Università degli Studi di Milano
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica e Comunicazione

Corso di laurea triennale in
Scienze e Tecnologie della Comunicazione Musicale

Anno Accademico 2008/2009

Implementazione di un prototipo per il montaggio video on-line

Elaborato finale di
Jacopo **SALVESTRINI**
matricola 679276

Relatore Luca Andrea **Ludovico**
Correlatore Adriano **Baratè**

Contenuti

I. Contesto

Il seguente elaborato illustra il funzionamento di un prototipo per il montaggio on-line di video, coinvolgendo anche altri contenuti multimediali, quali immagini e audio. Si affronteranno le diverse problematiche comportate sia dalla natura degli stessi contenuti, sia dal tipo di tecnologie e dal tipo di fruizione e utilizzo di questi.

1. Preambolo

Quando si parla di web spesso si deve considerare la sua natura distribuita; perlomeno due entità entrano in gioco in modo fondamentale, una, nascosta agli occhi dell'utente finale, il *server*, l'altra, invece ben visibile, il *client*. È da un decennio ormai che si vede come queste due entità si stiano sempre più separando in modo netto, e arricchendo, ognuna per proprio conto; il client fornisce sempre più azione e interazione immediata con l'utente, comprendendo anche effetti (grafici per lo più) che a volte disturbano la navigazione, oppure che invece la allietano, o ancora che ne rendono i suoi contenuti più accattivanti; il server invece, oltre che a rispondere, secondo "tradizione", direttamente alle richieste del client, fornendo una pagina statica, o dinamica, ed eventualmente interagendo con un database, può essere anche interrogato per un servizio specifico che va al di là della semplice pagina web: il *web service*.

Questo nuovo utilizzo della rete web prende le forme, ormai mature, del fenomeno del cosiddetto *Web 2.0*. Quindi il client da una parte e il web service dall'altra; il primo che interagisce con applicazioni sempre più simili, se non a volte anche più intuitive e accattivanti, a quelle desktop, il secondo che, di nascosto agli occhi dell'utente, serve da remoto le applicazioni; queste ultime sono perciò distribuite, da una parte proprio vicino all'utente e sulla stessa sua macchina, dall'altra nella nuvola di Internet, probabilmente dall'altro capo del mondo. Il web service e il client comunicano in maniera *asincrona*, proprio per dare impressione all'utente che queste due parti non siano distanti.

Client, web service, Web 2.0, asincronia, queste le parole chiave legate al prototipo che si andrà a illustrare; ma come possono servire a un montaggio video?

Un primo abbozzo di risposta viene dato nel prossimo paragrafo, in cui si inizia a configurare il motivo che ha spinto alla programmazione di questo prototipo, per poi addentrarsi e vedere sempre più da vicino la sua costituzione e il suo rapporto con le nuove tecnologie.

2. Un festival d'autunno

Dal 18 ottobre all'8 novembre 2009, nei quattro palcoscenici del Teatro Stabile di Torino, si svolse un festival teatrale intitolato "Prospettiva 09". Per l'occasione, teatro, danza, musica, arte e politiche si intrecciarono per dar voce a cinquanta spettacoli innovativi con talenti emergenti (tra cui Abattoir Fermé e Rafael Spregelburd) insieme ad artisti affermati (come Ascanio Celestini, Tim Etchells, Jan Fabre, René Pollesch, Armando Punzo), provenienti da sette differenti Paesi (Argentina, Belgio, Francia, Germania, Gran Bretagna, Italia, Stati Uniti)

[Prospettiva 2009].

Gli spettacoli del festival sono stati registrati, ripresi da video camere e immortalati da fotografi; per ognuno di essi ci sono dunque dei materiali multimediali, alcuni dei quali ("Paranoia", "I pescecani", "Concerto senza titolo", "Short ride in a fast machine" e "Too late!") sono stati forniti dagli organizzatori degli eventi al Laboratorio di Informatica Musicale (LIM), del Dipartimento di Informatica e Comunicazione (DI*Co*), affinché si realizzasse un sito web per renderli disponibili pubblicamente in rete.

3. Un sito web

Il sito web, introdotto nel paragrafo precedente, sarà strutturato in tre parti, ciascuna focalizzata su un differente modo di fruizione/manipolazione dei contenuti multimediali del festival. In particolare saranno a disposizione dell'utente dei contenuti video, con diversi punti di ripresa, videoclip e interviste agli artisti; tracce audio delle recitazioni e delle musiche; fotografie di scena; testi dei copioni in lingua originale e tradotti in italiano; testi e spartiti delle musiche.

C'è dunque una varietà abbondante di tipi di contenuti multimediali che richiede di conseguenza una varietà di fruizione degli stessi, che possono essere messi in relazione in vari modi. Ogni parte del sito web sarà dedicata alla fruizione diretta di questi contenuti, come entità a sé stanti, alla fruizione in cui questi sono messi in relazione sincronicamente, e infine alla manipolazione libera di questi da parte dell'utente. Nei prossimi paragrafi si affrontano le tre parti.

3.1. Fruizione diretta dei contenuti multimediali

In questa sezione del sito, denominata "*Enjoy*", l'utente ha la possibilità di accedere a tutti i contenuti multimediali degli spettacoli, ognuno classificato secondo lo spettacolo di provenienza e il formato, in maniera indipendente l'uno dall'altro.

3.2. Fruizione sincrona dei contenuti multimediali

In quest'altra parte, chiamata "*Interact*" è possibile interagire con tutti quei contenuti che sono legati tra loto da un certo istante di tempo, o evento, venendo così fruiti in maniera sincrona.

Per esempio per lo spettacolo "Paranoia" sono disponibili tre punti di ripresa (uno da una videocamera frontale e due da una laterale), una traccia audio dello spettacolo, un testo del copione di scena in lingua originale (spagnolo) e uno tradotto in italiano, e quattro fotografie di scena. Chiaramente ognuna delle tre riprese ha una propria inquadratura, ma è anche legata all'altra sincronicamente, ossia riprende il medesimo evento, e condivide una stessa traccia audio; parimenti il testo del copione corrisponde sincronicamente alla medesima traccia audio. Questi contenuti sono stati disposti in maniera tale che mentre l'utente visiona un punto di ripresa e ascolta le battute degli attori, può anche leggerle, scegliendo tra la lingua originale e la traduzione; può inoltre decidere di cambiare inquadratura, per esempio per meglio vedere una certa espressione dell'attore, oppure scegliere di vederne la relativa fotografia, se disponibile.

Anche qui i contenuti, sempre per agevolarne la fruizione, sono stati classificati per spettacolo e quindi per formato multimediale.

3.3. Manipolazione dei contenuti multimediali

Finalmente si giunge all'ultima parte del sito, chiamata "*Create*", e dedicata al prototipo in questione, che ha la scopo di permettere all'utente di montare uno o più video utilizzando i materiali multimediali appena descritti, a esclusione dei testi. Questa parte si presenta all'utente in tre sezioni; una lista dei materiali sulla destra, un riproduttore multimediale sulla sinistra e i comandi di manipolazione in alto.

La prima è a sua volta suddivisa in due sezioni (ognuna rappresenta una *Playlist*), di cui una con i contenuti originali predefiniti, l'altra con i file creati dall'utente in seguito a una manipolazione di un contenuto originale. Ognuna di esse è suddivisa in tre categorie, ciascuna dedicata ai contenuti audio, a quelli video (sequenze di immagini in movimento con o senza canale audio), e a immagini statiche; per ognuno di questi contenuti si presentano diverse possibilità di manipolazione.

Il riproduttore multimediale permette all'utente di controllarne la riproduzione con i comandi appositi di avvio alla riproduzione (*play*), pausa di questa (*pause*), di scegliere un punto temporale preciso (con un cursore – *slider*) da cui iniziare la riproduzione e/o una selezione, dove, in questo caso, l'utente sceglierà anche un secondo punto temporale di fine selezione; quest'ultima funzionalità per poter effettuare le manipolazioni descritte appena di seguito.

Sia per un file audio che per uno video è possibile regolare il volume, tagliare una porzione temporale creando un nuovo file da questa, oppure un nuovo file con questa esclusa, oppure sovrapporre un altro contenuto multimediale (*overlay*); per un video è inoltre possibile regolarne le dimensioni. Su un file audio si può sovrapporre solamente un altro file audio; su un video, sia un file audio, sia un altro video, sia un'immagine; le immagini possono essere usate solo come livello superiore a una sovrapposizione, non come base di questa. Per una sovrapposizione è possibile selezionare una porzione temporale del file multimediale usato come base e quindi, per il livello sovrapposto, regolare il volume dell'audio, con eventuali effetti di sfumatura (*fading* – per un video/immagine sarà visuale), sia iniziale che finale; per una sovrapposizione video/immagine (*visual overlay*) è possibile regolare anche l'opacità, la posizione all'interno del video base (con cinque opzioni: in alto a sinistra o a destra, in centro, oppure in basso a sinistra o a destra), infine è possibile ridimensionare relativamente alla grandezza del video base, scegliendo se avere come riferimento l'asse orizzontale oppure quella verticale di quest'ultimo (si rimanda al §III. 1.1.3. per i dettagli).

II. Tecnologie

Per realizzare il prototipo sono state utilizzate varie tecnologie web, appartenenti all'ambiente di sviluppo *Microsoft .NET Framework*, nella versione 3.5. Si è creata una soluzione con *Microsoft Visual Studio 2008* costituita da due progetti; un *Web Service* e un'applicazione *Silverlight*. Però, prima di presentare queste due tecnologie, si vuole proporre una breve digressione storica proprio per introdurne la seconda, in particolar modo.

1. Il contorno storico di Silverlight

In quest'era del Web 2.0, non si può non accennare a *Flash* della Adobe. Qualsiasi navigatore web si sarà senz'altro imbattuto in qualche applicazione Flash, sia in forma di sito completo, di animazione 2D, di avviso pubblicitario e sia di lettore multimediale; *YouTube* ne è un esempio sotto gli occhi di tutti. È da più di dieci anni ormai che Flash si diffonde sempre di più sulle scene web; questa tecnologia fu inizialmente sviluppata da Macromedia (1996) e in seguito acquisita da Adobe (2006); parallelamente, per il web, Microsoft realizzava *ASP* (Active Server Pages), fino al 2001, e l'anno dopo introdusse *ASP.NET* (tutt'ora in uso). Quest'ultima tecnologia (come si può evincere dal suffisso) è parte del Framework .NET, ossia di un'astrazione dove del codice comune fornisce funzionalità generiche e predefinite che possono essere estendibili per usi specifici; parte centrale del framework è il *CLI* (Common Language Infrastucture), che permette di programmare in linguaggi di programmazione ad alto livello, indipendentemente dall'architettura specifica di una macchina (della quale se ne occupa appunto il CLI). Dal momento che Flash, sin dalla sua prima comparsa, incontrava crescente e dirompente diffusione sia nel web, per la sua unicità e immediatezza accattivante interazione con l'utente, sia, di ovvia conseguenza, tra gli sviluppatori, naturalmente fu adottato anche da coloro che utilizzavano tecnologie Microsoft, in particolare ASP.

Con l'avvento del Framework .NET e di ASP.NET, questa branca di programmatori si trovava di fronte a sviluppare i siti web con notevoli agevolazioni e potenziati da una infrastruttura molto versatile che coinvolgeva numerose altre tecnologie, a eccezione di Flash, per il quale furono implementate (tuttora usate) delle librerie di compatibilità col framework. Tuttavia, già, per esempio, solo l'integrazione con Visual Studio (il principale strumento usato per sviluppare nel framework), era una soluzione accettabile, ma non delle migliori, dal momento che un'applicazione Flash richiede un linguaggio di scripting, ActionScript, esterno al framework (cfr. 1.2) [Piller 2010]. Dunque, che cosa avrebbe comportato introdurre nel Framework .NET una tecnologia simile a Flash? Il mondo di Silverlight ne è la risposta, ossia un incontro tra ASP.NET e Flash.

1.1. ASP.NET

ASP.NET, successore di ASP, è un framework web volto allo sviluppo di siti web dinamici, applicazioni web e servizi web, raggruppandosi a sua volta nel Framework .NET. Le pagine web dinamiche sono pagine web che presentano al client un diverso contenuto a seconda

delle sue richieste. Perciò, il client, attraverso il protocollo *HTTP* (solitamente con un metodo *GET* o *POST*), chiede a un server un certo file – p.e. una pagina web dinamica – il server (*Microsoft IIS*, in questo caso) accoglie la richiesta e cerca tra i moduli che ha installati quale corrisponde quello per un file con una determinata estensione (.aspx, in questo caso), trova il modulo che punta alle librerie ASP.NET, queste vengono eseguite per il file richiesto, che lo elaborano a seconda dei parametri di richiesta del client (in un GET, le variabili dopo il carattere "?" di un *URL*; in un POST le variabili presenti nel corpo di una richiesta HTTP; oppure secondo delle variabili *cookie*) o qualsiasi altra informazione legata alla richiesta che il server può dare al motore ASP.NET; questo può decidere di stabilire una connessione con una base dati (o con un file sul *file system*), per leggerne o modificarne le informazioni, e quindi costruire il contenuto della pagina web da restituire al server, il quale la include nel corpo della risposta HTTP.

Si è appena visto un possibile e molto comune scenario di interazione tra client e server web, con supporto per pagine web dinamiche ASP.NET, ma la caratteristica principale di questo framework web è la suddivisione delle pagine web in una parte grafica (file con estensione .aspx) e una parte dedicata al codice (lo stesso nome del file con estensione .cs.aspx o .vb.aspx), chiamata anche *code-behind* (o codice sottostante – alla pagina web). Il file dell'interfaccia grafica si presenta come un documento XML contenente codice *(X)HTML – (eXtensible) HyperText Markup Language* – interfogliato da codice ASP.NET, costituito a sua volta da dichiarazioni per definire la pagina (all'interno dei caratteri `<%@` e `%>`) e per rappresentare particolari controlli che interagiscono con il server e quindi col code-behind (tag con prefisso namespace `asp:`). Il file del code-behind contiene codice scritto in uno dei qualsiasi linguaggi .NET ad alto livello; i più diffusi sono C# (estensione .cs.aspx) e Visual Basic .NET (estensione .vb.aspx); caratteristica di questi è che sono utilizzati anche per sviluppare applicazioni desktop, per cui il programmatore può progettare sia applicazioni web sia applicazioni desktop senza bisogno di cambiare il suo linguaggio preferito.

Il code-behind è infine compilato in una o più DLL, per cui nella cartella del sito web sul server di rilascio è necessario mettere solo le pagine .aspx e le librerie DLL; il codice sorgente viene invece conservato in ambiente di sviluppo. Solitamente negli altri linguaggi di scripting web (compreso il precedente ASP) la compilazione avviene nel tempo stesso in cui la pagina web è richiesta dal client; pre-compilando quindi si risparmia tempo sulla compilazione, per cui il server dovrebbe rispondere più velocemente.

1.2. Adobe Flash

Flash è una piattaforma multimediale volta ad arricchire le pagine web con animazioni e interattività. Dal 1996, anno della sua prima comparsa, a oggi ha avuto notevoli cambiamenti. Inizialmente era una semplice applicazione di *authoring* volta a gestire la grafica vettoriale con la possibilità di animarla; col passare del tempo si vide come un linguaggio di scripting risultasse sempre più utile per velocizzare la creazione di animazioni, per cui si decise di dare sempre più importanza a questo aspetto, finché, con la versione 5 di Flash (2000), si realizzò una fase matura di questo linguaggio, ActionScript 1.0, con sintassi molto simile a JavaScript. Oggi ActionScript, ormai giunto alla versione 3.0, è un linguaggio più che maturo, orientato ad oggetti, con supporto di ereditarietà e interfacce, sistema di gestione degli eventi, supporto di *package* e *namespace* (per meglio organizzare le classi), supporto per le espressioni regola-

ri e per XML. Con ActionScript, e volendo anche – o solo – con l'applicazione di authoring della Adobe (Flash CS), si crea il proprio sito web, animazione, etc. e si compila il tutto in un file (con estensione .swf) che viene poi inserito in una pagina (X)HTML (codice sottostante), per essere fruito nel web.

```
<object data="/cartella/animazione.swf" type="application/x-shockwave-flash"
 width="640" height="480">
    <param name="movie" value="/cartella/animazione.swf" />
</object>
```

La pagina web viene caricata, durante la navigazione, da un *browser*, il quale vede che è presente un collegamento a un oggetto esterno e, se ne riconosce il contenuto, scarica il file e lo fa visualizzare nella pagina, altrimenti lo ignora. Perché il browser riconosca l'oggetto Flash, è necessario che sia installato un *plug-in*, il cosiddetto *Flash Player*. Per navigare in Internet si può scegliere tra numerosi browser, i più diffusi sono *Microsoft Internet Explorer* (mondo *Windows*), *Mozilla Firefox* (multi-piattaforma – *Linux*, *Mac OS X* e altri sistemi *Unix*, Windows), *Google Chrome* (multi-piattaforma), *Apple Safari* (Mac OS X e Windows) e *Opera* (multi-piattaforma); il Flash Player è distribuito dalla Adobe (esistono anche altre varianti indipendenti e *Open Source*, come *Gnash* e *Swfdec*, del mondo Linux, ma ancora incomplete), con una versione dedicata per ogni sistema operativo (dal momento che il player è a tutti gli effetti una macchina virtuale per ActionScript), Linux (di cui è disponibile, sebbene ancora in fase *alfa*, una versione a 64-bit) Mac OS X e Windows. Questo significa che, una volta ottenuto un browser compatibile con il player (quelli citati lo sono pienamente) e il player stesso, si può interagire con un oggetto Flash in una pagina web indipendentemente dal tipo di piattaforma.

Parallelamente a Flash, sin dal 2004, Macromedia, e in seguito Adobe, sviluppava un *Software Development Kit* (*SDK*) per costruire *Rich Internet Applications (RIA),* ossia applicazioni web aventi molte delle funzionalità di quelle desktop, basato sulla stessa piattaforma di Flash, dal nome *Flex*. L'SDK è disponibile gratuitamente presso il sito web dell'attuale produttore, con il quale è possibile compilare *MXML*, linguaggio conforme allo standard XML e usato per le parte grafica, e ActionScript, per la parte di codice logica lato client, producendo un file Flash SWF. Con Flex, oltre ad avere una ricca gamma di controlli preimpostati, un sistema di validazione dell'input dell'utente, funzionalità *drag & drop*, ed effetti vari, è possibile effettuare chiamate asincrone a servizi web – anche a quelli sviluppati in ASP.NET.

Si è visto dunque come Flash si colloca a tutti gli effetti nel mondo del Web 2.0, e come il paradigma di programmazione divida la parte grafica da quella logica di controllo, adottando linguaggi specifici [Wikipedia 2009].

2. Silverlight

Silverlight è un framework per applicazioni web che si colloca a sua volta all'interno del Framework .NET. Più si vede in superficie e più si può accomunare con Flash, perché infatti offre pressoché le stesse funzionalità (animazioni, streaming multimediale, gestioni eventi per l'interattività con l'utente, RIA, etc.).

