Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Sede di Modena

Corso di Laurea in Ingegneria Informatica – *Nuovo Ordinamento*

Reingegnerizzazione del Manifesto degli Studi con tecnologie AJAX

Relatore:
Prof. Sonia Bergamaschi

Candidato:
S.Ten. Jacopo Bestetti

Correlatore:
Prof. Maurizio Vincini

Anno Accademico 2008 - 2009

Parole chiave:

Portale

WebDB

AJAX

ASP

Indice

Elenco delle figure

Capitolo I

Introduzione

Il presente lavoro di tesi è frutto dell'attività di studio effettuata sul Manifesto degli Studi del portale della Facoltà di Ingegneria dell'Università di Modena e Reggio Emilia.

L'obiettivo è quello di studiare la reingegnerizzazione dei servizi offerti dalle pagine web attraverso le tecnologie AJAX al fine di migliorarne le prestazioni: tale studio viene poi tradotto nella realizzazione di uno dei servizi offerti dal portale, quello relativo all'offerta didattica, con la nuova tecnologia.

I capitoli II e III di questo documento introducono gli argomenti sviluppati e illustrano i fondamenti teorici su cui si basa il lavoro svolto, approfondito nel capitolo IV.

È utile illustrare brevemente la struttura del documento e l'organizzazione dei contenuti.

Capitolo II – In questo capitolo si descrive inizialmente il dominio di lavoro e il contesto applicativo. Si illustrano così le caratteristiche del portale della Facoltà e del Manifesto degli Studi. Successivamente si evidenziano le caratteristiche più importanti della tecnologia ASP, con la quale è realizzata attualmente la pagina web in questione. Si darà prima una definizione, poi si descriveranno le peculiarità e le funzionalità delle Active Server Pages. Da ultimo infine si presenteranno brevemente le caratteristiche della pagina

web attuale, *Manifesto.asp*, e della sua versione frutto dalla rielaborazione effettuata, *ManifestoAJAX.asp*.

Capitolo III – Questo capitolo è dedicato interamente alla descrizione approfondita delle tecnologie AJAX. Esse infatti non possono essere definite una vera e propria tecnologia, ma si parla piuttosto di una tecnica di programmazione che utilizza tecnologie informatiche già comunemente diffuse. Da una prima definizione e una contestualizzazione storica, l'attenzione si sposta verso l'illustrazione sui pregi e i difetti di questo insieme di tecnologie. Infine, si completa l'approfondimento dedicato ad AJAX con una descrizione dell'oggetto XMLHttpRequest, che è il cuore del funzionamento di questa tecnica.

Capitolo IV – Questo capitolo presenta la descrizione dettagliata di tutte le operazioni svolte per compiere la reingegnerizzazione del Manifesto degli Studi. Si effettueranno alcune considerazioni di carattere realizzativo, confrontando il codice sorgente originario della pagina web e quello realizzato come obiettivo del lavoro di tesi. La prima parte del capitolo è dedicata esclusivamente all'illustrazione delle modifiche apportate e delle novità introdotte. Si effettua un confronto diretto tra gli elementi costitutivi della pagina web nelle due versioni, si introducono specifiche ipotesi progettuali, sulla base delle quali si opera un rinnovamento della veste grafica del Manifesto degli Studi. La seconda parte invece presenta inizialmente un approfondimento sulla funzione AJAX, responsabile del recupero asincrono dei contenuti informativi relativi ad un certo corso di studi universitario. Successivamente si offrono delle considerazioni sul costo computazionale, che giustificano il netto miglioramento delle prestazioni a seguito dell'adozione delle tecnologie AJAX all'interno del codice della pagina web in questione. Si completa l'approfondimento illustrando i file di appoggio realizzati di proposito per consentire la corretta riuscita del lavoro.

Capitolo II

Il portale della Facoltà di Ingegneria e il Manifesto degli Studi

2.1 Il portale della Facoltà di Ingegneria

Il portale *www.ing.unimore.it* della Facoltà di Ingegneria dell'Università di Modena e Reggio Emilia è pensato e costruito per gli utenti della Facoltà, ed è necessario che si presti alle esigenze dei navigatori in modo adeguato e completo. Esso ricopre notevoli funzioni, e con la sempre più ampia e costante diffusione dell'informatizzazione dei contenuti tende a ricoprire un ruolo centrale nel rapporto che si deve necessariamente instaurare tra l'utente e l'Università.

A fronte dell'organizzazione dei contenuti che il sito offre, per la sua realizzazione si è pensato allora all'utilizzo di Active Server Pages. Questa implementazione permette una gestione più pronta e veloce delle informazioni, e risulta molto adeguata nel momento in cui la gran parte dei contenuti che si offrono agli utenti debba essere prelevata da un database, che, nel caso specifico, è quello della Facoltà. La scelta adottata, come conseguenza delle specifiche tecniche della tecnologia ASP, consente una perfetta interazione con un database, come avviene nel caso più generale di qualsiasi pagina del sito web della Facoltà, e come, più nel particolare, verrà illustrato di seguito all'interno della

pagina del Manifesto degli Studi della Facoltà di Ingegneria, dominio di studio e di elaborazione progettuale.

È importante sottolineare preliminarmente che l'implementazione adottata per la realizzazione del sito web corrente non include però un importante strumento in possesso dei programmatori, che permette indubbiamente di diminuire il tempo di attesa della risposta del e della sua successiva elaborazione da parte del browser. Stiamo parlando dell'insieme di tecnologie che vengono raccolte sotto il nome di AJAX. È degno di nota il fatto che ormai sempre più siti web utilizzano tale insieme di tecnologie, e che quindi è fondamentale che anche la stessa Università e soprattutto la Facoltà di Ingegneria restino al passo con l'innovazione.

2.2 Il Manifesto degli Studi

Il Manifesto degli Studi della Facoltà di Ingegneria è uno strumento molto importante per colui che lo utilizza e molto interessante per colui che lo implementa.

È uno dei servizi offerti dal portale della Facoltà, ma si distingue assieme ad altri pochi servizi per la centralità del suo ruolo.

Organizzato per corso di laurea, la sua funzione è quella di presentare tutte le informazioni relative ai corsi in atto di quel tale indirizzo di studi, riportando, suddivisi per anno di corso, e a loro volta per periodo, tutti gli insegnamenti previsti, siano essi obbligatori, facoltativi o fortemente consigliati. Relativamente a ciascun insegnamento individuato per il periodo di corso, di cui viene riportato anche l'orientamento temporale, la pagina web riporta i corrispettivi docenti che insegnano in tali corsi e inoltre i crediti universitari con cui viene valutato ciascun insegnamento.

Ulteriormente a queste informazioni, il Manifesto degli Studi riporta le eventuali note relative all'intero corso di studi, o relative ad un particolare anno o periodo.

Il Manifesto degli Studi diventa inoltre punto di partenza verso la navigazione delle pagine individuali dei docenti e degli insegnamenti, ed è necessario che la sua funzione possa essere parafrasata come un 'trampolino di lancio', ovvero uno strumento utile e immediato di guida e di indicazione.

La centralità e l'importanza scaturite dalla sua funzione rendono necessari tutti gli accorgimenti per proporre all'utenza un risultato sempre migliore, che sappia far fronte alla costante complessità dell'organizzazione didattica e alla rapida e incalzante riforma del ciclo universitario, sapendo rispondere con velocità all'enorme quantità di dati e informazioni richieste dall'utente, consentendogli di poter usufruire di uno strumento che sia all'altezza delle sue aspettative.

Da un utilizzo frequente di questo servizio si è notato come l'attuale versione soffra di prontezza nel rispondere alle richieste del client. Il Manifesto degli Studi, infatti, coerentemente con la grande quantità di informazioni che deve recuperare e con la sua implementazione, porta a compimento la sua funzione in un tempo che rimane obiettivamente elevato, senza però pregiudicare, è importante sottolinearlo, la sua efficacia. Se da una parte l'utilizzo di Active Server Pages sembra essere stata la scelta ideale per la sua realizzazione, dall'altra, a fronte della irrompente diffusione di AJAX e dei suoi benefici, cede il passo in confronto a sistemi che consentono una navigazione veloce, rapida e altrettanto efficace.

Con questo spirito, e con la volontà di adeguare uno strumento fondamentale con l'efficienza richiesta dal contesto, è stata pensata la reingegnerizzazione del Manifesto degli Studi della Facoltà di Ingegneria.

2.3 Active Server Pages

Contestualmente all'illustrazione del dominio di lavoro, si reputa utile fornire cenni sulla tecnologia ASP utilizzata per la realizzazione delle pagine del sito web della Facoltà di Ingegneria.

Active Server Pages (*Pagine Server Attive*, in genere abbreviato in **ASP**) è una tecnologia di casa Microsoft per la creazione di documenti Web gestibili con script lato server.

ASP non è un vero e proprio linguaggio, in senso stretto, come il PHP, ma è una tecnologia che adotta dei linguaggi che fungono da scripting per l'elaborazione delle stesse pagine.

In informatica, le Active Server Pages sono pagine web contenenti, oltre al puro codice HTML, degli script che verranno eseguiti dal server per generare runtime, ovvero a tempo di esecuzione, il codice HTML da inviare al browser dell'utente, e proprio per questo vengono in genere definite 'pagine web dinamiche'. In questo modo è possibile mostrare contenuti dinamici (ad esempio estratti da database che risiedono sul server web) e modificarne l'aspetto secondo le regole programmate negli scripts, il tutto senza dover inviare il codice del programma all'utente finale (al quale va inviato solo il risultato), con un notevole risparmio di tempi e banda. I linguaggi utilizzati sono VB.NET, che è il più utilizzato, C# e J#. Grazie a questi linguaggi il sistema dinamico può comunicare lato server con tutti gli oggetti presenti sul sistema. Le possibilità offerte dal sistema sono infatti fortemente orientate verso l'interfaccia con un corrispondente database, rendendo così possibile lo sviluppo di siti dinamici basati sulle informazioni contenute nel database. È possibile inoltre interfacciare le pagine ASP con qualsiasi tipo di database che abbia un driver OLE-db o ODBC, come ad esempio Access, SQL Server, MySQL, Oracle, Firebird, Sybase e tanti altri.

Funziona ufficialmente solo sul web server Microsoft Internet Information Services (IIS). Nonostante questo, per quanti utilizzano piattaforme GNU/Linux è disponibile un emulatore che consente di eseguire i codici ASP senza dover riscrivere l'intero progetto ma soltanto una piccola porzione di esso. Nel diffuso web server Apache, ad esempio, è

possibile utilizzare pagine dalla sintassi simile ad ASP installando e configurando il modulo Apache: ASP funzionante sulla base di Perl. In alternativa esiste inoltre un modulo generalmente noto come ChiliASP (su cui si basa il modulo SJSASP, Sun Java System Active Server Pages, disponibile sul sito di SUN).

Una caratteristica molto apprezzata dagli utilizzatori dell'interprete ASP è la semplice e comprensibile sintassi di programmazione che rende la curva di apprendimento di tale linguaggio poco ripida. L'interprete ASP, tuttavia, presenta alcuni limiti, specialmente di prestazioni.

Principali concorrenti di ASP sono il PHP, che funziona in modo molto simile, ma con una sintassi del tutto diversa dal VBScript, e l'unione di Perl e CGI, che è un meccanismo meno integrato nella sola pagina web e quindi più macchinoso da gestire, che tuttavia permette al programmatore più flessibilità.

ASP è stato ufficialmente abbandonato, seppure continua ancora ad essere supportato e può funzionare sulle ultime versioni di IIS, in favore di ASP.NET, ormai giunto alla versione 3.5, che consente di creare applicazioni web su piattaforma Microsoft che possono sfruttare anche funzionalità avanzate e, soprattutto, contare su un'infrastruttura molto più avanzata, qual è quella offerta dal .NET Framework in accoppiata con IIS 6.

2.3.1 Cenni storici

Il successo ottenuto dal ‘client-side scripting’ ha portato in breve tempo allo sviluppo di linguaggi di scripting anche per il lato server, iniziando prima con ambienti specializzati, come ad esempio Borland IntraBuilder dove è Javascript il linguaggio sever-side utilizzato e poi orientandosi verso linguaggi di scripting integrati nel server Web, come ad esempio in Netscape Enterprise Server.

Cavalcando questa tendenza, la Microsoft, rilasciando la versione 3.0 di Internet Information Server (IIS), ha introdotto sul mercato degli scripting server-side, la tecnologia

Active Server Pages (ASP), con l'idea di sfruttare non solo le potenzialità degli strumenti server dedicati alla connettività, ma anche attraverso la tecnologia COM (Common Object Model), sfruttare tutte le risorse che il server Microsoft ha a disposizione e coinvolgendo anche i linguaggi di scripting come Jscript e Vbscript.

L'importanza di un linguaggio lato server è quella della creazione della dinamicità e dell'interazione che le pagine di un sito offrono all'utente, consentendo di memorizzare dei dati su di un database, registrarsi ad un servizio ed utilizzarlo, ed altre caratteristiche che l'HTML da solo, o linguaggi di scripting lato client, non offrono.

2.3.2 Contesti applicativi e il suo funzionamento

Le pagine ASP sono completamente integrate con i file HTML, sono facili da creare e non necessitano di compilazione, sono orientate agli oggetti e usano componenti server ActiveX.

Possiamo affermare che ASP è lo strumento che riesce a far convogliare tutte le risorse disponibile sul server, per realizzare un sito Web (che nell'ottica ASP coincide con una applicazione) che possa sfruttare diverse tecnologie in modo trasparente.

Le pagine di un sito non sono più una semplice collezione di documenti HTML ma un insieme di pagine contenenti codice script interpretabile dal server Web, il quale effettua le elaborazioni specificate prima di inviare la pagina HTML risultante al browser che l'ha richiesta.

