli eventi tipici del HTML (come *on load*, *on input focus*, etc) viene creata una correlazione tra la parte grafica del interfaccia e gli elementi vocali. Un evento prevede un event handler, che specifica un'azione che deve essere eseguita in seguito al verificarsi di un particolare evento, in X+V spesso l'event handler corrisponde al form VoiceXML da attivare in seguito al evento.
L'utilizzo di XHTML e VoiceXML insieme permette agli sviluppatori di applicazioni web di aggiungere input e output vocale alle pagine web tradizionalmente basate solo sulla grafica.

Vediamo un semplice esempio di come grafica e voce vengono utilizzate in maniera coordinata e di come vengono correlate attraverso gli eventi XML; l'esempio riguarda un form di ricerca di voli aerei, in questo scenario l'utente riceve un prompt sia grafico che vocale, fornisce un input vocale che lo speech engine riconosce e trasforma in testo col quale riempie il campo, a questo punto il focus passa al prossimo field dove ha luogo la prossima interazione. La parte grafica (XHTML) per il campo di inserzione della città di partenza è molto semplice:

```
Departure city: <input type="text" id="from" name="departure_city"/>
```

La parte VoiceXML per il campo vocale corrispondente è un poco più complessa in quanto presenta:

- Un prompt vocale per porre una domanda all'utente;
- Una grammatica che elenca una lista di aeroporti;

- Una direttiva che indica allo speech engine dove mettere l'input ricevuto;
- Direttive in caso di fallimento (help, input non riconosciuto o nessun input).

La grammatica è un modo per indicare al motore di riconoscimento vocale quali parole e frasi sono gli input accettati dall'applicazione.

```
<vxml:form id="voice_city">
 <vxml:field name="field_city" id="field_city">
  <vxml:grammar src="city.grxml" type="application/srgs+xml"/>
  <vxml:prompt>Please enter your departure city</vxml:prompt>
  <vxml:catch event="help no match noinput">
         For example say Florence
  </vxml:catch>
  <vxml:filled>
   <vxml:assign name="document.getElementById('from')" expr="field_city"/>
  </vxml:filled>
 </vxml:field>
</vxml:form>
```

L'ultimo passo è aggiungere gli eventi XML ai tag XHTML che identificano le condizioni o l'evento che attiverà il codice vocale.

```
<input type="text" id="from" name="departure_city" ev:event="inputfocus"
 ev:handler="#field_city"/>
```

Quando il focus è sul campo from viene generato l'evento *inputfocus* che viene gestito dall'handler indicato che corrisponde al VoiceXML form.
Per sincronizzare la parte grafica con quella vocale viene utilizzato un tag proprio del linguaggio X+V che verrà descritto nella sezione 5.3.1.

Nell'esempio proposto sotto si mostra il codice completo, come si può notare la parte vocale è contenuta all'interno del tag *head*, mentre la parte grafica è contenuta all'interno del tag *body*, ottenendo in questo modo una completa e chiara separazione del codice relativo alle due modalità.

```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:ss="http://www.w3.org/2001/10/synthesis"
 xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
 xmlns:ev="http://www.w3.org/2001/xml-events"
 xmlns:vxml="http://www.w3.org/2001/vxml" xml:lang="en-US">
 <head>
  <title>XHTML + Voice example</title>
  <xv:sync xv:input="departure_city" xv:field="#field_city"/>
  <vxml:form id="voice_city">
   <vxml:field name="field_city" id="field_city">
```

```
    <vxml:grammar src="city.grxml" type="applicaiton/srgs+xml"/>
    <vxml:prompt>Please enter your departure city</vxml:prompt>
    <vxml:catch event="help no match noinput">
           For example say Florence
    </vxml:catch>
    <vxml:filled>
     <vxml:assign name="document.getElementById('from')" expr="field_city"/>
    </vxml:filled>
   </vxml:field>
  </vxml:form>
 </head>
 <body>
  <input type="text" id="from" name="departure_city" ev:event="inputfocus"
   ev:handler="#voice_city"/>
 </body>
</html>
```

5.2 Elementi VoiceXML supportati in X+V

5.2.1 Form

I form sono i componenti chiave della parte vocale dei documenti X+V, essi si occupano di fornire informazioni all'utente e di raccoglierne l'input. Un elemento form, che rappresenta un handler vocale attivato in risposta a un evento XHTML o VoiceXML, generalmente presenta un elemento `<filled>` che specifica le azioni da compiere quando uno o più campi sono stati riempiti. Questo elemento può apparire anche come figlio di un input item, in questo caso il comportamento specificato sarà conseguenza dell'immissione di solo quell'input. I form item sono i figli diretti dell'elemento form, questi vengono valutati nell'algoritmo di interpretazione dei form (FIA[4]). L'algoritmo FIA è un algoritmo di valutazione implicito costituito da una fase di inizializzazione seguita da un ciclo principale; nella fase di inizializzazione si ha un assegnamento degli opportuni valori alle variabili della form (ogni variabile form item viene inizializzata a *undefined*) e i contatori del numero di output verso l'utente sono settati a 1. Il ciclo principale si compone di tre fasi:

[4]Form Interpretation Algorithm

1. **selezione:** in questa fase l'algoritmo individua il dialogo successivo, vengono valutate le guardie dei form item e si sceglie il primo form item la cui guardia è false (undefined);

2. **raccolta:** una volta selezionato un form item si procede con la sua valutazione. Viene fatta una richiesta di input all'utente, verrà attivata l'opportuna grammatica e catturato l'input (o il possibile evento sollevato);

3. **elaborazione:** se il risultato della fase di raccolta è un input allora questo verrà associato al corrispondente campo della form e la variabile guardia del form item viene settata a true in modo che l'algoritmo non visiti nuovamente il form item. Quando tutti le variabili guardia dei form item sono uguali a true (tutti gli input sono stati inseriti) verrà avviata l'eventuale azione associata alla form. Se il risultato della fase di raccolta è un evento allora verrà eseguito l'handler corrispondente.

5.2.2 Field

L'elemento *field* è un figlio diretto di form, definisce un input field e formula un dialogo vocale tra l'utente e il browser. Field possiede un attributo *type* che specifica il tipo del campo, ad esempio il nome di una *built in grammar*. Le built in grammar sono delle grammatiche predefinite associate a un field; i valori supportati sono: boolean, date, digits, currency, number, phone e time. Specificando il tipo del field non è necessario associargli una grammatica in quanto ne verrà utilizzata una predefinita.

```
<vxml:field name="confirm_field" type="boolean">
 <vxml:prompt>
  Say yes to confirm your input
 </vxml:prompt>
 <vxml:filled>
  <vxml:prompt>Input confirmed</vxml:prompt>
 </vxml:filled>
</vxml:field>
```

5.2.3 Block

Un elemento *block* è un form item che viene usato per contenere del testo da sintetizzare o del contenuto eseguibile; generalmente gli elementi block vengono eseguiti solo una volta per form.

```
<vxml:block>
 Welcome to my multimodal application
</vxml:block>
```

Il padre di un elemento block può essere solo l'elemento form; in genere vengono usati per sintetizzare un testo all'inizio di un form, non per dare un prompt all'utente all'interno di un field, per questa funzione vengono usati gli elementi prompt descritti più avanti.

5.2.4 Catch

Il linguaggio VoiceXML prevede un meccanismo di cattura degli eventi; i tipi di eventi possibili sono:

- **no input:** nessun input rilevato dal sistema;
- **no match:** nessuna corrispondenza trovata tra l'input ricevuto e l'insieme dei valori riconosciuti dichiarati nella grammatica;
- **help:** richiesta di assistenza da parte dell'utente.

L'elemento *catch* cattura un evento sollevato da un elemento VoiceXML o dall'interprete.

```
<vxml:catch event="noinput" count="1">
 No input event, please say something
</vxml:catch>
```

Nell'attributo event si elenca la lista degli eventi da catturare, se la lista è vuota verranno catturati tutti gli eventi. L'attributo count rappresenta l'occorrenza dell'evento e permette di trattare in maniera differente successive occorrenze dello stesso evento.

5.2.5 Throw

Questo elemento solleva un evento, all'interno del form VoiceXML, che verrà propagato verso l'elemento HTML che ha invocato il form.

```
<vxml:throw event="some.event"/>
```

5.2.6 Grammar

Una grammatica è un'enumerazione in forma compatta di un insieme di dichiarazioni (parole e frasi) che costituiscono le risposte accettate a un dato prompt. Tutte le parole che si vogliono far riconoscere allo speech recognition engine devono essere incluse in una grammatica. Una grammatica può essere una semplice lista di parole o può essere definita in modo più flessibile in modo che sia capace di riconoscere frasi. Il linguaggio X+V supporta solo grammatiche di tipo JSGF[5] che è un formato, sviluppato dalla Sun ™Microsystem, per la definizione di grammatiche testuali indipendenti dalla piattaforma che adotta lo stile e le convenzioni del linguaggio Java in aggiunta alle classiche notazioni. All'interno di un documento X+V è possibile definire le grammatiche creandole all'interno dell'applicazione, in questo caso si parla di **inline grammar**.

```
<vxml:field name="name_field" id="name_field">
 <vxml:grammar>
  <![CDATA[
   #JSGF V1.0 iso-8859-1;
   grammar lastnames;
   public <lastnames> = Marco | Davide | Teresa | Anna;
  ]]>
 </vxml:grammar>
</vxml:field>
```

L'uso delle inline grammar è sconsigliato in quanto non è possibile il riuso in altri documenti.
Il modo migliore per definire una grammatica è definirla esternamente al documento X+V (**external grammar**) in quanto queste possono essere riutilizzate anche in altri documenti.

```
<vxml:grammar src="lastnames.jsgf"/>
```

[5]Java Speech Grammar Format

Un altro modo per definire una semplice grammatica è utilizzare l'elemento *option* che consente di definire un insieme di alternative all'interno di un elemento field.

```
<vxml:field id="departure_city_field">
 <vxml:prompt>
  Choose the departure city
 </vxml:prompt>
 <vxml:option>rome</vxml:option>
 <vxml:option>florence</vxml:option>
 <vxml:option>milan</vxml:option>
</vxml:field>
```

In questo caso gli input accettati sono quelli definiti all'interno degli elementi option.
Un altro semplice modo per inserire una grammatica è utilizzare l'elemento *rule* che definisce una regola con un nome; l'elemento *one-of* figlio di rule permette di costruire una grammatica come un insieme di frasi o parole ognuna delle quali è contenuta all'interno di un elemento *item*.

```
<vxml:field name="name_field">
 <vxml:prompt>Enter your name</vxml:prompt>
 <vxml:grammar root="name_grammar">
  <rule scope="public" id="name_grammar">
   <one-of>
    <item>Marco</item>
    <item>Teresa</item>
    <item>Davide</item>
    <item>Anna</item>
   </one-of>
  </rule>
 </vxml:grammar>
</vxml:field>
```

Nell'attributo root di grammar si specifica il nome dell'elemento rule che definisce la grammatica, il nome di rule è specificato nell'attributo id. Nell'esempio sopra si definisce una grammatica che accetta uno dei valori specificati nell'elemento input.

5.2.7 Assign

Quest'elemento assegna il valore di un'espressione a una variabile che può appartenere sia al form VoiceXML o al documento Html.

```
<assign expr="voice_navigator_field" name="window.location"/>
```

5.2.8 Clear

Resetta il valore di una o più variabile, inclusi i form items.

```
<clear namelist="city state zip"/>
```

5.2.9 Filled

In quest'elemento si specifica un'azione che deve essere eseguita dopo che un field o l'intero form sono stati riempiti.

```
<vxml:field id="departure_city_field">
 <vxml:prompt>
  Choose the departure city
 </vxml:prompt>
 ...
 <vxml:filled>
  <vxml:prompt>
        The departure city is <vxml:value expr="departure_city_field"/>
  </vxml:prompt>
 </vxml:filled>
</vxml:field>
```

5.2.10 Audio

Quest'elemento esegue un file audio specificato nell'attributo URL; nel caso i cui la risorsa audio non sia disponibile è possibile inserire un contenuto alternativo che può essere un testo da sintetizzare o un altro file audio.

```
<vxml:audio src="msg.waw">Welcome to multimodal system</vxml:audio>
```

Il testo 'Welcome to multimodal system' rappresenta il contenuto alternativo.

5.2.11 Enumerate

L'elemento enumerate viene utilizzato per leggere una lista di elementi option definiti in un field.

```
<vxml:field id="departure_city_field">
 <vxml:prompt>
  Choose the departure city
 </vxml:prompt>
 <vxml:option>rome</vxml:option>
 <vxml:option>florence</vxml:option>
 <vxml:option>milan</vxml:option>
 <catch event="nomatch">
```

```
  Your options are <enumerate/>.
 </catch>
</vxml:field>
```

5.2.12 Prompt

L'output del sistema all'interno di un dialogo interattivo viene dato principalmente attraverso l'elemento **prompt** il cui contenuto è modellato da SSML. Prompt ha 3 attributi molto importanti:

1. **bargein:** se settato a true indica che l'utente può interrompere la riproduzione del testo;

2. **timeout:** intervallo di tempo (in secondi o millisecondi) che la piattaforma attende per l'input dell'utente prima di sollevare un evento no input;

3. **count:** serve per la gestione del tapered prompting descritta in precedenza nella sezione 3.6.4. Ogni field possiede un contatore dei prompt che mantiene il numero di volte che il prompt è stato eseguito dal momento in cui il field è attivo. Count indica il numero di volte che un prompt deve essere eseguito nel field prima che il prompt col count considerato venga selezionato ed eseguito.

SSML è parte di un grande insieme di specifiche di markup per i Voice e Multimodal Browser sviluppati attraverso il processo open del W3C. E' sviluppato per fornire un ricco linguaggio di markup basato su XML per assistere la generazione di voci sintetizzate sia nel Web che in altre applicazioni. Il ruolo essenziale del linguaggio di markup è di dare agli sviluppatori un modo standard per controllare gli aspetti dell'output vocale come pronuncia, il volume, il tono, la frequenza, etc. attraverso piattaforme con differenti capacità di sintesi [8].
Gli elementi principali che SSML mette a disposizione sono:

- **prosody**: nel quale si specificano tutte le caratteristiche tipiche di una voce sintetizzata (tono, frequenza, volume, etc.);
- **emphasis**: permette di settare il livello di enfasi (forte, moderato, etc) del testo contenuto nei suoi tag;
- **voice**: nel quale si setta il genere e l'età della voce sintetizzata.

```
<vxml:prompt count="1" timeout="10s" bargein="true">
 <ss:prosody pitch="medium" volume="loud" rate="slow">
  <ss:emphasis level="moderate">
   <ss:voice gender="female" age="30">
    Choose your departure city
   </ss:voice>
  </ss:emphasis>
 </ss:prosody>
</vxml:prompt>
<vxml:prompt count="2" timeout="10s" bargein="true">
 Choose your departure city. For example say Florence
</vxml:prompt>
```

5.2.13 Reprompt

Incrementa di uno il contatore del prompt di un field e poi viene riprodotto il prompt col attributo count pari al valore incrementato.

```
<vxml:field id="departure_city_field">
 <vxml:prompt count="1">
  Choose the departure city
 </vxml:prompt>
 <vxml:prompt count="2">
        ...
 </vxml:prompt>
 ...
 <catch event="noinput">No input <reprompt/></catch>
</vxml:field>
```

5.2.14 Value

Quest'elemento valuta e restituisce un espressione ECMAScript che è inserita in un prompt.

```
...
 <script type="text/javascript">
  var saythis = "Hello, world!";
 </script>
 <vxml:form id="sayHello">
  <vxml:block>
```

```
    <vxml:value expr="saythis"/>
   </vxml:block>
  </vxml:form>
...
```

5.2.15 Property

Quest'elemento è utilizzato per settare le proprietà vocali per un form o un form item; le proprietà riguardano il comportamento della piattaforma vocale come il valori relativi al riconoscimento vocale, timeout, bargein, etc. Se una proprietà è settata a livello di form e poi a livello di un field, quella settata nel field sovrascrive la proprietà del padre.

```
<property name="timeout" value="10s"/>
```

5.3 XHTML + Voice tag

Gli elementi descritti in questa sezione sono propri del X+V e non sono derivati dal VoiceXML.