Più superficialmente, dunque, così come si presenta direttamente all'utente, è un plug-in per il browser (Internet Explorer, Firefox e Safari, su sistemi Windows o Mac OS X – per gli altri

sistemi UNIX c'è un progetto Open Source chiamato *Moonlight*, che offre un plug-in per Firefox ed è sviluppato all'interno del progetto *Mono* della *Novell*, che ha l'obiettivo di rendere portabile su sistemi UNIX, Linux e Mac OS X principalmente, il Framework .NET e i suoi componenti, adattandosi ai sistemi ospitanti, per esempio per quanto riguarda le loro interfacce grafiche molto differenti tra loro nell'architettura) scaricabile liberamente presso il sito apposito della Microsoft (o della Novell, per UNIX); è anche un oggetto annidato nel codice (X)HTML di una pagina web (esempio sottostante).

```
<object data="data:application/x-silverlight-2," type="application/x-silverlight-2"
 width="100%" height="100%">
    <param name="source" value="/cartella/applicazione.xap"/>
    <param name="onError" value="onSilverlightError" />
    <!-- Altri possibili parametri, come la versione minima richiesta, o un link per
         scaricare il plug-in, se non installato -->
</object>
```

Come per Flash, se il plug-in è correttamente installato, allora il browser scarica l'oggetto e lo fa eseguire al plug-in; in caso di errore (tag `param` con attributo `name="onError"`) il plug-in comunica al browser di eseguire una funzione JavaScript, solitamente preimpostata (`onSilverlightError`). L'oggetto Silverlight è un file ZIP compresso con estensione .xap, al cui interno sono presenti un file XAML, delle DLL ed eventuali file XML di configurazione. Il primo (AppManifest.xaml) contiene alcuni parametri di configurazione base che puntano alle DLL e specifica quale di queste è quella di entrata principale; dunque l'applicazione Silverlight in sé è una libreria eseguita dalla macchina virtuale del plug-in che, a seconda dei casi, si può servire di informazioni esterne di configurazione, presenti in file XML.

Andando più in profondità, nell'architettura del framework, si possono vedere due ambienti principali, uno per gestire la parte di interfaccia e l'altro per la gestione della logica. Il primo è il cosiddetto nucleo di rappresentazione (*presentation core*, a diretto contatto con l'utente), gestito attraverso codice *Extended Application Markup Language* (XAML, conforme allo standard XML) e composto a sua volta da una parte nucleare dedicata all'interfaccia utente (*UI Core*) con supporto di grafica vettoriale e animazione, testo e grafica *raster* (ossia di immagini); da una parte per la gestione di input da tastiera e da mouse, che è arricchito da utilità per disegnare, con differenti tratti, (*Ink*) su una superficie grafica impostata; da una per la gestione di elementi multimediali (tra cui audio: *WMA*, *AAC*, *MP3*; video: *WMV*, *VC-1, H.264*) e di eventuali diritti associati (*Digital Right Management*, o *DRM*); infine da una tecnologia per l'ingrandimento ad alta qualità di immagini (*Deep Zoom*, da due o tre megapixel a dimensioni di gigapixel).

Il secondo ambiente controlla il primo attraverso la logica del "codice gestito" (*managed code*), ossia quel codice gestito dal motore di esecuzione del *Common Language Runtime* (CLR), che, durante l'esecuzione (*runtime*), compila dal linguaggio intermedio (*Common Intermediate Language*, *CIL*, indipendente dal sistema operativo) in linguaggio nativo del sistema operativo. I linguaggi ad alto livello utilizzati sono di due tipi: linguaggi staticamente tipizzati e linguaggi dinamicamente tipizzati. Tra i primi i più utilizzati sono C# e Visual Basic .NET (spesso abbreviato in VB.NET) e sono compilati (prima del runtime) nel CIL; tra gli altri ci sono, fra i più usati, IronPython, Iron Ruby e JScript, che sono invece compilati dal *Dynamic Language Runtime* (*DLR*), una estensione del CLR appositamente pensata.

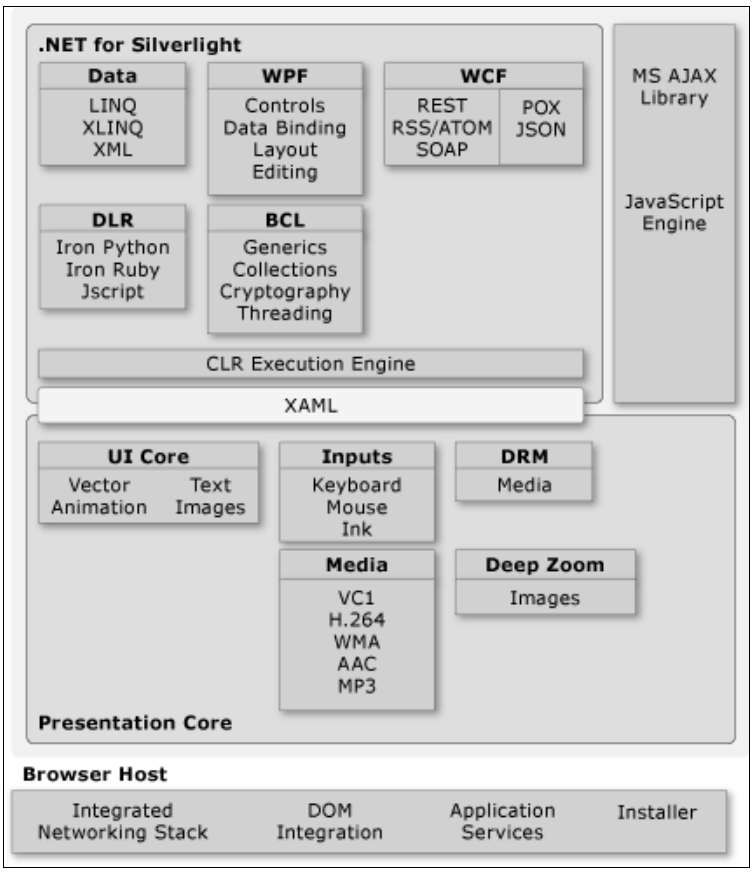

Figura 1. Architettura di Silverlight [MSDN 2010b]

Tutti i linguaggi del Framework .NET possono comunicare tra loro; dispongono di librerie per l'accesso ai dati, memorizzati in formato testo o, più formalmente, in XML (quindi interrogati da LINQ, cfr. §2.3), ma da Silverlight non è possibile accedere a un database (per questo ci vorrebbe un servizio web apposito, per esempio); librerie con funzionalità di base (*BCL*, *Base Class Library*); librerie per interagire con il nucleo di rappresentazione (*WPF*, *Windows Presentation Foundation*); infine vi sono librerie per comunicare con l'esterno (*WCF*, *Windows Communication Foundation*), p.e. con un web service (cfr. §3), per reperire dei dati da un database. Inoltre in Silverlight, ma al di fuori sia delle librerie .NET, sia del nucleo di rappresentazione, si trova un motore JavaScript, con relative librerie *AJAX* (*Asynchronous Javascript And Xml*), al fine di controllare il *DOM* (*Document Object Model*) [Goodman 2008] del documento della pagina web contente l'oggetto Silverlight, attraverso JavaScript, dal CLR; quindi, p.e., utilizzando codice C#.

In Silverlight si dà molta importanza alla gestione degli eventi, soprattutto per quel che riguarda i Controlli del WPF, i cambiamenti di stato dei contenuti multimediali, le animazioni, e le chiamate asincrone del WCF (cfr. §3.3).

Ogni linguaggio ad alto livello permette di rispondere a esigenze diverse; per il prototipo qui presentato è stato utilizzato C#, perché è il più versatile e il più documentato, ed inoltre è utilizzato presso il LIM; questo linguaggio sarà brevemente discusso nel §2.2.

Per quanto detto fin'ora, si può notare come Silverlight sia molto simile ad ASP.NET, in quanto è volto a costruire applicazioni web, si colloca nel Framework .NET, e utilizza un linguaggio CLR per la logica e un linguaggio XML per la rappresentazione dell'interfaccia utente; queste sono le somiglianze, mentre le differenze portano ad avvicinarsi a Flash, in quanto,

come questo, un oggetto Silverlight è eseguito completamente lato client, mentre in ASP.NET si trova ad essere spezzato tra client e server (la parte di logica rimane sul server, mentre la parte di interfaccia utente è sul client, con eventualmente dello script JavaScript/AJAX per le applicazioni web; per questo motivo Silverlight ha accesso in maniera ridotta alle librerie .NET) e perciò può fornire un'esperienza interattiva più ricca. Concludendo, Silverlight si dimostra essere una grande facilitazione per gli sviluppatori .NET, che ora hanno modo, a seconda di come si affermerà questa nuova tecnologia, di abbandonare Flash e la sua integrazione incompleta per abbracciare un'integrazione totale nel Framework .NET, dove, p.e., è possibile costruire un ambiente web in tre parti funzionali: un sito web dinamico ASP.NET, una o più applicazioni Silverlight, un Web Service che può comunicare sia con il sito, sia con l'applicazione; il tutto rimanendo all'interno del medesimo Framework e dei rispettivi strumenti di sviluppo (come Visual Studio .NET). Di seguito si continua approfondendo gli aspetti principali di questa tecnologia; per un riferimento completo si rimanda a [Bochicchio 2009] e al sito Microsoft dedicato agli sviluppatori [MSDN 2010a].

2.1. XAML

XAML è l'acronimo di eXtensible Application Markup Language, ossia un linguaggio conforme allo standard XML (eXtensible Markup Language, sviluppato dal W3C, consorzio internazionale dedicato agli standard web, dal quale nacque anche HTML e XHTML) introdotto da Microsoft nel 2003, in origine per rappresentare formalmente l'interfaccia utente del sistema Windows, poi esteso a supporto di tutto il Framework .NET, in alto al nucleo di rappresentazione e delle sue componenti (cfr. § sopra).

Il codice XAML può essere composto con qualsiasi programma per testo semplice, oppure con supporto di colorazione del codice (con sintassi XML), o con un IDE (Interactive Development Environment). Microsoft fornisce Visual Studio .NET per gli sviluppatori programmatori e Expression Blend per gli sviluppatori grafici; il primo facilita primariamente la creazione e modifica di codice sorgente (sia codice gestito che codice XAML), il secondo invece opera principalmente sull'interfaccia utente (i codici sorgenti sono in secondo piano) con numerosi e potenti controlli che agiscono direttamente sul codice XAML, senza che sia necessario che lo sviluppatore ne conosca la natura [Microsoft 2010]. In questa maniera è possibile migliorare la produttività assegnando una parte al programmatore e un'altra al grafico, che lavorano in parallelo. (Per la realizzazione del prototipo sono stati utilizzati entrambi.)

Attraverso markup è possibile dichiarare rapidamente, in un file "leggibile da un essere umano", un albero di oggetti per essere in seguito utilizzato dal codice gestito.

```
<UserControl x:Class="NomeNamespace.ClasseDellaPagina"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
     <!-- [...] -->
</UserControl>
```

Nell'esempio di codice sopra si vede come l'elemento radice, `UserControl`, abbia alcuni attributi di definizione di *namespace*, `xmlns`; il primo, senza suffisso, dichiara il formato XML del nucleo di rappresentazione (presentation core), e il secondo, con suffisso `:x`, dichiara il formato XML di collegamento con il codice gestito (notare che il valore dell'attributo `x:Class` punta a una classe del codice gestito).

2.1.1. Oggetti

Durante la compilazione di un progetto Silverlight, il compilatore XAML genera un file temporaneo con estensione .g.cs (o .g.vb, a seconda del linguaggio di codice gestito utilizzato, in Visual Studio è nella cartella obj/Debug o obj/Release relative alla cartella del progetto) per ogni file di XAML, affinché si crei un collegamento tra la classe del codice gestito e il rispettivo markup di interfaccia utente; in questa maniera si ricava un albero di oggetti dal file XAML, di cui a ogni suo elemento corrisponde una classe appartenente al tipo scelto di codice gestito (C# o VB.NET, p.e.).

2.1.2. Proprietà

Ad ogni attributo di un elemento del documento XAML corrisponde una proprietà della classe relativa a quell'elemento. Nel caso in cui una proprietà sia complessa, ossia di cui del semplice testo non sia sufficiente a descriverla, sono utilizzati ulteriori elemento (o *tag*), figli del tag-oggetto, aventi come prefisso il nome del tag genitore seguito da un punto "."; inoltre, nel caso in cui il tag-oggetto con i tag-proprietà abbia ulteriori oggetti come figli che possono fare riferimento alle proprietà complesse del genitore, questi hanno degli attributi gerarchizzati con il nome del tag genitore seguito da un punto "." come prefisso. Nell'esempio sottostante si considera il controllo `Grid` (griglia/tabella), che ha il compito di organizzare relativamente a se stesso la disposizione di altri controlli scegliendo tra righe (`Row`) e colonne (`Column`); esso possiede proprietà complesse.

```
<UserControl x:Class="NomeNamespace.ClasseDellaPagina"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<Grid>
     <Grid.RowDefinitions>
          <RowDefinition Height=”0.3333*”/>
          <RowDefinition Height=”0.3333*”/>
          <RowDefinition Height=”0.3333*”/>
     </Grid.RowDefinitions>
     <Grid.ColumnDefinitions>
          <ColumnDefinition Width="0.5*"/>
          <ColumnDefinition Width="0.5*"/>
     </Grid.ColumnDefinitions>
     <TextBlock Text="Sono nella 2a riga e 1a colonna" Grid.Row="1" Grid.Column="0"/>
     <TextBlock Text="Sono nella 1a riga e 2a colonna" Grid.Column="1"/>
     <TextBlock Text="Sono nella 3a riga e 2a colonna" Grid.Row="2" Grid.Column="1"/>
</Grid>
</UserControl>
```

In questo esempio il controllo ha due proprietà: `RowDefinitions` e `ColumnDefinitions`, aventi a loro volta rispettivamente tre e due proprietà che dichiarano tre righe e due colonne per la griglia. Ci sono tre altri elementi figli di `Grid`: tre `TextBlock`, che definiscono dei campi con contenuto testuale (definito nell'attributo `Text`), di cui ciascuno occupa una posizione

diversa all'interno della griglia, a seconda di come gli attributi relativi al genitore sono dichiarati (`Grid.Row` e `Grid.Column`: `0` ad indicare la prima riga o colonna, `1` la seconda, `2` la terza, e cosi' via; se questi attributi sono omessi si sottintende `0`).

2.1.3. Eventi

È possibile gestire gli eventi legati ai controlli dichiarati nel codice XAML attraverso attributi che hanno come valore il nome della funzione di riferimento appartenente alla classe dichiarata nell'attributo `x:Class` dell'elemento radice. Gli eventi possono prendere l'input dell'utente attraverso mouse e tastiera, oppure secondo stati di controlli a cui serve gestire uno stato, solitamente legati a una linea temporale (p.e. streaming video, quando questo è in riproduzione oppure in pausa, o in caricamento, etc.; o anche animazioni).

```
<Grid>
     <!-- [...] -->
     <TextBlock Text="Sono nella 2a riga e 1a colonna"
      Click=”cambiaTesto” Grid.Row="1" Grid.Column="0"/>
     <!-- [...] -->
</Grid>
```

L'esempio precedente è per il codice XAML, quello seguente per il codice gestito (C#).

```
namespace  NomeNamespace {
public partial class ClasseDellaPagina : UserControl {
     /* [...] */
     void cambiaTesto(object sender, RoutedEventArgs event) {
          TextBlock tb = sender as TextBlock;
          tb.Text = “Mi hai cliccato!”;
     }
     /* [...] */
}
}
```

Nel codice XAML l'attributo `Click` si riferisce a un evento, in particolare quando l'utente clicca col puntatore del mouse sul controllo; con lo scattare dell'evento viene chiamata la funzione `cambiaTesto`, che ha una forma tipica di quelle funzioni per gestire gli eventi, ossia con due parametri di ingresso: `sender`, un oggetto generico, e `event`, l'evento. Il primo punta all'oggetto che ha fatto scattare l'evento, in questo caso il controllo `TextBlock` (riconvertito dalla parola chiave `as` al tipo originale nella prima riga del codice della funzione), per cui è possibile accedere alle sue proprietà e manipolarle – nell'esempio si cambia semplicemente il contenuto testuale; il secondo accede alle proprietà dell'evento.

2.2. C#

C# è un linguaggio di programmazione orientato agli oggetti, a tipi statici e sicuri. Fa parte dei linguaggi del codice gestito del Framework .NET (insieme a VB.NET, C++.NET, JScript, e altri), per cui si avvale dei servizi forniti dal *Common Language Runtime* (CLR), tra cui gestione della memoria, gestione dei *thread*, trattamento delle eccezioni, *garbage collection*, e sicurezza. È molto simile a C e a C++, dei quali è l'ultimo successore.

Nei prossimi paragrafi vengono illustrate le caratteristiche fondamentali del linguaggio per comprendere il codice del prototipo in questione; si rimanda a [Schildt 2006] per un approfondimento (e a [Horstmann 2008], per vedere la sua relazione con C++) .

2.2.1. Namespace

In questo linguaggio non esistono variabili e funzioni con visibilità globale, per cui tutto deve appartenere a una classe, che solitamente fa parte a sua volta di un *namespace*. Un namespace è un raccoglitore di classi che serve a far sì che queste non vengano confuse quando il numero di classi utilizzate per un'applicazione aumenta; p.e. Posso avere una classe `Cherchio` sia nel namespace `FormeGeometricheDiAnna` che in `FormeGeometricheDiRoberto`, perché è possibile che Anna e Roberto abbiano implementato classi con lo stesso nome e funzionalità differenti, entrambe utili per il progetto di una terza persona, che potrà usare `FormeGeometricheDiAnna.Cerchio` e `FormeGeometricheDiRoberto.Cerchio` senza problemi di ambiguità. Solitamente per un namespace si assegna il nome della compagnia che ne ha programmato le classi, e si innestano altri namespace per i differenti progetti e sotto-progetti. All'interno del codice si può decidere di usare un namespace come riferimento in tre modi: specificando il nome completo (come `NomeCompagnia.Progetto.Sottoprogetto.Classe`), oppure dichiarando all'inizio del file del codice sorgente la parola chiave `using` seguita dal nome del namespace, oppure assegnando un namespace locale nella seguente forma.

```
using NSLocale = NomeCompagnia.Progetto;
namespace NamespaceNuovoOEsteso { }
```

La seconda riga dell'esempio mostra come creare o estendere un namespace; all'interno delle parentesi graffe si aggiungono le classi o ulteriori namespace, i quali possono essere dichiarati/estesi anche nella forma `NamespaceNuovoOEsteso.NamespaceFiglio`.

2.2.2. Common Type System (CTS)

C# ha un sistema unificato per i tipi, chiamato *Common Type System* (*CTS*); un sistema del genere implica che tutti i tipi, primitivi compresi, derivino dalla classe `System.Object` (namespace `System`), avendo così dei metodi comuni (come il `ToString()` per convertire in tipo stringa di testo).

Il CTS si suddivide in tipi valore (*value type*) e tipi riferimento (*reference type*). I primi derivano da `System.ValueType` di cui i più usati sono `int` (intero a 32-bit con segno), `float` (numero a virgola mobile IEEE a 32-bit), `char` (unità di carattere Unicode a 16-bit) `System.DateTime` (identifica un preciso punto nel tempo con precisione di un nanosecondo), `enum` (enumerazioni, ossia raccolta di variabili di tipo `int`) e `struct` (strutture definite dall'utente – ovvero il programmatore, in questo caso; tutti i valori primitivi – numeri e `char` – sono `struct`); hanno sempre un valore preimpostato e possono essere creati e copiati. Al contrario i tipi riferimento non possono essere copiati, dal momento che ogni istanza è diversa dall'altra, pur avendo lo stesso contenuto; alcuni esempi di questi tipi sono `object` (l'ultima classe base per tutte le altre classi C#), `System.String` (una stringa di caratteri Unicode) e `System.Array` (classe base per tutti gli array C#).