E' da sottolineare che il risultato della elaborazione del motore ASP è una pagina HTML standard che offre di conseguenza il vantaggio di essere indipendente dal tipo di browser utilizzato.

La figura sintetizza lo schema di funzionamento di un'applicazione ASP:

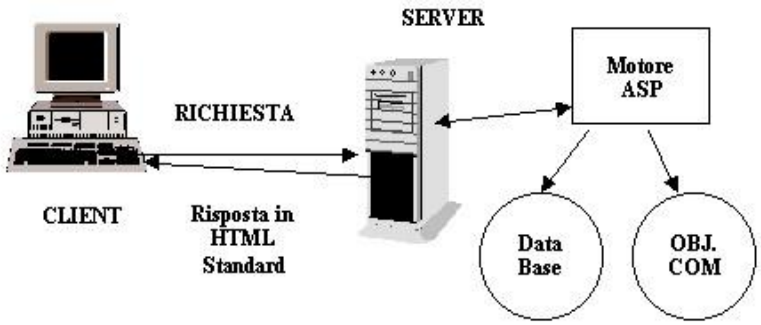

***Figura 2.1**: Schema di funzionamento di una applicazione ASP*

- Il browser richiede una pagina ASP;
- il Web server preleva la pagina ASP ed attiva l'interprete ASP per la pre-elaborazione della pagina, coinvolgendo eventualmente altri componenti presenti sul server;
- il risultato dell'elaborazione eventualmente altri componenti presenti sul server;
- il risultato dell'elaborazione viene inviato al browser tramite il protocollo http;

2.3.3 ASP e client/server

Tra i vantaggi di ASP, è stato messo in risalto la generazione di pagine Web in HTML standard, ottenendo in tal modo una completa indipendenza dell'applicazione Web dal particolare browser utilizzato. Questo aspetto comporta tuttavia un lavoro notevole per il server che deve generare pagine statiche. In un certo senso questa soluzione è in contrasto con il modello client/server secondo il quale il carico di lavoro di elaborazione viene suddiviso tra le due entità. L'introduzione stessa dei linguaggi di scripting client side aveva questo scopo. Con ASP si dovrebbe quindi dedurre un ritorno al modello centrato esclusivamente sul server. Invece ASP mette a disposizione del progettista, la possibilità di decidere quale parte del codice far eseguire dal client e quale dal server, magari decidendolo sulla base del browser utilizzato.

2.3.4 ASP e i suoi utilizzi

Definire i contesti di applicazione di ASP, equivale a definire i contesti in cui è opportuno utilizzare elaborazioni sul server. Tuttavia è possibile delineare alcuni tipi di applicazione per le quali ASP può essere una soluzione indicata.

L'utilizzo per cui era stata progettata la tecnologia ASP era originariamente quello di interfacciare il browser con un database, ma basandosi sul modello COM (Component Object Model), ASP è in grado di interagire con le applicazioni che seguono questo standard e riportare i risultati di questa interazione sotto forma di pagine Web. Ad esempio la richiesta di un utente di visualizzare una tabella si traduce in una richiesta da parte di ASP verso il database di accedere alla tabella in questione, rappresentarla internamente tramite un oggetto ADO e tradurre il contenuto della tabella in una serie di tag HTML. La possibilità di interagire con database ODBC compatibili, pone ASP come uno dei mezzi più efficaci per la realizzazione di applicazioni distribuite indipendentemente dal database utilizzato.

2.4 Manifesto.asp

Addentrandoci più nel particolare, affinché si possa successivamente realizzare un confronto esaustivo tra la versione originaria del Manifesto degli Studi e la versione prodotta attraverso l'adozione di AJAX, è utile cominciare ad analizzare la pagina web che è stata il punto di partenza per il lavoro svolto.

La pagina *Manifesto.asp* è una pagina fondamentale del sito web della Facoltà di Ingegneria dell'Università di Modena e Reggio Emilia, dal momento che, come già illustrato, riporta tutti gli insegnamenti, i relativi docenti e crediti universitari di un certo corso di studi, organizzati per anni di corso e, a loro volta, per periodi.

La pagina *Manifesto.asp* è stata realizzata attraverso la tecnica delle Active Server Pages. Essa infatti consta di uno script che viene eseguito dal server all'atto della richiesta da parte del client. In questo caso l'informazione cercata rappresenta la totalità dei dati relativi ad un certo corso di studi. L'esecuzione di tale script recupera le informazioni cercate, attraverso interrogazioni, presenti all'interno del codice, al database della Facoltà. Una volta esaurita l'esecuzione, tali informazioni vengono quindi inviate al client per poter essere organizzate attraverso la predisposizione di una veste grafica opportuna. Tale obiettivo è realizzato mediante la definizione di *Common Style Sheets*, ovvero 'fogli di stile'. Il client, che in questo caso è il browser, può allora usufruire di questi contenuti, e ovviamente li sottopone all'utente attraverso la visualizzazione sullo schermo.

È necessario puntualizzare che, fintanto che la richiesta è in corso, ovvero che lo script è in esecuzione e le interrogazioni al database si stanno susseguendo, al client non viene restituita nessuna ulteriore informazione. Il tempo di esecuzione dello script quindi si riversa sull'attesa del client, che rimane nello stato di attesa finché non viene esaurita la sua richiesta, e viene fornita una risposta completa. Questa attesa, in generale, può essere trascurabile o meno a seconda della velocità di esecuzione dello script ma soprattutto della quantità di informazioni da reperire. Nel caso della pagina in questione, si evidenzia che l'attesa è in generale elevata, a causa della grande quantità di dati da reperire attraverso interrogazioni innestate e interrogazioni su numeri molto elevati di istanze. Questa

considerazione sembra indubbiamente valida qualora all'interno di uno stesso corso di laurea si trovino più orientamenti didattici, ma si potrà comprendere come comunque rimanga valida anche nella più semplice delle ipotesi. Si potrà successivamente constatare come queste affermazioni trovino fondamento sia all'interno del codice stesso che nella pratica interazione con il portale del Manifesto degli Studi. Considerata allora l'interoperabilità di AJAX con la tecnologia ASP, si provvederà a mostrare come si possa rendere più efficiente il reperimento delle informazioni proposte dal portale attraverso una reingegnerizzazione con AJAX dello stesso.

Soffermandoci ad un primo alto livello di analisi del codice, si nota che il Manifesto degli Studi si presenta secondo la classica struttura di una pagina ASP. Nello script si riscontra infatti subito la presenza del tag `<html>`, al cui interno si individua il codice che compete in maniera più appropriata al reperimento delle informazioni.

Il contenuto del tag `<html>` è a sua volta coerentemente suddiviso in due tag, `<head>` e `<body>`.

Il tag `<head>` riporta tutte le specifiche e i file di appoggio per permettere la visualizzazione voluta delle informazioni ottenute. Queste impostazioni sono di pertinenza del browser. Il tag `<body>` invece è il corpo del codice, è quello che contiene la struttura della pagina e racchiude tutte le operazioni da effettuare per recuperare le informazioni. La pagina ASP è strutturata mediante la realizzazione di due soli div, elementi divisionali, denominati `<div id="visite">` e `<div id="content">`. Il `<body>` presenta inoltre, nella parte iniziale e conclusiva, i richiami e le chiamate ai componenti standard della pagina web del sito dell'Università, quali i frame laterale, superiore ed inferiore, nonché le procedure di autenticazione per permettere l'accesso al database e consentire le interrogazioni presenti.

All'atto della richiesta alla pagina *Manifesto.asp* si utilizza il passaggio di una variabile, chiamata `IdCSR`, ovvero l'identificativo univoco numerico del corso di studi, il cui valore identifica all'interno del database uno e un solo dato corso di studi. La gestione di tale variabile realizza in maniera altrettanto univoca tutte le interrogazioni al database presenti

all'interno del codice, permettendo di ottenere il risultato desiderato, ovvero l'utilizzo da parte del client dell'informazione complessiva richiesta e specifica, che in questo caso si concretizza nella visualizzazione a video del Manifesto degli Studi di un certo corso di studi. La selettività dei contenuti viene realizzata in una delle prime linee di codice del `<div id="content">`.

Infatti qui si incontra il comando

```
IdCSR = Request.QueryString("IdCSR")
```

che memorizza la variabile passata attraverso il link

```
http://.../Manifesto.asp?IdCSR=<%IdCSR%>
```

per effettuare tutte le interrogazioni previste contestualizzandole.

Al fine di ottenere un corretto funzionamento e quindi di un'esecuzione corretta dello script, è stato introdotto un controllo da effettuare in maniera prioritaria sul valore della variabile `IdCSR`, ottenuto mediante la funzione `VerificaID(IdCSR)` presente nella pagina *Utility.asp*, una pagina ASP di supporto, il cui approfondimento non viene però affrontato perché non risulta essere di interesse per lo scopo della tesi.

Prima di procedere con l'illustrazione delle caratteristiche della pagina ASP del Manifesto degli Studi reingegnerizzata attraverso l'introduzione dell'insieme di tecnologie Ajax, è molto utile formulare una specifica ipotesi di notazione testuale a scopo convenzionale.

Per semplicità e chiarezza, al fine di rendere più intuitivi i parametri di confronto che si andranno ad evidenziare tra le due versioni del Manifesto degli Studi, si definisce arbitrariamente *ManifestoAJAX.asp* la pagina ASP rielaborata attraverso AJAX, mentre invece si preferisce mantenere la denominazione originaria di *Manifesto.asp* per ciò che riguarda la versione originaria dello stesso. Si sottolineano però due importanti considerazioni. Innanzitutto, avendo adottato volontariamente una convenzione puramente testuale nell'identificare le due versioni, si può affermare con certezza che la

pagina *ManifestoAJAX.asp* è perfettamente sostitutiva della precedente, in quanto riporta la stesso contenuto informativo della prima versione, ma lo reperisce e lo presenta in forma differente e innovativa. Altrettanto importante è il fatto che il rinnovamento della pagina del Manifesto degli Studi è perfettamente trasparente nei confronti del sistema globale. Si utilizza infatti lo stesso e unico parametro `IdCSR` illustrato precedentemente, senza richiedere un necessario adeguamento del sistema alla modifica. Questo requisito non è affatto di poco conto, ma si potrà verificare come è stato comunque mantenuto.

2.5 ManifestoAJAX.asp

La pagina *ManifestoAJAX.asp* è quella che sostituisce la versione precedente del Manifesto degli Studi della Facoltà di Ingegneria. Come detto viene qui identificata con un nome alternativo per permettere chiarezza nella comprensione, ma va ripetuto che essa riporta in realtà lo stesso contenuto informativo della precedente versione. La differenza sostanziale introdotta con la reingegnerizzazione è il recupero asincrono di parte delle informazioni, effettuato in maniera dinamica attraverso l'utilizzo dell'insieme di tecnologie che prendono il nome di AJAX. L'ulteriore differenza, ovvero l'aspetto grafico ed estetico, si risolve semplicemente attraverso l'adozione di una visualizzazione mediante schede con transizione a scorrimento verticale. Si discuterà successivamente dell'introduzione di questa nuova veste grafica, sia per completezza di informazione che per chiarezza di esposizione, ma rimane evidente che le due versioni del Manifesto degli Studi differiscono completamente sotto l'aspetto del funzionamento. Questa differenza si concretizza da un lato a livello puramente implementativo, ossia all'interno della organizzazione del codice, sia a livello dell'interazione con l'utente, a cui è sottoposta la stessa informazione globale ma in un intervallo temporale che è sicuramente ridotto di molti istanti.

Volendo focalizzare l'attenzione sul centro di gravità della reingegnerizzazione del Manifesto degli Studi, è chiaro che bisogna evidenziare la funzione

```
ajax_loadContent(
      'panel-<%=nPanel%>',
      'FillTab.asp?IdCSR=<%=IdCSR%>&AnnoTab=<%=AnnoCorrente%>',
      'inizializzaPanel<%=nPanel%>'
)
```

che permette di reperire e proporre dinamicamente una parte dei contenuti del Manifesto in maniera asincrona. Infatti, a differenza della versione precedente dello script, le informazioni recuperate attraverso la chiamata di questa funzione non sono ottenute correntemente all'esecuzione dello script principale, ma successivamente, o per meglio dire, in maniera asincrona o in 'background', a partire dal momento in cui l'esecuzione del codice raggiunge la linea contenente la chiamata alla funzione proposta `ajax_loadContent()`. Analizzeremo più avanti e in maniera esaustiva la funzione che

permette il funzionamento asincrono, ma già ora è utile osservare che tale funzione presenta come uno dei due parametri un'ulteriore pagina ASP, evidentemente in qualche modo legata a *ManifestoAJAX.asp*. Si tratta della pagina *FillTab.asp*, che viene chiamata con il passaggio di due variabili. Questa pagina ASP permette di completare il contenuto informativo del Manifesto degli Studi, ottenendolo però in maniera dinamica. Dall'analisi del codice della pagina *ManifestoAJAX.asp* si evince infatti che una prima parte delle informazioni cercate viene recuperata necessariamente effettuando una normale richiesta HTTP, che pertanto dovrà essere completata prima di proporre al client lo sfruttamento dell'informazione cercata, mentre la restante parte delle informazioni sarà recuperata attraverso il motore AJAX con l'appoggio alla pagina *FillTab.asp*.

La pagina *FillTab.asp*, il cui codice sarà poi approfondito, contiene una versione adattata e semplificata del Manifesto degli Studi, in quanto essa produce, come risultato, tutte le tabelle, complete di tutte le informazioni e delle relative note, di un dato corso di studi, identificato dalla variabile `IdCSR`, ma di un solo unico dato anno esistente di quel corso di studi, passato come il parametro `AnnoTab`, che ciclicamente all'interno del codice assume il valore della variabile `AnnoCorrente`. L'informazione complessiva recuperata dalla chiamata alla pagina di appoggio viene proposta al'interno della relativa scheda, a fronte della scelta adottata di una organizzazione mediante tab dei contenuti.