5.3.1 Sync

L'elemento *sync* fornisce il supporto per la sincronizzazione dei dati inseriti via vocale o grafica.

```
<xv:sync xv:input="departure_city" xv:field="#field_city"/>
```

L'attributo *xs:input* denota il nome di input field XHTML; l'attributo *xv:field* riferisce l'id di un field all'interno di un form VoiceXML.
Questo elemento collega il valore di una proprietà di un input XHTML al field VoiceXML con il dato attributo id, ciò significa:

- L'input raccolto vocalmente viene restituito sia al field VoiceXML sia all'elemento input XHTML;
- I dati inseriti via tastiera aggiornano sia il field VoiceXML che l'elemento input XHTML;

- In un form VoiceXML attivo con più field, se l'utente seleziona un input field dall'interfaccia grafica, l'algoritmo FIA visiterà il corrispondente field VoiceXML come prossimo item.

L'elemento *sync* è un elemento del linguaggio X+V perciò non viene definito all'interno del form VoiceXML, ma è un figlio diretto dell'elemento **head**.

5.4 Tag VoiceXML non supportati in X+V

Come detto in precedenza X+V è un sottoinsieme del VoiceXML, per cui alcuni elementi VoiceXML non sono supportati:

- *vxml*: non esiste come elemento di root perché giustamente nell'X+V l'elemento root è *HTML*;
- *transfert* e *disconnect*: perché l'architettura di un'applicazione VoiceXML prevede una chiamata via linea telefonica al *Voice Gateway*, mentre nell'X+V si ha una normale comunicazione con un web server;
- *VoiceXML menu*: *menu*, *enumerate* e *choice* consentono di proporre delle scelte all'utente, recepire la sua decisione e reindirizzarlo opportunamente verso un altro field o verso un altro documento VoiceXML;
- *goto*: rimanda a uno specifico form all'interno del documento oppure a una pagina esterna all'applicazione;
- *link*: permette la transizione ad un nuovo documento (o dialogo) oppure può lanciare un evento. I link sono definiti globalmente e quindi sono attivabili da qualunque punto del documento;
- *submit*: viene utilizzato per sottomettere i campi di un form

L'assenza degli elementi strettamente legati all'architettura VoiceXML è normale, ma non supportare elementi importanti come goto, link e submit introduce delle difficoltà nell'implementare vocalmente semplici azioni come il passaggio da una pagina all'altra e la sottomissione dell'input dell'utente. Come

gli elementi VoiceXML non supportati sono stati resi in X+V verrà spiegato nel prossimo capitolo relativo alla trasformazione da CUI multimodal a X+V.

Capitolo 6

La trasformazione di generazione

6.1 XSLT

In questa sezione presento il linguaggio XSLT [15], l'analisi si focalizzerà sugli aspetti più interessanti per la trasformazione dalla CUI multimodal al linguaggio X+V.

6.1.1 Introduzione

L'XSLT è un linguaggio per trasformare documenti XML in altri documenti XML. XSLT è predisposto per essere utilizzato in congiunzione al linguaggio XPath 2.0, definito in [4], il quale permette di individuare i nodi all'interno di un albero XML.
Una trasformazione in XSLT è espressa attraverso uno *stylesheet* la cui sintassi è un documento XML well-formed; il termine stylesheet riflette il fatto che uno dei ruoli più importanti del XSLT è dare informazioni di stile a un albero XML sorgente e trasformandolo in un albero destinazione. La trasformazione è ottenuta attraverso un insieme di regole di template (*template rules*) che sono composte da:

- *pattern* nel quale si specificano una serie si condizioni su un nodo e il cui scopo è effettuare il controllo di corrispondenza con un nodo presente nell'albero di partenza;

- *template*: nel quale si definisce come deve essere composto l'albero di destinazione relativo al nodo che ha superato il controllo di corrispondenza del pattern.

```
<xsl:template match="cui:presentation">
 <html>
  <head><title>Presentation title</title></head>
  <body>Presentation content</body>
 </html>
</xsl:template>
```

Nell'esempio sopra si definisce una regola composta dal pattern *cui:presentation* nel quale si specifica la condizione per applicare il template, tra l'apertura e la chiusura del tag viene definito il template dove si specifica il codice da generare quando nell'albero di partenza viene trovato un nodo che rispetta la condizione definita nel pattern.
Un template può essere istanziato per un particolare elemento dell'albero sorgente per creare una parte dell'albero di destinazione e può contenere elementi XSLT che rappresentano istruzioni per creare parti dell'albero di destinazione. Quando un template è istanziato ogni istruzione è eseguita e rimpiazzata con la parte dell'albero destinazione che crea; le istruzioni possono selezionare e processare elementi discendenti nell'albero sorgente. Processando un elemento discendente viene creato un frammento dell'albero destinazione attraverso la ricerca di una template rule applicabile e istanziando il corrispondente template. Bisogna specificare che questi elementi vengono processati solo quando vengono selezionati attraverso l'esecuzione di una istruzione. L'albero destinazione viene costruito a partire dalla ricerca di una template rule per il nodo radice e istanziando il suo template. Nel processo di ricerca di una template rule applicabile possono essere trovate più template rule il cui pattern corrisponde a un dato elemento, ma solo una regola può essere applicata.

Il software responsabile della trasformazione da albero sorgente ad albero destinazione è detto processore XSLT; il processore integrato in MARIAE è XALAN (http://xml.apache.org/xalan-j/).

6.1.2 Definizione del foglio di stile

Il linguaggio XSLT mette a disposizione una serie di costrutti, descritti in questa sezione, che permettono di definire il foglio di stile, l'output del documento destinazione e le regole dei template.

6.1.2.1 xsl:stylesheet

Uno stylesheet è rappresentato da un elemento *xsl:stylesheet* in documento XML; questo elemento deve contenere un attributo *version* (nel nostro caso 1.0) e la dichiarazione dei namespaces [1] degli elementi utilizzati nel foglio di stile.
Gli elementi figli di stylesheet sono chiamati elementi *top-level*; i principali sono:

- *xsl:include*;
- *xsl:import*;
- *xsl:output*;
- *xsl:template*

Questi elementi verranno descritti nelle sezioni seguenti.

6.1.2.2 xsl:include

L'inclusione è uno dei meccanismi forniti dal XSLT per combinare fogli di stile; dal punto di vista semantico è equivalente a copiare un foglio di stile dentro a un altro esattamente nel punto in cui viene inserita la richiesta.

[1] I namespaces vengono utilizzati per disambiguare elementi omonimi appartenenti a XML schema differenti.

```
<xsl:include href="another_stylesheet.xsl"/>
```

L'attributo *href* consente di definire l'URI dovere reperire lo stylesheet da includere.

6.1.2.3 xsl:import

A differenza dell'inclusione, il processo di importazione ha effetto sulla priorità dei template e il foglio di stile importato ha una precedenza maggiore. Per esempio supponiamo che:

- Lo stylesheet A importa gli stylesheet B e C in questo ordine;
- Lo stylesheet B importa lo stylesheet D;
- Lo stylesheet C importa lo stylesheet E.

Dunque l'ordine di priorità tra gli stylesheet è: D, B, E, C, A.

6.1.2.4 Il modello di processo

Il processo di generazione dell'albero destinazione è ricorsivo. Per ogni nodo dell'albero sorgente vengono valutati tutti i template compatibili e viene scelto il migliore. La *risoluzione dei conflitti* avviene in 2 fasi:

- Inizialmente vengono scartati i template con priorità più bassa in base alle importazioni effettuate;
- In questa fase vengono scartati i template con priorità più bassa in base alle priorità impostate manualmente [2]

Alla fase di scelta del template segue la fase di istanziazione nella quale vengono eseguite le istruzioni presenti al suo interno.

[2] E' possibile impostare manualmente le priorità di un template attraverso l'attributo *priority* dell'elemento template.

6.1.2.5 xsl:output

Il processore XSLT può fornire in uscita un albero destinazione in diversi formati. Quelli supportati da questa versione di XSLT sono:

- XML;
- HTML;
- Testo.

La definizione del tipo di output può essere effettuata solo a livello *top-level* tramite l'elemento:

```
<xsl:output\>
```

Alcuni dei suoi attributi principali sono:

- *method:* può essere uno tra xml, html e text;
- *version:* definisce la versione del tipo di output scelto;
- *indent:* attiva (yes) o disattiva (no) l'indentazione;
- *enchoding:* specifica la codifica dei caratteri da utilizzare.

6.1.2.6 Pattern

Un pattern specifica un insieme di condizioni su un nodo; un nodo che soddisfa tali condizioni è compatibile (match) col pattern. [15].
I pattern più comuni per individuare nodi nell'albero sorgente sono:

- *grouping :* individua l'elemento specificato;
- ** :* seleziona qualsiasi elemento;
- */ :* seleziona l'elemento root;
- *single choice|multiple choice :* individua ogni elemento single choice e ogni elemento multiple choice;

- *presentation/grouping :* seleziona ogni elemento grouping figlio di presentation;
- *//single choice :* individua ogni elemento single choice indipendentemente dalla sua profondità;
- *single choice[@input='equivalence']/vocal selection :* seleziona gli elementi vocal selection il cui padre single choice ha l'attributo input che vale equivalence.

6.1.2.7 xsl:template

Una *template rule* è definita attraverso l'elemento *xsl:template*:

```
<xsl:template match="pattern" name="template_name" priority="int
    " mode="mode_name">
 template content
</xsl:template>
```

Gli attributi hanno il seguente significato:

- *match*: specifica il pattern;
- *name*: definisce il nome del template; viene usato per richiamare l'applicazione di un template per nome;
- *priority*: setta la priorità del template; l'utilizzo della priorità è stato spiegato nella sezione 6.1.2.4;
- *mode*: permette di specificare template con contenuto diverso ma con lo stesso pattern. Attraverso questo attributo un nodo dell'albero sorgente può produrre nodi con contenuto diverso nell'albero di destinazione.

Il contenuto all'interno dell'elemento *xsl:template* individua il testo che viene generato quando la regola viene applicata.
Vediamo un esempio di un documento XML:

```
<cui:presentation>
</cui:presentation>
```

Questo è stylesheet che definisce le regole per la trasformazione:

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
     /1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="cui:presentation">
   <html>
    content
   </html>
  </xsl:template>
 </xsl:stylesheet>
```

Il documento risultato della trasformazione è il seguente:

```
<html>
 content
</html>
```

6.1.2.8 xsl:apply-templates

Per istanziare un template si utilizza l'elemento:

```
<xsl:apply-templates>
```

Gli attributi possibili sono:

- *select*: specifica il path del nodo dell'albero sorgente da processare; in questo attributo è possibile utilizzare le espressioni XPath per selezionare il nodo. Se l'attributo select non è specificato vengono processati tutti i nodi figli del nodo corrente;
- *mode*: vengono richiamati solo i template con l'attributo mode uguale al mode specificato.

Vediamo un esempio:

```
<cui:presentation>
 <cui:grouping>
  ...
 </cui:grouping>
</cui:presentation>
```

Stylesheet utilizzato per la trasformazione:

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
     /1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="cui:presentation">
   <html>
    <head><title>Presentation</title></head>
    <body>
     <xsl:apply-templates select="cui:grouping"/>
    </body>
   </html>
  </xsl:template>

  <xsl:template match="cui:grouping">
   <div>
     ...
   </div>
  </xsl:template>
 </xsl:stylesheet>
```

Il risultato della trasformazione sarà:

```
<html>
 <head><title>Presentation</title></head>
  <body>
   <div>
    ...
   </div>
  </body>
</html>
```

6.1.2.9 xsl:call-template

Un template può essere richiamato anche per nome attraverso l'elemento

```
<xsl:call-template name="template_name"/>
```

Nell'attributo *name* si indica il nome del template da richiamare. Questa tecnica consente di definire un solo template per elementi il cui nome è diverso ma possiedono lo stesso contenuto ed è possibile richiamarlo più volte all'interno di template differenti.

6.1.3 Creazione dell'albero di destinazione

In questa sezione vengono descritti i costrutti che il linguaggio XSLT mette a disposizione per definire l'albero di destinazione.

6.1.3.1 xsl:element

Per creare un nuovo elemento all'interno di un template viene utilizzato il costrutto:

```
<xsl:element name="element_name" namespace="ns"
    use-attribute-sets="attr1 attr2" />
```

Gli attributi hanno il seguente significato:

- *name*: nome dell'elemento (obbligatorio);
- *namespace*: specifica la URI del namespace (opzionale);
- *attribute-sets*: è una lista di nomi di attributi separati da uno spazio.

6.1.3.2 xsl:attribute

Per creare gli attributi di un elemento di utilizza:

```
<xsl:attribute name="attr_name" namespace="ns" />
```

Il contenuto tra il tag di apertura e di chiusura di quest'elemento specifica il valore dell'attributo.

6.1.3.3 xsl:text

Un template può contenere dei nodi di testo specificati in questo modo:

```
<xsl:text disable-output-escaping="yes" />
```

L'attributo *disable-output-escaping* abilita o disabilita l'elaborazione della sequenza di escape; se vale yes allora i caratteri speciali vengono mandati in output come sono; altrimenti viene utilizzata la sequenza di escape. Il contenuto compreso tra il tag di apertura e di chiusura specifica il contenuto del nodo testuale.

6.1.3.4 xsl:value-of

Attraverso *xsl:value-of* è possibile selezionare il valore di un elemento XML e aggiungerlo all'output.

```
<xsl:value-of select="@id" disable-output-escaping="yes"/>
```

L'attributo *select* può contenere un'espressione XPath che specifica da quale nodo o attributo estrarre il valore; l'attributo *disable-output-escaping* ha lo stesso significato descritto nella sezione precedente.