2.2.3. Classi

Una classe si definisce con la parola chiave `class` seguita dal nome che le si vuole assegnare, può implementare una o più interfacce (dichiarate con il simbolo "`:`" tra il nome della nuova classe e quello delle interfacce), ma ereditare solo da un'altra classe (dichiarata nello stesso modo dell'interfaccia), e si può decidere di definire un costruttore (definito con lo stesso nome della classe) e un distruttore (molto meno usato, si dichiara come il costruttore, ma con prefisso "~") nel caso in cui si voglia sovrascrivere quelli creati automaticamente durante la compilazione.

Una classe assomiglia a una struttura (`struct`), ma la differenza più netta è che con la prima si crea un tipo riferimento, mentre con la seconda un tipo valore ed è obbligatorio dichiarare un costruttore.

Una classe può avere campi, proprietà, metodi e costanti. I primi sono variabili relative alla classe; le seconde sono tra il campo e il metodo, vengono dichiarate come un campo (tipo seguito dal nome, p.e. `int numeroIntero`) e in aggiunta hanno delle funzioni di lettura (`get`) e di scrittura (`set`) del loro contenuto che operano similmente a un metodo; un metodo invece è una funzione che può avere dei parametri in ingresso (variabili che vanno dichiarate e istanziate solo quando si chiama il metodo) e restituisce un tipo (`void`, nel caso non si voglia far restituire niente) che va dichiarato prima del nome del metodo; una costante è un campo statico il cui valore che deve essere inizializzato una sola volta (non può variare) e nel momento della dichiarazione.

Inoltre classe, campo, proprietà, metodo e costante possono avere dei modificatori di accesso *(access modifiers)*: `public`, `protected`, `private`, `internal` e `protected internal`, che vanno dichiarati prima di `class` oppure prima del tipo e nel caso in cui vengano omessi, si sottintende `private`. Classe, campo, proprietà e metodo possono essere usati staticamente (preceduti da `static`).

Per accedere ai campi, proprietà e metodi della classe all'interno della stessa, si può usare la parola chiave `this` seguita da "`.`" e dal nome dell'oggetto; se invece si desiderasse accedere agli oggetti ereditati si usa `base` al posto di `this`. Per istanziare una classe si utilizza la parola chiave `new`. Di seguito un esempio di un programma per interfaccia a linea di comando (*CLI*) che illustra quanto detto fin'ora.

```
using System;
namespace NuovoProgetto {
    // questo è un commento su una linea
    /* Questo è un commento
       su più
       linee  */
    // di default: English = 0, Italiano = 1 e Francaise = 2
    // ma si possono cambiare (p.e. Se si volessero usare come flag):
    // { English = 1, Italiano = 2, Francaise = 4, Deutschland = 8,
    //   Tedesco = Deutschland, ItalianoETedesco = Italiano | Tedesco }
    public enum LingueSupportate { English, Italiano, Francaise }
    public class A {
        // costante privata, inaccessibile al di fuori di questa classe
```

```
        const int min = 2;
        // campo protetto, accessibile solo da classi che ereditano da questa
        protected string saluto = "Hello world!";
        // metodo pubblico, accessibile dall'esterno (classi ereditanti comprese)
        public void HelloWorld() {
            Console.WriteLine(saluto);
        }
    }
    public class B : A {
        // proprietà pubblica collegata a saluto, della classe genitore A
        public string Saluto {
            get { return saluto; }
            set {
                // ... alcune operazioni ...
                // value è una parola chiave che indica il valore inserito dall'utente
                saluto = value;
            }
        }
        // costruttore
        public B(LingueSupportate linguaScelta) {
            switch (linguaScelta) {
                case LingueSupportate.Italiano:
                    base.saluto = "Ciao a tutti!";
                    break;
                case LingueSupportate.Francaise:
                    base.saluto = "Bonjour tout le monde!";
                    break;
                case LingueSupportate.English:
                default: break;
            }
        }
    }
    class Program {
        // Metodo di inizio esecuzione del programma in C# – come in Java
        // e a differenza di C++ – è  statico.
        static void Main() {
            B b = new B(LingueSupportate.Italiano);
            b.HelloWorld();
            // La console aspetta un'input da tastiera, quindi si chiude e
            // il programma esce
            Console.ReadKey();
        }
    }
}
```

2.2.4. Strutture di controllo

Naturalmente anche C# possiede delle strutture che controllano l'esecuzione del codice secondo condizioni che devono assolutamente restituire un tipo booleano. Queste condizioni e la loro sintassi sono le stesse di C/C++ (le condizioni semplici `if` [… `else if`] […`else`], `switch`, e i cicli `while`, `do` ... `while`, `for`), con l'aggiunta del ciclo `foreach`, derivato da `for`, che opera su qualsiasi oggetto provvisto di *iteratori*; come in C/C++ la parola chiave `break` interrompe un ciclo o un'istruzione `switch`, e `continue` salta all'iterazione successiva di un ciclo.

C# supporta anche la gestione delle eccezioni attraverso la struttura `try` … `catch` […`finally`]. All'interno del blocco di codice (quella parte di codice contenuta tra parentesi graffe, "`{`" e "`}`") di `try` si inserisce quel codice che si ritiene possa essere soggetto a errori di esecuzione del programma, e quindi possa generare eccezioni. Nel blocco di `catch` invece si mette quel codice da eseguire nel caso in cui una eccezione si verifichi; inoltre, subito dopo catch e tra parentesi tonde, si può dichiarare una variabile (oggetto derivante dalla classe `System.Exception`) nella quale vengono memorizzati i metodi e le proprietà relative all'errore generato; è possibile generare eccezioni personalizzate (con `throw new` e il nome della classe da istanziare, derivante da `System.Exception`). Nel blocco di `finally` è possibile inserire quel codice da eseguire indipendentemente dal verificarsi di un'eccezione.

2.2.5. Eventi

C# è anche un linguaggio di programmazione a eventi (*event-driven)*, molto utile per costruire applicazioni interattive, soprattutto per le interfacce grafiche, per gestire l'input dell'utente. Un evento viene fatto scattare in condizioni particolari e raccoglie una (ma anche più di una) classe speciale (`delegate` – delegato, di cui si definisce il tipo restituito, il nome e i parametri in ingresso, ma non il corpo) che punta a una funzione che "ascolta" l'evento (*event listener*) a cui è legata. Di seguito un esempio.

```
using System;
namespace NuovoProgetto {
// il delegato
public delegate void GestoreEventoOperazione(string messaggio);
public class A {
     // evento
     public event GestoreEventoOperazione EsitoOperazione;
     private string campo;
     public string Proprietà {
          set {
               campo = value;
               string s = "Il campo con valore [" + campo + "]";
               // verifica che il campo sia formattato secondo qualche criterio
               if (campo.Length == 8 && campo.StartsWith("UID")) {
                    EsitoOperazione(s + " è valido.");
               } else {
```

```
                    EsitoOperazione(s + " non è valido.");
                }
            }
        }
}
class Program {
    static void Main(string[] args) {
        A istanzaDiA = new A();
        // quando si istanzia la classe si collega il metodo con l'evento;
        // notare l'operatore +=, per aggiungere il metodo ad altri
        // eventualmente già associati.
        istanzaDiA.EsitoOperazione += new GestoreEventoOperazione(MostraMessaggio);
        istanzaDiA.Proprietà = "UID1234";
        Console.ReadKey();
    }
    // event listener, conforme al delegato
    static void MostraMessaggio(string mex) {
        Console.WriteLine(mex);
    }
}
}
```

2.3. LINQ

LINQ è l'acronimo di *Language Integrated Query* ed è un componente del Framework .NET (introdotto nella versione 3.0) che aggiunge capacità native di interrogazione (*query*) di dati ai linguaggi .NET. Le interrogazioni sono applicabili a qualunque oggetto il cui tipo sia una classe derivante direttamente o indirettamente dall'interfaccia `System.Collections.IEnumerable`. LINQ definisce un gruppo di nomi di metodi (*standard query operators*, o *standard sequence operators*), insieme a traduzioni di espressioni SQL verso espressioni utilizzanti questi nomi di metodi, espressioni lambda, tipi anonimi, e variabili implicitamente tipizzate.

2.3.1. Operatori ed estensioni

Alcuni operatori di interrogazione sono `Select`, `Where`, `OrderBy`, `First`, `Last`, etc., a cui corrispondono le espressioni SQL tradotte nelle parole chiave `select`, `where`, `orderby` e, in aggiunta, `from` (`first` e `last` sono escluse). Le espressioni lambda forniscono una semplice sintassi per funzioni anonime (quando una funzione viene assegnata direttamente a una variabile). I tipi anonimi sono delle classi senza nome aventi come membri solo campi in sola lettura. Infine le variabili implicitamente tipizzate sono quelle variabili il cui tipo è evinto dal compilatore, a seconda di come sono inizializzate; sono comunque fortemente tipizzate e devono essere inizializzate nel momento in cui si dichiarano. Segue un esempio riassuntivo.

```
int unNumero = 5;
// Versione con le espressioni SQL tradotte.
```

```
var risultati = from c in CollezioneDiQualcosa
                where c.Proprieta < unNumero * 2
                // es. di tipizzazione implicita
                select new {c.Proprieta, c.AltraProprieta};
foreach (var risultato in risultati) {
     Console.WriteLine(risultato.ToString());
}
// Stesso esempio con le espressioni lambda;
// in questo caso from non serve.
var results = CollezioneDiQualcosa
              .Where(c => c.Proprieta < unNumero * 2)
              .Select(c => new {c.Proprieta, c.AltraProprieta});
foreach (var risultato in risultati) {
     Console.WriteLine(risultato.ToString());
}
```

2.3.2. LINQ per diverse fonti di dati

LINQ può essere esteso per attingere potenzialmente a qualsiasi sorgente di dati. Ne sono un esempio *LINQ to SQL* (dove è usato per interrogare database *SQL Server* e *SQL Server Compact*), *LINQ to XML* e *LINQ to Objects*.

Il prototipo non prevede una connessione a un database, per cui qui LINQ to SQL non viene considerato. LINQ to Objects è utilizzato per interrogare collezioni (*collections*) di dati all'interno della memoria allocata per il programma, per cui sono interrogazioni locali. LINQ to XML converte un documento XML in una collezione di oggetti `XElement`, appartenente a `System.Xml.Linq.XDocument`. Silverlight non supporta la classe `XmlDocument`, normalmente utilizzata per gestire i documenti XML, ma implementa `XDocument` e `XElement`.

3. Web Service

Un web service è un sistema *software* progettato per supportare un'interazione tra macchine distribuite nel network ed è costituito da un'interfaccia descritta in un formato processabile da una macchina (in specifico *Web Services Description Language – WSDL*). Gli altri sistemi interagiscono con il web service, in un modo prescritto dalla sua descrizione, utilizzando messaggi *SOAP* (*Simple Object Access Protocol*), tipicamente veicolati attraverso HTTP con una serializzazione XML congiunta ad altri standard relativi al web.

Il Framework .NET, in particolare, implementa questo sistema software per mezzo dell'interfaccia *WCF* (*Windows Communication Foundation*).

3.1. Servizi

Un servizio WCF è composto da tre parti. Una classe dedicata che implementa il servizio da fornire, un ambiente che ospita il servizio, e uno o più "punti di confine" (*endpoints*) attraverso cui i client si connettono.

3.2. Endpoint

Tutte le comunicazioni del servizio WCF avvengono per mezzo degli endpoint che specificano un "contratto" (*Contract*) che definisce quale metodo della classe del servizio è accessibile attraverso l'endpoint; una volta stabilito il contratto tra client e servizio, l'endpoint può quindi esporre una gamma differente di metodi (cfr. §III. 2.2. , per degli esempi). Gli endpoint definiscono inoltre delle proprietà di legame (*Binding*) che specificano il modo in cui i dati sono trasferiti durante la comunicazione col client, e l'indirizzo (*Address*, *un URL – Uniform Resource Locator*) dove possono essere raggiunti.

Le proprietà di legame stabiliscono quali protocolli di comunicazione vengono usati per accedere al servizio, ed eventuali meccanismi di sicurezza. WCF supporta numerosi protocolli, tra cui HTTP (utilizzato per il prototipo) e *TCP*, e interagisce dall'endpoint con il client utilizzando un involucro SOAP, semplice XML che rende WCF indipendente dal tipo di piattaforma. Quando un client vuole accedere al servizio attraverso l'indirizzo di un endpoint, è necessario che conosca il contratto e che aderisca alle proprietà di legame specificate dall'endpoint stesso. Pertanto, sia client che server devono avere endpoint compatibili.

Dalla versione 3.5 del Framework .NET è incluso un codificatore che aggiunge a WCF il supporto del formato di serializzazione *JSON* (JavaScript Object Notation, un formato di scambio di dati strutturati e a "lettura d'uomo" derivato dal modo di annotazione degli oggetti in JavaScript), permettendo di servire richieste *AJAX* dalle pagine web.

3.3. Interazione con Silverlight

SOAP (*Simple Object Access Protocol*) è un protocollo di scambio di dati strutturati attraverso la rete, basato su XML e standard W3C. Questo protocollo è costituito da tre parti: un involucro (*envelope*), che costituisce l'elemento radice, che ha annidati, in eguale parentela, una testata (*header*) e un corpo (*body*). All'interno del corpo dell'involucro SOAP, sono annidate le informazioni di scambio tra client e web service, e, dal momento che queste sono strutturate in un qualunque altro formato XML, si utilizzano i namespace (solitamente il prefisso utilizzato per il protocollo è `soap:`, ma può variare a patto che si rispetti l'univocità dei namespace). Silverlight dispone di un client SOAP per interagire con un servizio web. Dopo che il client è stato opportunamente configurato (per dialogare con l'endpoint del servizio), questo può accedere ai metodi del servizio web effettuando chiamate asincrone che vengono gestite nella logica dell'applicazione attraverso metodi che ascoltano l'evento di conclusione della comunicazione.

Si rimanda a [Moroney 2008] per un a visione più dettagliata di Silverlight, ASP.NET e Web Service.

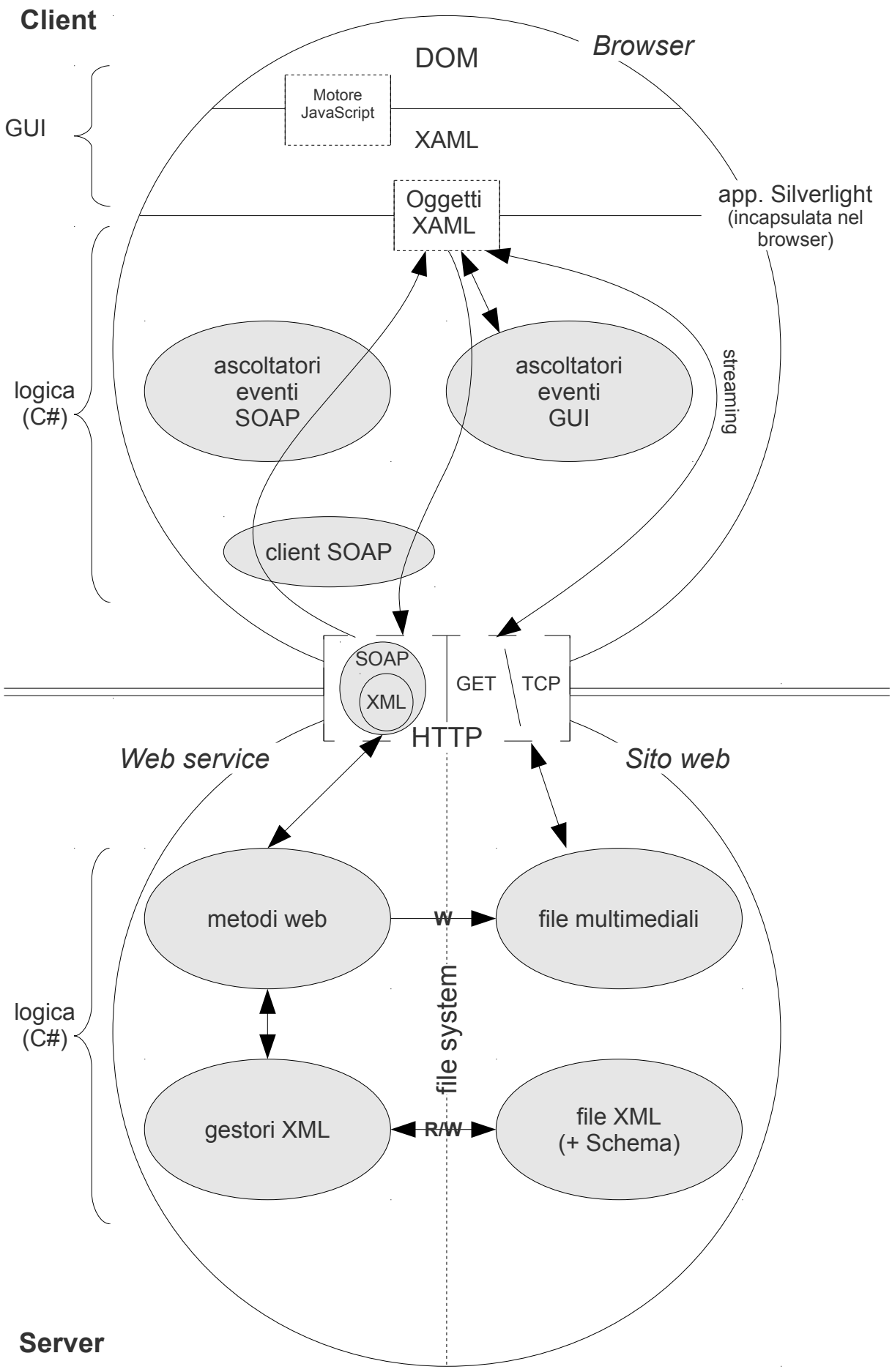

Figura 2: Struttura del prototipo

III. Il prototipo

Dopo l'introduzione alle tecnologie, ora si può vedere come queste sono state utilizzate per la realizzazione del prototipo.

Come si è già detto (cfr. §II), il prototipo è suddiviso in un'applicazione web e in un servizio web, dialoganti tra loro attraverso la rete, poiché il primo è sul client dell'utente, e il secondo sul server remoto, al quale l'utente si connette per mezzo di un browser. Quest'ultimo, dopo aver caricato la pagina web richiesta, carica anche l'applicazione Silverlight attraverso l'apposito plug-in. Dopodiché, l'utente può interagire con l'applicazione nel seguente modo.

L'applicazione Silverlight si è detto essere incapsulata all'interno del browser (Figura 2, qui a fianco), e quindi nel DOM (Document Object Model – il modello a oggetti del documento, manipolabile grazie a JavaScript) della pagina; l'utente interagisce direttamente con il livello più superficiale dell'applicazione, la parte grafica prodotta da XAML; questa parte viene convertita in una struttura a oggetti per poter essere gestita dalla logica delle classi C# (cfr. §2.1.1) – quanto fin qui detto, si può paragonare, concettualmente, con la struttura di una pagina web: codice per la grafica (X)HTML + *CSS* (*Cascading Style Sheets*), DOM per convertire l'interfaccia utente in oggetti gestibili da JavaScript (e tutto ciò rimane lato client), ma la differenza è che la pagina web è interpretata dal browser mentre questo la sta caricando, al contrario l'applicazione Silverlight è compilata e viene eseguita dal plug-in che funge da macchina virtuale. Nella parte di logica sono gestiti gli eventi legati all'interfaccia utente che a loro volta, se viene invocato quello corrispondente, possono inizializzare una chiamata asincrona SOAP verso il web service; ricevuta la risposta dal web service, il client SOAP chiude la connessione e fa scattare l'evento relativo, che è ascoltato da opportuni metodi che a loro volta aggiornano lo strato di grafica, attraverso gli oggetti XAML, con le informazioni appena ricevute dal servizio web. Gli oggetti XAML possono chiamare il server anche direttamente (senza passare per il client SOAP) nel caso in cui debbano reperire un contenuto multimediale; quindi la prima chiamata sarà un HTTP GET che stabilisce l'inizio dello *streaming* che poi avviene in *TCP* (*Transmission Control Protocol*).