All'interno della trasformazione che ha subito la pagina ASP del Manifesto, infatti, sebbene il risultato più evidente sia il cambiamento nella modalità di reperimento dei contenuti, in realtà l'adozione di AJAX è stata solo l'ultimo ma più importante passo verso il raggiungimento degli obiettivi fissati. Si è dovuto far fronte inizialmente alla gestione degli stessi contenuti ricorrendo però alla nuova veste grafica costituita da schede, che permette un rapporto più immediato con l'utente nella presentazione delle informazioni. Terminata questa rielaborazione, l'approfondimento si sposterà sulle modalità di richiesta asincrona dei contenuti, che costituisce poi il vero risultato del lavoro svolto.

Si fa notare come la pagina *ManifestoAJAX.asp* produca un risultato concreto attraverso il correlato necessario passaggio della già citata variabile `IdCSR`, identificativo numerico univoco di un dato corso di studi. Tale parametro risulta essere necessario come nel caso

precedente, e si mantiene altresì la procedura di gestione degli eventuali errori in cui si può incorrere da un'errata sintassi del link. A livello puramente sintattico, quindi, le due versioni del Manifesto degli Studi sono perfettamente sovrapponibili e intercambiabili tra loro, risultato ottimo per quanto riguarda l'avvicendamento tra di esse sul server della Facoltà. È necessario sottolineare però che la chiamata AJAX, che viene effettuata passando come uno dei parametri la pagina *FillTab.asp,* presenta, oltre alla già nota `IdCSR`, un'ulteriore variabile passata al suo interno, ovvero la variabile `AnnoTab`. Il mancato passaggio di entrambi questi parametri a *FillTab.asp* produce ovviamente un errore, ma è importante sottolineare come `AnnoTab` venga assegnato direttamente all'interno del codice di *ManifestoAJAX.asp*, senza quindi gravare sulla perfetta intercambiabilità tra le due versioni del Manifesto degli Studi.

Capitolo III

AJAX

Si sente molto parlare di AJAX e non sempre in modo chiaro. Il più delle volte però qualsiasi approfondimento parte dal presupposto che chi legge sappia già molto di JavaScript o dello stesso AJAX.

Si vuole quindi far luce sulla questione, spiegando dettagliatamente l'acronimo tanto mormorato e le sue potenzialità al fine di fornire ai meno aggiornati o meno preparati le conoscenze per usare questa tecnica di sviluppo, ma soprattutto di comprendere le capacità e i conseguenti miglioramenti apportati da un suo utilizzo all'interno di un sito web o, ad esempio, del Manifesto degli Studi della Facoltà di Ingegneria.

3.1 Cenni storici

L'acronimo **AJAX**, che significa esattamente **Asynchronous JavaScript And XML** (JavaScript asincrono ed XML), è stato enunciato per la prima volta da Jesse Garrett, nel 18 Febbraio 2005, come titolo di un post all'interno del suo blog.

Non si tratta di una nuova tecnologia né di un'invenzione bensì di un concetto utilizzato per sviluppare applicativi avanzati e particolari. Il concetto è in parte espresso nell'acronimo stesso, cioè un utilizzo asincrono di Javascript che, attraverso l'interfacciamento con XML, può permettere ad un client di richiamare informazioni lato server in modo veloce e trasparente, allargando gli orizzonti delle 'Rich Internet Applications'.

Queste applicazioni fino a poco tempo fa erano legate principalmente alle tecnologie Adobe-Macromedia Flash o Java (con le applet). Entrambe purtroppo non sempre interpretabili dai client degli utenti e troppo spesso usate a sproposito con il solo scopo di stupire, discorso che spesso e purtroppo vale anche oggi.

In alternativa a queste tecniche di interazione client/server, quando nel 1996 venne introdotto l'iframe in Internet Explorer 3, molti sviluppatori sfruttarono quest' ultimo modificando l'attributo sorgente della pagina racchiusa e simulando così un refresh trasparente di una parte di contenuti, il che emulava, in modo abbastanza sporco, un'interazione asincrona.

Nel 1998 Microsoft cominciò a sviluppare una tecnologia, chiamata Remote Scripting, con lo scopo di creare una tecnica più elegante per richiamare contenuti differenti ed è in questo periodo, seppur con nome differente, che AJAX venne utilizzato per la prima volta, per poi evolversi in versioni più mature fino a diventare un oggetto vero e proprio, noto ora come XMLHttpRequest.

Il motivo principale di tanto successo è che solo ultimamente il Remote Scripting ha suscitato lo stupore degli addetti ai lavori nel vedere cosa Google fosse riuscita a fare all'interno dei suoi applicativi senza necessità di Flash Player o Java Virtual Machine, mantenendo comunque la compatibilità con molteplici browser di utenti che per diversi motivi non potevano usufruire di Javascript.

Come spesso è accaduto negli ultimi anni molti hanno preso spunto da Google, ed il web, noto per la sua immediatezza propositiva, è stato in grado in poco più di un anno di regalarci numerosi esempi di applicativi basati su AJAX, esagerando in alcuni casi

nell'utilizzo ma considerando molto spesso e fin da subito, a differenza di quanto accadde per Flash ed Applets anni prima, due delle problematiche più diffuse del web attuale: accessibilità ed usabilità.

3.2 Definizione e funzionamento

AJAX è una tecnica multi-piattaforma utilizzabile su molti sistemi operativi, architetture informatiche e browser web, ed esistono numerose implementazioni open source di librerie e framework.

La tecnica AJAX utilizza una combinazione di:

- **HTML** (o XHTML) e **CSS** per il markup e lo stile;
- **DOM** (Document Object Model) manipolato attraverso un linguaggio ECMAScript come JavaScript o JScript per mostrare le informazioni ed interagirvi;
- l'oggetto **XMLHttpRequest** per l'interscambio asincrono dei dati tra il browser dell'utente e il web server. In alcuni framework Ajax e in certe situazioni, può essere usato un oggetto Iframe invece di XMLHttpRequest per scambiare i dati con il server e, in altre implementazioni, tag `<script>` aggiunti dinamicamente (JSON);
- in genere viene usato XML come formato di scambio dei dati, anche se di fatto qualunque formato può essere utilizzato, incluso testo semplice, HTML preformattato, JSON e perfino EBML. Questi file sono solitamente generati dinamicamente da script lato server.

Come DHTML o LAMP, Ajax non è una tecnologia individuale, piuttosto è un gruppo di tecnologie utilizzate insieme.

Le applicazioni web che usano Ajax richiedono browser che supportano le tecnologie necessarie appena elencate. Questi browser includono: Mozilla, Firefox, Opera, Konqueror,

Safari, Internet Explorer e Chrome. Tuttavia, per specifica, Opera non supporta la formattazione degli oggetti XSL.

Le classiche applicazioni Web lavorano in questo modo: molte delle operazioni non sono altro che una richiesta HTTP inoltrata al web server. Il server fa l'elaborazione - raccolta dati, manipolazione numeri, dialogo con eventuali sistemi legacy (datati) - e ritorna una pagina HTML al client. Questo è un modello adottato sin dalle origini del Web, considerato medium ipertestuale, anche se purtroppo esso rende il Web ottimale per l'ipertesto ma non necessariamente così ottimale anche per le applicazioni software.

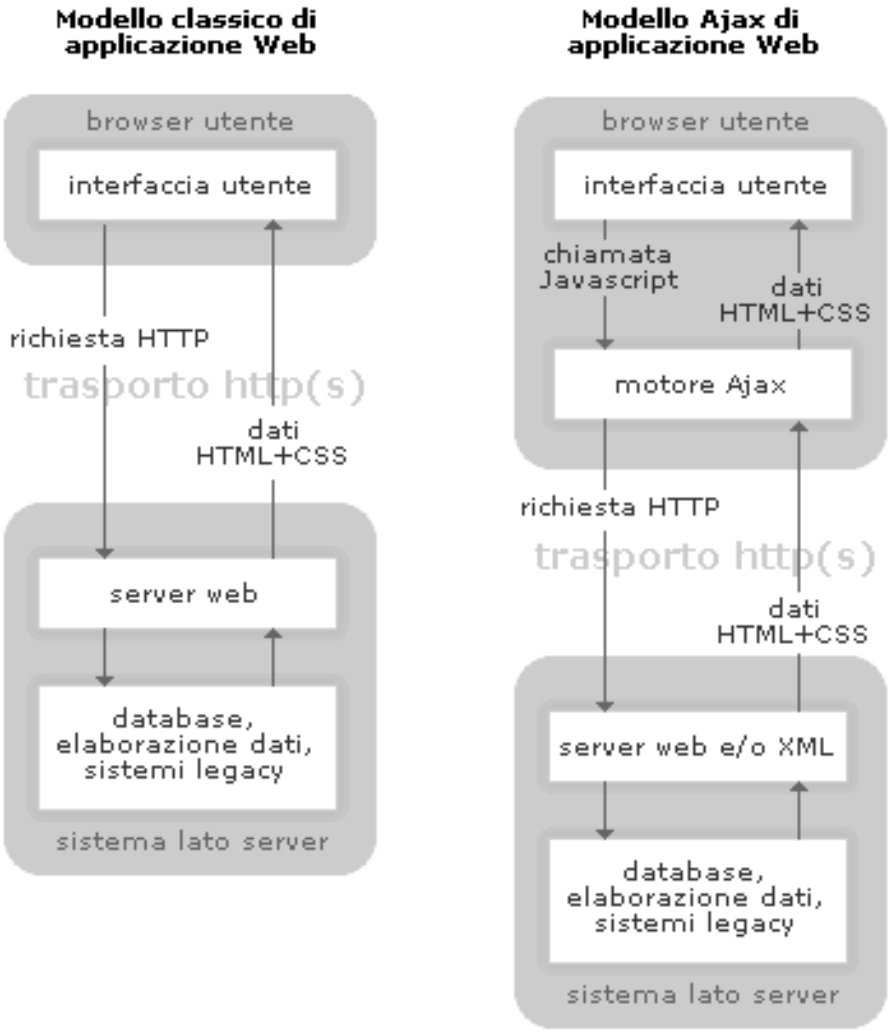

***Figura 3.1**: Modello tradizionale delle applicazioni Web (alla sinistra) confrontato col modello Ajax (alla destra).*

Questo approccio crea una miriade di soluzioni e quesiti tecnici che annebbiano quelli inerenti la User Experience. Mentre il server sta elaborando, l'utente aspetta, essendone, tra l'altro, costretto a farlo ad ogni richiesta.

Un'applicazione Ajax elimina la natura del Web ad essere contraddistinta da momenti alternati di dialogo-risposta che fanno attendere l'utente, introducendo un intermediario - un motore Ajax - tra l'utente e il server. Può sembrare di aggiungere un livello all'applicazione che la rende meno veloce a rispondere, anche se in realtà è vero il contrario.

Invece infatti di caricare una classica pagina web, ad inizio sessione, il browser carica il motore Ajax - scritto in Javascript e usualmente associato ad un frame invisibile. Questo motore è responsabile della comunicazione tra l'interfaccia utente e il server, mettendoli in comunicazione tra di loro. Il motore Ajax, indipendentemente dalla comunicazione con il server, permette all'interazione dell'utente con l'applicazione di essere asincrona. Così l'utente non è mai succube della finestra del browser bianca e dell'icona indicante il caricamento, aspettando che il server esegua le operazioni.

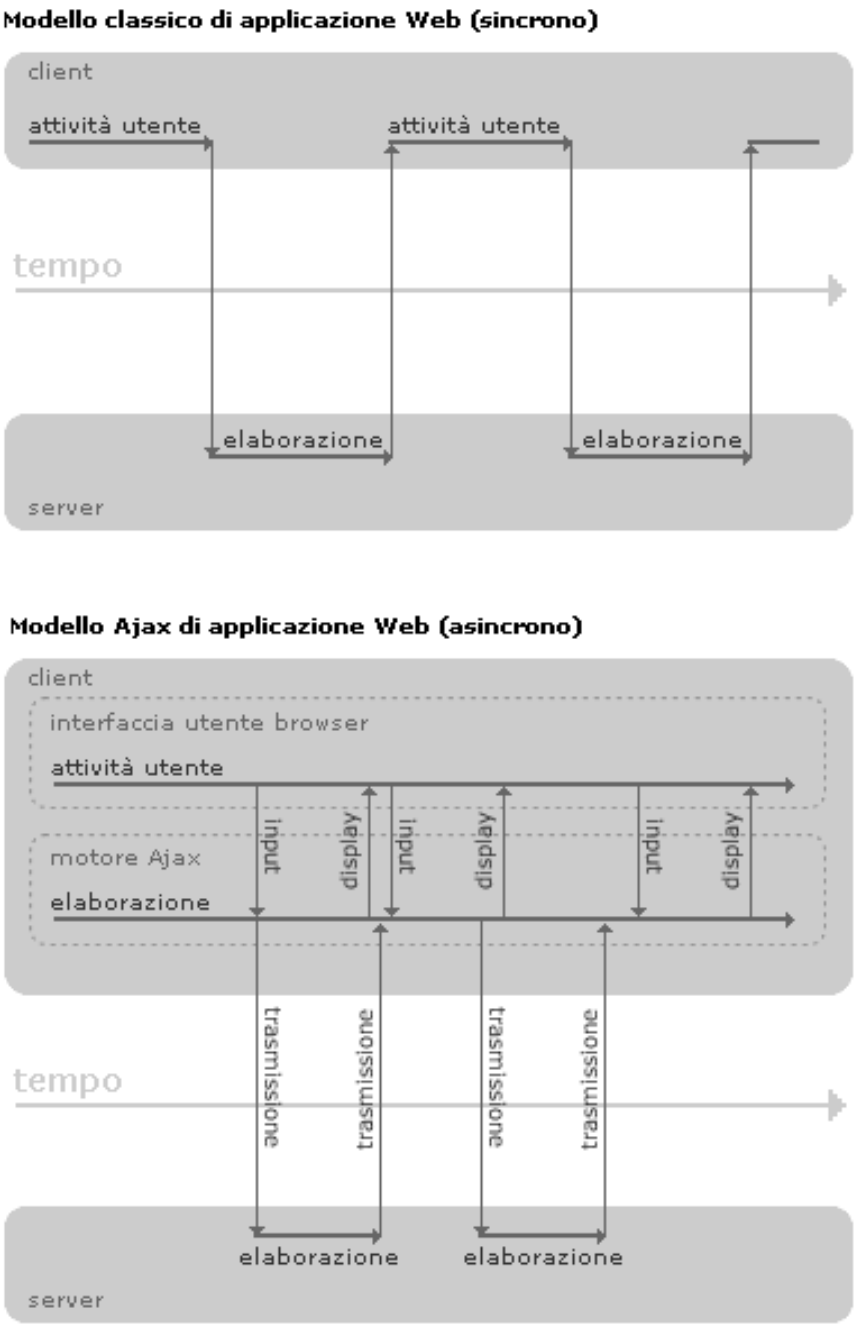

***Figura 3.2**: Modello di interazione sincrona di un'applicazione web (sopra) confrontato con il modello asincrono Ajax (sotto).*