Quest'esempio vuole essere un riassunto dei costrutti presentati in questa sezione:

```
<cui:presentation>
 <cui:grouping id="container">
  <cui:description>
   Hello world!
  </cui:description>
 </cui:grouping>
</cui:presentation>
```

Stylesheet utilizzato per la trasformazione:

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
     /1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="cui:presentation">
   <html>
    <head><title>Presentation</title></head>
    <body>
     <xsl:apply-templates select="cui:grouping"/>
    </body>
   </html>
  </xsl:template>

  <xsl:template match="cui:grouping">
   <xsl:element name="div">
    <xsl:attribute name="id">
     <xsl:value-of select="@id"/>
    </xsl:attribute>
    <xsl:call-template name="description"/>
   </xsl:element>
  </xsl:template>

  <xsl:template name="description">
```

```
    <xsl:value-of select="."/>
  </xsl:template>
 </xsl:stylesheet>
```

Il risultato della trasformazione sarà:

```
<html>
 <head><title>Presentation</title></head>
  <body>
   <div id="container">
    Hello world!
   </div>
  </body>
</html>
```

6.1.4 Costrutti di supporto

In questa sezione verranno descritti i costrutti utilizzati per direzionare il flusso di esecuzione all'interno di un documento XSLT.

6.1.4.1 xsl:for-each

Quest'elemento permette di ciclare all'interno di un documento XSLT.

```
<xsl:for-each select="node_name"/>
```

L'attributo *select* specifica l'espressione XPath per selezionare i nodi su cui applicare il template.

```
<cui:presentation>
 <cui:single_choice id="city">
  <cui:choice_element value="Abbasanta">
  <cui:choice_element value="Pisa">
  <cui:choice_element value="Vasto">
 </cui:single_choice>
</cui:presentation>
```

Stylesheet utilizzato per la trasformazione:

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
     /1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="cui:presentation">
   <html>
    <head><title>Presentation</title></head>
    <body>
```

```
    <xsl:apply-templates select="cui:single_choice"/>
   </body>
  </html>
 </xsl:template>

 <xsl:template match="cui:single_choice">
  <xsl:element name="select">
   <xsl:attribute name="id">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
   <xsl:for-each select="cui:choice_element">
    <option><xsl:value-of select="@value"/></option>
   </xsl:for-each>
  </xsl:element>
 </xsl:template>
```

Il risultato della trasformazione sarà:

```
<html>
 <head><title>Presentation</title></head>
  <body>
   <select id="city">
    <option>Abbasanta</option>
    <option>Pisa</option>
    <option>Vasto</option>
   </select>
  </body>
</html>
```

6.1.4.2 xsl:if

Questo elemento permette di valutare una condizione e produrre un determinato output a seconda se questa è verificata o no.

```
<xsl:if test="expression"/>
```

L'attributo *test* permette di specificare l'espressione booleana da valutare.

6.1.4.3 xsl:choose

Questo costrutto è equivalente al costrutto *if-then-else* comune a tutti i linguaggi di programmazione; permette di valutare condizioni multiple.

```
<xsl:choose>
 <xsl:when test="expression">
  ...
 </xsl:when>
 <xsl:otherwise>
```

```
    ...
  </xsl:otherwise>
</xsl:choose>
```

All'interno di *choose* è possibile specificare quanti elementi *when* si vogliono, ognuno con una condizione diversa.

6.2 Trasformazione da CUI multimodal a X+V

La trasformazione da CUI multimodal verso un documento X+V avviene attraverso diversi fogli di stile XSLT. Per ogni elemento dell'interfaccia concreta è stato definito un mappaggio verso un elemento X+V; in questa sezione verranno presentate le scelte che ho effettuato e verranno mostrate porzioni di codice della trasformazione.

6.2.1 Applicazione delle CARE properties

Come nel livello della progettazione dell'interfaccia multimodale concreta, anche nella fase di generazione le proprietà CARE giocano un ruolo molto importante; prima di trasformare ogni interattore controllo il valore degli attributi multimodali (input, prompt, feedback e output) e a seconda del loro valore il risultato della trasformazione è differente; i valori possibili sono:

- **Vocal Assignment**: viene generata solo la parte vocale e non quella grafica:
- **Graphical Assignment**: viene generata solo la parte grafica;
- **Equivalence**: l'input viene raccolto in entrambi i modi, viene quindi generato, per la parte vocale, un field VoiceXML, un elemento input per la parte grafica e un elemento X+V per sincronizzare le due modalità;
- **Complementarity e Redundancy**: la distinzione tra questi 2 valori è significativa a livello logico/descrittivo, a livello di codice generato

non vi è differenza in quanto viene generata sia la parte vocale che quella grafica, la differenza sta nel contenuto.

6.2.2 Tipo di output

Come introdotto nella sezione 6.1.2.5 le trasformazioni XSLT producono in output un albero diversi formati; poiché un documento X+V è un documento XML well-formed viene naturale che il tipo di output sia XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:cui="http://giove.isti.cnr.it"
 xmlns:vxml="http://www.w3.org/2001/vxml"
 xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
 xmlns="http://www.w3.org/1999/xhtml">

 <xsl:output method="xml" version="1.0" indent="yes"/>

</xsl:stylesheet>
```

6.2.3 Utilizzo dei modi

Nella generazione della parte vocale si possono trovare situazioni diverse a seconda che ci di trovi all'interno di un form VoiceXML o no. Nell'applicazione vocale viene generato un unico form VoiceXML e dialoghi vengono mappati come dei *field* se è previsto un inserimento di input (cioè se l'attributo input ha il valore equivalence) o in caso contrario come semplici elementi *block*.

Per differenziare la situazione in cui l'elemento che devo generare si trovi già all'interno di un form VoiceXML utilizzo l'attributo *mode* di un template XSLT (sez.6.1.2.7), i valori possibili sono:

- **normal**: l'elemento non si trova all'interno di un form VoiceXML, crea l'elemento form e richiama gli altri template con il modo form;
- **form**: l'elemento si trova all'interno di un form, non c'è bisogno di creare il form;

- **interface.presentation**: è il mode base per richiamare i template che generano la parte grafica dell'interfaccia.

Per richiamare tutti i template con un certo mode si utilizza il comando XSLT:

```
<xsl:apply-templates mode='form'/>
```

6.2.4 Inclusione

La generazione della parte grafica dell'interfaccia è stata definita in precedenza nel foglio di stile **DesktopCUIToXHTMLtrans.xsl**; il codice relativo alla trasformazione vocale è stata definita in due fogli di stile separati: uno per i template con mode normal (**MultimodalDesktopCUItoXV**) e uno per i template con mode form (**MultimodalDesktopCUItoXVForm**). In questo modo la definizione della trasformazione grafica e vocali sono ben separate ed è stato possibile riutilizzare parte di codice pre-esistente.
Ho deciso di utilizzare l'inclusione in quanto non avevo necessità di introdurre una priorità tra i template.

6.2.5 Mappaggio

6.2.5.1 Interface

L'elemento *Interface* non genera codice X+V, ma richiama il template per la presentation.

```
<xsl:template match="cui:interface">
 <xsl:apply-templates select="cui:presentation" mode="normal"/>
</xsl:template>
```

6.2.5.2 Presentation

Una *presentation* rappresenta una pagina X+V, all'interno del tag head viene richiamato per nome il template per generare gli elementi per la sincronizzazione tra parte grafica e vocale (*sync all*) e vengono applicati i template

con mode normal per generare la parte vocale della pagina. All'interno del tag body vengono invece richiamati i template per la generazione della parte grafica con mode differente da quelli per la parte vocale.

```
<xsl:template match="cui:presentation" mode="normal">
 <html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:vxml="http://www.w3.org/2001/vxml"
 xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
 xmlns:ss="http://www.w3.org/2001/10/synthesis"
 xml:lang="en-US">
 <head>
  <title><xsl:value-of select="@name"/></title>

  <xsl:call-template name="sync_all"/>

 <xsl:apply-templates mode="normal"/>
 </head>
 <body>
  <xsl:attribute name="ev:event">load</xsl:attribute>
 <xsl:attribute name="ev:handler">
   #<xsl:value-of select="//cui:grouping[1]/@id"/>_form
  </xsl:attribute>

 <xsl:apply-templates select="*" mode="interface.presentation" /
     >
  </body>
 </html>
</xsl:template>
```

Tra gli attributi del tag body ci sono *ev:event* e *ev:handler*:

```
<body ev:event="load" ev:handler="#voice_form_id">
```

Questo indica che quando la pagina viene caricata (cioè si verifica l'evento XML load) viene chiamato l'handler che gestisce l'evento; l'handler è il form VoiceXML con l'id indicato.

6.2.5.3 Sync

Come spiegato nella sezione 5.3.1, il tag *sync* si occupa della sincronizzazione tra la parte grafica e vocale. Per ogni interattore (esclusi quelli only output) il template sync controlla il valore degli attributi multimodali, l'elemento sync relativo all'interattore considerato viene generato solo se input

è uguale a *equivalence* e feedback è diverso da *vocal assignment*; si considera il valore dell'attributo input e feedback perché se l'input è solo grafico (*graphical assignment*) non è presente la parte vocale da sincronizzare, ugualmente se il feedback è solo vocale (*vocal assignment*) non c'è parte grafica da sincronizzare.

```
<xsl:template name="sync_all">
 <xsl:for-each select="//cui:single_choice">
  <xsl:if test="@input = 'EQUIVALENCE' and @feedback != '
      VOCAL_ASSIGNMENT'>
   <xv:sync>
    <xsl:attribute name="xv:input"><xsl:value-of select="@id"
        /></xsl:attribute>
    <xsl:attribute name="xv:field">#<xsl:value-of select="@id"/>
        _field</xsl:attribute>
  </xv:sync>
 </xsl:if>
 </xsl:for-each>
 ...
 <!-- QUESTO VIENE FATTO PER TUTTI GLI INTERATTORI -->
</xsl:template>
```

6.2.5.4 Presentation settings

Le proprietà della presentazione, descritte nella sezione 3.6.1, specificano le proprietà di sintetizzazione e proprietà del riconoscitore per la parte vocale; per la parte grafica vengono specificate colore o immagine di background e impostazioni sui font.
Tra i settings si individuano due categorie:

- **elementi mappati**: sono quegli elementi che possiedono un corrispondente diretto nel codice VoiceXML o XHTML, si tratta delle proprietà del riconoscitore vocale e delle impostazioni grafiche.

- **elementi riferiti**: sono le proprietà di sintetizzazione, non hanno una mappatura diretta ma vengono riferiti da altri elementi per estrarne informazioni. Ad esempio saranno utilizzati all'interno dei template prompt, nel caso in cui le proprietà di sintetizzazione non siano specificate localmente nell'elemento si risalirà sino alle proprietà della presen-

tazione per settare le impostazioni vocali; nelle sezioni seguenti verrà data una spiegazione più dettagliata sul loro utilizzo.

Nell'esempio sotto verrà mostrato come vengono utilizzati nella trasformazione gli elementi con una mappatura diretta.

```
<xsl:template name="cui:presentation_settings/cui:vocal_settings
    /speech_recognizer">
 <vxml:properties>
  <xsl:attribute name="timeout">
   <xsl:value-of select="@timeout"/>
  </xsl:attribute>
 </vxml:properties>
 ...
</xsl:template>
```

Le altre proprietà settabili riguardano il bargein e il livello di confidenza del riconoscimento vocale.
Le proprietà grafiche vengono settate nell'attributo *style* del tag body.

```
<body>
 <xsl:attribute name="style">
  font-family : <xsl:value-of select="cui:presentation_settings/
      cui:font_settings/@name"/>;
  background-color : <xsl:value-of select="
      cui:presentation_settings/cui:background/
      cui:background_color"/>
 </xsl:attribute>
 ...
<body>
```

Le altre proprietà grafiche settabili sono: *background image*, varie proprietà che riguardano i font come *name*, *color*, *size*, *style*, etc.

6.2.5.5 Eventi

Gli eventi definiti nella CUI possiedono una corrispondenza diretta nel codice VoiceXML:

```
<xsl:template match="cui:events" mode="form">
 <vxml:noinput>
  <vxml:prompt>
  <xsl:attribute name="bargein">false</xsl:attribute>
  <xsl:value-of select="cui:noinput/@message"/>
```

```
  </vxml:prompt>
  <xsl:if test='cui:noinput/@reprompt'>
   <vxml:reprompt/>
  </xsl:if>
 </vxml:noinput>

 <vxml:nomatch>
  <vxml:prompt>
  <xsl:attribute name="bargein">false</xsl:attribute>
  <xsl:value-of select="cui:nomatch/@message"/>
  </vxml:prompt>
  <xsl:if test='cui:nomatch/@reprompt'>
   <vxml:reprompt/>
  </xsl:if>
 </vxml:nomatch>

 <vxml:help>
  <vxml:prompt>
  <xsl:attribute name="bargein">false</xsl:attribute>
  <xsl:value-of select="cui:help/@message"/>
  </vxml:prompt>
  <xsl:if test='cui:help/@reprompt'>
   <vxml:reprompt/>
  </xsl:if>
 </vxml:help>
</xsl:template>
```

Per ogni evento è stato settato l'attributo bargein a false per impedire che l'utente interrompa l'ascolto del messaggio vocale. Lo sviluppatore può scegliere attraverso l'attributo *reprompt* se riproporre l'ultimo dialogo ascoltato.

6.2.5.6 Grouping

Come rendere vocalmente l'interattore grouping è stato spiegato nella sezione 3.6.3; nella porzione di codice in seguito mostro l'esempio del template grouping con mode normal, vale a dire che non è all'interno di un form VoiceXML, in questo caso creo il form VoiceXML con attributo id uguale all'attributo id del grouping.

```
<xsl:template match="cui:grouping" mode="normal">
 <vxml:form>
  <xsl:attribute name="id">
  <xsl:value-of select="@id"/>_form
  </xsl:attribute>
```

```
  <xsl:if test="@output = 'REDUNDANCY' or @output = '
      VOCAL_ASSIGNMENT'">
   <xsl:apply-templates select="child::cui:grouping_start"/>
 </xsl:if>

  <xsl:apply-templates mode="form"/>

  <xsl:if test="@output = 'REDUNDANCY' or @output = '
      VOCAL_ASSIGNMENT'">
  <xsl:apply-templates select="child::cui:grouping_end"/>
 </xsl:if>

 </vxml:form>
</xsl:template>
```

Gli elementi *grouping start* e *grouping end* segnalano l'inizio e la fine del grouping vengono generati solo se l'attributo *output* assume un valore diverso da *graphical assignment*; tra il grouping start e il grouping end c'è il comando `<xsl:apply-templates/>` che richiama i template relativi al mode form.

L'inserimento di suoni per indicare l'inizio e la fine di un raggruppamento è ottenuto con l'elemento `<audio>`:

```
<vxml:block>
 <vxml:audio>
  <xsl:attribute name="src">
  <xsl:value-of select="cui:path"/>
 </xsl:attribute>
 <!-- ALTERNATIVE CONTENT -->
 <xsl:value-of select="cui:alternative_content/cui:content"/>
 </vxml:audio>
</vxml:block>
```

Nel caso in cui il file audio non sia disponibile è possibile inserire un testo alternativo da sintetizzare specificato nell'elemento `alternative content`. L'inserimento di una pausa tra è ottenuto con l'elemento SSML [3] `<break>`:

```
<xsl:template name="cui:insert_break">
 <vxml:block>
  <vxml:prompt>
```

[3]SSML:Speech Synthesis Markup Language, è un linguaggio di markup basato su XML orientato alla conversione del testo in parlato; la sua funzione è specificare l'intonazione, il ritmo, le pause, la velocità, il genere e l'enfasi della voce con la quale viene sintetizzato un testo.