Lato server c'è il sito web che fornisce al browser la pagina principale contenente XHTML statico con il collegamento all'applicazione Silverlight, l'applicazione Silverlight, i file multimediali, e il web service. Quest'ultimo espone al client i metodi web necessari per manipolare i file multimediali. I metodi scartano dall'involucro SOAP le informazioni XML che vengono date da gestire ad appositi metodi o classi che convertono l'XML (per mezzo di LINQ) in oggetti C# e, quando necessario, leggono o memorizzano le informazioni in file XML, usati come base-dati, la cui validazione viene effettuata tramite LINQ e XML Schema; a questo punto il metodo web continua con la propria operazione, che può comprendere la manipolazione e codifica di file multimediali (per mezzo dell'SDK di Expression Encoder), e impacchetta un documento XML per la risposta in un involucro SOAP che viene spedito al client.

Questi sono dunque in linea generale i processi del prototipo che nei prossimi paragrafi saranno espressi più dettagliatamente con anche esempi di codice sorgente, per i quali si adotteranno le seguenti convenzioni stilistiche: i commenti sono in corsivo e di colore grigio; i puntini di sospensione "…" stanno a indicare del codice omesso per praticità di lettura; tutti i metodi sono in grassetto; le parti sottolineate attirano l'attenzione su punti di cui si parla in seguito.

1. Applicazione web

Si è dunque detto che l'applicazione Silverlight è ospitata all'interno del browser e viene eseguita dal relativo plugin; l'esempio seguente mostra il codice HTML che annida l'oggetto Silverlight.

```
<!-- L'oggetto è annidato in un tag div con id univoco in modo che possa essere gestita la
     visualizzazione nella pagina HTML per mezzo dei CSS -->
<div id="silverlightControlHost">
     <!-- il tag object con gli attributi che definiscono il tipo di applicazione contenuto,
          in modo tale che il browser sa che plugin adoperare -->
     <object data="data:application/x-silverlight-2," type="application/x-silverlight-2"
      width="100%" height="100%">
          <!-- questo parametro indica l'url dove l'applicazione silverlight si trova; in
               questo caso è relativa rispetto a questa pagina web -->
          <param name="source" value="slbin/Torino.xap"/>
          <!-- in caso di errore, ovvero eccezione non gestita, si chiama la funzione
               JavaScript "onSilverlightError" (omessa in questo esempio e presente nel tag
               head della pagina)  -->
          <param name="onError" value="onSilverlightError" />
          <!-- La versione minima del player Silverlight con la quale è possibile eseguire
               l'applicazione -->
          <param name="minRuntimeVersion" value="3.0.40624.0" />
          <!-- Nel caso in cui la versione del player fosse più bassa oppure questo non fosse
               nemmeno installato nel browser, allora viene visualizzata, nello spazio della
               pagina dedicato all'applicazione, un'immagine col simbolo di silverlight che fa
               da collegamento esterno per scaricare il plug-in dal sito Microsoft -->
          <a href="http://go.microsoft.com/fwlink/?LinkID=149156&v=3.0.40624.0"
           style="text-decoration:none">
               <img src="http://go.microsoft.com/fwlink/?LinkId=108181"
                alt="Get Microsoft Silverlight" style="border-style:none"/>
          </a>
     </object>
</div>
```

Quando il browser riconosce che è annidato un oggetto Silverlight sceglie il rispettivo plug-in che scarica l'applicazione dal server e quindi la esegue.

1.1. Interfaccia utente

Una volta caricata l'applicazione Silverlight all'interno della pagina web, si mostra all'utente un'interfaccia composta da quattro entità (Figura 3, a fianco). 1) Sulla destra una *playlist* in cui sono elencati i file multimediali, 2) sulla sinistra un lettore multimediale che riproduce questi file, 3) in alto i comandi per il montaggio audio/video, e 4) degli elementi temporaneamente nascosti che compaiono per dare messaggi all'utente. Tutti i controlli che seguono appartengono al namespace CLR (Common Language Runtime) `System.Windows.Controls`.

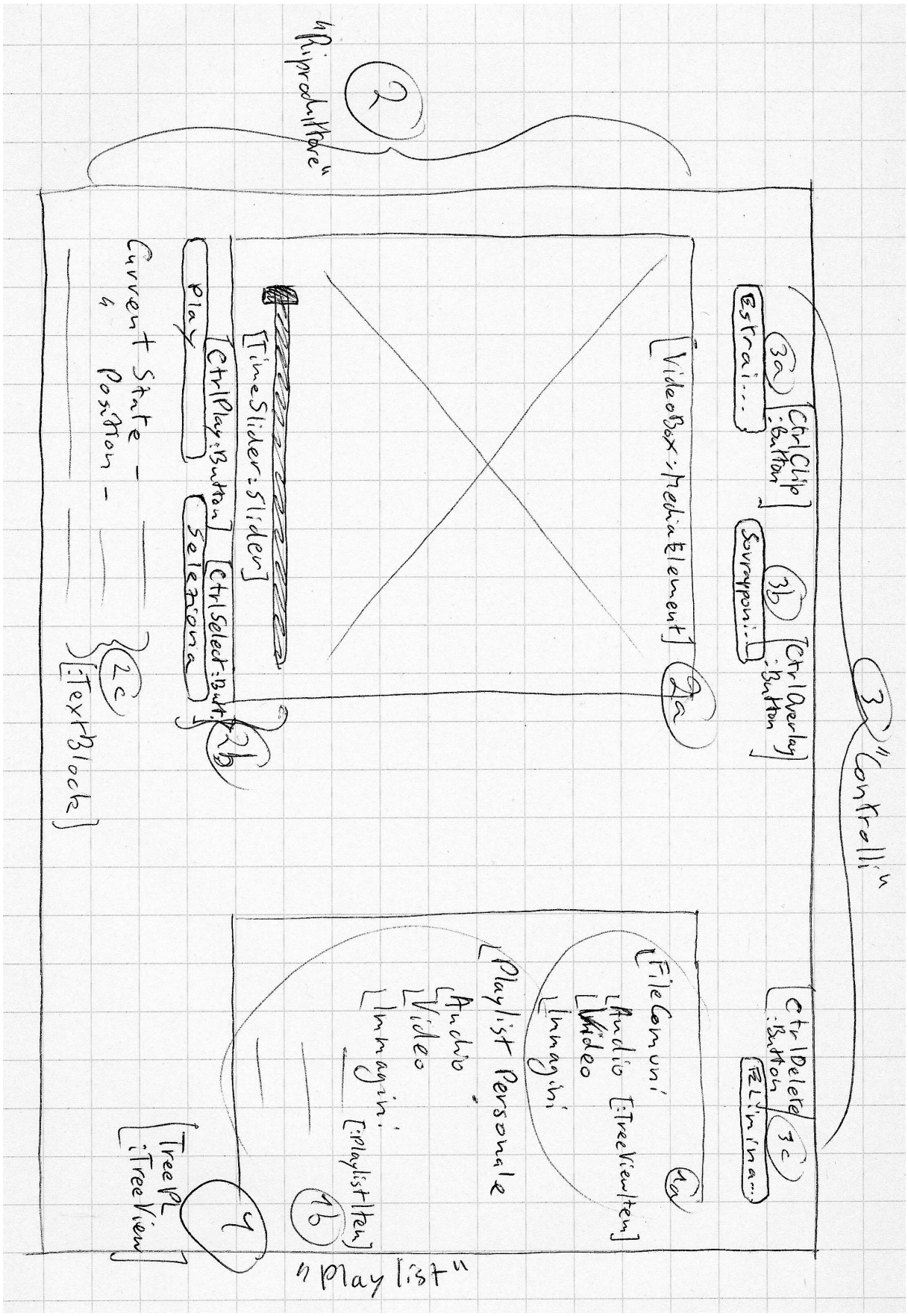

Figura 3: Struttura GUI dell'applicazione

1.1.1. Playlist

La Playlist è un controllo `TreeView`, ossia è una lista ad albero, qui divisa in due parti: una playlist per tutti gli utenti e una invece personale per ogni utente. Ciascuna è ramificata in tre categorie: audio, video e immagini, per raccogliere le rispettive tre tipologie multimediali; ogni categoria a sua volta contiene i nomi dei file multimediali che l'utente può cliccare per riprodurli o per le azioni di manipolazione.

1.1.2. Riproduttore multimediale

Il riproduttore multimediale è un controllo `MediaElement` (ovvero "elemento multimediale") che consiste solamente di una cornice di dimensioni personalizzabili e di una classe per lo streaming multimediale e i metodi e le proprietà necessari per poterlo controllare. Per questa ragione sono stati forniti all'utente dei controlli di riproduzione: uno cursore (`Slider`, posto in basso e sulla cornice dell'elemento multimediale) che funziona da indicatore temporale di riproduzione e, con l'intervento del puntatore, da posizionatore per saltare a un determinato istante temporale; un pulsante (`Button`, posto appena sotto alla cornice) avviare la riproduzione ("*play"*), sospenderla ("*pause"*) e riavviarla dopo che è terminata ("*play again"*); un pulsante ("seleziona" a destra di quelo precedente) per selezionare due istanti temporali dalla riproduzione, creando così un intervallo. Infine vi sono dei campi testuali (`TextBlock`, messi sotto i pulsanti) che indicano lo stato di riproduzione, il tempo di riproduzione che trascorre e la durata totale (espressi in minuti, secondi e millesimi di secondo), e l'inizio e la fine dell'intervallo di tempo selezionato (sempre espressi in minuti, secondi e millesimi di secondo).

1.1.3. Controlli

L'utente può usufruire di tre controlli-pulsanti per il montaggio on-line di audio/video. 1) "Estrai", che ricava dall'intervallo selezionato un nuovo file; 2) "Sovrapponi" che sovrappone un file audio, video o immagine sull'intervallo selezionato; 3) "Elimina", che cancella i file dalla playlist personale. I primi due sono inizialmente disattivati e si attivano quando l'utente seleziona un intervallo di tempo; se premuti compare una finestra di dialogo (un controllo per-

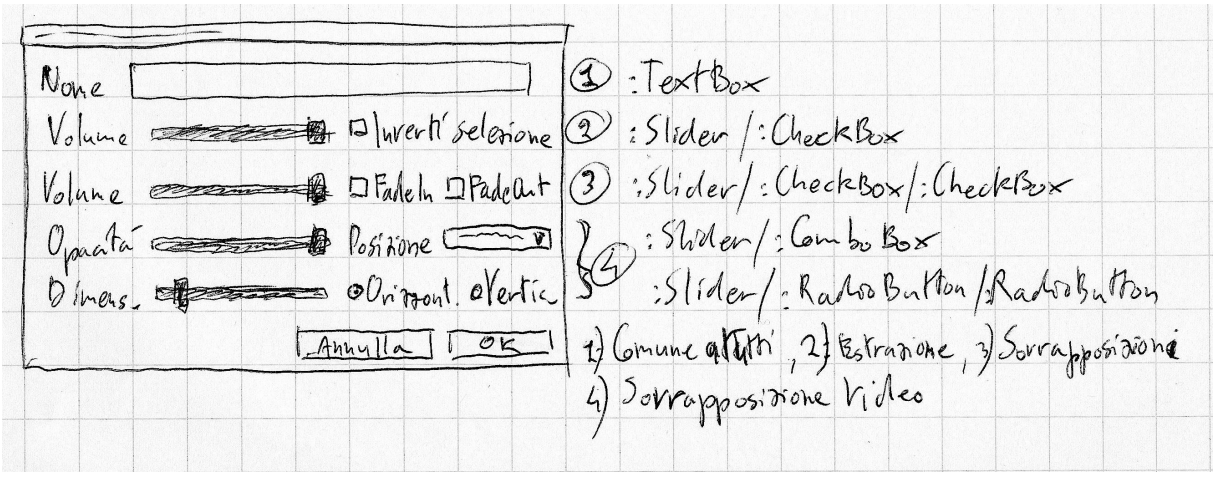

Figura 4: Struttura GUI della finestra per le manipolazioni audio/video

sonalizzato che eredita da `ChildWindow`) che permette di annullare o confermare l'operazione e di impostare il nome del nuovo file da creare e dei parametri di configurazione differenti a seconda. Se si è premuto 1) si può invertire la selezione, per cui l'intervallo selezionato viene escluso dal nuovo file, e il volume sonoro. S è 2) a essere premuto e il file caricato dal riproduttore è solo audio allora è possibile sovrapporre solo un altro file audio impostando il volume e eventuali effetti di sfumatura all'inizio e alla fine *(fade in* e *fade out),* se invece è stato caricato un video è possibile sovrapporre sia audio che video; nel primo caso con le stesse opzioni di prima, nel secondo con l'aggiunta di opacità, posizione (con cinque scelte predefinite), ridimensionamento relativo al video base, in rapporto al suo asse verticale o orizzontale. Premendo 3) invece l'utente può selezionare dalla propria playlist tutti quei file che vuole eliminare permanentemente; lo sfondo dei nomi dei file selezionati si colora di un rosso pallido e quindi si può scegliere se annullare o procedere con l'operazione.

1.1.4. Messaggi

Sono stati adottati due modi per notificare l'utente di messaggi di informazione o di errore. Nel caso di errore irreversibile, per cui non è più possibile utilizzare l'applicazione, se non aggiornando la pagina, oppure durante l'attesa delle risposte del web service, compare un livello con sfondo bianco semitrasparente, che copre tutta l'area dell'applicazione, avente nel centro il testo del messaggio; in questo modo non si dà modo all'utente di interagire o permanentemente o solo durante il periodo di attesa. Tutti gli altri messaggi vengono annunciati tramite una finestrella di avviso (`System.Windows.MessageBox`) con solo i pulsanti di chiusura.

1.2. Logica

Sì è visto come gli oggetti XAML siano il ponte tra l'interfaccia utente e la logica applicativa. Tuttavia, affinché ci sia un'interazione tra gli oggetti e la logica, è necessario che gli eventi dei primi siano ascoltati da appositi metodi; questi ultimi intervengono sugli oggetti XAML direttamente oppure tramite ulteriori metodi o classi di supporto (nel caso di funzionalità più complesse). A seconda dell'evento vengono invocati i metodi del web service per mezzo del client SOAP, le cui risposte sono gestite da appositi metodi che ascoltano l'evento di chiusura della chiamata SOAP; vi sono inoltre dei metodi di appoggio anche per il client SOAP. Le azioni della logica che richiedono una notifica (messaggi ed errori) verso l'utente sono gestite da appositi metodi.

Dal momento che tutta la logica è stata costruita per servire le azioni dell'utente attraverso l'interfaccia, è chiaro che la gestione degli eventi assume un posto cardinale; perciò nei paragrafi seguenti saranno affrontati uno a uno, con il codice XAML per primo (file principale "MainPage.xaml") e il rispettivo codice gestito di seguito (file "MainPage.xaml.cs").

1.2.1. Eventi UI generici

Il primo evento a manifestarsi è quello avvenuto caricamento degli oggetti XAML dell'applicazione.

```
<!-- UserControl è l'elemento radice, il cui codice gestito è la classe MainPage del
```

```
    namespace Torino (il nome in codice del prototipo). Per ragioni di spazio sono stati
    omessi alcuni attributi; invece è stato lasciato “Loaded” (ossia “caricato”) che
    contiene il riferimento al nome del metodo che ascolta l’evento. -->
<UserControl x:Class="Torino.MainPage"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Width="800" Height="500" Loaded="userControl_Loaded">
    <!-- [...] -->
</UserControl>
void userControl_Loaded(object sender, RoutedEventArgs e) {
    // Metodo di appoggio che disabilita alcuni o tutti i controlli, a eccezione della
    // Playlist e del Pulsante di eliminazione dei file multimediali da questa.
    enableControls(false, true);
    // Si inizializza a 0.05 secondi la durata di un istante dell’oggetto MediaStoryBoard.
    MediaStoryBoard.Duration = new Duration(TimeSpan.FromMilliseconds(50));
    // Si assegna all’oggetto TimeSlider (Figura 3, sezione 2b) il metodo che ascolta
    // l’evento di quando il pulsante sinistro del mouse viene premuto. Questo è l’unico modo
    // per assegnare questo tipo di evento con la classe di questo oggetto.
    TimeSlider.AddHandler(Slider.MouseLeftButtonDownEvent,
         new MouseButtonEventHandler(timeSlider_MouseLeftButtonDown), true);

    // Si visualizza all’utente il messaggio di caricamento delle playlist,
    showMessage(messageTypes.loadingPlaylist);
    // che vengono reperite dal client SOAP; original è la playlist dei file multimediali
    // predefiniti, mentre USERNAME è una costante che contiene il nome dell’utente a cui è
    // associata la relativa playlist.
    ssclient.GetPlaylistAsync("original");
    ssclient.GetPlaylistAsync(USERNAME);
}
```

In futuro la costante `USERNAME` del codice sopra potrà essere sostituita, per supportare più di un utente, da una variabile di sessione popolata in seguito all’identificazione dell’utente. `ssclient` è un’istanza del client SOAP, che si vedrà più avanti nel §1.2.5. `MediaStoryBoard` è un’istanza della classe di animazione e serve a gestire un evento in un periodo temporale, di cui è necessario esprimere la durata, che può essere controllata da metodi di partenza, pausa, ripresa e fermata; può essere inoltre associato un ascoltatore ad avvenuto completamento dell’animazione; se in questo ascoltatore si fa ripartire l’animazione si genera così un ciclo di riproduzione; nell’esempio precedente si è visto che la durata è impostata a 0,05 secondi, questo significa che, col ciclo di riproduzione, si ottengono 20 cicli al secondo, sufficienti perché l’occhio veda un’animazione fluida.

```
<!-- Questo elemento è diretto discendente della radice e descrive una proprietà complessa
    (cfr. §II. 2.1.2. ) di questa, per cui Storyboard è una risorsa, non un controllo. -->
<UserControl.Resources>
    <Storyboard x:Name="MediaStoryBoard" Completed="mediaStoryBoard_Completed"/>
    <!-- [...] -->
</UserControl.Resources>
```

```
void mediaStoryBoard_Completed(object sender, EventArgs e) {
     // Solo se il lettore multimediale è in fase di riproduzione,
     if (VideoBox.CurrentState == MediaElementState.Playing) {
          // il valore dello slider viene sincronizzato con l'istante temporale
          // del lettore multimediale,
          TimeSlider.Value = VideoBox.Position.TotalSeconds;
          // e l'animazione riparte.
          MediaStoryBoard.Begin();
     }
}
```

L'oggetto `MediaStoryBoard` serve dunque per animare il cursore (`TimeSlider`, Figura 3, sezione 2b) insieme alla riproduzione multimediale per indicarne visivamente lo stato di avanzamento. `VideoBox` è il nome dell'oggetto che riproduce i contenuti multimediali, al quale sono associati due eventi: quando un nuovo contenuto viene aperto (`MediaOpened`) e al cambiamento dello stato di riproduzione (`CurrentStateChanged` – fermo, pausa, riproduzione, caricamento, etc.).

```
<!-- L'elemento è all'interno della griglia (attributi Grid.*) che lo posiziona a sinistra -->
<MediaElement CurrentStateChanged="videoBox_StateChanged" x:Name="VideoBox"
     Grid.Row="0" AutoPlay="false" Margin="8" Grid.ColumnSpan="2" Volume="1"
     MediaOpened="videoBox_MediaOpened"/>
void videoBox_MediaOpened(object sender, RoutedEventArgs e) {
     // Quando il file multimediale è aperto per la riproduzione, sono impostati il valore e
     // il punto di partenza (.Minimum) del cursore temporale a 0;
     TimeSlider.Value = TimeSlider.Minimum = 0;
     // invece il valore massimo (punto di arrivo) che il cursore può raggiungere è identico
     // alla durata, in secondi, del file video/audio;
     TimeSlider.Maximum = VideoBox.NaturalDuration.TimeSpan.TotalSeconds;
     // dopodiché i controlli del lettore sono riabilitati;
     enableControls(true);
}
void videoBox_StateChanged(object sender, EventArgs e) {
     // A true se il riproduttore ha raggiunto la fine.
     bool eom = (VideoBox.Position.TotalSeconds ==
                 VideoBox.NaturalDuration.TimeSpan.TotalSeconds);
     // nei campi testuali sotto i pulsanti del lettore vengono aggiornati lo stato e
     mediaState.Text = "Current state - " + VideoBox.CurrentState.ToString();
     // la posizione dello stesso.
     writeCurrentPosition();
     // Gestione di alcuni stati del lettore
     switch (VideoBox.CurrentState) {
          case MediaElementState.Paused:
               /* ... */
               break;
          case MediaElementState.Stopped:
               /* ... */
```

```
                break;
            case MediaElementState.Buffering:
                /* ... */
                break;
            case MediaElementState.Playing:
                /* ... */
                break;
            default:
                /* ... */
                break;
        }
}
```

Legati al lettore multimediali c'è sia il cursore temporale (messo in moto dall'oggetto di animazione, come si è visto poco fa), sia il pulsante che controlla la riproduzione/pausa/riproduzione da capo del lettore; nel primo caso l'oggetto di animazione viene messo in movimento se il video è in stato di riproduzione (`MediaElementState.Playing`), invece in pausa se lo stato è di pausa o di caricamento (`MediaElementState.Paused` o `MediaElementState.Buffering`), oppure fermato (con stato `MediaElementState.Stopped`); nel secondo caso il pulsante assumerà funzione di riproduzione per uno stato `.Stopped` o `.Paused`, con `eom = false`, di pausa con `.Playing`, di riproduzione da capo con `.Paused` e `eom = true`, nessuna funzionalità in tutti gli altri casi.