Ogni azione dell'utente che normalmente genera infatti una richiesta HTTP prende la forma di una chiamata Javascript verso il motore Ajax. Ogni risposta che non richiede la chiamata diretta al server - come validare alcuni dati, modificare dati in memoria - è un lavoro

compiuto dal motore Ajax. Se il motore ha bisogno di interagire con il server per rispondere - invio dati da processare, caricare un interfaccia aggiuntiva, ricevere nuovi dati - il motore rende queste richieste asincrone, usualmente utilizzando XML, senza bloccare l'interazione tra utente e applicazione.

Ajax non risulta essere pura tecnica ma anche un'ottima pratica per la creazione di applicazioni molto utili. Non è una delle tante tecnologie utilizzate solo nei laboratori. Le applicazioni Ajax possono avere dimensioni diverse, dalla più semplice come Google Suggest, alla più complessa come Google Maps.

La maggiore sfida che Ajax pone davanti ai nostri occhi non riguarda la risoluzione dei problemi tecnici da esso derivati, ma il dimenticare tutte le limitazioni che riguardavano il Web, per lanciarsi verso la creazione di applicazioni robuste, veloci e sempre più simili alle applicazione Desktop.

3.3 Confronto con applicazioni tradizionali

Le applicazioni Ajax possono inviare richieste al web server per ottenere solo i dati che sono necessari (generalmente usando SOAP e JavaScript per mostrare la risposta del server nel browser). Come risultato si ottengono applicazioni più veloci, dato che la quantità di dati scambiati fra il browser ed il server si riduce. Anche il tempo di elaborazione da parte del web server si riduce poiché la maggior parte dei dati della richiesta sono già stati elaborati.

Un esempio concreto: molti siti usano le tabelle per visualizzare i dati. Per cambiare l'ordine di visualizzazione dei dati, con un'applicazione tradizionale l'utente dovrebbe cliccare un link nell'intestazione della tabella che invierebbe una richiesta al server per ricaricare la pagina con il nuovo ordine. Il web server allora invierebbe una nuova query

SQL al database ordinando i dati come richiesto, la eseguirebbe, prenderebbe i dati e ricostruirebbe da zero la pagina web rinviandola integralmente all'utente. Usando le tecnologie Ajax, questo evento potrebbe preferibilmente essere eseguito con uno JavaScript lato client che genera dinamicamente una vista dei dati con DHTML.

3.4 Pro e contro

Come per le applicazioni DHTML, anche le applicazioni AJAX devono essere testate su più browser per verificarne la compatibilità. Inoltre è richiesto che nel client sia attivato Javascript. Il vantaggio di usare AJAX è la grande velocità alla quale un'applicazione risponde agli input dell'utente.

Un problema abbastanza degno di nota è che, senza l'adozione di adeguate contromisure, le applicazioni AJAX possono rendere non utilizzabile il tasto "*indietro*" del browser: con questo tipo di applicazioni, infatti, non si naviga da una pagina all'altra, ma si aggiorna di volta in volta una singola parte del medesimo documento. Proprio per questo i browser, che sono programmi *orientati alla pagina*, non hanno possibilità di risalire ad alcuna di tali versioni "intermedie". Google, ad esempio, nella sua Google Maps, ha sviluppato una possibile soluzione al problema: invece di usare XMLHttpRequest quando l'utente clicca sul bottone di ricerca, il risultato della ricerca viene inviato in un iframe invisibile, dal quale le informazioni sono portate nella pagina visibile.

In ogni modo, un attento design delle applicazioni AJAX permette di risolvere totalmente o in parte questi aspetti negativi.

3.4.1 Vantaggi lato client

L'impressione di avere a che fare con pagine apparentemente più interattive, dinamiche e professionali è praticamente immediata. L'effetto novità si aggiunge prepotentemente alla lista, stuzzicando la curiosità del navigatore.

Questi aspetti apparentemente futili hanno permesso a tecnologie come Flash di affermarsi nel contesto web. Questo perché l'interesse collettivo è la ricerca di informazioni ma si preferisce navigare dove le informazioni sono presentante nel modo più semplice e veloce possibile.

Ci sono librerie dedicate, facilmente integrabili in applicativi AJAX e capaci di impressionare gli utenti in modo analogo a quello proposto dal plug-in grafico di Adobe.

La compatibilità praticamente totale con i browser più diffusi ed il supporto nativo, o integrabile, dell'oggetto XMLHttpRequest contribuiscono ulteriormente allo sviluppo di Rich Internet Applications, allargando i campi di impiego per gli sviluppatori e coinvolgendo sempre di più i navigatori.

Ottenere modifiche dinamiche e leggere sulle pagine, come i suggerimenti sui campi di ricerca, in grado di suggerire parole presenti e velocizzare l'inserimento, mappe, votazioni, statistiche, amministrazione, lezioni, chat, giochi e chi più ne ha più ne metta, ecco il motivo di tanto successo ed ecco perché gli utenti possono trarre numerosi vantaggi da Ajax.

3.4.2 Svantaggi lato client

Gli utenti che necessitano di tecnologie assistite e coloro che non dispongono di connessioni veloci sono i più penalizzati.

I primi perché non possono, attualmente, godere del cambio parziale di contenuti e non saranno quindi in grado di sfruttarne le potenzialità. I secondi devono caricare, almeno al

primo accesso, una mole di dati consistente rappresentata dai vari file JavaScript e potrebbero non essere in grado di sfruttare niente di quanto caricato.

Si potrebbe non disporre di un browser aggiornato, oppure in situazioni di JavaScript disabilitato, di assenza dell'oggetto XMLHttpRequest. In questi casi si scaricano centinaia di bytes che non verranno mai usati a causa del controllo a posteriori utilizzato per conoscere la reale compatibilità con Ajax.

Questi aspetti puramente tecnici sono spesso raggirabili, grazie ad esempio ad una stesura degradabile dell'applicativo da parte dello sviluppatore o grazie al caricamento dinamico del Javascript proposto dopo aver verificato la compatibilità del browser. Ve n'è uno però esclusivamente pratico, probabilmente il più noto nonché fastidioso: l'uso dei tasti '*avanti*' e '*indietro*' presenti nei browser.

Qualsivoglia interazione asincrona, dà comunque la sensazione di aver cambiato stato alla pagina e l'abitudine a tornare indietro, qualora il risultato non dovesse essere quello sperato, è intrinseca nel navigatore nonché lecita.

Aprendo direttamente una pagina ricca di interazioni asincrone non sarà possibile, nemmeno volendo, cliccare sul tasto '*indietro*' del browser per tornare allo stato precedente, si verrà reindirizzati invece alla pagina precedente, il che causare disorientamento. In questi casi, la consapevolezza di aver usato AJAX, qualora il navigatore sia preparato, non può certo aiutare, poiché anche l'aggiornamento della pagina non serve a tornare allo stato precedente ma solo allo stato iniziale.

Oltre ad essere un vincolo di navigazione, questo problema è anche un vincolo per l'indicizzazione o la possibilità di segnalare ad altri la pagina visualizzata. Questo accade perché il comando gestito da JavaScript non è portabile come un link e scrivere ad un amico l'indirizzo attualmente visitato ed eventualmente modificato tramite AJAX non è possibile se non dando delle indicazioni precise sulle operazioni svolte una volta arrivati nella pagina in oggetto.

La soluzione a questi problemi non è semplice ed i motivi sono diversi.

- In primo luogo creare un gestore eventi in grado di aggiungere cambiamenti alla history del browser può risultare un'operazione complessa per via dei diversi standard o funzioni presenti nei diversi browser.
- In secondo luogo l'indicizzazione di un solo link, per chi non ha JavaScript o non può usufruire di tale tecnologia, diventerebbe pressapoco impossibile.

Una libreria nota e molto potente che sembra abbia risolto almeno in parte i problemi è quella di *backbase*, dove il caricamento incrementale del contenuto, attraverso la visualizzazione asincrona dello stesso, sembra essere la soluzione definitiva, gestendo sotto-gerarchie e le varie sezioni tramite una serie di informazioni appese dopo il carattere #, il quale permette solitamente di indicizzare sotto-contenuti di una pagina tramite l'uso di elementi con id univoco ma che può essere gestito senza problemi dal JavaScript proposto.

Tuttavia oggi questi non sono ancora tecnicamente sormontabili. Se un utente che naviga una sottosezione passa il link ad un altro utente, potrebbero infatti esserci comunque problemi di compatibilità tra browser.

Questo avviene perché se JavaScript può verificare il link e gestirlo di conseguenza, il server ospitante non potrà fare altro che mostrare la pagina richiesta, ignorando giustamente tutto ciò che non fa parte del reale indirizzo richiesto, ovvero tutto il testo eventualmente presente dopo il carattere #.

3.4.3 Vantaggi lato server

È noto che usare fogli di stile esterni piuttosto che in linea ha l'effetto di decimare la banda divorata mensilmente dai navigatori, l'utilizzo di AJAX potrebbe fare altrimenti.

Infatti con AJAX il server non ha la necessità di trasferire tutta la pagina per ogni interazione, ma solo la porzione necessaria all'operazione richiesta. Questo rende sensibilmente più veloce l'interazione per l'utente e favorisce il risparmio di banda.

C'è una ricaduta anche in termini di calcoli da effettuare. Prendiamo il caso di un portale ricco di informazioni. Reperirle tutte per poter affrontare una interazione 'tradizionale' può richiedere uno stress relativo per diverse manciate di decimi di secondo (database, webservices etc.). Con AJAX le richieste sono puntuali ed il server può rispondere in modo più efficiente.

Siti con migliaia di utenti simultanei non dovranno quindi operare su tutte le parti dell'applicativo, migliorando la possibilità di gestire mole di utenti elevata.

Allo stesso tempo una parte dei calcoli può essere data in gestione al browser, così da sfruttare la potenza del PC-client e distribuire il carico su tutti, piuttosto che sul solo host.

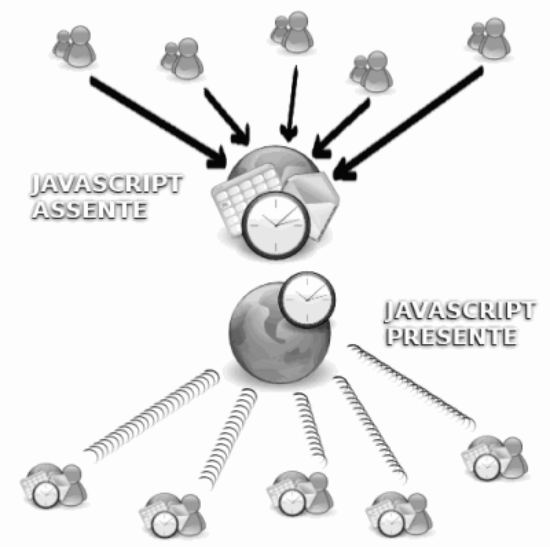

***Figura 3.3**: Carico del server con e senza AJAX*

Non bisogna ovviamente esagerare, pena il rallentamento del client qualora i calcoli dovessero essere troppi, va scremato minuziosamente l'utile dal superfluo e soprattutto non si dovrebbero creare applicativi così complessi da richiedere diversi secondi di attesa prima di poter essere utilizzati.

3.4.4 Svantaggi lato server

Il principale rovescio della medaglia è causato dall'abuso di AJAX, gli sviluppatori meno accorti mettono a disposizione troppe interazioni su una sola pagina. Inoltre, se non si controllano le operazioni sul client il server rischia di ritrovarsi sovraffollato di richieste probabilmente inutili.

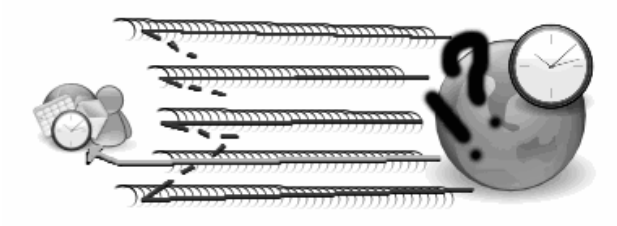

***Figura 3.4**: Chiamate 'a vuoto'*

3.5 XMLHttpRequest

Se parliamo di AJAX, oggi, parliamo di un oggetto specifico: **XMLHttpRequest**. A seconda del browser usato prende nomi differenti o viene richiamato in maniera differente.