```
    <ss:break>
    <xsl:attribute name="time"><xsl:value-of select="@length"/>s<
        /xsl:attribute>
   </ss:break>
   </vxml:prompt>
  </vxml:block>
</xsl:template>
```

L'elemento `insert keyword` consiste nella sintetizzazione di un testo e verrà descritto nella sezione relativa agli elementi di tipo *vocal* (sez. 6.2.5.7).
Le proprietà di sintetizzazione specificate nel grouping non sono mappate in maniera esplicita ma verranno utilizzate per definire gli elementi in altri template, come avviene con le proprietà della presentation.

Graficamente, a seconda dei valori specificati nelle proprietà, un grouping può essere reso come un *fieldset*, una lista non ordinata, una lista ordinata o un div; la parte grafica del grouping viene generata solo se l'attributo multimodale *output* è diverso da *vocal assignment*:

```
<xsl:template match="cui:grouping" mode="interface.presentation"
    >
<xsl:if test='@output != "VOCAL_ASSIGNMENT" '>
 <xsl:choose>
 <xsl:when test="cui:properties/@fieldset = 'true'">
  <fieldset>
   ...
  </fieldset>
 </xsl:when>
 <xsl:when test="cui:properties/@bullet = 'true'">
  <ul>
   <xsl:for-each select="child::*">
    <li> ... </li>
   </xsl:for-each>
  </ul>
 </xsl:when>
 <xsl:when test="@ordering = 'true'">
  <ol>
   <xsl:for-each select="child::*">
    <li> ... </li>
   </xsl:for-each>
  </ol>
 </xsl:when>
 <xsl:otherwise>
  <div>
```

```
    <xsl:apply-templates select="*" mode="interface.presentation"
        />
   </div>
  </xsl:otherwise>
  </xsl:choose>
</xsl:if>
</xsl:template>
```

6.2.5.7 Interactor: only output

Questi interattori hanno il compito di fornire un output all'utente e per farlo utilizzano gli elementi **vocal**, **sound** e **prerecorded message**, gli ultimi due sono equivalenti in quanto si occupano di riprodurre un file audio e si differenziano solo per il contenuto del file riprodotto. Gli elementi sound e pre-recorded message vengono mappati con l'elemento *audio* come descritto nella sezione precedente (inserimento suoni).

```
<xsl:template match="cui:only_output">
 <xsl:if test="@output != 'GRAPHICAL_ASSIGNMENT'">
  <xsl:apply-templates name="vocal"/>
  <xsl:apply-templates select="sound"/>
  <xsl:apply-templates select="prerecorded_message"/>
 </xsl:if>
</xsl:template>
```

Come descritto nella sezione 3.6.4 l'elemento vocal possiede l'elemento content e un text path, inoltre è possibile definire le proprietà di sintetizzazione del messaggio vocale. Per la definizione delle proprietà di sintetizzazione si utilizza un modello bottom-up: se le proprietà non sono state definite localmente nell'elemento vocal allora salgo di livello e controllo che siano definite nell'antenato grouping; se anche in questo caso non sono definite allora salgo a livello di presentation e se presenti utilizzo quelle. Se le proprietà di sintesi non sono definite a nessun livello allora non vengono specificate e si utilizzano quelle di default previste dalla piattaforma.

```
<xsl:template match="cui:speech" name="vocal" mode="form">
 <vxml:prompt count="@counter"
  timeout="@timeout"s
  bargein="<xsl:value-of select="cui:vocal_properties/
      cui:bargein/@active"/>
```

```
  <ss:prosody pitch="cui:vocal_properties/@pitch" volume="
      cui:vocal_properties/@volume" rate="cui:vocal_properties/
      @rate">
   <ss:emphasis level="cui:vocal_properties/@emphasis">
    <ss:voice gender="cui:vocal_properties/@gender" age="
        cui:vocal_properties/@age">
     <xsl:value-of select="cui:content"/>
    </ss:voice>
   </ss:emphasis>
  </ss:prosody>
 </vxml:prompt>
</xsl:template>
```

Nell'esempio proposto si mostra la situazione più semplice nella quale le proprietà sono settate localmente; nel caso in cui le proprietà non siano settate localmente si utilizza l'espressione XPath *ancestor::element name* che permette di risalire l'albero sino all'elemento indicato. Gli elementi *prosody*, *emphasis* e *voice* sono tutti elementi SSML e permettono di settare le proprietà relative alla sintetizzazione di un testo come *pitch*, *rate*, *emphasis*, *gender* e *voice*.

Al posto di specificare il testo da sintetizzare nel elemento `<content>` è possibile indicare il path di un documento che contiene il testo; l'unico vincolo in questo caso è dato dal fatto che XSLT è in grado di leggere solo documenti XML validi, dunque il documento deve avere questo formato:

```
<?xml version="1.0"?>
<root>
 Testo da sintetizzare
</root>
```

Per leggere il contenuto di un file XML si utilizza il comando XSLT:

```
<xsl:value-of select="document(@text_path)" />
```

Andiamo ad analizzare l'interattore only output chiamato *description* il quale contiene un elemento *table* (sez.4.2.3) che viene reso graficamente come una tabella con un header, un body e un footer.

```
<xsl:template match="cui:table">
 <xsl:if test="parent::cui:description/@output != '
     VOCAL_ASSIGNMENT'">
```

```
  <table>
   <xsl:apply-templates select="cui:head"/>
   <xsl:apply-templates select="cui:body"/>
   <xsl:apply-templates select="cui:footer"/>
  </table>
 </xsl:if>
</xsl:template>
```

Il codice XSLT per la trasformazione del body è il seguente:

```
<xsl:template match="cui:body">
 <xsl:for-each select="child::cui:row">
   <tr>
    <xsl:for-each select="child::cui:cell">
     <td><xsl:value-of select="cui:text/cui:string"/></td>
    </xsl:for-each>
   </tr>
  </xsl:for-each>
</xsl:template>
```

E' interessante mostrare come è stato reso vocalmente l'elemento table; sono state considerate due varianti per sintetizzare una tabella:

- *linear browsing*: la tabella viene resa sintetizzando riga per riga (6.1);
- *intelligent browsing*: per ogni riga e ogni cella del body viene sintetizzata prima la corrispondente cella dell'header e poi la cella del body (fig.6.2), in modo da rendere più semplice la comprensione dei dati contenuti nella cella.

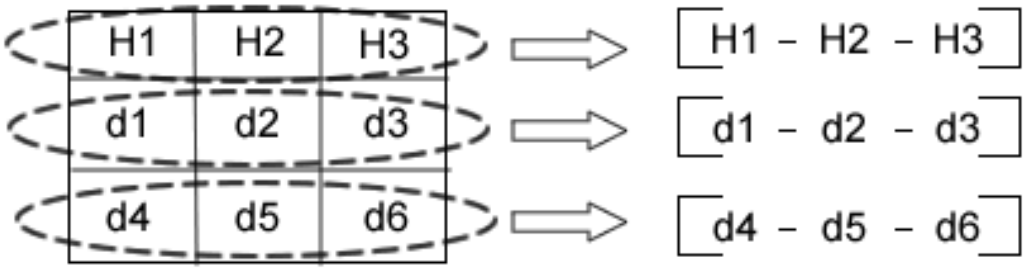

Figura 6.1: Table linear browsing

Nel caso di linear browsing, se le righe sono molte, è difficile ricordare a quale intestazione dell'header corrisponde la cella sintetizzata.

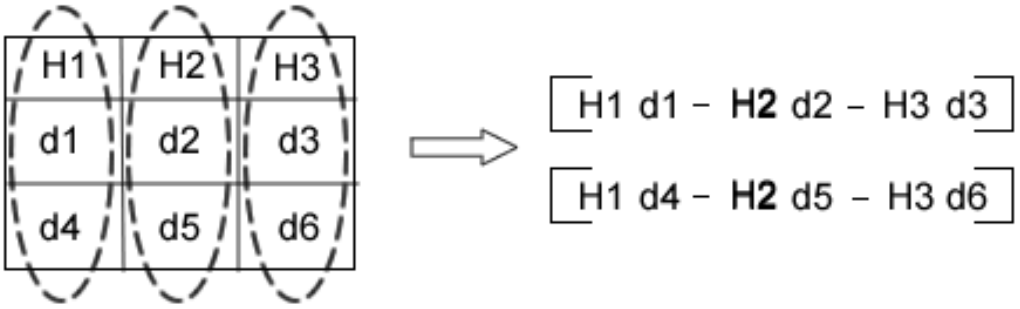

Figura 6.2: Table intelligent browsing

Se nella definizione della tabella non è presente l'header allora viene utilizzato il *linear browsing*, altrimenti la scelta preferenziale è l'*intelligent browsing*.

```
<xsl:template match="cui:table">
 <xsl:if test="@output != 'GRAPHICAL_ASSIGNMENT'">
  <xsl:for-each select="cui:body/cui:row">
   <xsl:for-each select="cui:cell">
    <!-- HEADER -->
    <xsl:variable name="position">
     <xsl:value-of select="position()"/>
    </xsl:variable>
  <xsl:value-of select="ancestor::cui:table/cui:head/cui:cell[
      position() = \$position]/cui:vocal_cell/cui:content"/>
  <ss:break time='500ms'/>
  <!-- CELL CONTENT -->
  <xsl:value-of select="child::cuivocal:cell/cui:speech/
      cui:content"/>
  <xsl:for-each/>
 <xsl:for-each/>
 </xsl:if>
</xsl:template>
```

Il codice mostrato sopra è relativo alla trasformazione della tabella con modalità di sintetizzazione *intelligent browsing*: se la modalità della tabella non è solo grafica allora faccio un ciclo sulle celle di tutte le righe del body, per ognuna di queste prelevo dall'header il contenuto della cella corrispondente e il contenuto della cella considerata. Tra la sintetizzazione dell'header e del contenuto inserisco una pausa di mezzo secondo per separare i due elementi. La sintetizzazione della tabella con modalità *linear browsing* cambia di

poco, la differenza sta solo nel fatto che non viene sintetizzato il contenuto dell'header ma solo quello del body.

6.2.5.8 Interactor: selection

Gli interattori di selezione si dividono in *single choice* e *multiple choice*.
Gli elementi single choice possono essere dei *radio button*, *list box*, *drop down list* o *image map*:

```
<xsl:template match="cui:single_choice">
 <xsl:if test="@input != 'VOCAL_ASSIGNMENT'">
  <xsl:apply-templates select="cui:radio_button"/>
  <xsl:apply-templates select="cui:list_box"/>
  <xsl:apply-templates select="cui:drop_down_list"/>
  <xsl:apply-templates select="cui:image_map"/>
 </xsl:if>
</xsl:template>
```

Vediamo in particolare come viene trasformato graficamente un elemento *drop down list*:

```
<xsl:template match="cui:drop_down_list">
 <xsl:if test="../@prompt != 'VOCAL_ASSIGNMENT'">
  <xsl:apply-templates select="cui:label"/>
 </xsl:if>
 <xsl:if test="@input != 'VOCAL_ASSIGNMENT'">
  <select name='../@id' id='../@id'>
   <xsl:for-each select="../cui:choice_element">
    <option value='@value'>
     <xsl:value-of select='cui:label/cui:text/cui:string'/>
    </option>
   </xsl:for-each>
  </select>
 </xsl:if>
</xsl:template>
```

Si nota che prima di generare qualsiasi porzione di codice si controlla sempre il valore degli attributi multimodali e in base al valore si genera o no la parte. Vocalmente non c'è differenza tra un radio button, una drop down list o una list box; una single choice viene resa sempre con un field VoiceXML, che rappresenta dove viene inserito l'input, con all'interno un eventuale prompt e feedback. Come visto nella sezione 3.6.5, non è previsto l'inserimento di una grammatica da parte dell'utente perché gli input possibili sono limitati dalle

scelte a disposizione ed è facilmente generabile una grammatica in maniera automatica.

```
<xsl:template match="cui:vocal_selection">
 <xsl:if test="../@input != 'GRAPHICAL_ASSIGNMENT'>
  <vxml:field id='../@id'>

   <xsl:if test="../@prompt != 'GRAPHICAL_ASSIGNMENT'">
    <xsl:call-template name='vocal'/>
   </xsl:if>

   <xsl:for-each select="../cui:choice_element">
    <vxml:option>
     <xsl:value-of select="@value"/>
    </vxml:option>
   </xsl:for-each>

   <xsl:if test="../@feedback != 'GRAPHICAL_ASSIGNMENT'">
    <vxml:filled>
   <vxml:prompt>
    <xsl:value-of select="cui:vocal_feedback/cui:content"/>
    <vxml:value expr="../@id"/>
   </vxml:prompt>
  </vxml:filled>
   </xsl:if>
  </vxml:field>
 </xsl:if>
</xsl:template>
```

Come si nota dall'esempio sopra la grammatica che riconosce gli input utente è generata ciclando sulle possibili scelte (*choice element*) e utilizzando l'elemento `<vxml:option>`.
Graficamente un elemento multiple choice può essere un check box o una list box:

```
  <xsl:template match="cui:multiple_choice">
  <xsl:if test="@input != 'VOCAL_ASSIGNMENT'">
   <xsl:apply-templates select="cui:check_box"/>
   <xsl:apply-templates select="cui:list_box"/>
  </xsl:if>
 </xsl:template>
```

L'implementazione dell'elemento multiple choice dal punto di vista vocale è simile a quella del single choice; la differenza sta nella generazione della grammatica per il riconoscimento dell'input. Nella generazione della gram-

matica del single choice la situazione era semplice in quanto l'utente poteva scegliere un elemento solo tra un insieme di scelte, nel multiple choice l'utente può scegliere più elementi in un ordine qualsiasi, questo introduce una certa complessità nel generare tutte le disposizioni semplici senza ripetizioni.

La soluzione adottata per rappresentare vocalmente un elemento multiple choice consiste nel creare un *field* per ogni possibile input e per ognuno di essi chiedere all'utente se vuole selezionare dato elemento.