1.2.2. Eventi dei controlli del lettore multimediale

In questo paragrafo si espongono le funzionalità dei controlli di riproduzione del lettore multimediale: riproduzione/pausa/riproduzione da capo ("play", "pause", "play again", del controllo `CtrlPlay`); selezione di un intervallo temporale del file che aperto dal lettore ("seleziona", controllo `CtrlSelect`); spostamento del cursore temporale (controllo `TimeSlider`) col puntatore del mouse, e conseguente riposizionamento del punto di riproduzione.

```
<!-- Gli elementi seguenti sono nella stessa griglia di MediaElement -->
<Button x:Name="CtrlPlay" Click="ctrlPlay_Click" Content="Play"
 Margin="0,8,4,8" Grid.Row="1"/>
<Button x:Name="CtrlSelect" Click="ctrlSelect_Click" Margin="4,8,0,8"
 Content="Seleziona" Grid.Row="1" Grid.Column="1"/>
<!-- Si ricorda che il metodo timeSlider_MouseLeftButtonDown dell'evento MouseLeftButtonDown è
     stato definito all'avvenuto caricamento dell'applicazione (cfr. §1.2.1). -->
<Slider x:Name="TimeSlider" Margin="8,0,8,8" Grid.ColumnSpan="2" Height="16"
 VerticalAlignment="Bottom" UseLayoutRounding="True" Opacity="0.6"
 ValueChanged="timeSlider_ValueChanged"/>

void ctrlPlay_Click(object sender, RoutedEventArgs e) {
     // Questo campo è importante in questa classe, poiché tiene traccia di quale controllo
     // è stato premuto per ultimo, facilitando così la gestione delle azioni da compiere.
     lastPressedControl = editControls.play;
     // Istruzione di scelta del tipo di funzionalità del controllo, condizionata dagli stati
```

```
     // di VideoBox (memorizzati nel campo “dothis”, un enumeratore).
     switch (dothis) {
          /* ... */
     }
}
void ctrlSelect_Click(object sender, RoutedEventArgs e) {
     // Campo importante, vedere commento metodo sopra.
     lastPressedControl = editControls.selection;
     /* ... */
     // “clipped” è un’istanza della classe ClipSpan che memorizza le proprietà di selezione;
     // se la selezione non ha impostato l’inizio oppure (è impostato insieme alla fine),
     if (!clipped.IsFromSet || (clipped.IsFromSet && clipped.IsToSet)) {
          // se l’inizio e la fine sono impostati
          if (clipped.IsFromSet && clipped.IsToSet) {
               /* ... cancello le proprietà di inizio e di fine da “clipped” e cancello il
                      contenuto del campo testuale che mostra all’utente i due limiti
                      temporali di selezione (reset); ... */
          }
          /* ... imposto la proprietà di inizio. ... */
     // Altrimenti, se la fine non è impostata,
     } else if (!clipped.IsToSet) {
          /* ... imposto la fine, aggiusto l’inizio e la fine (p.e. se la fine risulta <
                 dell’inizio, questi vengono ribaltati per coerenza) e scrivo nel campo
                 testuale i due estremi di selezione. ... */
     }
}
void timeSlider_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) {
     // Quando il cursore temporale è premuto dall’utente, si mette in pausa il video;
     VideoBox.Pause();
}
void timeSlider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e) {
     /* ... mentre l’utente tiene premuto il cursore e ne varia la posizione,
            il video si sincronizza di conseguenza, e il campo testuale che mostra all’utente
            il tempo trascorso viene aggiornato con i nuovi valori temporali. ... */
}
```

Per definire un intervallo temporale si usa la classe `TimeSpan`, precisa al millisecondo, impostata con un valore di zero quando istanziata. Sopra si è appena visto un riferimento alla classe di supporto `ClipSpan` (istanziata come “`clipped`”), che è stata creata con i seguenti membri: `From` e `To`, di tipo `TimeSpan`, che indicano un istante temporale per l’inizio e la fine di una selezione; `IsFromSet` e `IsToSet`, di tipo `bool`, che verificano che le proprietà di inizio e fine siano state valorizzate dall’utente, e non preimpostate; `Adjust()`, che opera se sia l’inizio che la fine sono valorizzati dall’utente, cancella il valore finale in caso sia uguale a quello iniziale, e invece ribalta i valori se l’inizio è maggiore della fine; e `Clear()` che azzera tutto e pone a `false` le proprietà di verifica.

Inoltre si è visto un campo (`lastPressedControl`) usato per tener traccia dell’ultimo con-

trollo premuto dall'utente. Il suo valore è un tipo `enum` e può assumere i seguenti valori: `none` ("nessuno", usato quando l'operazione legata a un pulsante è conclusa), `clip` ("taglio", dopo che è stato premuto il pulsante "Estrai"), `overlay` ("sovrapposizione", associato al pulsante "Sovrapponi"), `delete` ("cancella", pulsante "Elimina"), `play` ("riproduci", pulsante "play/pause/play again"), `selection` ("selezione", pulsante "Seleziona").

1.2.3. Eventi della Playlist

Il controllo per la playlist è il più complesso, fungendo in alcune operazioni da tramite per altri controlli, e per questa ragione gli si dedica un paragrafo intero. Esso ha una struttura ad albero con i nomi delle playlist come rami più esterni, le categorie multimediali come rami figli e infine le foglie sono i nomi dei file, e assume quattro funzionalità a seconda del contesto interattivo con l'utente: 1) al cliccare di un ramo mostra il contenuto, se chiuso o viceversa lo nasconde; 2) al cliccare di una foglia il lettore multimediale riproduce il relativo file; 3) se l'utente ha premuto il pulsante di cancellazione dei file, allora può selezionare diverse foglie dal controllo (naturalmente il collegamento con il lettore è disattivato) per scegliere quali eliminare; 4) se l'utente ha premuto il pulsante di sovrapposizione, allora può selezionare la foglia con il nome de file da sovrapporre sul file audio/video caricato dal lettore.

```
<!-- Il controllo è disposto nella cella di destra della griglia principale; alcuni attributi
     sono stati omessi. -->
<controls:TreeView x:Name="TreePL" Margin="0,8,8,0" Grid.Column="1" Grid.Row="1"
 SelectedItemChanged="tree_ItemChanged">
     <!-- Questi sono i due rami principali, contenenti, il primo, la playlist con i contenuti
          predefiniti e, il secondo, quella con i contenuti personalizzati, manipolati
          dall'utente. -->
     <controls:TreeViewItem Header="File comuni"/>
     <controls:TreeViewItem Header="Playlist personale"/>
</controls:TreeView>
void tree_ItemChanged(object sender, RoutedPropertyChangedEventArgs<object> e) {
     // Casting dell'oggetto mittente;
     TreeView tv = sender as TreeView;
     // se l'oggetto è null oppure una particolare variabile è a true,
     // allora si esce dal metodo.
     if (tv.SelectedItem == null || isTVSwitchedOff) return;
     string type = tv.SelectedItem.GetType().Name;
     // Se l'elemento selezionato è una foglia, riconosciuto perché di tipo PlayListItem, una
     // classe creata appositamente per estendere un elemento TreeView, allora
     if (type == "PlaylistItem") {
          PlaylistItem pi = tv.SelectedItem as PlaylistItem;
          // e se l'ultimo controllo premuto è quello di sovrapposizione, allora
          if (lastPressedControl == editControls.overlay) {
               // aggiunge la foglia alla lista delle foglie selezionate dall'utente,
               this.selectedItems.Add(pi);
               // istanzia la finestra con le opzioni, che cambia il suo contenuto a seconda
               // che la foglia selezionata sia associata a un file audio o video/immagine e
```

```
            WindowPrompt pr = new WindowPrompt((
                pi.ItemType == PlaylistItem.MediaType.Audio
                ? WindowPrompt.WindowType.audiooverlay
                : WindowPrompt.WindowType.visualoverlay));
            // le assegna un metodo per gestire l'evento di chiusura.
            pr.Closed += new EventHandler(prompt_Closed);
            // Dal momento che l'utente ha selezionato una nuova foglia, diversa
            // da quella caricata nel lettore, si riseleziona la foglia originale;
            // il metodo sotto tiene conto che, al selezionare di una foglia, scatta ancora
            // questo metodo che lo contiene e pertanto, prima della selezione, assegna
            // true a  isTVSwitchedOff (vedere più in alto in questo codice), e dopo lo
            // rimette a false.
            selectPlayingItem();
            // Si mostra la finestra delle opzioni e
            pr.Show();
            // si esce da questo metodo.
            return;
        // Se invece l'ultimo pulsante premuto è quello per l'eliminazione, allora
        } else if (lastPressedControl == editControls.delete) {
            // e se la foglia selezionata non è contenuta nella lista, allora
            if (!selectedItems.Contains(pi)) {
                // la foglia è aggiunta alla lista e
                this.selectedItems.Add(pi);
                // il colore del suo sfondo cambia in rosso pallido;
                pi.Background = new SolidColorBrush(Color.FromArgb(128, 255, 192, 192));
            } else {
                // altrimenti, se la foglia è già presente nella lista (ossia l'utente
                // l'ha già selezionata per l'eliminazione, per cui se la riseleziona
                // si presume che ha cambiato idea e voglia invece escluderla) si toglie
                // da questa e
                selectedItems.Remove(pi);
                // si ripristina il colore originale dello sfondo.
                pi.Background = new SolidColorBrush(Colors.Transparent);
            }
            // se ci sono foglie nella lista, allora l'utente ne ha selezionata almeno una
            // da eliminare, per cui compare il comando relativo per l'eliminazione;
            CtrlDelete_Y.IsEnabled = (this.selectedItems.Count > 0);
            // si riseleziona la foglia il cui file è aperto dal lettore (vedere più sopra
            // in questo esempio) e si esce.
            selectPlayingItem(pi);
            return;
        }
        /* ... se né il controllo di sovrapposizione e né quello di eliminazione sono stati
               premuti, allora significa che l'utente è fuori da quel contesto di
               interazione e sta selezionando una foglia per riprodurre il file associato
```

```
                nel lettore. ... */
    // Infine, se l'utente non ha selezionato una foglia, ma un ramo (perché di tipo
    // TreeViewItem), allora
    } else if (type == "TreeViewItem") {
         /* ... Si espande o riduce (azioni l'una con l'altra esclusive) il ramo e si
                riseleziona sempre la foglia del file aperto dal lettore (se l'utente ha
                precedentemente selezionato la foglia)  ... */
    }
}
```

Nell'esempio precedente si è incontrato l'oggetto `SelectedItems`, che è una semplice lista che memorizza gli elementi selezionati dall'utente, ossia le foglie del controllo `TreeView`; per la funzionalità di sovrapposizione la lista è popolata con un solo elemento, invece per l'eliminazione con più elementi, quelli che vuole eliminare l'utente (dal momento che la selezione è multipla le foglie selezionate hanno lo sfondo che cambia di colore per essere distinte); ogni foglia è di tipo `PlaylistItem`, ossia una classe creata appositamente per memorizzare le informazioni riguardanti i file a queste associate. `WindowPrompt` è un'altra classe personalizzata per gestire in un modo pulito e efficiente i controlli di un'istanza del tipo predefinito `WindowChild`; è la finestra per impostare i parametri per l'estrazione o la sovrapposizione. Nella funzionalità di eliminazione si è incontrato l'oggetto `ctrlDelete_Y`: è il pulsante per la conferma dell'eliminazione. Le classi `PlaylistItem` e `WindowPrompt` e l'oggetto `ctrl Delete_Y` verranno visti più da vicino, rispettivamente nel §1.2.5 e gli altri due nel prossimo.

1.2.4. Eventi dei controlli che invocano il client SOAP

Ci sono tre controlli con i quali l'utente può interagire e comunicare con il web service: `ctrlClip` (avente come testo "Estrai"), `ctrlOverlay` (testo "Sovrapponi") e `ctrlDelete` ("Elimina"). Quest'ultimo ha un comportamento particolare, ossia, quando viene premuto, scompare e al suo posto appare un campo testuale, che domanda sulla conferma di eliminazione, seguito da un pulsante per annullare l'operazione (`ctrlDelete_N`) e da un altro per confermarla (`ctrlDelete_Y`); oltre alla comparsa di questi pulsanti, vengono disattivate la sezione del lettore multimediale e la playlist dei contenuti predefiniti, che si riattiveranno a operazione conclusa; il controllo della playlist viene impostato per questo tipo di contesto interattivo; l'utente può quindi selezionare quei nomi dei file che vuole eliminare e, se ne sceglie almeno uno, allora si attiva (cfr. § sopra) il pulsante di conferma di eliminazione che invoca il client SOAP. Al premere del pulsante di annullamento operazione o quello della conferma, fa sì che questi due controlli e il campo testuale ritornino nascosti e riappaia il pulsante originale "Elimina".

```
<!-- StackPanel è un contenitore in cui i suoi figli si dispongono automaticamente impilati o
     orizzontalmente o verticalmente, questo è orizzontale; è disposto nella parte in alto a
     destra della griglia principale.  -->
<StackPanel Margin="0" Grid.Column="1" Orientation="Horizontal" HorizontalAlignment="Right">
     <!-- Pulsante di eliminazione; -->
     <Button x:Name="CtrlDelete" Height="22" Margin="8" Width="120" Content="Elimina..."
      HorizontalAlignment="Right" Click="ctrlDelete_Click" IsEnabled="False"/>
     <!-- contenitore annidato, che ospita i pulsanti di annullamento e conferma operazione;
```

```
          il contenitore è inizialmente invisibile (Visibility="Collapsed"). -->
     <StackPanel x:Name="CtrlDelete_SP" Orientation="Horizontal" Visibility="Collapsed">
          <TextBlock HorizontalAlignment="Right" Text="Elimina selezionati?"/>
          <Button x:Name="CtrlDelete_N" Content="Annulla" Click="ctrlDelete_N_Click"/>
          <!-- Inizialmente questo pulsante è disabilitato (IsEnabled="False"). -->
          <Button x:Name="CtrlDelete_Y" Content="Elimina" Click="ctrlDelete_Y_Click"
           IsEnabled="False"/>
     </StackPanel>
</StackPanel>
void ctrlDelete_Click(object sender, RoutedEventArgs e) {
     // Casting dell'oggetto mittente (un pulsante);
     Button b = sender as Button;
     // se il lettore è in riproduzione, allora
     if (VideoBox.CurrentState == MediaElementState.Playing
        || VideoBox.CurrentState == MediaElementState.Buffering)
          // si fermo;
          VideoBox.Pause();
     // nasconde il pulsante e mostra lo StackPanel contenente gli altri pulsanti;
     b.Visibility = Visibility.Collapsed;
     CtrlDelete_SP.Visibility = Visibility.Visible;
     // memorizzo che questo è l'ultimo pulsante a essere stato premuto dall'utente, quindi
     // disabilito i controlli a eccezione dei pulsanti per la conferma/annullamento e della
     // playlist personale (enableControls si serve di lastPressedControl per determinare
     // quali controlli nascondere); l'utente selezionerà le foglie dalla playlist, il cui
     // controllo risponderà sempre a seconda di lastPressedControl (cfr. § sopra).
     lastPressedControl = editControls.delete;
     enableControls(false);
}
void ctrlDelete_N_Click(object sender, RoutedEventArgs e) {
     // se l'utente annulla l'operazione, si ripristinano le impostazioni iniziali: gli sfondi
     // delle foglie della playlist ritornano transparenti, la lista che memorizza le foglie
     // selezionate è ripulita, i pulsanti di conferma/annullamento sono nascosti nuovamente e
     // riappare il pulsante originale "Elimina".
     ctrlDelete_reset();
}
void ctrlDelete_Y_Click(object sender, RoutedEventArgs e) {
     string fnames = "";
     // memorizzo la sorgente dei file legati alle figlie in una stringa
     foreach (PlaylistItem item in selectedItems) {
          fnames += item.Src + ":";
     }
     fnames = fnames.TrimEnd(':');
     // dal momento che si sono memorizzate le informazioni legate prese dalla lista delle
     // foglie selezionate, si può ripristinare lo stato iniziale dei controlli (vedere metodo
     // precedente);
```

```
    ctrlDelete_reset();
    // si mostra un messaggio di attesa e si chiama il metodo del web service attraverso il
    // client SOAP (ssclient).
    showMessage(messageTypes.loadingDelete);
    ssclient.DeleteAsync(fnames);
}
```

Il controllo di sovrapposizione (`ctrlOverlay`) permette all'utente di selezionare una foglia della playlist, che rappresenta un file audio, video o un'immagine presenti sul server del sito web, al fine di aggiungere questo file in una selezione temporale (presa con `ctrlSelect`, §1.2.2) del file aperto dal lettore, come livello di sovrapposizione. Una volta selezionata la foglia desiderata, appare una finestra di dialogo che offre diverse opzioni per personalizzare la modifica; l'utente quindi può scegliere se annullare l'operazione oppure procedere e contattare il web service attraverso il client SOAP, premendo i rispettivi pulsanti della finestra.