Ad oggi l'oggetto XmlHttpRequest è supportato in Internet Explorer 5.0+, Safari 1.2, Mozilla 1.0 / Firefox, e Netscape 7. Questo oggetto non è definito come standard dal W3C, il quale definisce alcune funzionalità simili ma non ancora implementate in alcun browser.

Per questo motivo il W3C stesso è costretto a suggerire a chiunque abbia la necessità di effettuare una richiesta HTTP lato browser, di utilizzare l'oggetto XmlHttpRequest.

Nel caso di Internet Explorer, ad esempio, questo oggetto è restituito da un ActiveXObject mentre nei browser alternativi più diffusi (Mozilla, Safari, FireFox, Netscape, Opera ed altri)

XMLHttpRequest è supportato nativamente, cosa che dovrebbe accadere anche per IE dalla versione 7.

Questo oggetto permette di effettuare la richiesta di una risorsa (con HTTP) ad un server web in modo indipendente dal browser. Nella richiesta è possibile inviare informazioni, ove opportuno, sotto forma di variabili di tipo GET o di tipo POST in maniera simile all'invio dati di un form.

La richiesta è asincrona, il che significa che non bisogna necessariamente attendere che sia stata ultimata per effettuare altre operazioni, stravolgendo sotto diversi punti di vista il flusso dati tipico di una pagina web.

Generalmente infatti il flusso è racchiuso in due passaggi alla volta, richiesta dell'utente (link, form o refresh) e risposta da parte del server per poi passare, eventualmente, alla nuova richiesta da parte dell' utente.

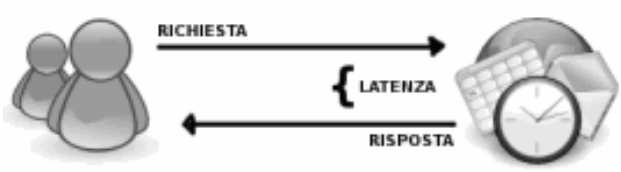

***Figura 3.5**: Rappresentazione di una richiesta ad un server con latenza e risposta*

Il terzo ed inevitabile incomodo di questo ciclo è l'attesa che trascorre tra una richiesta dell' utente e la risposta del server. Con l'aggiunta di AJAX si perde questa linearità e mentre l'utente è all'interno della stessa pagina le richieste sul server possono essere numerose e completamente indipendenti.

Nulla infatti vieta, teoricamente, di effettuare decine di richieste simultanee al server per operazioni differenti con o senza controllo da parte del navigatore.

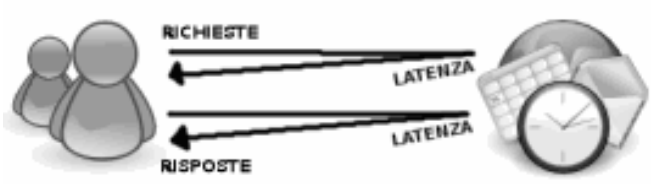

***Figura 3.6**: Rappresentazione di più richieste ad un server con latenza e risposta*

Ciò che resta del vecchio flusso, il tempo di attesa, passa spesso in secondo piano ed in molti tipi di interazione è praticamente impercettibile. Ma attenzione poiché questo tempo è anche uno dei maggiori problemi dell'utilizzo di AJAX, sia per gli sviluppatori sia per i navigatori.

I primi potrebbero trovarsi in difficoltà qualora l'operazione asincrona dovesse attendere assolutamente una risposta al fine di completare una serie di operazioni più o meno sensibili mentre i secondi potrebbero non avere idea di cosa stia accadendo alla pagina chiudendola ignari di aver richiesto un'informazione.

Il tipo di risposta che l'oggetto si aspetta dopo una chiamata non deve essere necessariamente di tipo XML o letta come tale ma che può essere semplicemente testuale, in contro tendenza con l'acronimo stesso ma non per questo inusuale.

Sebbene cominciare ad utilizzare AJAX sia abbastanza semplice e le conoscenze necessarie potrebbero essere di livello molto basso, tutt'altro discorso vale per creare applicativi ed interazioni avanzate.

Definito l'oggetto, è indispensabile conoscerne metodi e parametri al fine di sfruttarlo al meglio a seconda delle necessità.

La lista dei metodi è diversa da browser a browser, per questo in genere si usano solo quelli presenti in Safari (browser per Mac), il quale supporta nativamente l'oggetto XMLHttpRequest ma con meno metodi rispetto gli altri.

Ecco quindi una lista, in ordine alfabetico, dei metodi supportati da tutti i browser Ajax compatibili:

- *abort*
- *getAllResponseHeaders*
- *getResponseHeader*
- *open*
- *send*
- *setRequestHeader*

Questi metodi permettono di effettuare le varie operazioni asincrone. Nonostante siano state elencate nella precedente lista, sarebbe utile analizzarle più approfonditamente secondo il loro utilizzo, al fine di capire al meglio a cosa servono, quali parametri richiedono e perché. Questo ulteriore approfondimento però non viene realizzato perché esula dalle finalità della presentazione del funzionamento delle tecnologie Ajax nel capitolo.

Per muovere i primi passi con Ajax non è indispensabile conoscere a fondo (X)HTML, CSS e nemmeno XML. Addirittura è possibile sapere poco anche di Javascript, ma almeno i concetti base sul cosa è una variabile o una funzione e quali sono gli eventi più comuni definibili all'interno dei tag della pagina sono necessari.

È necessario essere in grado di popolare una parte di una pagina con del testo definito in JavaScript. In particolare bisogna conoscere l'attributo *innerHTML* di un elemento o le specifiche DOM create appositamente per aggiungere, popolare o modificare elementi della pagina.

Nel caso di innerHTML, non si fa altro che sovrascrivere il contenuto di un elemento del documento visualizzato e tutto quello che ci serve è una funzione in grado di restituirci l'elemento richiesto.

Le funzioni più note per fare questo sono due, *getElementById* e *getElementsByTagName* ed entrambe possono essere sfruttate per usare il metodo innerHTML oppure il DOM.

Entrando nei dettagli, *getElementById* è un metodo dell'oggetto *document*, ovvero la pagina visualizzata, e cerca al suo interno un tag `<html>` con un attributo, `id`, assolutamente univoco.

Capitolo IV

Reingegnerizzazione del Manifesto degli Studi

In questo capitolo si affrontano in maniera più approfondita e dettagliata le operazioni svolte per attuare la reingegnerizzazione del Manifesto degli Studi della Facoltà di Ingegneria che ci eravamo prefissati di realizzare come scopo realizzativo della tesi.

Ripercorrendo quanto illustrato finora, si dovrebbe essere ora in grado di comprendere almeno le caratteristiche peculiari degli elementi sia originariamente presenti nel codice, che quelli introdotti con la rielaborazione effettuata.

Si richiama comunque la finalità della reingegnerizzazione della pagina ASP in questione, ovvero la volontà di sopperire alla lentezza della risposta da parte del server attraverso l'introduzione nel codice delle funzionalità migliorative delle tecnologie Ajax.

Verrà quindi illustrato dettagliatamente il codice sorgente della pagina ASP *ManifestoAJAX.asp*, mettendolo costantemente a confronto con quello della versione precedente del Manifesto degli Studi, per evidenziarne le caratteristiche identificative.

Per consentire dei termini di paragone efficaci, durante l'illustrazione si sfrutterà, fintanto che risulta essere possibile, la struttura implementativa del codice, seguendo quindi gli elementi interni e le divisioni funzionali individuate. Qualora non fosse possibile effettuare

un confronto diretto tra tag corrispondenti, ci si preoccuperà di illustrare l'elemento introdotto e di rapportarlo con parte di codice della versione precedente che era preposto ad operare una funzione simile.

4.1 Realizzazione della reingegnerizzazione e confronto tra le versioni

Viene ora svolta una attenta analisi nei dettagli degli elementi costituitivi della pagina ASP originaria e di quella frutto della rielaborazione effettuata. In questa parte di capitolo si proporrà la soluzione al problema riscontrato in fase progettuale circa la scarsa efficienza del Manifesto degli Studi in termini prestazionali. Come già anticipato, la risoluzione del problema avverrà effettuando un confronto per quanto più possibile coerente con la struttura del codice della pagina ASP.

4.1.1 Tag `<HTML>`

Il tag `<html>` è l'elemento principale di una pagina ASP. Analizzando sia la pagina *Manifesto.asp* che la sua successiva elaborazione *ManifestoAJAX.asp* ci si accorge che esso rimane invariato, proprio a causa della sua funzione.

Tutto il contenuto di un documento HTML è compreso all'interno dei marcatori `<html></html>` che, in altre parole, hanno il compito di aprire e chiudere il file. Questi tag indicano al browser che il documento è marcato in HTML, anche se i browser più recenti (Netscape 3 e 4, MsIe 3,4 e 5) riescono ugualmente ad interpretare i tag successivi.

Detto questo esistono comunque due buone ragioni per inserire sempre i tag <html></html> all'interno del documento:

- HTML non è l'unico linguaggio di contrassegno presente sul Web e quindi il browser rischia di non interpretare correttamente i tag, confondendoli con altri linguaggi di marcatura.
- Gli utenti che usano browser vecchi rischiano di vedere il documento mal formattato.

Esso racchiude sia il tag <head> che il <body>.

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="it" >
      ...
</html>
```

4.1.2 Tag <HEAD>

Il tag <head> è posto subito dopo l'apertura del tag <html> e racchiude l'intestazione vera e propria del documento, cioè tutte le informazioni necessarie al browser, al Webserver ed ai motori di ricerca. Il tag <head> è il primo elemento letto dal browser e all'intero và inserito il titolo del documento e altre informazioni.

Analizzando i tag <head> dei due codici è immediato notare come già essi differiscano nei loro contenuti e nelle informazioni che recano.

La pagina originale *Manifesto.asp* riporta un tag <head> così articolato:

```
<head>
 <meta http-equiv="content-type" content="text/html; charset=iso-8859-
 1" />
 <title>Manifesto del corso di studi</title>
 <link rel="stylesheet" type="text/css" href="CSS/.../imposta.css" />
 <link rel="stylesheet" type="text/css" href="CSS/.../contenuto.css"/>
 <link rel="stylesheet" media="print" type="text/css"
 href="CSS/print.css" />
</head>
```

a differenza della sua successiva versione che implementa il tag in maniera più ricca:

```
<head>
 <meta http-equiv="Content-type" content="text/html; charset=iso-8859-
 1" />
 <title>MANIFESTO del corso di studi</title>
 <script type="text/javascript"
 src="Utility/.../JQfunction.js"></script>
 <script type="text/javascript"
 src="Utility/.../JQScrollTo.js"></script>
 <script type="text/javascript" src="Utility/.../AJAX.js"></script>
 <script type="text/javascript"
 src="Utility/.../inizializza.js"></script>
 <script type="text/javascript">inizializzaPanel1()</script>
 <link rel="stylesheet" type="text/css" href="CSS/.../imposta.css" />
 <link rel="stylesheet" type="text/css" href="CSS/.../contenuto.css"/>
 <link rel="stylesheet" type="text/css"
 href="CSS/.../manifestoAjax.css" />
 <link rel="stylesheet" media="print" type="text/css"
 href="CSS/print.css" />
</head>
```

ove si evidenziano le modifiche apportate nella versione successiva attraverso l'introduzione di ulteriori linee.

È utile approfondire il significato delle linee di codice riportate, cominciando dal tag `<title>`. Questo tag riporta il nome che assumerà la scheda del browser una volta completato il caricamento della pagina. I tag `<link>` invece permettono di far riferimento ai contenuti dei fogli di stile appositamente realizzati per la formattazione del contenuto della pagina. In questo caso si nota la presenza di un ulteriore tag `<link>` nell'head del *ManifestoAJAX.asp* che punta al file `manifestoAjax.css`, evidentemente un foglio di stile ulteriore, il cui contenuto sarà approfondito successivamente, necessario per la realizzazione della visualizzazione delle informazioni attraverso schede scorrevoli. Un'ulteriore differenza è chiaramente la presenza di cinque tag `<script>`. I primi quattro di questi permettono di caricare uno script esterno contenuto in un apposito file .js, mentre il quinto consente l'esecuzione della funzione `inizializzaPanel<%=nPanel%>()` che come vedremo è legata alla struttura a schede della pagina e al suo funzionamento a scorrimento.

4.1.3 *Tag* <BODY>

L'elemento <body> rappresenta il corpo del documento e va posto immediatamente dopo alla chiusura del tag </head>. All'interno di esso si sviluppa il corpo del documento. E' il corpo centrale dello script, ed in entrambe le pagine ASP in questione mantiene la stessa sintassi, ma non chiaramente lo stesso contenuto.

Si presenta attraverso le linee di codice:

```
<body class="container">
    ...
</body>
```

Si nota come il tag venga specificato dalla classe 'container', dichiarata nel foglio di stile 'imposta.css', dove è a sua volta specificato però anche il suo stesso contenuto.

A questo punto si può cominciare ad approfondire il codice all'interno del tag <body>, dal momento che le differenze sostanziali tra la pagina *Manifesto.asp* e la sua evoluzione *ManifestoAJAX.asp* vengono realizzate attraverso la manipolazione del codice qui dentro contenuto.

La prima parte del contenuto del <body> non è ritenuta di rilevanza nell'ambito del confronto tra le due versioni. Essa propone dapprima l'*header*, ovvero l'intestazione della pagina, che è comune a qualsiasi pagina del sito dell'Università, così come il frame laterale sinistro, e successivamente i meccanismi di autenticazione per poi poter accedere ed interrogare il database. Viene anche creato un div definito dall'identificativo 'visite' che permette di sfruttare degli strumenti di statistica degli accessi e degli aggiornamenti. Al termine dello studio del <body> non si farà nemmeno riferimento ad un ulteriore parte responsabile dell'inserimento del *footer* della pagina, sempre comune a qualsiasi pagina web del sito, e sempre per irrilevanza nelle finalità dell'approfondimento.

È buono definire anche il tag , dal momento che entra in gioco più volte nell'illustrazione del lavoro realizzato. Il tag è un elemento della pagina ASP che

permette una divisione univoca del contenuto al suo interno dal resto della pagina. È possibile fornirgli degli attributi che competono alla formattazione del suo contenuto.

4.1.3.1 `<div id="content">`

Questo div è il primo elemento che si incontra durante l'analisi comune delle pagine *Manifesto.asp* e *ManifestoAJAX.asp*. Sebbene esso sia presente in entrambe le versioni, si noterà come abbia delle finalità completamente differenti.

Innanzitutto è bene mostrare che esso è definito da un identificativo univoco dal nome '`content`', facente riferimento alla classe '`content`' del foglio di stile '`imposta.css`'.

Nella versione originaria del Manifesto degli Studi tale div è il solo e unico elemento della pagina ASP a contenere tutte le informazioni che le successive circa 750 linee di codice saranno in grado di individuare. Infatti l'intero codice che compete alla vera e propria ricerca e richiesta delle informazioni relative ad un dato corso di studi è contenuto all'interno di tale div.

La differenza sostanziale con il div omonimo della pagina *ManifestoAJAX.asp* sta nel fatto che esso non è più l'unico e solo contenitore di tutte le informazioni, ma che in poco più di 200 linee di codice opera una doppia funzione.

Rielaborando infatti il codice originario, si è provveduto a selezionare le informazioni fondamentali e identificative del corso di studi, quali il tipo di laurea, il nome del corso, l'anno accademico, la legenda per consentire una corretta lettura del Manifesto, e le eventuali note relative al corso. Nella figura sottostante è stato evidenziato nel riquadro rosso il solo elemento div in questione,

***Figura 4.1**: Particolare del contenuto dell'elemento* `<div id="content">`.

riportandolo però nella sola versione della pagina *ManifestoAJAX.asp*, tralasciando di proposito di evidenziare il corrispettivo nella versione originaria a causa della misura spropositata che assumeva.

Si può notare come il corpo centrale delle informazioni relative agli insegnamenti segua il riquadro evidenziato, ovvero il div in corso di analisi.

Approfondendo il codice relativo a questo primo elemento, si possono notare alcune peculiarità, che spiegano anche la precedente affermazione del duplice ruolo del div rielaborato.

Una prima differenza si nota nel fatto che nella seconda versione del Manifesto, tutte le informazioni reperite ed evidenziate precedentemente dalla figura sono raccolte attraverso l'inizializzazione di un solo oggetto di tipo `RecordSet`:

```
Set ObjRs  = server.CreateObject("ADODB.RecordSet")
```

Un'altra notevole differenza si può trovare nel fatto che l'anno relativo all'ultimo anno di corso di un dato corso non viene definito staticamente all'interno del codice, come avveniva

nella versione precedente, ma la variabile viene assegnata successivamente ad una interrogazione:

```
StrSql =   "SELECT    max(AnnoCorso) AS AnnoFine " &_
           "FROM VIEW_INSEGNAMENTI_PERIODI2 " &_
           "WHERE     IDCSR = " & IdCSR & " " &_
           "AND Disattivato <> 1"

ObjRs.Open StrSql, CN, adOpenStatic, adLockReadOnly

     'Assegno AnnoFine leggendolo dal database
     AnnoFine = NullToInt(ObjRs("AnnoFine"))

ObjRs.Close
```

Ciò può sembrare un'operazione gravosa che appesantisce il già non snello onere computazionale, ma in realtà è un passo necessario per due ragioni: la prima è la adozione della visualizzazione a schede voluta, che prevede la realizzazione dinamica del numero delle tab sulla base della durata di un corso; la seconda è che a seguito delle riforme introdotte circa la durata e l'organizzazione degli ordinamenti didattici dell'università italiana, questa assegnazione mediante interrogazione garantisce l'immunità da errori e soprattutto l'indipendenza del codice dai mutamenti esterni.

Successivamente a questo passaggio di codice, un'ulteriore attenzione richiede la parte che compete a verificare qual è il primo curriculum non nullo, ovvero qual è il primo anno non nullo presente all'interno degli anni di corso di un dato corso di studi. Questo accorgimento è sempre in funzione della riforma in atto dell'Università, ed ha sempre le stesse finalità dichiarate di rendere immune da modifiche dirette il codice. Durante questi passaggi infatti, avvalendosi di una variabile utilizzata come flag definita `Stampato`, si verifica qual è il primo curriculum non nullo e il relativo anno lo si memorizza nella variabile `AnnoCorrente`. Questa variabile assume un ruolo fondamentale all'interno di tutto il codice della pagina *ManifestoAJAX.asp*, perché è il riferimento che verrà passato alla funzione Ajax e da cui dipende il corretto funzionamento della pagina web. Per questo motivo si ritorna a sottolineare la scelta di rendere indipendente l'assegnamento di tale variabile.

Si aggiunge solo ora, dal momento che non è termine di confronto, che viene effettuato un necessario controllo sul parametro `IdCSR` passato nel link alla pagina del Manifesto degli Studi.

È necessario e importante descrivere un ulteriore controllo che viene effettuato all'interno dell'elemento in corso di analisi, la cui assenza pregiudica la correttezza di parte delle novità introdotte con la reingegnerizzazione del Manifesto degli Studi. L'importanza di tale controllo verrà chiarita di seguito, ma dal momento che è qui che viene effettuato, è corretto riportarlo in questo paragrafo. Si definisce una variabile `ContentNOTNull`, il cui valore viene assegnato come `True`. Si approfitta poi, per non gravare sulla complessità globale del codice, per verificare che esista almeno un anno di corso, del corso di studi selezionato con `IdCSR`, che sia non nullo, ovvero le cui istanze contengono almeno una qualche informazione. Tale controllo si effettua in cascata alla prima interrogazione del database, ovvero quella che seleziona tutte le informazioni che possono essere visualizzate nell'ultima figura presentata. Per chiarezza si riporta la parte del codice in questione:

```
iF not ObjRs.EOF then
            ...
      else
            ContentNOTNull = False
end iF
```

dove è chiaramente da intendersi che al posto della punteggiatura compaia tutta quella parte di codice preposta ad effettuare le ulteriori operazioni descritte in questo paragrafo, quale l'interrogazione sul primo curriculum non nullo, l'assegnazione delle variabili `AnnoCorrente` ed `AnnoFine`, e quanto appena approfondito.

Il valore che da questo punto in poi assumerà la variabile `ContentNOTNull` opera un bivio all'interno delle successive operazioni da effettuare. Si ritiene però di approfondire questo argomento più avanti nel corso della trattazione, in quanto la sua contestualizzazione in modo adeguato consente maggiore chiarezza.

Il div analizzato della pagina *ManifestoAJAX.asp* si esaurisce a questo punto, avendo compiuto la funzione di ricerca delle informazioni generiche relative al corso di studi

identificato da `IdCSR`, e la funzione di assegnazione dinamica ed indipendente delle variabili `AnnoCorrente` e `AnnoFine`, nonché avendo effettuato degli importanti controlli prioritari sulle istanze e sui valori specificati.

Nella versione originale, che non sopperisce alle ultime funzioni relative alle variabili proposte, a questo punto comincia la parte di codice predisposta al *fetching* dell'informazione consistente nelle tabelle riportanti gli insegnamenti e i periodi, con visualizzazione incolonnata ordinata per anni di corso.

Dal momento che d'ora in poi sarebbe impossibile mantenere un costante confronto tra div corrispondenti, si illustreranno soltanto le novità degli elementi introdotti dalla nuova versione e, qualora ve ne sia il bisogno, si confronterà parte di codice con il codice originario ma operante la stessa o le stesse funzioni.

Prima di proseguire con l'analisi degli elementi div realizzati all'interno della pagina ASP del Manifesto degli Studi, si evidenzia innanzitutto un controllo preventivo che viene effettuato a priori dei meccanismi di realizzazione delle tab sulla concretezza delle istanze del database. La traccia di questo controllo rimane conservata tramite il valore assunto dalla variabile `ContentNOTNull`, come già specificato. Non è ancora stata però presentata l'importanza e né la centralità di tale operazione. A fronte della dichiarata volontà di realizzare un'interfaccia nuova attraverso cui proporre i contenuti informativi recuperati, ovverosia attraverso l'adozione di schede, è necessario dichiarare una scelta arbitraria fatta durante la rielaborazione. Le tab infatti sono pensate affinché la prima di esse, associata ad al primo anno di corso non nullo presente nel corso di studi selezionato, venga creata di default. Questa scelta può sembrare sbagliata, ma si può comprendere come invece essa rende più agevole l'esecuzione del codice. Il controllo preventivo operato in precedenza, infatti, è realizzato in cascata ad una operazione già prevista. Ciò consente così di aggiungere poche linee di codice per ottenere un risultato molto importante, senza appesantire di molto l'esecuzione. Se si pensasse invece di effettuare il controllo in questione a questo punto del codice si introdurrebbe una ulteriore interrogazione che in realtà non può essere neanche in parte sfruttata per altri scopi.

Ecco allora giustificata la scelta progettuale. Ricordando la scelta arbitraria di realizzare la prima tab di default, sulla base del valore assunto dalla variabile `ContentNOTNull` è possibile capire se esiste o meno un qualche anno del corso di studi che non abbia associate istanze nulle, potendo così ovviamente realizzare almeno la tab di default così come è pensata.

Per far ciò, tutto il codice che scritto da questo punto in poi, è discriminato da un semplice bivio, implementato attraverso questi passaggi elementari:

```
if ContentNOTNull then
      %>
      <div id="scroller-header">
            ...
      </div>
      <div id="scroller-body">
            <div id="mask">
                  <div id="panel">
                        ...
                  </div>
            </div>
      </div>
      <%
end if
```

Viene evidenziato in grassetto il codice in questione, mentre si accenna al suo contenuto in grigio per permettere una comprensione immediata del significato assunto da tale controllo. Nel caso in cui tutti gli anni presenti all'interno di un certo corso di studi assumano un valore nullo, non viene realizzata nessuna tab, nemmeno quella di default, come si può chiaramente immaginare.

Si tratta ora di approfondire il contenuto nel caso, ovviamente più significativo, in cui esistano dei contenuti da visualizzare che non siano nulli.

4.1.3.2 `<div id="scroller-header">`

Con questo div vengono presentate le ulteriori, ma non ultime, novità introdotte dalla nuova versione. Come è stato già detto, escludendo ancora la trattazione della gestione dinamica dei contenuti attraverso Ajax, uno dei primi obiettivi è quello di produrre una visualizzazione a schede dei contenuti. La prima versione *Manifesto.asp* propone la totalità dei contenuti in un unico div, recuperandoli dal server mediante interrogazioni cicliche che producono elementi tabellari uno di seguito all'altro. Con la nuova versione *ManifestoAJAX.asp* si mira, sia per necessità che per migliorare l'aspetto e la qualità della pagina web, ad una gestione a schede dei contenuti.

Per ottenere questo risultato è stato necessario realizzare tanti elementi contenitori quanti sono gli anni di corso di cui ottenere informazioni, e quindi realizzare una struttura adatta alla navigazione tra di essi.

A fronte della scelta di realizzare una navigazione a schede che dipendano sia dalle loro associazioni che dai loro contenuti, risulta interessante presentare la modalità di creazione delle tab attraverso cui sfogliarle.

Il div a cui si fa riferimento è l'elemento `<div id="scroller-header">` . La formattazione di tale elemento viene gestita attraverso una delle voci del foglio di stile creato appositamente per la nuova versione del Manifesto degli Studi, chiamato `manifestoAjax.css`.

Si evidenzia che la prima tab viene creata di default, e vi si assegna il link così definito:

```
<a href="#panel-1" rel="panel" class="selected">
     <%
     select case AnnoCorrente
          case 1 %>ANNO PRIMO<%
          case 2 %>ANNO SECONDO<%
          case 3 %>ANNO TERZO<%
          case 4 %>ANNO QUARTO<%
          case 5 %>ANNO QUINTO<%
          end select
     %>
</a>
```

Le successive tab vengono create invece in funzione della necessità, esplicitata attraverso il valore assunto dalla variabile

```
nTabNeeded = AnnoFine - AnnoCorrente +1
```

Le successive tab infatti vengono create attraverso un ciclo, differenziandosi dalla prima, creata di default, per quanto riguarda l'attributo `class="selected"`, attributo specificato nel foglio di stile '`manifestoAjax.css`'. Tale attributo permette di visualizzare in evidenza la prima tab all'atto del completamento del caricamento della pagina web.

L'esempio concreto della creazione delle tab lo si può individuare nel riquadro rosso in figura, che mostra la realizzazione di due tab, di cui la prima è dotata dell'attributo '`selected`':

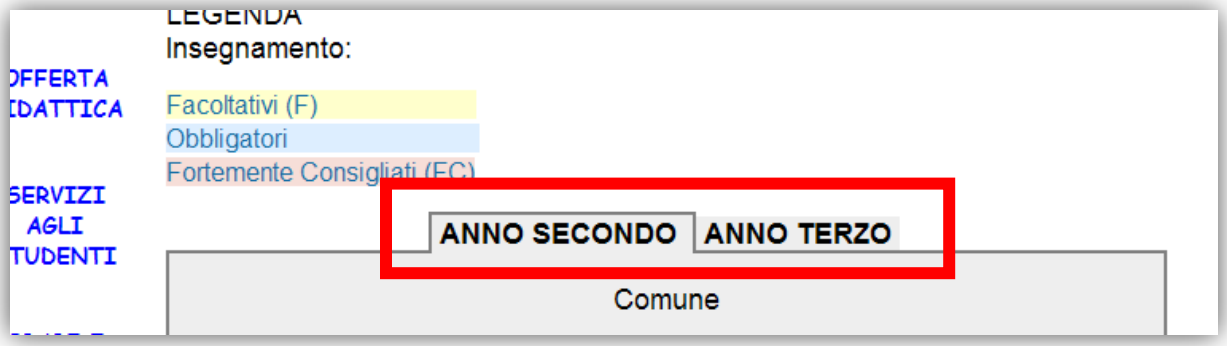

***Figura 4.2**: Particolare del contenuto dell'elemento `<div id="scroller-header">`*

Al termine del ciclo di creazione delle tab necessarie, si sottolinea che si tiene traccia del numero di tab create attraverso la variabile `nTabCreated`, poiché risulta necessario non perdere questo valore al fine di creare un adeguato numero di elementi preposti ad accogliere i contenuti successivamente richiesti.

Si evidenzia che le tab create posseggono dei riferimenti (`href`) interni, individuati dal simbolo #, a degli elementi chiamati `panel`, che sono contenuti nella stessa pagina in corso di analisi.