```
<xsl:template match="cui:multiple_vocal_selection">
 <xsl:if test="../@prompt != 'GRAPHICAL_ASSIGNMENT'">
  <vxml:block>
   <xsl:call-template name='vocal'/>
  </vxml:block>
 </xsl:if>

 <xsl:if test="../@input != 'GRAPHICAL_ASSIGNMENT'>
  <xsl:for-each select="../cui:choice_element">
   <vxml:field type='boolean' id='position()'>
    <vxml:prompt>
     Do you wanna choose <xsl:value-of select="@value"/>?
    </vxml:prompt>
   </vxml:field>
  </xsl:for-each>
 </xsl:if>

 <xsl:if test="../@feedback != 'GRAPHICAL_ASSIGNMENT'">
  <vxml:filled>
   <xsl:attribute name="namelist">
    <xsl:for-each select="../cui:choice_element">
     <value-of select="position()"/>
    </xsl:for-each>
   </xsl:attribute>
  <vxml:prompt>
   <xsl:value-of select="cui:vocal_feedback/cui:content"/>
   <xsl:for-each select="../cui:choice_element">
   <!-- elenco choice element selezionati -->
  </vxml:prompt>
  </vxml:filled>
 </xsl:if>
</xsl:template>
```

Se l'attributo prompt non assume il valore *assegnamento grafico* allora genero un elemento block col prompt da sintetizzare; se il valore di input è equivalente allora per ogni input possibile (specificato attraverso gli elementi *choice*

element) creo un field specificando che è di tipo booleano, in questo modo non devo definire una grammatica ma utilizzo le *built in grammar* messe a disposizione dal linguaggio X+V. In ogni field viene richiesto all'utente se selezionare l'elemento e l'utente può rispondere con *yes*, *ok*, *true*, *no*, *false* o *wrong*; dopo che tutti i field sono stati riempiti, se è previsto un feedback viene sintetizzato un testo che riepiloga gli elementi scelti.

6.2.5.9 Interactor: edit

Gli interactor edit possono essere di 3 tipi: *text edit*, *numerical edit full* e *numerical edit in range* (sez. 3.5.7 e 3.6.6).
Elementi text edit hanno un mappaggio diretto con i tag HTML della parte grafica dell'interfaccia:

```
<xsl:template match="cui:text_area">
 <xsl:if test="../@prompt != 'VOCAL_ASSIGNMENT'">
  <xsl:apply-templates select="cui:label">
 </xsl:if>

 <xsl:if test="../@input != 'VOCAL_ASSIGNMENT'>
  <textarea>
   <xsl:attribute name="rows"><xsl:value-of select="@rows"/></
       xsl:attribute>
  <xsl:attribute name="cols"><xsl:value-of select="@length"/></
      xsl:attribute>
  </textarea>
 </xsl:if>
</xsl:template>
```

```
<xsl:template match="cui:text_field">
 <xsl:if test="../@prompt != 'VOCAL_ASSIGNMENT'">
  <xsl:apply-templates select="cui:label">
 </xsl:if>

 <xsl:if test="../@input != 'VOCAL_ASSIGNMENT'>
  <input type='text'>
   <xsl:attribute name="size"><xsl:value-of select="@length"/></
       xsl:attribute>
  <xsl:attribute name="value"><xsl:value-of select="@text"/></
      xsl:attribute>
  </input>
 </xsl:if>
</xsl:template>
```

Dal punto di vista vocale gli elementi text edit sono simili agli elementi selection, sono resi dando un prompt all'utente, ricevendo la risposta da quest'ultimo e fornendo un eventuale feedback riguardo all'input ricevuto. Si differenziano dagli elementi selection per la varietà degli input possibili, in questo caso è difficile generare automaticamente una grammatica per il riconoscimento degli input, deve essere lo sviluppatore a fornirne una definita in un file esterno o elencando un insieme di input riconosciuti e poi a partire da questi nella fase di trasformazione viene generata la grammatica.

```
<xsl:template match="cui:vocal_textual_input">
 <vxml:field>
  <xsl:if test="../@prompt != 'GRAPHICAL_ASSIGNMENT'">
   <xsl:apply-templates select="cui:request">
  </xsl:if>

  <xsl:if test="../@input != 'GRAPHICAL_ASSIGNMENT'>
   <xsl:apply-templates select="cui:grammar"/>
  </xsl:if>

  <xsl:apply-templates select="cui:events" mode="form"/>

  <xsl:if test="../@feedback != 'GRAPHICAL_ASSIGNMENT'>
   <xsl:apply-templates select="cui:vocal_feedback"/>
  </xsl:if>
</xsl:template>
```

Nella generazione della grammatica si verifica se è stata definita una grammatica esterna e in caso si legge il path del file; altrimenti si controlla se esiste una grammatica interna definita nella CUI specificando l'elenco di input riconosciuti (*grammar option*) e la si trasforma in una grammatica supportata da X+V attraverso i costrutti *grammar-rule-one of*, per l'elenco degli input riconosciti viene usato l'elemento *item*.

```
<xsl:template match="cui:grammar">
 <xsl:if test="child::cui:external_grammar>
  <vxml:grammar>
  <xsl:attribute name="src">
   <xsl:value-of select="cui:external_grammar/@src"/>
  </xsl:attribute>
  </vxml:grammar>
 </xsl:if>

 <xsl:if test="child::cui:internal_grammar>
```

```
  <vxml:grammar root="internal_grammar">
   <rule scope="public" id="internal_grammar">
    <one-of>
     <xsl:for-each select="cui:grammar_option">
      <item>
    <xsl:value-of select="@grammar_option"/>
   </item>
     </xsl:for-each>
    </one-of>
   </rule>
  </vxml:grammar>
 </xsl:if>
</xsl:template>
```

Graficamente gli elementi numerical edit vengono specializzati con gli elementi *text field* e *spin box*, il primo viene trasformato nello stesso modo dell'elemento text field per il testo normale (text edit), il secondo viene reso con un input field e due bottoni per decrementare e incrementare il numero:

```
<xsl:template match="cui:spin_box">
 <xsl:if test="../@prompt != 'VOCAL_ASSIGNMENT'">
  <xsl:apply-templates select="cui:label">
 </xsl:if>

 <xsl:if test="../@input != 'VOCAL_ASSIGNMENT'>
  <button onclick='decr(<xsl:value-of select="../@id"/>)'>-</
      button>
  <input>
   <xsl:attribute name="id"><xsl:value-of select="../@id"/></
       xsl:attribute>
   <xsl:attribute name="size"><xsl:value-of select="@length"/></
       xsl:attribute>
  <xsl:attribute name="value"><xsl:value-of select="@value"/></
      xsl:attribute>
  </input>
  <button onclick='incr(<xsl:value-of select="../@id"/>)'>+</
      button>
 </xsl:if>
</xsl:template>
```

Gli elementi numerical edit vengono mappati vocalmente alla stessa maniera degli elementi text edit; in alcune situazioni non è necessario definire una grammatica per il riconoscimento dell'input numerico ma è possibile sfruttare un insieme predefinito di grammatiche fornite dall'linguaggio VoiceXML (sez. 5.2.2). Se l'input non appartiene ai tipi di input predefiniti allora è

possibile inserire una grammatica. L'attributo *built in grammar* della definizione vocale dell'elemento numerical edit viene mappato nell'attributo *type* dell'elemento *vxml:field*:

```
<xsl:template match="cui:vocal_numerical_input">
 <vxml:field>
  <xsl:attribute name="type">
   <xsl:value-of select="@built_in_grammar"/>
  </xsl:attribute>

  <xsl:if test="../@prompt != 'GRAPHICAL_ASSIGNMENT'">
   <xsl:apply-templates select="cui:request">
  </xsl:if>

  <xsl:if test="../@input != 'GRAPHICAL_ASSIGNMENT'>
   <xsl:apply-templates select="cui:grammar"/>
  </xsl:if>

  <xsl:apply-templates select="cui:events" mode="form"/>

  <xsl:if test="../@feedback != 'GRAPHICAL_ASSIGNMENT'>
   <xsl:apply-templates select="cui:vocal_feedback"/>
  </xsl:if>
</xsl:template>
```

Nell'elemento numerical in range bisogna aggiungere un controllo per verificare che l'input immesso sia entro il range di valori definito negli attributi *min* e *max* della descrizione logica, se l'input è fuori dal range si sintetizza un messaggio e viene riproposto il prompt. Il codice prodotto sarà del tipo:

```
<vxml:field name="id_field" type="number">

 <!-- VOCAL REQUEST -->

 <!-- EVENTS -->

 <vxml:filled>
  <vxml:if cond="(id_field < MIN) | (id_field > MAX)">
   <vxml:prompt>Sorry, value not in range. Retry</vxml:prompt>
   <vxml:reprompt/>
  </vxml:if>
 </vxml:filled>
</vxml:field>
```

6.2.5.10 Interactor:control

L'elemento *navigator* consente il passaggio da una presentation a un'altra, graficamente viene mappato con l'elemento XHTML `<a href=''/>`.

```
<xsl:template match="cui:navigator">
 <xsl:if test="../@input != 'VOCAL_ASSIGNMENT'>
  <a>
   <xsl:attribute name="href">
    <xsl:variable name="idname"><xsl:value-of select="@id"/></
        xsl:variable>
    <xsl:value-of select="ancestor::cui:presentation/
        cui:connections/cui:elementary_conn[@interactor_id = \$
        idname]/@presentation_name"/>
   </xsl:attribute>

   <xsl:apply-template select="cui:button"/>
   <xsl:apply-template select="cui:text_link"/>
   <xsl:apply-template select="cui:image_link"/>
  </a>
 </xsl:if>
</xsl:template>
```

Come introdotto nella sezione 3.5.3 le connessioni indicano quale sarà la presentazione verso la quale un interattore fa transire. Nella trasformazione di un navigator bisogna risalire di livello nell'albero sino alla connection con lo stesso id del navigator e leggere l'attributo *presentation name* che indica il valore da inserire nell'attributo *href* del tag `<a>`.

Dal punto di vista vocale rendere un elemento navigator è più difficoltoso in quanto nel linguaggio X+V è assente il costrutto VoiceXML *goto* che permette di eseguire un dialogo contenuto in un altro form o in un documento esterno. L'unico modo di implementare un link vocale è utilizzare l'oggetto javascript *location*: Location è un oggetto che può essere acceduto attraverso la proprietà location dell'oggetto *Window* e contiene informazioni sulla URL corrente della pagina. Cambiando il valore della proprietà *window.location* è possibile indicare l'URL della pagina da caricare.

Per dar modo all'utente di poter attivare un link in qualsiasi dialogo attivo viene definita una grammatica globale che riconosce le parole specificate nell'attributo *label* dell'elemento vocal navigator (solo se l'attributo multi-

modale input vale equivalence). Se il navigator prevede un prompt vocale questo viene reso con un elemento *block*. Nella trasformazione del navigator viene definito un solo field VoiceXML che assume il valore assegnatoli dalla grammatica in caso di riconoscimento dell'input vocale, in questo caso viene assegnata alla proprietà window.location l'URL della pagina ricavato dalla connection corrispondente al navigator.

```
<vxml:form id="container_form">
 <vxml:grammar>
  <![CDATA[
         #JSGF V1.0;
         grammar vocal_navigator_grammar;
         public <vocal_navigator_grammar> =
          home {\$.voice_navigator_field='home.mxml'}|
          download {\$.voice_navigator_field='download.mxml'};
  ]]>
 </vxml:grammar>

 <vxml:block>
  <vxml:prompt>
   Say home to go to the home page
  </vxml:prompt>
 </vxml:block>

 <vxml:block>
  <vxml:prompt>
   Say download to go to the download page
  </vxml:prompt>
 </vxml:block>

 ...

 <vxml:field name="voice_navigator_field" id="voice_navigator_field">
  <vxml:filled>
   <vxml:prompt>You go to
    <vxml:value expr="voice_navigator_field"/> page
         </vxml:prompt>
   <assign name="window.location" expr="voice_navigator_field"/>
  </vxml:filled>
 </vxml:field>
</vxml:form>
```

Una delle funzioni dell'elemento activator è sottomettere al server i dati raccolti all'interno di un form. Dal punto di vista grafico un activator può essere mappato semplicemente attraverso un elemento HTML *button*. Come nel caso dell'elemento navigator, la trasformazione vocale di un activator è difficoltosa a causa del fatto che X+V non presenta l'elemento VoiceXML *submit* utilizzato per inviare i dati raccolti a un server.

```
<xsl:template match="cui:activator">
```

```
<vxml:field id='activator_id'>
 <vxml:option>
  <xsl:value-of select="child::vocal_activator/@label"/>
 </vxml:option>

 <xsl:if test="@prompt != 'graphical_assignment'">
  <xsl:apply-templates select="child::vocal_activator/
      cui:prompt" mode="vocal_activator"/>
 </xsl:if>

 <vxml:filled>
  <xsl:if test="@feedback != 'none'">
   <xsl:apply-templates select="child::vocal_activator/
       cui:vocal_feedback"/>
  </xsl:if>

  <xsl:if test="@input != 'graphical_assignment'">
   <xsl:variable name="id_form">
    document.forms['<xsl:value-of select="
        ancestor::cui:relation/@id"></xsl:value-of>'].submit();
   </xsl:variable>
   <vxml:value>
    <xsl:attribute name="expr"><xsl:value-of select="\$id_form
        "/></xsl:attribute>
   </vxml:value>
  </xsl:if>
 </vxml:filled>
</vxml:field>
</xsl:template>
```

Come si vede nella porzione di codice proposta sopra un activator è reso vocalmente come un field: se l'attributo prompt è diverso da *graphical assignment* viene fornito all'utente un prompt vocale; attraverso l'elemento VoiceXML *option* viene specificato l'unico input riconosciuto che provoca l'attivazione dell'activator, il valore dell'input è preso dall'attributo label del vocal activator. Per ovviare all'assenza dell'elemento submit ho utilizzato javascript: l'oggetto javascript *form* rappresenta un form HTML (esiste un oggetto form per ogni form HTML definito nel documento) e possiede un metodo *submit* che provoca la sottomissione dei dati esattamente come se si cliccasse su un bottone submit grafico. Attraverso l'elemento VoiceXML *value* (sez. 5.2.14) è possibile eseguire un'espressione javascript, l'espressione da eseguire sarà quella che, attraverso l'elemento form, provoca la sottomissione.

```
<vxml:value expr="document.forms.['form_id'].submit()"/>
```

Capitolo 7

Integrazione del linguaggio in MARIAE

Lo scopo di questo capitolo è mostrare come dalla definizione del linguaggio si è arrivati alla sua integrazione nell'Authoring Environment MARIAE.

7.1 Procedimento

Il primo passo consiste nella definizione degli elementi del linguaggio utili per la descrizione delle interfacce multimodali. Il linguaggio è stato definito tramite XSD, raffinando l'interfaccia astratta e introducendo nuovi elementi e attributi che caratterizzano le interfacce multimodali. L'esempio sotto definisce l'elemento *text edit*:

```
<xs:complexType name="text_edit_type">
 <xs:complexContent>
 <xs:extension base="text_edit_type">
  <xs:sequence>
  <xs:choice>
   <xs:element name="text_field" type="text_field_type"/>
   <xs:element name="text_area" type="text_area_type"/>
  </xs:choice>
  <xs:element name="vocal_textual_input" type="
      vocal_text_input_type"/>
  </xs:sequence>
  <xs:attribute name="input" type="care_value_type"/>
  <xs:attribute name="prompt" type="care_value_type"/>
```

```
  <xs:attribute name="feedback" type="care_value_type"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Gli elementi dell'interfaccia astratta vengono integrati con la definizione degli elementi grafici e vocali e specificando gli attributi CARE.