Tornando al §1.2.3, dove viene illustrato il controllo della playlist, si può rivedere il blocco di codice dedicato all'operazione di sovrapposizione in cui, al cliccare di una foglia, compare una finestra di dialogo, la quale è logicamente strutturata nel seguente modo. Una classe che eredita da `System.Windows.Controls.ChildWindow` la forma e i controlli di annullamento o conferma, e aggiunge altri controlli che verranno nascosti/visualizzati a seconda delle operazioni da supportare, che sono tre: 1) l'estrazione (vedere più avanti in questo paragrafo), 2) la sovrapposizione di un file audio su un altro file audio o su uno video, e 3) la sovrapposizione di un file video o di un'immagine su un altro file video (sovrapposizione visuale). Sempre nel precedente §1.2.3, è illustrato nel codice di esempio il criterio di scelta tra le operazioni 2) e 3). Vi è poi un'altra classe (`WindowPrompt`), creata per far da tramite tra la classe della pagina principale (`MainPage`) e la finestra, che dispone di tutte le proprietà necessarie per agevolare il reperimento dei valori delle opzioni (elencate in Figura 4). Inoltre nel § sotto l'utilizzo di queste proprietà è illustrato con un esempio, in cui si vede come queste rispecchino la struttura XML che si andrà ad avvolgere nell'involucro SOAP.

```
<!-- L'elemento radice della finestra di tipo ChildWindow. (Omessi alcuni attributi.) -->
<basics:ChildWindow xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 x:Class="Torino.Prompt" Title="Prompt">
    <!-- [...] -->
    <Grid x:Name="GridRoot"  UseLayoutRounding="True">
        <!-- [...] Ci sono tre righe all'interno della finestra: la prima per i controlli
                   comuni alle tre operazioni, la seconda per l'operazione 1) e l'altra per
                   le operazioni 2) e 3). [...] -->
        <Grid.RowDefinitions> <!-- [...] --> </Grid.RowDefinitions>
        <!-- [...] Controlli preimpostati di conferma (OK) e chiusura (Annulla)
                   (due tipi Button); e controlli di diversi tipi comuni alle tre operazioni
             [...] -->
        <!-- Griglia annidata nella prima riga della griglia genitore (Grid.Row="1")  -->
        <Grid x:Name="GridClip" Height="18" Margin="0" Grid.Row="1">
            <!-- [...] Ci sono i controlli per l'operazione 1). [...] -->
        </Grid>
        <!-- Griglia annidata nella seconda riga della griglia genitore (Grid.Row="2");  -->
```

```
          <Grid x:Name="GridOverlay" Margin="0" Grid.Row="2" Height="100">
               <Grid.ColumnDefinitions>
                    <!-- [...] ha due colonne [...] -->
               </Grid.ColumnDefinitions>
               <Grid.RowDefinitions>
                    <!-- [...] e tre righe. [...] -->
               </Grid.RowDefinitions>
               <!-- [...] Ci sono i controlli per l'operazione 2), nella prima riga e 3),
                         nella seconda e terza riga. [...] -->
          </Grid>
     </Grid>
</basics:ChildWindow>
public partial class Prompt : ChildWindow {
     public Prompt() {
          // Come tutte le classi che ereditano da un controllo ne vengono inizializzati i
          // componenti nel costruttore.
          InitializeComponent();
     }
     private void OKButton_Click(object sender, RoutedEventArgs e) {
          // Questa finestra memorizza con questa proprietà se l'utente ha premuto il pulsante
          // di conferma (true) o
          this.DialogResult = true;
     }
     private void CancelButton_Click(object sender, RoutedEventArgs e) {
          // di annullamento (false).
          this.DialogResult = false;
     }
}
// Ecco la classe che gestisce la finestra.
public class WindowPrompt {
// Campo valorizzato con un'istanza della finestra.
private Prompt prompt = new Prompt();
// Usato per specificare il tipo di operazione da supportare: rispettivamente op. 1), 2) e 3).
public enum WindowType { clip, audiooverlay, visualoverlay }
// Lista delle possibili posizioni di un'immagine o video all'interno della cornice
// del video base, e rispettiva proprietà.
public enum VisualOverlayPositions {
     topleft = 0,
     topright = 1,
     center = 2,
     bottomleft = 3,
     bottomright = 4
}
public VisualOverlayPositions VisualOverlayPosition {
     get {
```

```
          // dal momento che nella finestra la lista delle posizioni ha gli elementi con degli
          // indici interi, si è fatto in modo che questi siano compatibili con gli indici
          // del tipo enum sopra, i cui elementi hanno nomi convertibili in stringhe (con
          // opportuno metodo System.String.ToString()) compatibili col documento XML.
          return (VisualOverlayPositions)prompt.optPosition.SelectedIndex;
     }
}
// Per gestire le opzioni di sfuma (fading) all'inizio e/o alla fine, oppure nessuno dei due,
// sono stati utilizzati dei flag che sfruttano gli operatori bitwise per le associazioni di
// più elementi di un enumerazione, dal momento che nella finestra si propone una scelta
// multipla per queste opzioni. Per identificare questo particolare uso di un enum, viene
// utilizzato l'attributo [Flags].
[Flags]
public enum OverlayFadings {
     none = 0, fadein = 1,
     fadeout = 2, both = fadein | fadeout
}
public OverlayFadings OverlayFading {
     get {
          // Si sfrutta sempre la relazione tra indice intero, rispettivo nome
          // dell'enumerazione, conversione in stringa per la struttura XML (i nomi sono
          // sempre stati scelti affinché siano uguali a quelli utilizzati nell'XML).
          return (OverlayFadings)(
               // Il casting è necessario poiché nella finestra IsChecked è un booleano che
               // supporta anche il valore nullo ("bool?", per cui può avere tre stati).
               ((bool)prompt.optFadeIn.IsChecked ? 1 : 0)
             | ((bool)prompt.optFadeOut.IsChecked ? 2 : 0)
          );
     }
}
// Ecco un altro booleano con tre stati possibili; qui serve per collegare lo stato della
// finestra con quello di questa classe
public bool? DialogResult { get { return prompt.DialogResult; } }
// L'evento per gestire il momento di chiusura della finestra. Praticamente, a livello
// pubblico di chi chiama questa classe (WindowPrompt) sembra che questa stessa sia la
// finestra che in realtà è qui invocata (non ereditata, per avere più ordine tra i membri)
public event EventHandler Closed;

/* ... Altre proprietà e campi non degni di nota. ... */

public WindowPrompt(WindowType type) {
     // Inizialmente il pulsante di conferma è disabilitato (perché c'è un campo obbligatorio)
     prompt.OKButton.IsEnabled = false;
     /* ... Si assegna l'evento di chiusura, */
     prompt.Closed += new EventHandler(prompt_Closed);
```

```
     // e un altro evento per gestire il cambiamento di valore all'interno del campo
     // obbligatorio (quello in cui l'utente inserisce il nome del nuovo file da creare), per
     // effettuare una validazione su di esso.
     prompt.optFilename.TextChanged += new TextChangedEventHandler(optFilename_TextChanged);
     // Nel caso in cui la tipologia di operazione sia
     switch (type) {
          case WindowType.audiooverlay:
               /* ... la 2) (sovrapp. audio) allora elimino tutti i controlli per l'operazione
                     3), che non servono, riconoscibili perché annidati in due righe di una
                     griglia, conoscibili a priori (vedere rispettivo codice XAML); ... */
               break;
          case WindowType.visualoverlay:
               /* ... se invece l'operazione scelta è la 3) (sovrapposizione visuale) nascondo
                     la griglia contenente i controlli per l'operazione 1); ... */
               break;
          case WindowType.clip:
          default:
               /* ... infine, per l'operazione 1), si nascondono i controlli delle altre
                     operazioni. ...  */
               break;
     }
}
// Metodo di validazione del campo testuale per specificare il nome del nuovo file.
void optFilename_TextChanged(object sender, TextChangedEventArgs e) {
     // Casting del mittente.
     TextBox tb = sender as TextBox;
     // Per la validazione si utilizza un'espressione regolare.
     System.Text.RegularExpressions.Regex rx =
     // Sono ammessi solo i caratteri di lettere e numeri e il trattino "-" e "_";
     new System.Text.RegularExpressions.Regex("^[-0-9A-Z _]+$"
          // l'espressione è insensibile alle maiuscole o minuscole.
          , System.Text.RegularExpressions.RegexOptions.IgnoreCase);
     if (!rx.IsMatch(tb.Text)) {
          /* ... Se mentre l'utente digita un carattere invalido per il nome del file, lo
                sfondo del controllo si tinge di un rosso e il pulsante di conferma
                rimane/diventa disattivato; ... */
     } else {
          /* ...  altrimenti ritorna del colore preimpostato (memorizzato inizialmente in un
                  campo) e il pulsante di conferma si attiva. ... */
     }
}
// Metodo di tramite per l'evento di chiusura della finestra.
void prompt_Closed(object sender, EventArgs e) { Closed(this, e); }
// Metodo di tramite per mostrare la finestra.
public void Show() { prompt.Show(); }
```

```
}
```

Il pulsante per l'estrazione (`ctrlClip`) permette all'utente di estrarre una certa porzione (si rimanda al §1.2.2, a proposito della selezione temporale) di audio/video dal file aperto dal lettore multimediale. Sia quest'ultimo pulsante, dedicato all'operazione 1), sia il pulsante visto prima, dedicato alle operazioni 2) e 3) (questo attraverso una prima interazione con la playlist), fanno comparire la finestra `WindowPrompt`. Di seguito un esempio che mostra gli eventi gestiti per i due pulsanti e per la chiusura della finestra.

```
<!-- Anche questo elemento è annidato nella griglia principale. Non ci sono le proprietà
     complesse Grid.Row e/o Grid.Column, perché è nella prima cella, quindi sono usati i
     valori predefiniti (="0"). (Alcuni attributi sono stati omessi.)  -->
<StackPanel Margin="0" Orientation="Horizontal" HorizontalAlignment="Center">
     <Button x:Name="CtrlClip" Content="Estrai..."  Click="ctrlClip_Click"/>
     <Button x:Name="CtrlOverlay" Content="Sovrapponi..." Click="ctrlOverlay_Click"/>
</StackPanel>
// Metodo dell'evento di pressione sul tasto per l'operazione 1) (estrazione).
void ctrlClip_Click(object sender, RoutedEventArgs e) {
     // Si contestualizza il tipo d'interazione dell'utente.
     lastPressedControl = editControls.clip;
     // Se i valori di selezione (cfr. §1.2.2) sono impostati, allora
     if (clipped.IsFromSet && clipped.IsToSet) {
          // compare la finestra impostata per l'operazione 1),
          WindowPrompt pr = new WindowPrompt(WindowPrompt.WindowType.clip);
          // si assegna l'evento che scatta quando questa si chiude, e si mostra la finestra;
          pr.Closed += new EventHandler(prompt_Closed);
          pr.Show();
     } else {
          // altrimenti all'utente compare un messaggio di avvertimento che dice di
          // selezionare una porzione del file in lettura.
          showMessage(messageTypes.warningSelection);
     }
}
// Metodo dell'evento di pressione sul tasto per le operazioni 2) o 3).
void ctrlOverlay_Click(object sender, RoutedEventArgs e) {
     // Se i valori di selezione sono impostati, allora
     if (clipped.IsFromSet && clipped.IsToSet) {
          // si contestualizza il tipo d'interazione dell'utente,
          lastPressedControl = editControls.overlay;
          // si disabilitano alcuni controlli;
          enableControls(false);
          // (a questo punto l'utente dovrebbe selezionare una foglia dalla playlist, il cui
          // evento di selezione fa mostrare la finestra.)
     } else {
          // altrimenti si visualizza il messaggio di avvertimento
          showMessage(messageTypes.warningSelection);
     }
```

```
}
// Questo metodo gestisce l'evento di chiusura della finestra ed è condiviso da tutte e tre le
// operazioni.
void prompt_Closed(object sender, EventArgs e) {
     // Casting dell'oggetto mittente.
     WindowPrompt wp = sender as WindowPrompt;
     // Se l'utente ha annullato l'operazione, si cancella la lista delle foglie selezionate,
     // si riattivano i controlli precedentemente disabilitati e si esce.
     if (wp.DialogResult != true) {
          /* ... */
     }
     // Se si è nel contesto dell'operazione 1), allora
     if (lastPressedControl == editControls.clip) {
          // compare il relativo messaggio di attesa
          showMessage(messageTypes.loadingClip);
          // e si spedisce, per mezzo del client SOAP, la struttura XML con le opzioni;
          ssclient.NewMediaAsync(xNewmedia(wp));
     // altrimenti, nel contesto delle opp. 2) e 3),
     } else if (lastPressedControl == editControls.overlay) {
          // compare il relativo messaggio di attesa,
          showMessage(messageTypes.loadingOverlay);
          // si spedisce, per mezzo del client SOAP, la struttura XML con le opzioni
          ssclient.NewMediaAsync(xNewmedia(wp, this.selectedItems[0]));
          // e si cancella la lista delle foglie selezionate (date le operazioni ci sarà una
          // sola foglia).
          selectedItems.Clear();
     }
     /* ... Si cancellano i valori di selezione temporale e il contenuto del campo testuale
            che mostra all'utente i due istanti selezionati, e si riabilitano i controlli
            precedentemente disabilitati  ... */
}
```

Si è appena vista un'invocazione del metodo `xNewmedia`. Serve per costruire la struttura XML secondo le proprietà della finestra (variabile `wp`) appena chiusa e – solo nel caso delle operazioni 2) e 3) – le proprietà della foglia selezionata da sovrapporre (la prima della lista). Viene esposto nel prossimo paragrafo.

1.2.5. Eventi SOAP

Prima di affrontare i gestori degli eventi veri e propri si propongono una classe di appoggio (`PlayListItem`) e due metodi di appoggio (`drawTree` e `xNewmedia`). La prima è utilizzata dalle foglie della playlist ed estende `System.Windows.Controls.TreeViewItem`., usato per ramificare un oggetto `TreeView` (la playlist, appunto); in questo modo è possibile memorizzare delle informazioni specifiche (oltre al nome) relative alle foglie che rappresentano i file multimediali, in rapporto con la struttura XML di interscambio con il web service. Le proprietà supportate: `MediaType`, di tipo enum; `ItemType`, di tipo `MediaType`, che può assumere il

valore di `Audio`, `Video` o `Image` e identifica il tipo di contenuto multimediale (audio, video o immagine); `Filename` ,di tipo `string` come le prossime proprietà, che indica il nome del file; `Fileext`, per l'estensione del file; `Src`, per l'indirizzo web, relativo rispetto all'indirizzo dello'host, del file; `Desc`, per la descrizione del file (ora è inutilizzato, ma potrebbe essere utile in un futuro); `Category`, che indica il nome della categoria del file (rispetto ai tre tipi, audio, video e immagine); infine il costruttore accetta in ingresso `Src`, `Desc` e `Category`, che le inizializza subito, e deduce da `Src` l'estensione (e quindi il tipo multimediale) e il nome del file che "battezza" anche la foglia (impostando una proprietà ereditata).

Della classe `PlayListItem` se ne serve il metodo `drawTree` che disegna i rami e le foglie della playlist secondo le informazioni ricevute in seguito a una chiamata SOAP.

```
// Prende in ingresso l'elemento XML contenente le informazioni della playlist da disegnare
// e un ramo (Playlist predefinita o Playlist personale) del controllo ad albero (TreeView)
// per espanderlo con i dati XML.
void drawTree(XElement pl, TreeViewItem tv) {
     /* ... Si impostano alcune variabili e si cancellano eventuali rami disegnati da una
            chiamata precedente di questa funzione. Con LINQ si individuano i nodi con le
            informazioni sui file multimediali; quindi si cicla su di essi. */
     foreach (XElement media in imedia) {
          if (media.Parent.Attribute("name").Value != category) {
               /* ... Se c'è una nuova categoria, questa si aggiunge all'albero e prende il
                      nome dall'attributo "name" del nodo genitore (che rappresenta una
                      categoria); category è il nome della categoria corrente e viene
                      aggiornata in questo blocco. ... */
          }
          if (item != null) {
               /* ... item è l'oggetto TreeViewItem della categoria: se esiste si può
                      aggiungergli una foglia di tipo PlayListItem  ... */
          }
     }
}
```

Il metodo `xNewmedia` costruisce la struttura XML da spedire via SOAP per creare un nuovo file multimediale (audio o video) come risultato delle operazioni di estrazione o di sovrapposizione.

```
// Primo overload
XElement xNewmedia(WindowPrompt wp) {return xNewmedia(wp, null); }
// In ingresso la finestra che l'utente ha utilizzato per personalizzarsi un'estrazione o una
// sovrapposizione, e una foglia della playlist usata nel caso della sovrapposizione.
// Restituisce un elemento XML che supporta LINQ.
XElement xNewmedia(WindowPrompt wp, PlaylistItem item) {
     string xml;
     // Si inizializza la variabile locale contenente il documento XML, specificando il
     // namespace di riferimento;
     xml = "<newmedia xmlns=\"urn:schemas-torino:newmedia\">\n"
        // popolando gli elementi con proprietà della finestra,
        + " <filename>" + wp.Filename + "</filename>\n"
```

```
        + " <base>\n"
        // del file aperto dal lettore (playingItem)
        + "  <path isaudio=\"" + (playingItem.ItemType == PlaylistItem.MediaType.Audio
              ? "true" : "false") + "\">" + playingItem.Src + "</path>\n"
        // e della selezione che prende una porzione temporale dal file.
        + "  <selection start=\"" + clipped.From.TotalSeconds.ToString(nfi) + "\" end=\""
              + clipped.To.TotalSeconds.ToString(nfi) + "\"" + (wp.ClipInverted
              ? "isinverted=\"true\"" : "") + "/>\n"
        + " </base>";
    // Se una foglia è stata passata tra i parametri d'ingresso (è quindi un'operazione di
    // sovrapposizione), allora
    if (item != null) {
         // arricchisco il documento XML
         xml += " <overlay>\n"
             // con le proprietà di questa;
             + "  <path isaudio=\"" + (item.ItemType == PlaylistItem.MediaType.Audio
                   ? "true" : "false") + "\">" + item.Src + "</path>\n"
             + "  <gain>" + ((double)wp.AudioOverlayGain / 10).ToString(nfi) + "</gain>\n"
             + "  <fading>" + wp.OverlayFading.ToString() + "</fading>\n";
         // se inoltre la foglia non rappresenta un file multimediale, allora è una
         // sovrapposizione visuale e si continue ad arricchire il documento,
         if (item.ItemType != PlaylistItem.MediaType.Audio) {// Video options
              xml +=
                   "  <opacity>" + ((double)wp.VisualOverlayOpacity / 10).ToString(nfi)
              + "</opacity>\n"
              + "  <position>" + wp.VisualOverlayPosition.ToString() + "</position>\n"
              + "  <size>" + wp.VisualOverlaySize.ToString() + "</size>\n"
              + "  <orientation>" + wp.VisualOverlayOrientation.ToString()
                   + "</orientation>\n";
         }
         xml += " </overlay>\n";
    }
    // che viene completato, con la chiusura del tag radice, e restituito al chiamante.
    xml += "</newmedia>";
    return XElement.Parse(xml);
}
```

È importante notare che il metodo web del web service non accetta tipi `XElement`, ma `XmlDocument`, di cui il primo supporta LINQ, mentre il secondo no. Viceversa Silverlight non supporta `XmlDocument`, ma non c'è bisogno di alcuna conversione in quanto ciò che si trasmette tra applicazione e servizio web non sono degli oggetti, bensì una struttura XML, ossia del testo con una particolare sintassi, che poi sarà convertita in `XElement` o `XmlDocument` (e viceversa, in caso di spedizione) a seconda che si tratti di Silverlight o web service rispettivamente. Nel §2.1 si vedrà più a fondo come le strutture XML sono state gestite.