```
<a href="#panel-<%=nTab%>" rel="panel">
```

Tali elementi sono creati appositamente per la visualizzazione a schede che si intende realizzare. La loro realizzazione avviene all'interno del successivo div che si incontra nel codice, ed è a quel punto che verranno approfonditi.

4.1.3.3 `<div id="scroller-body">`

Questo ulteriore elemento è il contenitore vero e proprio di tutte le informazioni che verranno reperite. Si può perfino paragonare al div '`content`' della pagina *Manifesto.asp*, che come si è già detto racchiudeva tutto il contenuto informativo ricevuto come conseguenza dell'apertura del Manifesto degli Studi. In realtà, ritornando alla nuova versione del codice, sebbene tutte le informazioni ora vengano convogliate all'interno di questo div, e sebbene esso abbia quasi la stessa funzione del corrispettivo, questo viene strutturato in maniera completamente differente, a causa della scelta di adoperare una visualizzazione a schede la cui navigazione tra di esse avvenga per scorrimento.

Realizzazione delle schede

Si ritiene opportuno introdurre ora a livello teorico l'idea alla base della rielaborazione grafica che ha permesso il raggiungimento dell'obiettivo prefissatoci con la scelta voluta. Al fine di realizzare schede contenenti una qualche informazione generica, si è provveduto a suddividere il div in questione, ovvero `<div id="scroller-body">`, in tanti elementi più piccoli, chiamati `panel-<%=nPanel%>`, posti uno sotto l'altro. Il numero di questi `panel-<%=nPanel%>` viene calcolato all'atto dell'esecuzione dello script, in base al numero di anni accademici presenti all'interno del corso di studi selezionato:

```
while nPanel <= nTabCreated
```

```
            %>
            <div id="panel-<%=nPanel%>">
                  ...
            </div>
            <%
            ...
wend
```

Prima di realizzare però tali elementi `panel-<%=nPanel%>`, ci si preoccupa di creare una maschera, chiamata `mask` affinchè possa essere visualizzabile un solo `panel-<%=nPanel%>` per volta nella visualizzazione a schermo. La funzione della maschera la si comprende meglio dalla figura seguente. Tutti questi elementi più piccoli sono creati all'interno di un ulteriore elemento contenitore chiamato `panel`, la cui lunghezza è pari alla somma delle lunghezze dei singoli sottoelementi appena descritti. Questa struttura è dettata dalla necessità di dover implementare una funzione che permetta lo scorrimento tra i sottoelementi, filtrati dalla maschera, che ne lascia visualizzare uno alla volta. Le funzioni preposte ad operare tutti i meccanismi di scorrimento e formattazione verranno presentate successivamente, ma non sono il vero obiettivo dello studio. Si giustifica però la struttura presentata sulla base di queste.

La figura seguente illustra bene il funzionamento della `mask`. Essa infatti apre una finestra virtuale al di sopra dei singoli sottoelementi, consentendo di poter filtrare il risultato e i contenuti presenti. Dall'immagine statica non si percepisce il fatto che essa non è di dimensioni prefissate, ma che viene inizializzata alla lunghezza del singolo sottoelementi ogniqualvolta si passi per scorrimento da un sottoelemento all'altro.

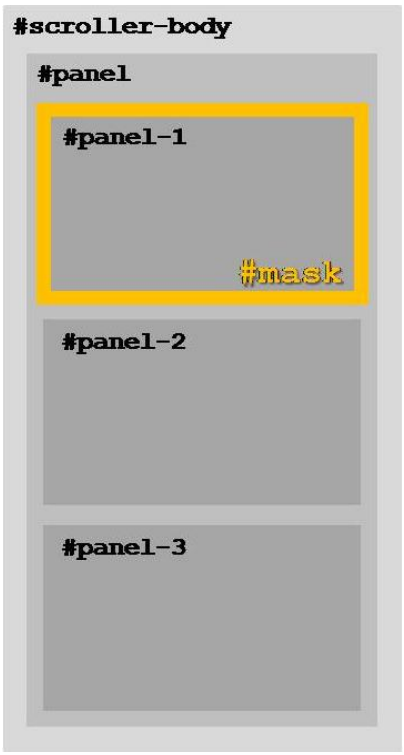

***Figura 4.3**: Schema della struttura utilizzata per la realizzazione di schede a transizione scorrevole*

Al fine di concludere la trattazione della creazione degli elementi rappresentativi delle schede, si riporta di seguito il codice che individua gli elementi in figura:

```
<div id="scroller-body">
      <div id="mask">
            <div id="panel">

                  ...

            </div>
      </div>
</div>
```

Un esempio concreto della descrizione appena effettuata può essere trovato nella figura successiva, che evidenzia attraverso il riquadro rosso l'elemento `mask`, e, contemporaneamente, l'elemento `panel-<%=nPanel%>`.

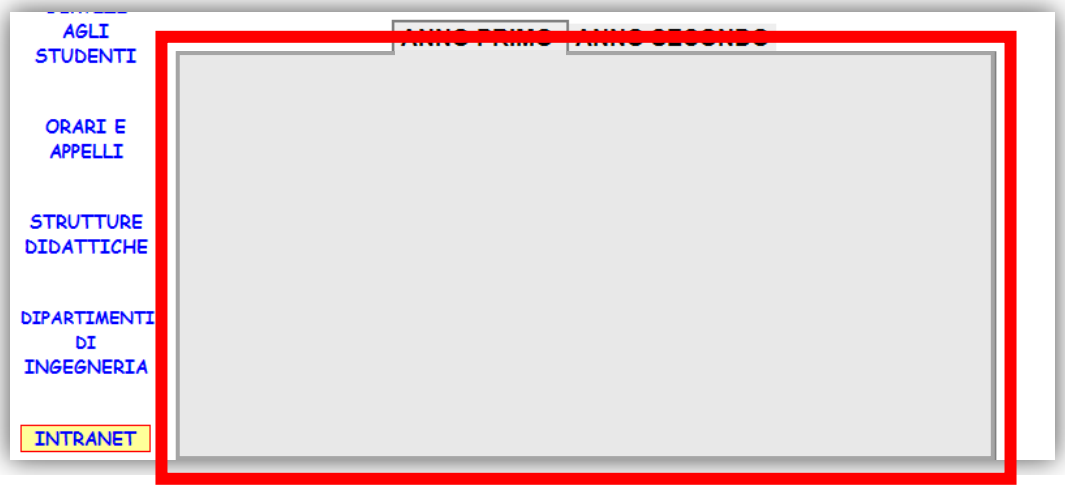

***Figura 4.4**: Particolare del contenuto dell'elemento* `<div id="scroller-body">`

A questo punto è possibile cominciare ad analizzare il vero contenuto dei vari elementi appena descritti. Ci preoccupiamo di descrivere i differenti elementi `panel-<%=nPanel%>`, poiché come si è già detto i contenitori superiori non aggiungono informazioni ma sono necessari solo per la visualizzazione intenzionalmente adottata.

Si sottolinea ora come sia differente l'implementazione del reperimento dei contenuti all'interno del primo dei sottoelementi e dei successivi. Nel caso in cui si considera il primo dei `panel-<%=nPanel%>`, allora il codice, che è stato debitamente rielaborato rispetto alla versione precedente, viene eseguito di default. Ciò significa che il suo contenuto non viene caricato dinamicamente, ma che è necessario attendere la risposta del server per poter visualizzare la pagina web attraverso l'interfaccia del proprio browser. Nel caso in cui invece si stia eseguendo, a seguito dell'avanzamento del ciclo, codice interno ai sottoelementi successivi, ecco che entra in gioco il fetching dinamico dei contenuti, attraverso l'utilizzo della funzione preposta realizzata con le tecnologie Ajax.

È utile analizzare separatemante tali casi per poter trarre conclusioni più precise.

4.1.3.4 Caso `panel-<%=nPanel%>` *di default*

Come già anticipato, il contenuto del sottoelemento che viene creato per primo è richiesto a tempo di esecuzione, e non dinamicamente come avviene per i successivi. Esso presenta quindi una parte di codice adattata alla sua funzione che permetterà intuitivamente di selezionare una sola parte del contenuto informativo globale da recuperare. Tale informazione è ovviamente quella legata alla propria tab di riferimento, ovvero quella del primo anno di corso le cui istanze non siano nulle.

Innanzitutto è bene individuare due sottosezioni del codice adattato. Infatti in esso si possono chiaramente distinguere una prima parte a cui compete la selezione del curriculum completo relativo all'orientamento comune, mentre la seconda che si occupa di selezionare tutti i successivi curricula degli orientamenti alternativi. Sebbene possa sembrare verbosa la scelta di mantenerli separati, in realtà in termini computazionali è migliore di quella che si avrebbe accorpandoli. Si dovrebbe introdurre infatti una ulteriore interrogazione al database, che individui il solo numero di orientamenti presenti, per poi diventare quest'ultimo la condizione di uscita del ciclo che recupererbbe le informazioni ciclicamente. La soluzione avanzata nella versione precedente del Manifesto degli Studi consente invece, a fronte di una apparente verbosità del codice, di risparmiare su una interrogazione, che, per quanto semplice, appesantisce il già sovraccarico onere computazionale.

Affinchè si individui la differenza nella struttura dei due codici,si riporta la versione originaria, che dunque viene privata del ciclo for che la caratterizzava

```
for Anno = AnnoInizio to AnnoFine + 1
      ...
Next
```

in favore della possibilità di effettuare così una interrogazione diretta del database, dal momento che si è già a conoscenza di quale sia il primo anno che non presenta istanze non nulle.

Il codice della sezione in corso di studio si completa con la selezione di tutte le informazioni relative al primo anno di corso individuato, ovverosia i periodi, la loro durata, gli insegnamenti previsti in ognuno di essi, i docenti che tengono questi corsi, i crediti formativi che essi offrono e le eventuali note dei singoli orientamenti come quelle del corso di studi.

Per quanto riguarda tali operazioni, si sottolinea che il codice sorgente della versione precedente della pagina non è stato ulteriormente rielaborato. Tale scelta è stata effettuata infatti sulla base della considerazione della già buona efficacia del codice esistente. L'adeguamento apportato è consisitito sostanzialmente alla riduzione di una struttura ciclica ad una selezione mirata, effettuata attraverso la variabile `AnnoCorrente`.

4.1.3.5 Caso `panel-<%=nPanel%>` *con AJAX*

Si tratta ora di affrontare una delle sezioni più interessanti della rielaborazione operata sul codice originario. Si è appena visto l'insieme di operazioni che vengono eseguite all'interno dell'elemento associato alla prima tab. In questo caso il contenuto informativo veniva recuperato all'atto dell'esecuzione dello script, e fintanto che non fosse stato disponibile, il client non avrebbe ottenuto risposta. Premesse anche le scelte operate circa la modifica della veste grafica, si è ora in grado di approfondire in maniera esaustiva e dettagliata quello che all'inizio dell'analisi è stato esposto come 'centro di gravità' della reingegnerizzazione effettuata sul Manifesto degli Studi della Facoltà di Ingegneria. Si sta parlando ovviamente delle tecnologie Ajax che consentono il recupero asincrono e dinamico dei contenuti informativi, offrendo all'utente un servizio completo che risulta essere evidentemente più veloce, ma altrettanto efficace. Come si potrà comprendere, la novità introdotta è il codice contenuto all'interno degli elementi `panel-<%=nPanel%>`, ad eccezione del primo, di cui si è già trattato.

Per semplicità e chiarezza si riporta subito la struttura del codice in questione, in modo da permettere già intuitivamente di impossessarsi delle sue peculiarità, e di osservare

l'immediatezza dello stesso affinché si possa giustificare la parafrasi precedente circa la centralità delle tecnologie Ajax.