Il passaggio successivo è il procedimento di binding, attraverso il generatore *xjc* incluso in JAXB[1] [12], tra la rappresentazione XML dello schema e le corrispondenti classi Java che rappresentano lo schema. Ogni modifica allo schema implica che le classi java devono essere rigenerate. Il meccanismo di mapping viene implementato attraverso l'uso di annotazioni, ad esempio *@XmlRootElement* e *@XmlElement* annotano le classi che rappresentano rispettivamente l'elemento radice del file XML e un elemento generico.
La parte di codice presentata sotto rappresenta la trasformazione, tramite xjc, della definizione XML dell'elemento text edit:

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "text_edit_type", propOrder = {
    "textField",
    "textArea",
    "vocalTextualInput"
})
public class TextEditType {
 @XmlElement(name = "text_field")
 protected TextFieldType textField;

 @XmlElement(name = "text_area")
 protected TextAreaType textArea;

 @XmlElement(name = "vocal_textual_input")
 protected VocalTextInputType vocalTextualInput;

 @XmlAttribute
 protected CareValueType input;
 @XmlAttribute
 protected CareValueType prompt;
 @XmlAttribute
 protected CareValueType feedback;
 ...
 /* GETTERS E SETTERS PER OGNI ELEMENTO E ATTRIBUTO */
 public TextFieldType getTextField() {
  return textField;
 }
 public void setTextField(TextFieldType value) {
  this.textField = value;
 }
```

[1] Java Architeture for XML Binding

```
  ...
}
```

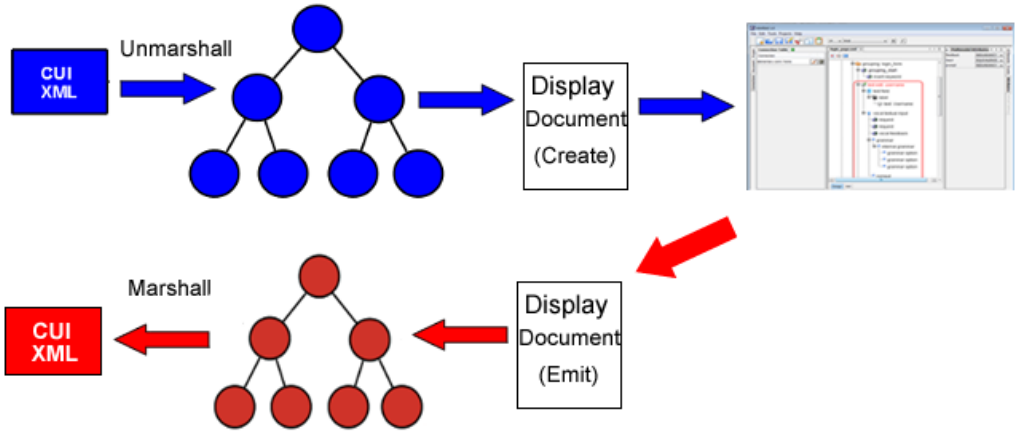

Figura 7.1: MARIAE: marshall e unmarshall

I passi successivi sono schematizzati nella figura 7.1. Per la realizzazione della vista grafica occorre che il file XML che descrive una generica interfaccia multimodale venga mappato in memoria in una struttura ad albero, questo viene ottenuto attraverso un'operazione di *unmarshalling* utilizzando le classi create col generatore JAXB. A partire dall'albero ottenuto con l'unmashalling viene creato un altro albero corrispondente alla struttura intermedia chiamata *DocumentDisplay* che contiene tutte le informazioni utili per disegnare gli elementi, come le icone da utilizzare, la dimensione del rettangolo che rappresenta l'elemento, il testo e il colore del rettangolo e la lista dei figli; inoltre contiene un riferimento al corrispondente nodo nell'albero ottenuto dal documento XML e un riferimento al padre. La struttura DocumentDisplay rappresenta il modello della vista grafica secondo il pattern *Model View Controller*.

Il procedimento inverso, vale a dire il passaggio dalla visualizzazione grafica al documento XML corrispondente, viene ottenuto attraverso l'operazione di *marshalling*.

Oltre che della generazione delle classi Java, mi sono occupato anche di realizzare il DocumentDisplay relativo al mio schema: il *MultimodalCuiDocumentDisplay*.

7.2 Document Display

La definizione di questo documento consiste nell'implementazione, per ogni classe generata, di 2 metodi:

- *create*: permette di costruire l'albero che verrà utilizzato per disegnare la vista, incapsulando ogni nodo dell'albero ottenuto dall'unmashalling dentro il relativo nodo dell'albero visuale e aggiungendo le informazioni relative alle icone, i colori e tutto ciò che è necessario per disegnare il nodo;
- *emit*: questo metodo permette di riportare nell'albero Java le modifiche apportate dall'utente nell'albero visuale.

Vediamo un esempio del metodo create e uno del metodo emit relativi all'elemento text edit:

```
private void createTextEdit(TextEditType text, DisplayTreeNode parent) {
 DisplayTreeNode node = new DisplayTreeNode(
  LanguageManager.getString("MultimodalCui.TextEditType"),
  icons.getString("MultimodalCui.TextEditType"),
  text,
  parent,
  docType);

 node.setColor(new Color(Integer.decode(
  icons.getString("MultimodalCui.TextEditType.Color"))));
 parent.addChild(node);

 createTextArea(text.getTextArea(), node);
 createTextField(text.getTextField(), node);

 createVocalTextInput(text.getVocalTextualInput(), node);

 createAttribute(text.getInput(), node, attrType.input);
 createAttribute(text.getPrompt(), node, attrType.prompt);
 createAttribute(text.getFeedback(), node, attrType.feedback);
}
```

```
private void emitTextEdit(UserObjectWrapper uow, DisplayTreeNode node) {
 /* Creo un nuovo elemento */
 TextEditType interactor = new TextEditType();

 /* Scorro la lista dei figli */
 for (DisplayTreeNode child : node.getChildren()) {
  Attribute attr = child.getAttribute();

  if (attr != null){
   attrType a = attrType.valueOf(attr.getName());

   switch (a) {
    case input:
     interactor.setInput(CareValueType.valueOf(attr.getValue()));
     break;
    case prompt:
     interactor.setPrompt(CareValueType.valueOf(attr.getValue()));
     break;
    case feedback:
     interactor.setFeedback(CareValueType.valueOf(attr.getValue()));
     break;
   }
  }else{
   Object userobj = child.getUserObject();
   Class childClass = userobj.getClass();