Sono stati creati tre metodi che ascoltano gli eventi di completamento delle trasmissioni tra client SOAP e web service: `ssclient_GetPlaylistCompleted`, `ssclient_NewMedia`

Completed e ssclient_DeleteCompleted. Durante gli eventi gestiti da questi due ultimi metodi e dopo il caricamento dell'applicazione (cfr. §1.2.1), sono i momenti in cui si reperiscono le informazioni per le playlist con una chiamata SOAP verso il metodo web service relativo (GetPlaylist); a completamento della risposta scatta l'evento ascoltato dal primo metodo appena elencato, che, se non si verificano errori invoca drawTree (si è visto poco fa in questo paragrafo) per disegnare l'albero della playlist.

```
void ssclient_GetPlaylistCompleted(object sender, GetPlaylistCompletedEventArgs e){
     try {
          // e.Result è il documento XML (come Xelement) spedito dal web service.
          XAttribute type = e.Result.Attribute("type");
          // Si gestiscono eventuali errori che ha riscontrato il web service
          if (type != null && type.Value != "info") {
               showMessage(messageTypes.errorPlaylist);
               return;
          }
          // Nel caso in cui la playlist è quella predefinita (non ha il nome utente come
          // attributo dell'elemento radice), si disegna il ramo corrispondente del controllo
          // per le playlist.
          if (e.Result.Attributes(USERNAME).Count() < 1) {
               drawTree(e.Result, TreePL.Items[0] as TreeViewItem);
          } else {
               // Altrimenti si disegna la playlist personale dell'utente, si riabilitano i
               // controlli associati e si cancellano i messaggi di attesa.
               drawTree(e.Result, TreePL.Items[1] as TreeViewItem);
               CtrlDelete.IsEnabled = TreePL.IsEnabled = true;
               showMessage(messageTypes.empty);
          }
     } catch (Exception ex) {
          // Si evitano gli errori non gestiti.
          abortSoapCall(messageTypes.errorPlaylist, false, ex);
     }
}
```

La gestione degli errori e il metodo di appoggio abortSoapCall saranno considerati nel prossimo paragrafo.

Gli altri due eventi SOAP sono gestiti da ssclient_NewMediaCompleted e ssclient DeleteCompleted; il primo scatta come risposta alle operazioni di estrazione e di sovrapposizione e chiama il metodo web service NewMedia, mentre il secondo come risposta dell'operazione di eliminazione e chiama Delete. Poiché sono identici (a eccezione dei messaggi d'errore), di seguito si mostra solo il primo.

```
void ssclient_NewMediaCompleted(object sender, NewMediaCompletedEventArgs e) {
     try {
          // Si gestiscono eventuali errori che ha riscontrato il web service
          if (e.Result.Attribute("type").Value == "info") {
               showMessage(messageTypes.loadingPlaylist);
               // si effettua una chiamata SOAP per aggiornare gli elementi della playlist
```

```
            // personale (la sola che l'utente può modificare)
            this.ssclient.GetPlaylistAsync(USERNAME);
        } else {
            showMessage(messageTypes.errorNewmedia, true);
        }
    } catch (Exception ex) {
        // Si evitano gli errori non gestiti.
        abortSoapCall(messageTypes.errorNewmedia, true, ex);
    }
}
```

1.2.6. Gestione errori e messaggi

È stato previsto anche un essenziale sistema per informare l'utente sia mentre attende che le sue operazioni di montaggio e gestione dei file vengano completate, sia per avvisarlo nel caso in cui manchino delle informazioni perché possa effettuarsi l'operazione scelta. In questo sistema sono inclusi anche i messaggi di errore incontrati dal web service, sia quelli non gestiti e incontrati durante la comunicazione tra client SOAP e web service. Il metodo principale è `showMessage`, che utilizza un tipo enumerazione per distinguere i messaggi da mostrare all'utente; il metodo si avvale sia della finestra preimpostata `System.Windows.MessageBox` per gli errori e gli avvisi (rispettivamente "*error*" e "*warning*"), sia di uno speciale riquadro (gestito dal metodo `toggleAsyncLayer`), creato in XAML, e nascosto, che appare per i messaggi di attesa e in caso di errori irreversibili; questo livello scompare ad attesa terminata, ma per gli errori rimane visibile, poiché ricopre tutta l'area dell'applicazione con uno sfondo traslucido che impedisce all'utente di interagire.

```
// il primo parametro è l'oggetto il tipo enumerazione, il secondo serve a cancellare i
// messaggi, il terzo per aggiungere del testo di un messaggio (in caso di errori
// irreversibili di chiamate SOAP).
void showMessage(messageTypes type, bool clear, string addMessage) {
    // Variabile che determina se far visualizzare (=true) il la finestra di sistema;
    bool isbox = false;
    // stringa per il messaggio.
    string s = "";
    switch (type) {
        // Messaggi di errore.
        case messageTypes.errorDelete:
            isbox = true;
            s = "Errore durante l'eliminazione.";
            break;
        case messageTypes.errorNewmedia:
            isbox = true;
            s = "Errore durante l'operazione.";
            break;
        case messageTypes.errorPlaylist:
            // Questo è un errore irreversibile, infatti isbox rimane =false, poiché se la
```

```
                // playlist non è caricata non ci sono i file per essere manipolati.
                s = "Errore durante il caricamento della playlist.";
                this.Cursor = Cursors.Arrow;
                break;
           // Messaggi di caricamento.
           case messageTypes.loadingClip:
                s = "Estraendo...";
                break;
           case messageTypes.loadingDelete:
                s = "Eliminando...";
                break;
           case messageTypes.loadingOverlay:
                s = "Sovrapponendo...";
                break;
           case messageTypes.loadingPlaylist:
                s = "Caricamento playlist...";
                break;
           // Messaggio di avviso.
           case messageTypes.warningSelection:
                isbox = true;
                s = "Prima selezionare un porzione temporale.";
                break;
           // Per tutti gli altri casi (messageTypes.empty), il riquadro scompare.
           default:
                toggleAsyncLayer("");
                break;
      }
      if (!isbox) {
           // Se non si usa la finestra di sistema, si utilizza il riquadro.
           toggleAsyncLayer(s);
      }
      // Aggiunge il messaggio personalizzato, se c'è.
      if (addMessage != "") s += "\n" + addMessage;
      // La finestra di sistema si vede anche con il messaggio personalizzato valorizzato.
      if (isbox || addMessage != "")
           MessageBox.Show(s);
      // In questo caso si fa scomparire il riquadro.
      if (clear) toggleAsyncLayer("");
}
```

Se si torna per un momento al paragrafo precedente si può notare che i metodi che ascoltano gli eventi SOAP hanno dei blocchi `try/catch` per catturare quegli errori imprevisti durante la comunicazione con il web service. In caso si verifichi un'eccezione questa viene gestita dal metodo di appoggio `abortSoapCall` che funge da tramite con `showMessage`, al quale passa il messaggio dell'eccezione (InnerException) come parametro di messaggio personalizzato e interrompe tutte le altre comunicazioni SOAP (dal momento che l'errore è irreversibile).

2. Web Service

Tutte le comunicazioni SOAP avvengono in maniera asincrona, permettendo al web service, una volta ricevute le istruzioni dal client, di operare in maniera indipendente e di rispondere quando è pronto; inoltre avvengono attraverso messaggi XML incartati nell'involucro SOAP durante la spedizione e scartati nella ricezione; questi messaggi fanno parte del metodo scelto di comunicazione e a loro volta possono contenere altro XML o semplice testo. Poiché il web service è stato configurato in modo tale da restituire sempre delle strutture XML, la logica applicativa di Silverlight le parserizza producendo oggetti `XElement` e quella del web service oggetti `XmlDocument` (cfr. §1.2.5). Il web service espone dei metodi accessibili da Internet che interagiscono, nella logica interna, con altri metodi di appoggio; tra questi sono stati creati dei gestori di strutture XML e XML Schema per manipolare e validare documenti XML, riducendo così le possibilità di errori imprevisti.

2.1. XML

Si è scelto di costruire tre diverse strutture XML, ognuna formalizzata da uno Schema [Binstock 2003], sia per ragioni di sicurezza, sia di chiarezza dell'organizzazione dei dati con altri possibili sviluppatori che interverranno (sono stati utilizzati anche dei namespace), e sia di utilità per la caratteristica modulare di XML.

2.1.1. Playlist

Per tenere traccia dei file multimediali associati a un utente e delle relative modifiche, si è pensato di creare dei documenti XML memorizzati sul file system e contenenti le informazioni necessarie. Si è deciso di chiamare questi documenti "playlist" in quanto contengono delle liste di file multimediali da essere riprodotti in un apposito lettore. I dati memorizzato sono il nome dell'utente associato (a eccezione che sia una playlist condivisa da tutti gli utenti – per quei contenuti multimediali predefiniti) e la lista dei file con il percorso relativo alla cartella radice del sito (in questo modo è possibile una gestione biunivoca: lato client basta aggiungere all'inizio del percorso l'host e lato server il percorso sul file system) e una breve descrizione (attualmente non utilizzata, ma potrebbe tornare utile in futuro), organizza in categorie (secondo la tipologia dei contenuti, ma il criterio potrà in futuro variare, dando anche possibilità di organizzazione personalizzata per ogni utente). Di seguito un esempio di file con questa struttura ("playlist").

```
<?xml version="1.0" encoding="utf-8"?>
<!-- L'attributo username indica il nome dell'utente. -->
<playlist xmlns="urn:schemas-torino:playlist" username="username">
  <!-- Ecco una prima categoria con il suo nome,  -->
  <category name="audio (wma)">
    <!-- e la lista dei file multimediali associati. -->
    <media src="/Playlists/Audio/pas02b.wma" desc="" />
    <media src="/Playlists/Audio/eccezionale.wma" desc="" />
    <media src="/Playlists/Audio/canone.wma" desc="" />
```

```
    <media src="/Playlists/Audio/taglio-1.wma" desc="" />
  </category>
  <category name="video (wmv)">
    <media src="/Playlists/Video/introduzione.wmv" desc="" />
  </category>
</playlist>
```

2.1.2. Nuovo file multimediale

Quest'altra struttura ("newmedia") XML invece non è memorizzata sul file system, ma serve per organizzare le istruzioni, spedite dell'applicazione Silverlight, della creazione di un nuovo file multimediale in seguito a un'estrazione o a una sovrapposizione.

```
<?xml version="1.0" encoding="utf-8" ?>
<newmedia xmlns="urn:schemas-torino:newmedia">
    <!-- 1) nome del nuovo file; -->
    <filename>montaggio-1</filename>
    <!-- 2) file di base, con percorso (path) e selezione temporale (selection, con valori di
            inizio [start] e fine [end] decimali da 0 a secondi di durata del file). -->
    <base>
        <path isaudio="false">/Playlists/Audio/videopredefinito.wmv</path>
        <selection start="0" end="8.987" />
    </base>
    <!-- 3) File per la sovrapposizione, con percorso (path) e a seguire le opzioni. -->
    <overlay>
        <path isaudio="false">/Playlists/Video/introduzione.wmv</path>
        <!-- Volume; -->
        <gain>1</gain>
        <!-- tipo di sfumatura - valori possibili: none, fadein, fadeout, both; -->
        <fading>fadein</fading>
        <!-- livello di opacità - un valore decimale da 0 a 1; -->
        <opacity>1</opacity>
        <!-- posizione all'interno dell'area del file di base – valori possibili: topleft,
             top right, center, bottomleft, bottomright (ossia agli angoli o al centro); -->
        <position>center</position>
        <!-- dimensione relativa rispetto all'area del file di base: è il numeratore di una
             frazione in dodicesimi, per cui i valori sono interi e vanno da 1 a 12; -->
        <size>3</size>
        <!-- il ridimensionamento avviene considerando l'asse verticale (valorizzato con
             "vertical") dell'area di base o quella orizzontale (horizontal). -->
        <orientation>horizontal</orientation>
    </overlay>
</newmedia>
```

La struttura è divisa in tre parti: 1) impostazione del nome del file; 2) il percorso del file di base e la selezione di un suo intervallo temporale; 3) il percorso del file che si sovrappone alla base, con le varie impostazioni di effetti, posizionamento, ridimensionamento, e così via. Le

sezioni 2) e 3) vengono utilizzate in maniera diversa a seconda che si tratti di un'estrazione o di una sovrapposizione. Nel primo caso la sezione 3) deve essere omessa e dalla sezione 2) si ricavano le informazioni necessarie per estrarre un intervallo di tempo dal file e crearne uno nuovo. Nell'ultimo caso invece la sezione 3) serve per associare il file di sovrapposizione su quello base della sezione 2), in un determinato intervallo di tempo di questa, e applicarvi le varie proprietà. Se l'attributo `isaudio` di 3) è valorizzato a vero allora si tratta di una sovrapposizione audio, altrimenti visuale; infine, se l'attributo `isaudio` di 2) è uguale a vero allora saranno accettate solo sovrapposizioni di uguale formato, altrimenti anche visuali.

2.1.3. Messaggi

I metodi del web service possono rispondere con le strutture XML viste sopra oppure con una struttura ("message") molto semplice di messaggi generici.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- L'elemento radice con l'unico attributo che può essere valorizzato con error, warning o
     info. -->
<message xmlns="urn:schemas-torino:message" type="info">
  <![CDATA[Operazione avvenuta con successo!]]>
</message>
```

C'è solo un elemento, la radice, con un unico attributo, il tipo di messaggio, che può essere di errore, di avviso o di informazione (come nell'esempio); il testo del messaggio è sempre incapsulato in un campo CDATA, in modo tale che non si verifichino problemi con caratteri XML riservati.

2.2. Metodi web

La logica del web service è strutturata in tre parti: tre metodi pubblici esposti all'esterno; vari metodi generici di supporto; e metodi di supporto per trattare informazioni in XML. La logica interagisce, da una parte, con Internet, e dall'altra con il file system (cfr. figura a p. 26); il file system è utilizzato come una sorta di base dati in cui vengono utilizzati dei file XML per organizzare. Sul file system si possono individuare due sezioni logiche: una per i file XML e XML Schema (cfr. §2.1.1), l'altra per quelli multimediali. I primi organizzano i secondi, i quali, a livello fisico, sono memorizzati in cartelle apposite, una per categoria (audio, video, immagini), più un'altra (al di fuori della gestione XML) usata come cartella temporanea per la codifica di un nuovo file. I tre metodi esposti a Internet (distinti nel codice con l'attributo `[WebMethod()]`) sono `GetPlaylist`, `Delete` e `NewMedia`, affrontati nei prossimi paragrafi.

2.2.1. Playlist

Si è già più volte visto come i file multimediali sono organizzati in playlist, per mezzo di documenti XML. Si può aggiungere che ogni playlist ha il proprio nome come file fisico che corrisponde al nome dell'utente, nel caso delle playlist personalizzate, mentre per quella dei contenuti predefiniti ha un nome fisso preimpostato a "original" (l'estensione ".xml" è sottintesa). Questo metodo web accetta in ingresso, come unico parametro, una stringa contenente il

nome della playlist da inviare all'applicazione.

```
[WebMethod()]
public XmlDocument GetPlaylist(string playlistName) {
     XmlDocument d = new XmlDocument();
     try {
          // Si carica la playlist dal file system.
          d.Load(mapFilePath(playlistName, plpath));
          // All'interno della logica del web service si può utilizzare LINQ (vedere il tipo
          // XDocument sottostante), ma non come tipo di risposta verso Internet; qui sotto si
          // valida la playlist caricata.
          validatePlaylist(XDocument.Parse(d.OuterXml));
     } catch (Exception ex) {
          xmlMessage(d, ex.Message, messageType.error);
     }
     return d;
}
```

Il metodo di supporto generico `mapFilePath` fa sì che si ottenga, dal nome della playlist, il corrispondente percorso sul file system; invece il metodo di supporto XML `validate Playlist` verifica che il file XML caricato sia conforme allo Schema relativo. Le eccezioni e i metodi per i messaggi, come `xmlMessage`, saranno discussi nel §2.2.4.

2.2.2. Eliminazione

Quest'altro metodo web cancella alcuni file multimediali, associati a un utente, dal file system, e aggiorna il file XML che li organizza. L'unico parametro di ingresso è una stringa contenente i percorsi relativi dei file separati dal carattere dei due punti ":".

```
[WebMethod()]
public XmlDocument Delete(string filenames) {
     XmlDocument doc = new XmlDocument();
     try {
          int i, n;
          // Si individua il luogo della playlist nel file system.
          string plsrc = mapFilePath(USERNAME, plpath);
          // Si crea un array dal parametro di ingresso.
          string[] fnlist = filenames.Split(':');
          // Documento XML con supporto di LINQ
          XDocument xdoc = XDocument.Load(plsrc);
          XNamespace ns = xdoc.Root.GetDefaultNamespace();
          XElement xel;
          n = fnlist.Length;
          // Si cicla sull'array dei percorsi dei file.
          for (i = 0; i < n; i++) {
               // Query LINQ per trovare nella playlist l'elemento avente nell'attributo src
               // il percorso del file.
               var q = from c in xdoc.Descendants(ns + "media")
```

```
                    where c.Attribute("src").Value == fnlist[i]
                    select c;
               // Se la query non trova niente si esce con un errore,
               if (q.Count() < 1)
                    throw new Exception("Could not find the selected media item.");
               // altrimenti si cancella il file e il corrispondente elemento nella playlist.
               xel = q.First();
               System.IO.File.Delete(mapFilePath(xel.Attribute("src").Value));
               xel.Remove();
          }
          // Si salva il documento della playlist e si crea il messaggio da spedire al client
          xdoc.Save(plsrc);
          xmlMessage(doc, "success", messageType.info);
     } catch (Exception ex) {
          xmlMessage(doc, ex.Message, messageType.error);
     }
     return doc;
}
```

2.2.3. Nuovo file multimediale

Il processo di creazione di un nuovo file multimediale si struttura in quattro parti: 1) validazione e conversione in oggetti del documento XML contenente le informazioni per il risultato finale; 2) inizializzazione delle proprietà principali del nuovo elemento multimediale; 3) impostazione delle proprietà specifiche per 3a) una sovrapposizione (*audio overlay*) audio, o 3b) una sovrapposizione visuale (*visual overlay*), o 3c) un'estrazione (*clip*) di audio/video in un certo intervallo di tempo, o 3d) un'estrazione invertita, per cui l'intervallo di tempo selezionato sarà escluso dal nuovo file; 4) inizializzazione del processo di codifica del nuovo elemento, codifica di questo nella cartella di lavoro dedicata, e spostamento del nuovo file nella relativa cartella di categoria. Il metodo web accetta in ingresso il documento XML "newmedia" la cui struttura è stata discussa nel §2.1.2.

```
[WebMethod()]
public XmlDocument NewMedia(XmlDocument newMediaDocument) {
     XmlDocument doc = new XmlDocument();
     try {
          // 1) validazione e conversione in oggetti del documento XML e
          newmediaReader read = new newmediaReader(
               Xdocument.Parse(newMediaDocument.OuterXml),
               // localizzazione nel fyle sistem del rispettivo schema.
               mapFilePath("/Schemas/newmedia.xsd"));
          if (!read.IsValid) throw new Exception(read.Error);

          // 2) Inizializzazione delle proprietà principali del nuovo elemento multimediale:
          mediaPath outputPath = (read.BasePathIsaudio ? apath : vpath);
          // creazione di un nuovo elemento multimediale, ottenuto dal file base del doc. XML;
```

```
mxe.MediaItem item = new mxe.MediaItem(mapFilePath(read.BasePath));
// il file sarà codificato secondo un formato Windows Media;
item.OutputFormat = new mxe.WindowsMediaOutputFormat();
item.OutputFileName = read.Filename + outputPath.Ext;

// se non è stato riconosciuto il profilo audio del file originale
// se ne imposta uno predefinito (WMA);
if (item.SourceAudioProfile != null) {
     item.OutputFormat.AudioProfile = item.SourceAudioProfile;
} else {
     item.OutputFormat.AudioProfile = new mxep.WmaAudioProfile();
}
// se il file base non è un audio, e
if (!read.BasePathIsaudio) {
     // non ne è stato riconosciuto il profilo video, allora
     // si imposta uno predefinito (VC1) con un bitrate costante a 1000kbs.
     if (item.SourceVideoProfile != null) {
          item.OutputFormat.VideoProfile = item.SourceVideoProfile;
     } else {
          item.OutputFormat.VideoProfile = new mxep.AdvancedVC1VideoProfile() {
               Size = item.MainMediaFile.VideoStreams[0].VideoSize,
               Bitrate = new mxep.ConstantBitrate(1000)
          };
     }
}

// 3) Impostazione delle proprietà specifiche per
// [valore preimpostato a un secondo per gli effetti di sfumatura audio]
TimeSpan fd = TimeSpan.FromSeconds(1.0);
if (read.IsOverlay) {
     // 3b) una sovrappone video:
     if (read.IsVisualOverlay) {
          // poiché si è identificata una sovrapposizione video, se il file base è
          // audio, si genera un errore;
          if (read.BasePathIsaudio) throw new
          Exception("Si può sovrapporre un video solo su un altro video.");
          // dichiarazione delle variabili di dimensione e di posizione dello
          // strato da sovrapporre;
          int x, y, w, h;
          // si assegna il nome del file da usare come strato,
          item.OverlayFileName = mapFilePath(read.OverlayPath);
          // il periodo in cui visualizzarlo,
          item.OverlayStartTime = TimeSpan.FromSeconds(read.BaseSelectionStart);
          item.OverlayEndTime = TimeSpan.FromSeconds(read.BaseSelectionEnd);
          // il volume sonoro,
```

```
                item.OverlayAudioGainLevel = read.OverlayGain;
                // gli eventuali effetti audio di sfumatura,
                switch (read.OverlayFading) {
                     case "fadein":
                          item.OverlayFadeInDuration = fd;
                          break;
                     case "fadeout":
                          item.OverlayFadeOutDuration = fd;
                          break;
                     case "both":
                          item.OverlayFadeInDuration =
                          item.OverlayFadeOutDuration = fd;
                          break;
                     default:// none
                          break;
                }
                // l'opacità dello strato,
                item.OverlayOpacity = read.OverlayOpacity;
                // le dimensioni relative all'asse verticale (divisa in 12 parti) del
                // video base,
                if (read.OverlayOrientation == "vertical") {
                     h = item.VideoSize.Height * read.OverlaySize / 12;
                     w = (int)Math.Round((double)h * item.OverlayRect.Width
                          / item.OverlayRect.Height);
                // oppure le dimensioni rispetto all'asse orizzontale (stessa divisione),
                } else {
                     w = item.VideoSize.Width * read.OverlaySize / 12;
                     h = (int)Math.Round((double)w * item.OverlayRect.Height
                          / item.OverlayRect.Width);
                }
                // posizione del video-strato sull'area del video-base.
                switch (read.OverlayPosition) {
                     case "topleft":
                          x = y = 0;
                          break;
                     case "topright":
                          y = 0;
                          x = item.VideoSize.Width - w;
                          break;
                     case "center":
                          y = (int)Math.Round((double)(item.VideoSize.Height - h) / 2);
                          x = (int)Math.Round((double)(item.VideoSize.Width - w) / 2);
                          break;
                     case "bottomleft":
                          y = item.VideoSize.Height - h;
```

```
                    x = 0;
                    break;
               default:// bottomright
                    y = item.VideoSize.Height - h;
                    x = item.VideoSize.Width - w;
                    break;
          }
          item.OverlayRect = new System.Drawing.Rectangle(x, y, w, h);
     // 3a) una sovrapposizione audio:
     } else {
          // assegnazione del file audio da usare come strato sovrapposto;
          item.AudioOverlayFileName = mapFilePath(read.OverlayPath);
          // periodo in cui si sente la sovrapposizione;
          item.AudioOverlayStartTime =
               TimeSpan.FromSeconds(read.BaseSelectionStart);
          item.AudioOverlayEndTime = TimeSpan.FromSeconds(read.BaseSelectionEnd);
          // volume sonoro dello strato;
          item.AudioOverlayGainLevel = read.OverlayGain;
          // eventuali effetti di sfumatura.
          switch (read.OverlayFading) {
               case "fadein":
                    item.AudioOverlayFadeInDuration = fd;
                    break;
               case "fadeout":
                    item.AudioOverlayFadeOutDuration = fd;
                    break;
               case "both":
                    item.AudioOverlayFadeInDuration = fd;
                    item.AudioOverlayFadeOutDuration = fd;
                    break;
               default:// none
                    break;
          }
     }
} else {
     // 3d) Un'estrazione invertita:
     if (read.BaseSelectionIsinverted) {
          // memorizzazione del contorno finale della selezione temporale;
          mxe.Clip c = new mxe.Clip(
               TimeSpan.FromSeconds(read.BaseSelectionEnd),
               item.Sources[0].Clips[0].EndTime);
          // per ogni elemento multimediale c'è inizialmente una porzione temporale
          // (clip = taglio), che corrisponde con la durata complessiva
          // dell'elemento, per cui si va a ridurre il termine della porzione
          // iniziale con l'inizio della selezione;
```

```
                    item.Sources[0].Clips[0].EndTime =
                         TimeSpan.FromSeconds(read.BaseSelectionStart);
                    // e si aggiunge la porzione salvata (in questo modo l'intervallo di tempo
                    // selezionato – espresso nel documento XML – viene eliminato).
                    item.Sources[0].Clips.Add(c);
               // 3c) Un'estrazione
               } else {
                    // si reimposta la porzione di tempo originale con i parametri di
                    // selezione.
                    item.Sources[0].Clips[0].StartTime =
                         TimeSpan.FromSeconds(read.BaseSelectionStart);
                    item.Sources[0].Clips[0].EndTime =
                         TimeSpan.FromSeconds(read.BaseSelectionEnd);
               }
          }
          // 4) inizializzazione del processo di codifica del nuovo elemento,
          mxe.Job job = new mxe.Job();
          // si imposta la cartella di lavoro, senza creare sottocartelle,
          job.OutputDirectory = Server.MapPath(WORKPATH);
          job.CreateSubfolder = false;
          // si associa l'elemento multimediale appena impostato
          job.MediaItems.Add(item);
          // e si inizia con il processo di codifica,
          job.Encode();
          // infine si sposta il file codificato dalla cartella di lavoro alla relativa
          // cartella di categoria.
          System.IO.File.Move(Server.MapPath(WORKPATH + read.Filename + outputPath.Ext)
               , Server.MapPath(outputPath.Folder) + read.Filename + outputPath.Ext);
          // Si crea il messaggio di risposta, dopo aver aggiornato la playlist (metodo
          // updatePlaylist, che si avvale di LINQ), che viene incapsulata.
          xmlMessage(doc, updatePlaylist(read.Filename, outputPath), messageType.info);
     } catch (Exception ex) {
          xmlMessage(doc, ex.Message, messageType.error);
     }
     return doc;
}
```

Per questo metodo sono state utilizzate le librerie dall'SDK di Microsoft Expression Encoder, che permettono di gestire la codifica multimediale [Microsoft 2009]. Come si è visto in questo lungo esempio di codice sorgente, sono stati utilizzati dei profili audio e video, perché necessari per mantenere le impostazioni di codifica iniziali dei file multimediali. Si è scelto di utilizzare solo formati Windows Media (quindi WMA e WMV) dal momento che sono pienamente supportati dall'SDK; quindi le condizioni per cui, nel codice d'esempio, si impostano dei profili predefiniti se non riconosciuti quelli originali non dovrebbero essere mai verificate.

Inoltre nell'esempio è stata introdotta la classe `newmediaReader`, che valida e trasforma il documento XML "newmedia" in una struttura a oggetti, avente proprietà in sola lettura. Di se-

guito si propone l'esempio del costruttore.

```
// La classe viene inizializzata con il documento XML e l'indirizzo del rispettivo Schema.
public newmediaReader(XDocument doc, string schemaUri) {
     // Si inizializza la classe per lo Schema,
     XmlSchemaSet schema = new XmlSchemaSet();
     schema.Add(null, schemaUri);
     // quindi si valida il documento con lo Schema, passando alla funzione la classe con lo
     // Schema e una funzione, dichiarata da un'espressione lambda, che viene invocata in caso
     // di errore, e da cui si memorizza in due campi, che poi saranno letti da due proprietà
     // di questa classe, il messaggio d'errore e la conferma dell'invalidità del documento;
     doc.Validate(schema , (o, e) => {
          error = e.Message;
          isvalid = false;
     });
     // se non è valido si esce.
     if (!isvalid) return;
     // Si impostano dei campi poi usati per impostare le proprietà (che sono tutte in sola
     // lettura).
     this.doc = doc;
     this.ns = doc.Root.GetDefaultNamespace();
     this._base = doc.Root.Element(this.ns + "base");
     // Si evince il tipo di operazione: estrazione o sovrapposizione,
     if (this.isoverlay = (doc.Root.Element(this.ns + "overlay") != null)) {
          this._overlay = doc.Root.Element(this.ns + "overlay");
          // sovrapposizione video o audio.
          this.isvisualoverlay = (this._overlay.Element(this.ns + "opacity") != null
               && !this.OverlayPathIsaudio ? true : false);
     }
}
```

2.2.4. Gestione errori e messaggi

I messaggi sono gestiti da una funzione (`xmlMessage`) che incapsula il messaggio all'interno di codice XML, nel formato descritto nel §2.1.3, creando un documento che poi viene spedito dal web service come risposta. Come protezione da errori nel codice XML si utilizzano le validazioni con Schema; per catturare gli errori, sia previsti (generati con "`throw new Exception([messaggio])`") sia imprevisti, in ogni metodo web sono stati inseriti blocchi `try/catch` che creano dei messaggi d'errore con il metodo `xmlMessage`.

3. Il prodotto finale

Il prototipo è stato chiamato con il nome in codice “Torino”.

È un sito web composto, dalla cartella radice, da

1. una pagina XHTML statica (“Default.htm”) avente il collegamento con l’oggetto Silverlight,
2. una pagina ASP .NET Web Service (“Service.asmx”),
3. e da un file di configurazione del web service (“Web.config”);
4. una cartella (“bin”) contenente il file binario del web service (“MediaService.dll”);
5. una cartella (“Playlist”) contenente
    1. i documenti XML che organizzano i file multimediali secondo una playlist predefinita e una personalizzata per l’utente (rispettivamente “original.xml” e “username.xml”), e
    2. una cartella (“Audio”) con tutti i file audio (“*.wma”),
    3. una cartella (“Video”) con tutti i file video (“*.wmv”),
    4. una cartella (“Images”) con tutte le immagini (“*.jpg”) e
    5. una cartella (“Working”) temporanea che può contenere sia file audio sia video, usata durante la codifica per la creazione di nuovi file;
    6. una cartella (“Schemas”) con dei documenti XML Schema (“message.xsd”, “newmedia.xsd” e “playlist.xsd”) che definiscono formalmente la struttura dei documenti XML di interscambio tra client e web service;
6. infine un’ultima cartella contenente l’applicazione Silverlight (“Torino.xap”).

Torino.xap è un file compresso ZIP, al cui interno sono contenuti dei file binari (“Torino.dll”, e altre librerie di supporto), un documento XML di configurazione dell’applicazione (“AppManifest.xaml”) e un altro documento XML invece di configurazione del collegamento col web service, dove viene specificato l’accesso con l’endpoint, dal quale un metodo della classe `MainPage` dell’applicazione ne ricava l’host in cui questa è ospitata.

```
<!-- [...] -->
<client>
     <!-- Modificare l’attributo address con l’attuale indirizzo URL del web service. -->
     <endpoint address="http://dominio/Service.asmx" binding="basicHttpBinding"
      bindingConfiguration="ServiceSoap" contract="MediaService.ServiceSoap"
      name="ServiceSoap" />
</client> <!-- [...] -->
Uri makeAbsoluteUri(string relativePath) {
     // ssclient è un’istanza del client SOAP.
     return new Uri("http://" + ssclient.Endpoint.Address.Uri.Host
          + ":" + ssclient.Endpoint.Address.Uri.Port + relativePath);
}
```

3.1. Esempio di messaggi SOAP

Ora si propone un esempio che mostra la comunicazione SOAP/HTTP tra browser e web service. Richiesta HTTP POST del client.

```
POST http://localhost:49653/Service.asmx HTTP/1.1
Host: localhost:49653
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.3) Gecko/20100401
            Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Content-Length: 238
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/GetPlaylist"
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <GetPlaylist xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="http://tempuri.org/">
        <playlistName>original</playlistName>
    </GetPlaylist>
  </s:Body>
</s:Envelope>
```

Si può notare come in questo caso il prefisso per il namespace SOAP sia "`s:`", l'elemento radice è `envelope`, non c'è il figlio `header`, ma c'è `body`; il sistema del web service vi incapsula quindi un messaggio XML avente come radice il nome del metodo web invocato e come figlio i parametri d'ingresso (qui uno solo); c'è la testata `SOAPAction` che specifica il tipo di metodo web da invocare (tempuri.org è un namespace/URL predefinito); ASP .NET Web Services ammette tag *case insensitive*. Risposta HTTP del web service.

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/9.0.0.0
Date: Sun, 11 Apr 2010 23:45:01 GMT
X-AspNet-Version: 2.0.50727
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 687
Connection: Close
<?xml version="1.0" encoding="utf-8"?>
<soap:envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:body>
    <getplaylistresponse xmlns="http://tempuri.org/">
      <getplaylistresult>
        <playlist xmlns="urn:schemas-torino:playlist">
          <category name="audio (wma)">
            <media src="/Playlists/Audio/baroqueloop90z.wma" desc=""/>
```

```
            <media src="/Playlists/Audio/audiosample.wma" desc=""/>
          </category>
          <category name="video (wmv)">
            <media src="/Playlists/Video/Wildlife.wmv" desc=""/>
          </category>
        </playlist>
      </getplaylistresult>
    </getplaylistresponse>
  </soap:body>
</soap:envelope>
```

Anche in questo caso non è presente l'elemento header; il messaggio XML annidato ha la radice col nome del metodo web seguito da "response" e un figlio con lo stesso nome ma seguito da "result", che sta a indicare che cosa il metodo web ha restituito, che qui è un documento XML.

3.2. Nota conclusiva

I problemi cardine che si sono affrontati sono stati 1) un alto grado di interattività, 2) capacità di fruizione di contenuti multimediali (decodifica di questi per la riproduzione), 3) capacità di manipolazione di contenuti multimediali (codifica di questi) e 4) rendere il punto precedente compatibile con il primo. I punti 1) e 2) sono stati soddisfatti grazie a Silverlight e le sue capacità, mentre il punto 3) per mezzo delle librerie dell'SDK di Expression Encoder. Applicare il punto 4) interamente nell'applicazione Silverlight sarebbe stato impossibile, in quanto l'applicazione risiede sul client e avrebbe dovuto accedere al file system dell'utente per la scrittura, cosa non permessa per ragioni di sicurezza; in questo modo l'utente avrebbe dovuto esporre la propria macchina alle operazioni di codifica e allo scaricamento delle relative librerie e dei file multimediali. Tutti processi, questi, che consumano banda di rete e risorse della sua macchina; invece sfruttare la natura distribuita di internet significa lasciare all'utente i controlli per comandare comodamente qualcosa che sta a distanza e configurato apposta per servirlo. Qui entra in gioco ASP. NET Web Services, con SOAP e le comunicazioni asincrone.

Queste dunque le risposte per soddisfare anche il punto 4), ma va considerato anche un altro aspetto. La manipolazione di video richiede un processo di codifica che impiega molto tempo (anche decine di minuti, a seconda delle dimensioni del file), ma ciò non dipende né dall'interfaccia, né dal servizio web, ma piuttosto dalle librerie di codifica. L'utente che naviga nel web non è abituato ad aspettare diversi minuti perché si completi qualcosa, pertanto in un futuro si potrebbero applicare delle migliorie per ridurre il tempo di attesa.

Risorse di consultazione

[**Binstock 2003**] C. Binstock, D. Peterson, S. Mitchell, M. Wooding, C. Dix e C. Galtenberg, *The XML schema complete reference*, Addison-Wesley, Boston, MA, USA, 2003.
[**Bochicchio 2009**] D. Bochicchio, C. Civera, A. Leoncini e M. Leoncini, *Silverlight 3.0*, Hoepli, Milano, 2009.
[**Goodman 2008**] D. Goodman, *Dynamic HTML : the definitive reference*, O'Reilly, Sebastopol, CA, USA, 2008.
[**Horstmann 2008**] C. Horstmann e T.A. Budd, *Big C++*, John Wiley, Hoboken, NJ, USA, 2008.
[**Microsoft 2009**] "Expression Studio 3 (x86)" (DVD), *Expression Encoder SDK User Guide*, Microsoft, 2009.
[**Microsoft 2010**] Microsoft, *Expression Blend Training Videos*, 2010, http://expression.microsoft.com/en-us/cc197141.aspx.
[**Moroney 2008**] L. Moroney, *Beginning Web Development, Silverlight, and ASP.NET AJAX: From Novice to Professional*, Apress, New York, NY, USA, 2008.
[**MSDN 2010a**] Microsoft Developer Network, *Silverlight 3*, 2010, http://msdn.microsoft.com/en-us/library/cc838158%28VS.95%29.aspx.
[**MSDN 2010b**] Microsoft Developer Network, *Silverlight Architecture*, 2010, http://msdn.microsoft.com/en-us/library/bb404713(v=VS.95).aspx.
[**Piller 2010**] Piller M, *Introduction to Flex 4 and .NET Integration*, 2010, http://cookbooks.adobe.com/post_Introduction_to_Flex_4_and__NET_Integration-16930.html.
[**Prospettiva 2009**] Teatro Stabile di Torino, *Prospettiva 09*, 2009, http://prospettiva.teatrostabiletorino.it/.
[**Schildt 2006**] H. Schildt, *C# 2.0: the complete reference*, McGraw-Hill/Horborne, New York, USA, 2006.
[**Wikipedia 2009**] Wikipedia, *Adobe Flash*, 2009, http://en.wikipedia.org/wiki/Adobe_Flash.