   if (childClass.equals(TextAreaType.class)) {
    emitTextArea(
     new UserObjectWrapper(interactor, elemType.TextEditType), child);
   } else if (childClass.equals(TextFieldType.class)) {
    emitTextField(
     new UserObjectWrapper(interactor, elemType.TextEditType), child);
   } else if (userobj instanceof VocalTextInputType) {
    emitVocalTextInput(
     new UserObjectWrapper(interactor, elemType.TextEditType), child);
   }
  }
 }
 /* Il nodo generato viene appeso al padre */
 switch (uow.getUserObjectType()) {
  case PresentationType:
   ((PresentationType) uow.getUserObjectWrapper()).setTextEdit(interactor);
   break;
   ...
 }
}
```

Il risultato dell'integrazione è visibile nella figura 7.2.

Come spiegato nella sezione 4.1 per ogni interattore e operatore di composizione è stato individuato un valore che ogni attributo multimodale assume di default. L'authoring environment in corrispondenza di questi elementi imposta il valore di default degli attributi multimodali e se questo prevede la coesistenza di entrambe le modalità vengono aggiunti all'interfaccia gli elementi che le implementano. Nella figura 7.2 l'elemento text edit ha co-

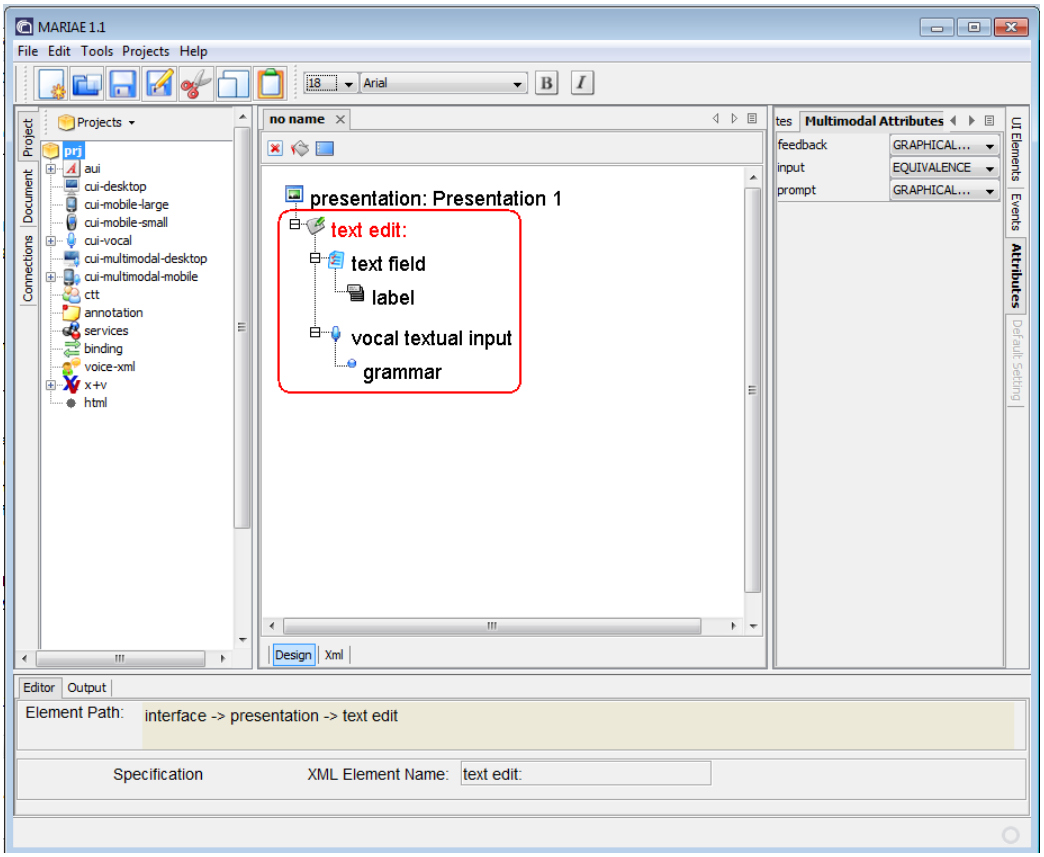

Figura 7.2: MARIAE: integrazione multimodal CUI

me figli l'elemento *vocal textual input* e *text field* perché il valore di default dell'attributo input è equivalence e prevede entrambe le modalità.

Terminata la progettazione grafica è possibile salvare il file XML che descrive logicamente l'interfaccia multimodale, questo file potrà essere preso in input dal processore XSLT per la generazione dell'interfaccia finale.

Capitolo 8

Un esempio di applicazione

In questo capitolo verranno presentate le principali scelte operate durante la fase di progettazione e di trasformazione attraverso un esempio di applicazione multimodale per la gestione di dispositivi domotici.

L'applicazione è composta da diverse *presentation*:

1. *login*: inserimento delle credenziali per l'autenticazione;
2. *lista delle stanze*: scelta della stanza da monitorare;
3. *gestione stanza*: vengono presentati i dispositivi monitorati ed è possibile selezionare un dispositivo della stanza per modificarne i settaggi.
4. *gestione dispositivo*: vengono presentate le caratteristiche del dispositivo ed è possibile cambiarle.

Per creare un'interfaccia multimodale attraverso l'authoring environment bisogna scegliere il tipo di documento dal menù file, le scelte possibili sono due: *Multimodal Desktop* e *Multimodal Mobile*. Una volta scelto il tipo di documento è possibile iniziare a definire l'interfaccia scegliendo gli elementi dal pannello destro, che presenta tutti gli elementi definiti nel linguaggio, e trascinandoli nel pannello centrale che contiene gli elementi dell'interfaccia. La struttura delle presentation dell'applicazione è composta da tre grouping

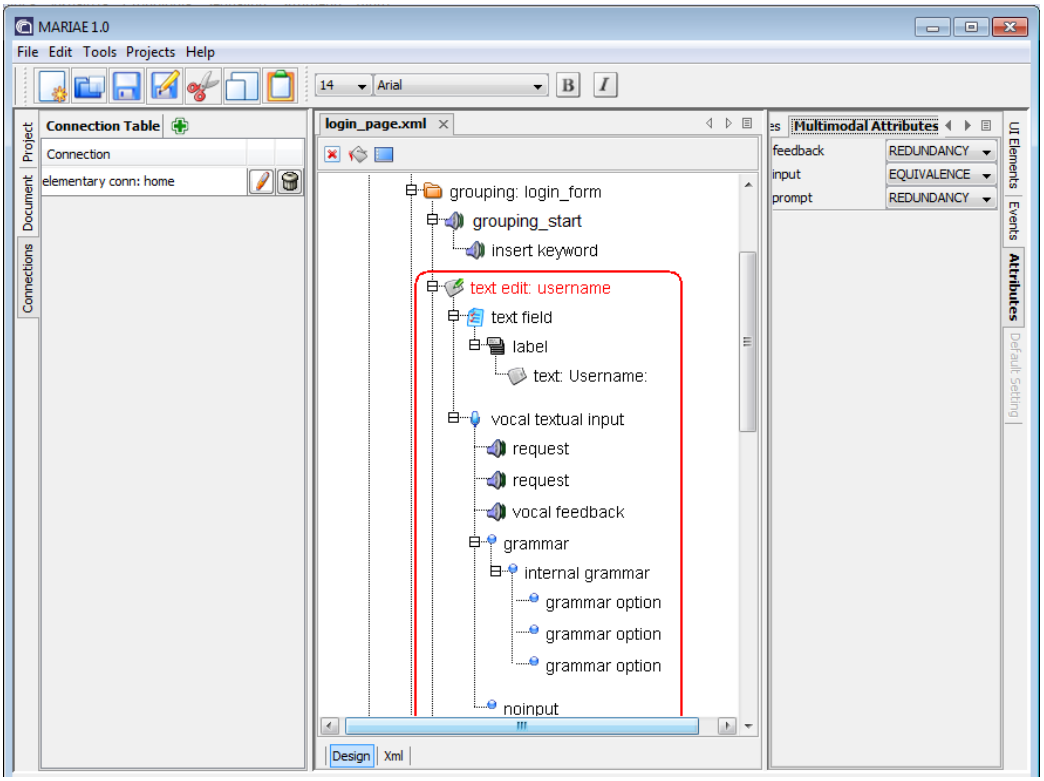

Figura 8.1: MARIAE: design dell'interfaccia multimodale

fondamentali: *header*, *body* e *footer*, all'interno del raggruppamento body bisogna inserire un ulteriore grouping che contiene i vari input field; all'interno di questo grouping si definisce un elemento vocale *grouping start* che attraverso l'inserimento di una parola chiave avvisa l'utente dell'inizio del form di login (viene sintetizzato il testo 'Start login form!'). Nella definizione del form di login bisogna inserire due elementi *text edit* per l'inserimento del nome utente e password; sulla parte destra della figura 8.1 viene mostrato il pannello degli attributi multimodali con i valori relativi ad ogni stadio dell'interazione per l'interattore che si occupa dell'inserimento dello username. Nella parte centrale col colore rosso è selezionato l'elemento *text edit* specializzato graficamente attraverso un *text field* e vocalmente attraverso l'elemento *vocal textual input* perchè i valori degli attributi multimodali prevedono entrambe le modalità. Per fornire all'utente un prompt vocale si

assegna all'attributo prompt il valore *redundancy* e l'authoring environment inserisce automaticamente un elemento *request* attraverso il quale è possibile specificare il testo del prompt da sintetizzare. Per implementare la tecnica del tapered prompting (introdotta nella sezione 3.6.4) è possibile inserire manualmente ulteriori elementi *request* con attributo *count* differente : il primo elemento request fa la domanda semplice: 'Insert your username', nel caso in cui l'utente non inserisca un input entro l'intervallo di timeout o inserisca un input non riconosciuto viene riprodotto l'elemento request con count uguale a 2, il secondo prompt fornisce informazioni più precise rispetto al primo. Per permettere l'inserimento dell'input sia vocalmente che graficamente lo sviluppatore deve settare il valore dell'attributo multimodale input a *equivalence*, l'authoring environment inserirà come figlio di vocal textual input un elemento grammar necessario per indicare alla piattaforma vocale gli input vocali riconosciuti. Il linguaggio per la definizione delle interfacce multimodali fornisce due modi per definire una grammatica: l'elemento *external grammar* nel quale indicare il percorso di un file contenente la grammatica, in alternativa è possibile definire la grammatica internamente all'interfaccia (*internal grammar*) definendo gli input accettati all'interno degli elementi *grammar option*.

MARIAE genera automaticamente il codice XML che descrive l'interfaccia multimodale, nella figura 8.2 è mostrata la porzione di codice relativa al grouping visualizzato dal design della figura precedente.
Dal menù tools dell'authoring environment è possibile trasformare l'interfaccia appena definita nell'interfaccia finale, il risultato della trasformazione è mostrato nella figura 8.3.

Nel esempio seguente voglio mostrare l'utilizzo delle *connection* per passare da una presentazione all'altra; prima di definire una connection è necessario aggiungere all'interfaccia il navigator che provoca il caricamento della nuova presentation. La figura 8.4 mostra la definizione di una connessione attraverso il designer di MARIAE: l'attributo *interactor id* individua il navigator precedentemente definito e *presentation name* indica il nome della presen-

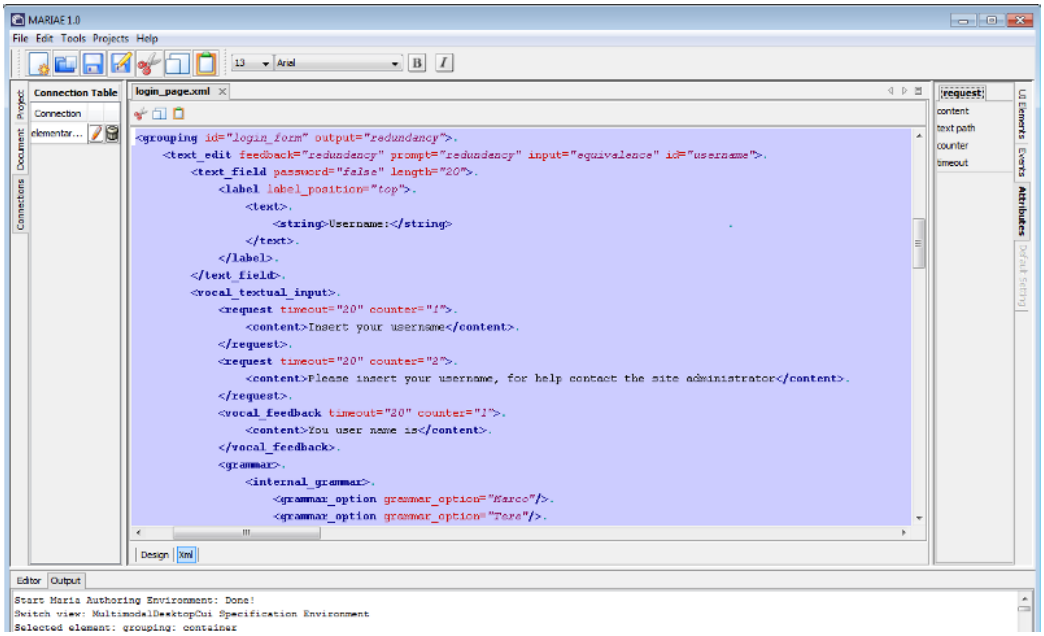

Figura 8.2: MARIAE: codice XML

tazione verso cui andare. Assegnando agli attributi multimodali prompt e feedback il valore redundancy viene aggiunto al navigator un elemento *vocal navigator* e i suoi figli *prompt* e *feedback*, mentre dal punto di vista grafico il navigator è implementato attraverso un elemento *image link*.

Nella figura 8.5 è mostrato il codice xml prodotto, come si può vedere ci sono più elementi connection (uno per ogni navigator definito) per cui in fase di trasformazione bisogna trovare la connection il cui interactor id è uguale a quello del navigator che si sta trasformando.

Il risultato della trasformazione è visibile nella figura 8.6, il grouping che contiene i navigator prevede un messaggio vocale ('Select the room you want to monitor') che introduce i navigator. Per ogni navigator viene sintetizzato un prompt che specifica l'input da inserire per passare alla nuova presentazione ('Say living to go to the living room').

Vediamo ora un esempio della pagina di gestione di una stanza relativo alla stanza *living room*: inizialmente si definisce una tabella con i dispositivi

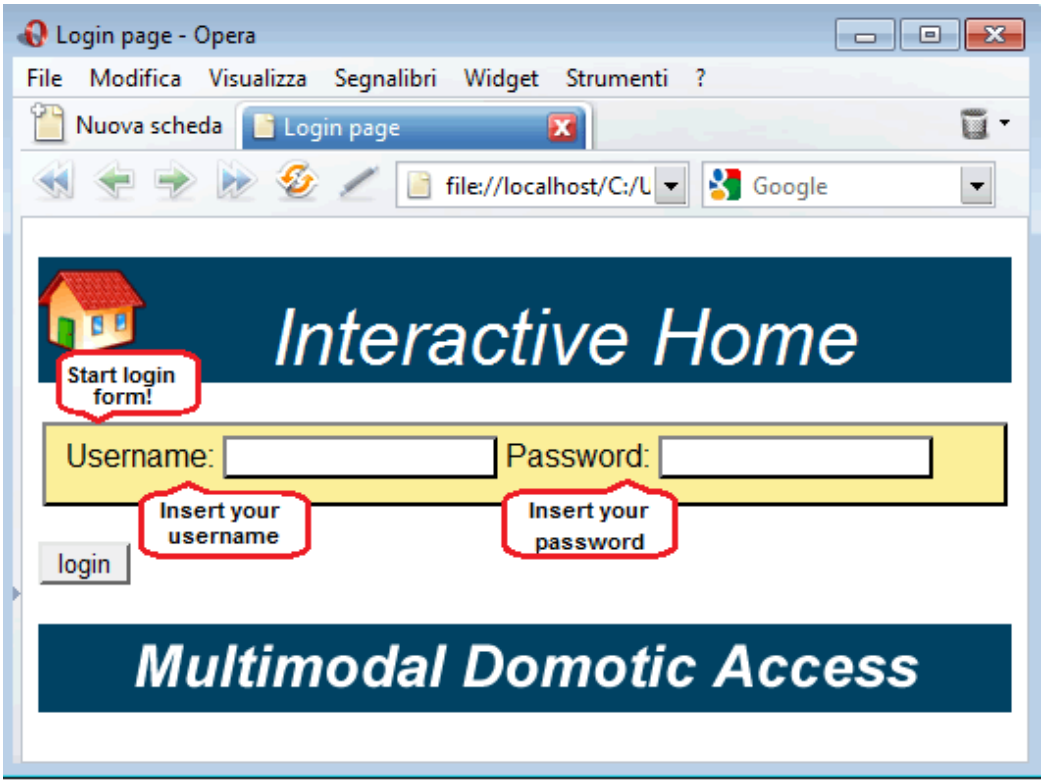

Figura 8.3: Interfaccia multimodale generata

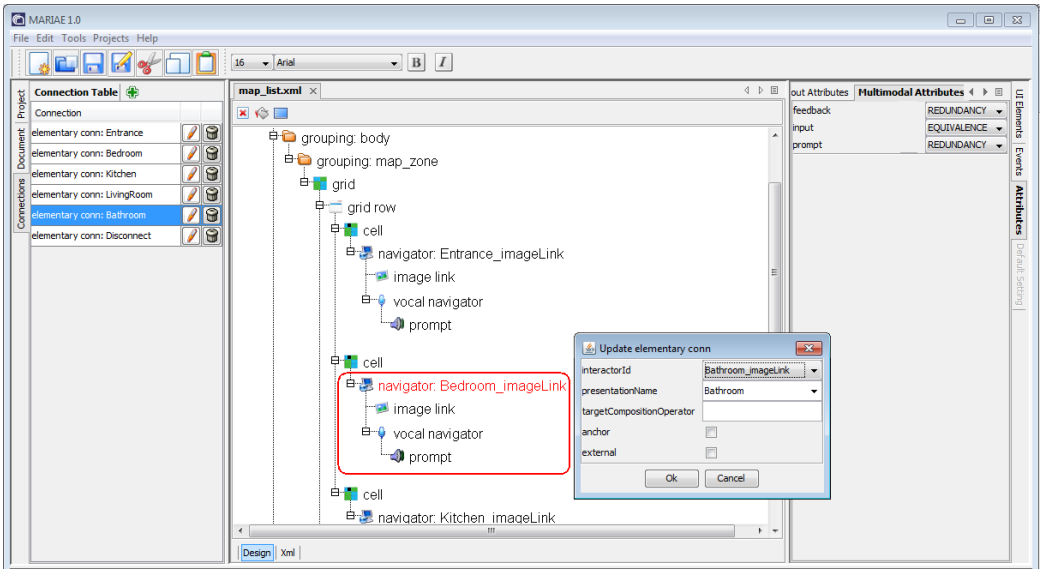

Figura 8.4: MARIAE: connection

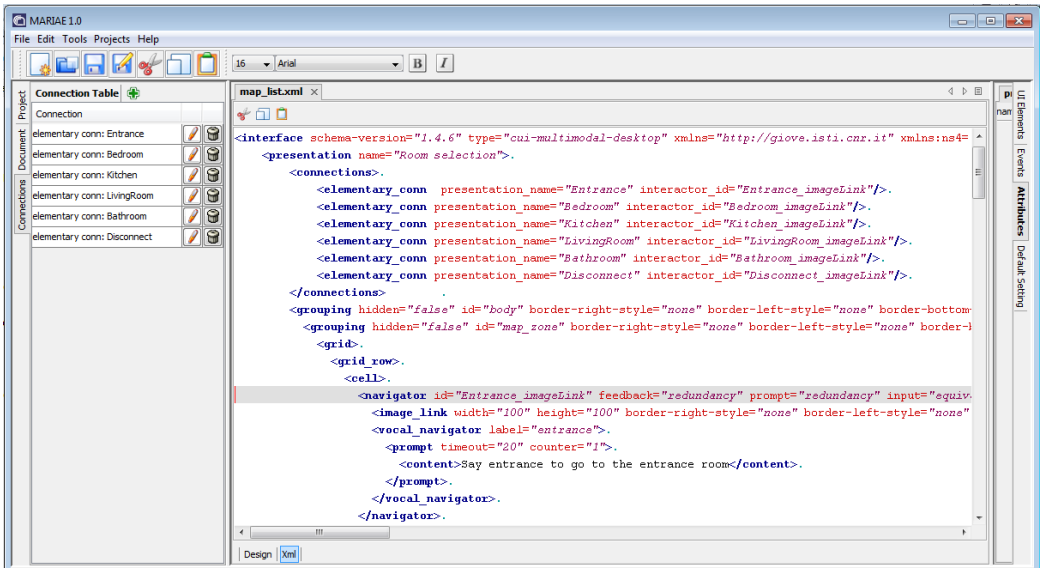

Figura 8.5: MARIAE: codice XML per le connection

Figura 8.6: Interfaccia multimodale generata

monitorati; per ogni dispositivo si specifica il nome, lo stato e i dettagli. Accanto alla tabella dei dispositivi si inserisce un elemento single choice, implementato graficamente con una *drop down list*, per scegliere un dispositivo

e modificarne le impostazioni.

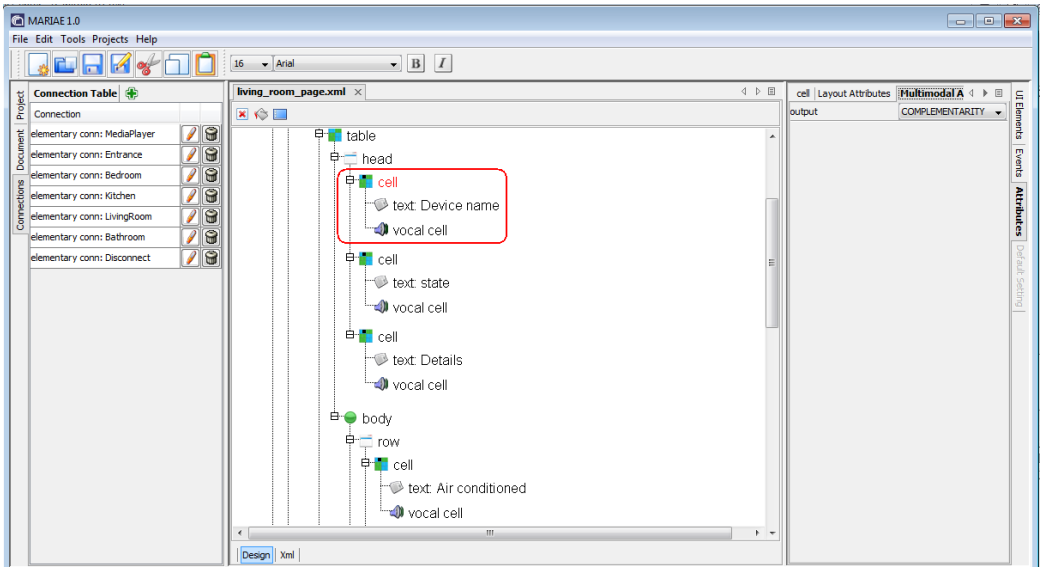

Figura 8.7: MARIAE: definizione di una tabella

Nella figura 8.7 viene mostrata la definizione della tabella nel designer di MARIAE, come si nota è presente la definizione grafica e vocale dell'elemento *head* e dell'elemento *body*, di conseguenza la tabella verrà sintetizzata con la modalità *intelligent browsing* (sez. 6.2.5.7).

Presentiamo un semplice caso d'uso relativo alla presentation di gestione della stanza nel quale viene mostrato come viene sintetizzata la tabella:

```
C: Livingroom page
C: Monitored devices

//INTELLIGENT BROWSING
C: Device name: Air conditioned
C: State: The device is on
C: Details: Temperature 20 degrees
//2nd ROW
C: Device name: Media Player
C: State: The device is on
C: Details: Playing Album Litfiba 3

C: Select the device to monitor
U: Media Player
C: You choose the device media player

C: Say submit to submit the form
U: Submit
C: You have submit the form
```

Il risultato della trasformazione è visibile nella figura 8.8.

Figura 8.8: Gestione della stanza

L'ultimo esempio riguarda la presentation responsabile della gestione dei dispositivi relativo al caso del dispositivo media player (fig. 8.9).

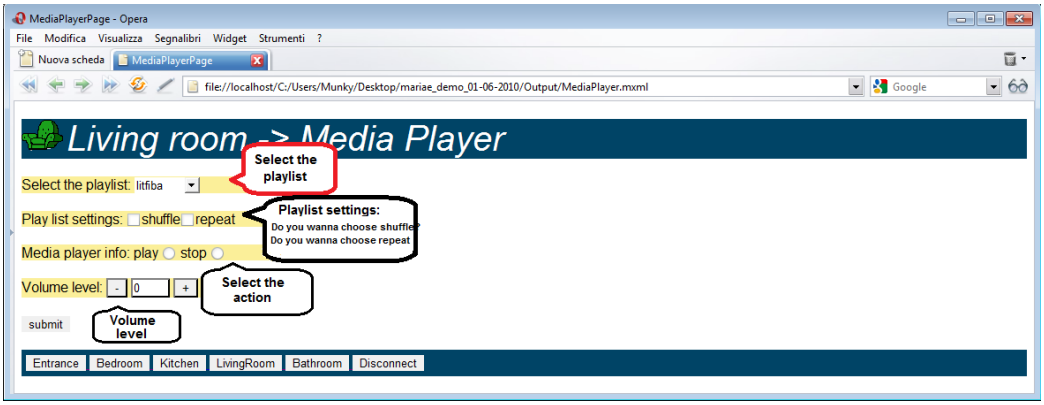

Figura 8.9: Gestione del dispositivo

Vediamo un semplice caso d'uso che mostra come vengono trasformati gli elementi *multiple choice*:

```
C: Media player settings

C: Select the playlist
U: Litfiba
C: You selected the playlist litfiba

// MULTIPLE CHOICE
C: Playlist settings:
C: Do you wanna choose shuffle?
```

```
U: Yes
C: Do you wanna choose repeat?
U: No

C: Select the action
U: Play

C: Volume level
U: 50

C: Say submit to submit the form
U: Submit
```

Per ultimo viene mostrata la presentation di gestione della stanza living room relativa alla piattaforma mobile. Come detto in precedenza la piattaforma mobile possiede ridotte risorse grafiche anche a causa del fatto che gli schermi sono di dimensioni inferiori; per questo motivo si incoraggia gli sviluppatori a utilizzare la sintetizzazione vocale maggiormente rispetto a quanto avviene alla piattaforma desktop. Nel esempio della presentation di gestione di una stanza visto in precedenza per la piattaforma desktop, vi era una tabella con i dispositivi presenti nella stanza accanto a un elemento selection per la selezione del device da monitorare; la tabella è un elemento che può occupare molto spazio nello schermo, per cui nella versione mobile della presentation è possibile dare una definizione della tabella solo vocale in modo da risparmiare le risorse grafiche (fig.8.10).
Il risultato della trasformazione è visibile nella figura 8.11; presentando la tabella solo vocalmente si riesce a far entrare la presentation in un'unica schermata evitando lo scrolling orizzontale e rendendo la pagina maggiormente fruibile per i dispositivi mobile.

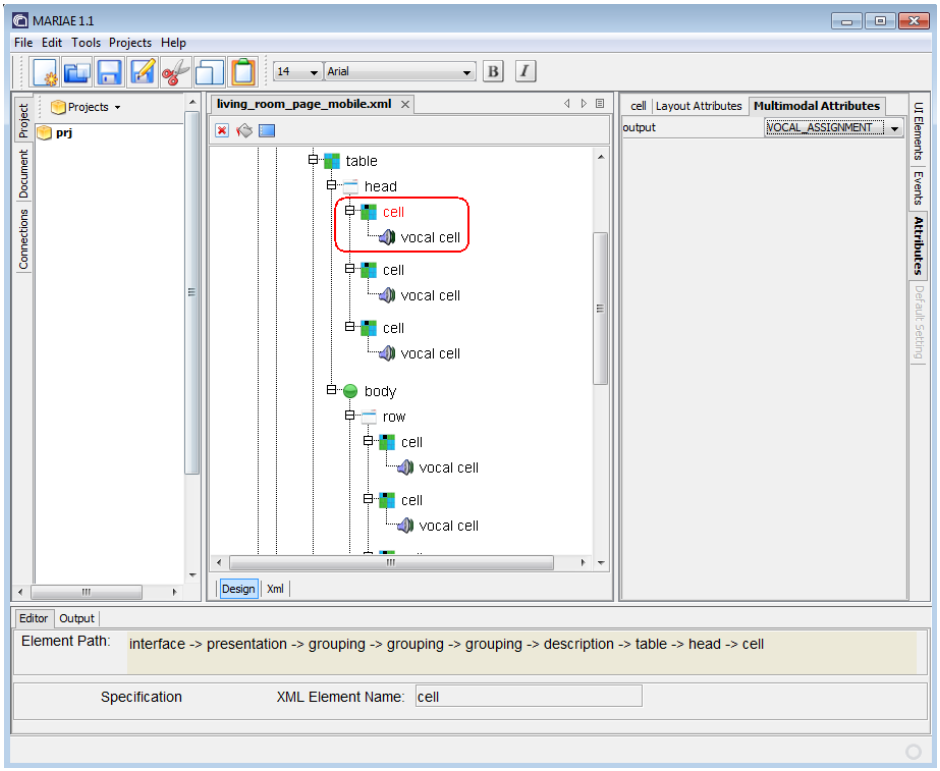

Figura 8.10: MARIAE: definizione di una tabella per la piattaforma mobile

Figura 8.11: Presentation living room per piattaforma mobile

Capitolo 9

Conclusioni

Nel corso della tesi è stata mostrata la progettazione di un linguaggio logico, basato su XML, che permette la descrizione di interfacce multimodali composte da grafica e voce; a partire da un linguaggio che descrive in maniera astratta un interfaccia (AUI) il linguaggio da me sviluppato ne eredita le caratteristiche e lo espande fornendo il supporto per le interfacce multimodali. Attraverso le proprietà CARE vengono descritte le modalità di interazione tra la parte vocale e quella grafica; queste proprietà hanno dei valori standard, ma l'utente può decidere di selezionare valori non considerati in partenza e utilizzare così differenti modi di interazione. In seguito viene esposto un metodo, basato su template XSLT, per trasformare la descrizione logica dell'interfaccia in un'interfaccia multimodale reale. Il codice prodotto dalla trasformazione è XHTML+Voice ed è essenzialmente basato su due standard del W3C: XHTML e VoiceXML.
Il linguaggio è stato integrato all'interno del tool MARIAE permettendo di ottenere la descrizione logica attraverso il semplice drag and drop degli elementi; MARIAE permette di applicare la trasformazione all'interfaccia appena creata in maniera semplice e diretta.

Gli sviluppi futuri riguarderanno:

- Test di usabilità per valutare come il processo di sviluppo è facilitato con l'approccio scelto;

- Implementazione di una trasformazione che supporti l'adattamento da interfaccia grafica desktop a interfaccia multimodale;
- Estendere il linguaggio e l'ambiente per fornire un supporto a modalità differenti;
- Sviluppo di un processo automatico che da un file o un database riesca a costruire una grammatica sollevando lo sviluppatore dalla definizione manuale delle grammatiche;
- Supporto a lingue diverse dall'inglese;
- Definizione degli attributi grafici all'interno di un foglio di stile esterno e non nell'attributo *style* come avviene ora, in modo da separare i contenuti dalla formattazione.

Bibliografia

[1] Service annotations for user interface composition. http://www.servface.eu/.

[2] Openinterface framework. http://www.oi-project.org/.

[3] J. Axelsson, C. Cross, H.W. Lie, G. McCobb, T.V. Raman, and L.Wilson. XHTML+Voice Profile 1.0. Recommendation, World Wide Web Consortium (W3C), 2001. See http://www.w3.org/TR/xhtml+voice/.

[4] Anders Berglund, Scott Boag, Donald D. Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. Xml path language (xpath) 2.0. World Wide Web Consortium, Recommendation REC-xpath20-20070123, January 2007.

[5] Paul V. Biron and Ashok Malhotra, editors. *XML Schema Part 2: Datatypes*. W3C Recommendation. W3C, second edition, October 2004.

[6] Richard A. Bolt. "put-that-there": Voice and gesture at the graphics interface. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 262–270, New York, NY, USA, 1980. ACM.

[7] T. Bray, J. Paoli, C. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). Recommendation, World Wide Web Consortium (W3C), 2008. See http://www.w3.org/TR/2008/REC-xml-20081126.

[8] Daniel C. Burnett, Mark R. Walker, and Andrew Hunt Scansoft. Speech synthesis markup language (ssml) version 1.0, September 2004.

[9] P. R. Cohen, M. Johnston, D. McGee, S. L. Oviatt, J. Clow, and I. Smith. The efficiency of multimodal interaction: A case study, 1998.

[10] Philip R. Cohen. The role of natural language in a multimodal interface. In *UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology*, pages 143–149, New York, NY, USA, 1992. ACM.

[11] Joelle Coutaz, Laurence Nigay, D. Salber, A. Blandford, J. May, and R. M. Young. Four easy pieces for assessing the usability of multimodal interaction: The CARE properties. In *Proceedings of INTERACT95*, pages 115–120, Lillehammer, June 1995.

[12] Ort Ed and Mehta Bhakti. Java architecture for xml binding (jaxb), March 2003.

[13] David C. Fallside and Priscilla Walmsley. Xml schema part 0: Primer second edition. W3C Recommendation, October 2004.

[14] Mikko Honkala and Mikko Pohja. Multimodal interaction with xforms. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 201–208, New York, NY, USA, 2006. ACM.

[15] James Clark. Xsl transformations (xslt) version 1.0. Technical report, W3C, 1999.

[16] Kouichi Katsurada, Yusaku Nakamura, Hirobumi Yamada, and Tsuneo Nitta. Xisl: a language for describing multimodal interaction scenarios. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces*, pages 281–284, New York, NY, USA, 2003. ACM.

[17] James A. Larson, T. V. Raman, Dave Raggett, Michael Bodell, Michael Johnston, Sunil Kumar, Stephen Potter, and Keith Waters. W3C

multimodal interaction framework. W3C Note, may 2003. `http://www.w3.org/TR/mmi-framework/`.

[18] James Lin and James A. Landay. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1313–1322, New York, NY, USA, 2008. ACM.

[19] Stéphane H. Maes and Vijay Saraswat. Multimodal interaction requirements. World Wide Web Consortium, Note NOTE-mmi-reqs-20030108, January 2003.

[20] G. L. Martin. The utility of speech input in user-computer interfaces. *Int. J. Man-Mach. Stud.*, 30(4):355–375, 1989.

[21] Zeljko Obrenovic, Dusan Starcevic, and Bran Selic. A model-driven approach to content repurposing. *IEEE MultiMedia*, 11:62–71, 2004.

[22] Sharon Oviatt. Ten myths of multimodal interaction. *Commun. ACM*, 42(11):74–81, 1999.

[23] Fabio Paternò. Teresa. `http://giove.cnuce.cnr.it/tools/TERESA/`.

[24] Fabio Paternò and Federico Giammarino. Authoring interfaces with combined use of graphics and voice for both stationary and mobile devices. In *AVI '06: Proceedings of the working conference on Advanced visual interfaces*, pages 329–335, New York, NY, USA, 2006. ACM.

[25] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In *INTERACT '97: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, pages 362–369, London, UK, UK, 1997. Chapman & Hall, Ltd.

[26] Fabio Paternò and Carmen Santoro. One model, many interfaces. In *CADUI*, pages 143–154, 2002.

[27] Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16(4):1–30, 2009.

[28] Ben Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[29] Michael Sperberg-McQueen and Henry S. Thompson. Xml schema 1.1 status. `http://www.w3.org/XML/Schema`.

[30] Henry Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. Recommendation, World Wide Web Consortium, May 2001. See `http://www.w3.org/TR/xmlschema-1/`.

Acronimi

AJAX Asynchronous Javascript And XML

ASR Automatic speech recognition

AUI Abstract User Interface

CARE Complementarity, Assignment, Redundancy, Equivalence

CNR Consiglio Nazionale della Ricerca

CSS Cascading Style Sheet

CTT ConcurTaskTrees

CUI Concrete User Interface

DTD Document Type Definition

DTMF Dual Tone Multi-frequency

FIA Form Interpretation Algorithm

GUI Graphical User Interface

HCI Human Coumputer Interaction

HIIS Human Interfaces in Information Systems

HTML HyperText Markup Language

ISTI Istituto di Scienza e Tecnologie dell'Informazione

JAXB Java Architeture for XML Binding

JSGF Java Speech Grammar Format

MARIA Model b-Ased descRiption of Interactive Application

MARIAE MARIA authoring Environment

MMUI Multimodal User Interface

OICDL OpenInterface Component Description Language

OIDE Openinterface Interaction Development Environment

PDA Personal Digital Assistant

SERVFACE Service Annotations for User Interface Composition

SFE Service Front End

SGML Standard Generalized Markup Language

SMIL Synchronized Multimedia Integration Language

SOA Service Oriented Architeture

SSML Speech Synthesis Markup Language

SVG Scalable Vector Graphics

UI User Interface

WIMP Windows Icons Menus Pointers

WSDL Web Services Description Language

XHTML eXtensible HyperText Markup Language

XISL eXtensible Interaction Scenario Language

XML Extensible Markup Language

XSD XML Schema Definition

XSLT eXtensible Stylesheet Language Transformations