```
<div id="panel-<%=nPanel%>">
     <br />
     <%
     IF nPanel = 1 then
          ...
     ELSE
        <script type="text/javascript">
          ajax_loadContent(
            'panel-<%=nPanel%>',
            'FillTab.asp?IdCSR=<%=IdCSR%>&AnnoTab=<%=AnnoCorrente%>',
            'inizializzaPanel<%=nPanel%>'
          )
        </script>
     <%
     end IF
     %>
</div>
```

È immediato osservare quanto tale codice differisca da quello dell'elemento precedente.

Nell'elemento in corso di approfondimento infatti il codice consta di poche linee, che operano una funzione quanto fondamentale tanto semplice. Si tratta della chiamata alla funzione

```
     ajax_loadContent(
          'panel-<%=nPanel%>',
          'FillTab.asp?IdCSR=<%=IdCSR%>&AnnoTab=<%=AnnoCorrente%>',
          'inizializzaPanel<%=nPanel%>'
          )
```

Tale chiamata viene effettuata attraverso il passaggio di tre parametri, come si può evincere direttamente dal testo presentato. Approfondiremo di seguito la chiamata alla funzione necessaria per il caricamento dinamico dei contenuti e la stessa funzione completa, nonché i parametri necessari alla corretta esecuzione del corrispondente codice.

Si deve innanzitutto evidenziare il fatto che il codice completo dell'elemento `panel-<%=nPanel%>`, senza fare distinzione tra i due casi esplicitati, ovvero di quello associato

alla prima tab e di quello relativo alle successive, è contenuto all'interno di un ciclo, di cui si è già trattato in precedenza, che qui si ricorda:

```
while nPanel <= nTabCreated
      ...
wend
```

Tale ciclo ovviamente termina quando la condizione non è più verificata, quindi indipendentemente dai contenuti. È chiaro però che se l'esecuzione del codice contenuto non prevede operazioni in background, ma coerentemente con l'ordine del codice da processare, il ciclo impiega un tempo per concludersi completamente dipendente dal codice sorgente che vi è associato. Ciò si può infatti comprendere se si osserva il caso in cui `nPanel = 1`, ovvero nel caso dell'esecuzione di default al runtime.

Ora invece, introducendo la semplice funzione di caricamento dei contenuti in modo asincrono, è consentito così completare l'esecuzione del ciclo in maniera corretta, completa e allo stesso tempo molto rapida, perché permette al client di ottenere una risposta in tempi brevi, e quindi di poter usufruire dell'informazione globale ricevuta, ma di poterla aggiornare, in questo caso espandere, senza gravare sull'attesa. Questo è lo scopo già dichiarato dell'utilizzo delle tecnologie Ajax, e qui si può già cominciare a capire tale aspetto.

Facendo due esempi concreti, ponendosi nelle seguenti ipotesi iniziali:

- necessità di creazione di 3 tab, a cui associare tre anni di corso, in ordine dal primo al terzo anno, associati a tre relativi elementi della pagina ASP;
- uguale informazione presentata dai suddetti elementi all'utente, ovvero una sola tabella contenente tutte le informazioni di un dato anno di corso;
- definendo **T** il tempo necessario al recupero del contenuto informativo di uno degli elementi in questione;
- utilizzo della stessa implementazione del codice adottata per il primo elemento, ovvero quella che recupera i dati a tempo di esecuzione in maniera sincrona, per tutti e tre gli elementi.

È possibile allora affermare che il client potrà sfruttare l'informazioni globale ricevuta dopo un tempo almeno pari a **3*T**, ma che nella pratica risulta essere sicuramente superiore. Dopo tale intervallo temporale il client viene svincolato dallo stato di attesa, possiede tutte le informazioni richieste, e, ad esempio, può visualizzarle a video.

Modificando ora la quarta e ultima ipotesi nel modo seguente:

- utilizzo della implementazione del codice uguale a quella presentata, ovvero recupero dei dati a tempo di esecuzione per il primo elemento e recupero dinamico per i successivi attraverso l'adozione delle tecnologie Ajax.

È ora possibile invece affermare che il client otterrà il primo terzo delle informazioni dopo un intervallo di durata almeno pari a **T**, venendo svincolato dallo stato di attesa, potendo cominciare a sfruttarle. Conseguentemente poi, dopo un intervallo ulteriore di durata non minore di **T**, ma in generale superiore, il client otterrà in maniera asincrona il secondo terzo, con la differenza però che esso è già stato privato della necessità di attendere la risposta. Considerazioni simili si possono fare per il terzo e ultimo elemento. Globalmente adesso, dopo un tempo totale superiore sicuramente a **3*T**, dal momento dell'intervento esterno di funzioni di appoggio dettate dall'adozione delle tecnologie Ajax che devono essere eseguite per consentire tale risultato, il contenuto informativo della pagina ASP d'esempio coincide perfettamente, ma in termini prestazionali risulta essere invece completamente differente. È chiaro allora come la pagina realizzata sotto le prime tre ipotesi e la quarta modificata sia dichiaratamente più prestazionale.

Questo esempio concreto è poi in realtà una esemplificazione teorica del funzionamento delle due versioni del Manifesto degli Studi, quella precedente, e quella ottenuta dalla reingegnerizzazione.

4.2 Funzione `ajax_loadContent()`

Si è appena potuto osservare come il contenuto dei sottoelementi successivi al primo, processato di default a tempo d'esecuzione, dipenda strettamente ed esclusivamente dalla funzione che sfrutta le tecnologie Ajax per rendere dinamico il fetching delle informazioni. È interessante ora studiare tale funzione, comprendendo il suo funzionamento e quindi la sua novità.

Per discuterne più compiutamente, si mostrano subito i parametri necessari al suo funzionamento. Successivamente si analizzerà il codice di tale funzione.

La chiamata alla funzione `ajax_loadContent()` avviene attraverso il passaggio di tre parametri.

```
ajax_loadContent(
      'panel-<%=nPanel%>',
      'FillTab.asp?IdCSR=<%=IdCSR%>&AnnoTab=<%=AnnoCorrente%>' ,
      'inizializzaPanel<%=nPanel%>'
)
```

Il primo è il nome identificativo del sottoelemento che andrà a contenere le informazioni recuperate in maniera asincrona dalla chiamata della funzione. Il secondo invece è una pagina ASP di appoggio contenente un apposito codice dovutamente rielaborato per permettere l'identificazione dei contenuti informativi richiesti. Il terzo invece è il nome di una funzione, variabile rispetto al valore `nPanel`, che permette, una volta completato il recupero asincrono dei contenuti, di modificare i parametri della relativa scheda di visualizzazione.

La funzione `ajax_loadContent()` risiede all'interno di uno script realizzato debitamente per consentire di usufruire delle potenzialità delle tecnologie Ajax, chiamato `AJAX.js`. Tale script contiene tutte le funzioni necessarie al reperimento dinamico e asincrono dei contenuti.

4.2.1 Parametri della funzione

Per completezza si ritiene utile presentare tutta la funzione in questione, richiamando eventualmente le ulteriori funzioni utilizzate solamente sotto l'aspetto funzionale, per non appesantire la trattazione. Ritenendo però di favorire la comprensione chiarendo prima i parametri utilizzati nella chiamata della funzione, si preferisce descrivere prima questi.

- **`'panel-<%=nPanel%>'`**

Come già anticipato, questo parametro, al variare del valore assunto dalla variabile `nPanel`, identifica univocamente uno dei sottoelementi creati appositamente per contenere date informazioni, relative al titolo della tab che vi viene associata. Passando alla funzione l'identificativo nominale del div, si attua una selezione del campo all'interno della pagina in cui caricare i contenuti.

- **`'FillTab.asp?IdCSR=<%=IdCSR%>&AnnoTab=<%=AnnoCorrente%>'`**

Questo parametro è invece più complesso del precedente. Alla funzione che esegue il caircamento dinamico viene qui passata una pagina ASP caratterizzata da due specifici valori assegnati alle due seguenti variabili: `IdCSR` e `AnnoTab`. Tali variabili sono fondamentali per il funzionamento del recupero asincrono.

Innanzitutto, analizzando questo parametro, viene individuata la pagina ASP *FillTab.asp*. Essa viene considerata come una pagina di appoggio, il cui codice consente di recuperare parte della totalità del contenuto informativo da visualizzare all'interno del Manifesto degli Studi. Si tratta di una versione adattata del codice originario, in quanto anch'essa produce come risultato una o più tabelle complete di tutte le informazioni relative ai differenti orientamenti presenti all'interno di uno specificato anno di corso di un dato corso di studi.

La prima caratteristica evidente è che tale pagina porta a termine l'esecuzione del codice se e solo se viene chiamata attraverso due valori validi assegnati alle variabili che le permettono il funzionamento. Infatti si memorizzano subito tali valori attraverso

```
IdCSR = Request.QueryString("IdCSR")
AnnoTab = Request.QueryString("AnnoTab")
```

che vengono recuperati dalla stringa

```
?IdCSR=<%=IdCSR%>&AnnoTab=<%=AnnoCorrente%>
```

dove `AnnoCorrente` assume il valore che è stato spiegato all'inizio del codice della pagina *ManifestoAJAX.asp* quando si opera l'assegnamento attraverso il comando

```
AnnoCorrente = AnnoInizio
```

Questo valore poi viene automaticamente incrementato all'interno del ciclo già analizzato in precedenza, per consentire la naturale esecuzione del recupero dei dati.

È importante sottolineare come l'utilizzo di questa pagina di appoggio, e conseguentemente delle tecnologie Ajax, risulti completamente trasparente al sistema attuale, dal momento che la variabile `IdCSR` rimane utilizzata nello stesso modo della versione precedente, mentre la nuova variabile `AnnoCorrente`, che caratterizza poi `AnnoTab`.

Analizzando invece più oltre il codice della pagina di appoggio, si evidenzia come venga eliminato il controllo sull'anno corrente presente invece nel codice della pagina principale della rielaborazione, ovvero quella da cui parte la chiamata alla funzione, dal momento che il controllo sugli anni di corso viene effettuato preventivamente. Questo controllo viene individuato nel codice attraverso la condizione dell'if che viene evidenziata in grassetto. Una differenza di rilievo è anche individuabile nel ciclo for, qui presentato in grassetto ma appartenente esclusivamente alla prima versione del Manifesto degli Studi, che viene qui tralasciato dal momento che si operano interrogazioni specializzate sugli anni di corso.

```
for Anno = AnnoInizio to AnnoFine + 1
      if CInt(AnnoTab) < AnnoFine + 1 then
            StrSql =" SELECT ... "
      else
            StrSql =" SELECT ... "
      end if
      ...
Next
```

La descrizione della pagina *FillTab.asp* è sostanzialmente completata, anche perché non introduce particolari novità, ma realizza solamente una più specifica serie di interrogazioni non operando però volutamente in maniera ciclica.

- **`'inizializzaPanel<%=nPanel%>'`**

Per completare la descrizione della funzione responsabile del recupero dinamico dei contenuti, è necessario soffermarsi su quest'ultimo parametro. Esso infatti è il nome di una funzione contenuta all'interno del codice dello script chiamato `inizializza.js`. Le funzioni contenute in questo script effettuano le stesse operazioni di quella che si è incontrata all'inizio all'interno dei tag `<head>`, ovvero `inizializzaPanel1()`. Il suo compito infatti è quello di impostare l'altezza del relativo elemento della pagina al termine del caricamento dinamico, intuitivamente espandendone le dimensioni. Si approfondirà più avanti tale operazione, avendo così anche chiaro il funzionamento teorico della funzione Ajax prossima all'essere descritta in maniera particolareggiata.

4.2.2 Codice

Si ritiene ora opportuno presentare tutto il codice della funzione `ajax_loadContent()`, dal momento che presenta le caratteristiche peculiari delle tecnologie Ajax.

```
var enableCache = true;
var jsCache = new Array();
var dynamicContent_ajaxObjects = new Array();

/************************************************************************/
function ajax_loadContent(divId, url, callbackOnComplete)
/************************************************************************/
function ajax_loadContent(divId, url, callbackOnComplete) {

  if (enableCache && jsCache[url]) {
    document.getElementById(divId).innerHTML = jsCache[url];
    ajax_parseJs(document.getElementById(divId))
    evaluateCss(document.getElementById(divId))
    if (callbackOnComplete) {
      executeCallback(callbackOnComplete);
    }
    return;
  }

  var ajaxIndex = dynamicContent_ajaxObjects.length;

  document.getElementById(divId).innerHTML = 'Attendere la risposta del
  server...';

  dynamicContent_ajaxObjects[ajaxIndex] = new sack();

  if (url.indexOf('?') >= 0) {
    dynamicContent_ajaxObjects[ajaxIndex].method = 'GET';
    var string = url.substring(url.indexOf('?'));
    url = url.replace(string, '');
    string = string.replace('?', '');
    var items = string.split(/&/g);
    for (var no = 0; no < items.length; no++) {
      var tokens = items[no].split('=');
      if (tokens.length == 2) {
        dynamicContent_ajaxObjects[ajaxIndex].setVar(tokens[0], tokens[1]);
      }
    }
    url = url.replace(string, '');
  }

  dynamicContent_ajaxObjects[ajaxIndex].requestFile = url;
  dynamicContent_ajaxObjects[ajaxIndex].onCompletion = function() {
    ajax_showContent(divId, ajaxIndex, url, callbackOnComplete);
    };
  dynamicContent_ajaxObjects[ajaxIndex].runAJAX();
}
```

Parti del codice della funzione presa in considerazione sono stati evidenziati in grassetto poichè esse rappresentano ulteriori necessarie funzioni presenti all'interno del file `AJAX.js` ma che sono trasparenti alla implementazione eseguita sul codice del Manifesto degli Studi, e che quindi non si ha interesse ad approfondire.

Si sottolinea innanzitutto come la funzione in questione presenti tre parametri.

Per l'esecuzione della funzione si provvede inizialmente a definire tre variabili: `enableCache`, `jsCache` e `dynamicContent_ajaxObjects`.

La funzione esegue allora, sottoposte a un condizionamento sulla memoria cache del browser, tre chiamate a funzioni, presenti all'interno dello stesso file, evidenziate in grassetto, che permettono rispettivamente di individuare chiamate a script esterni da parte della pagina passata come parametro e di caricarli, di memorizzare le formattazioni imposte nei fogli di stile relativi e infine di preparare l'esecuzione di quella funzione che verrà messa in atto solo al termine della chiamata di caricamento dinamico.

Nel caso in cui il condizionamento individuato nel passaggio precedente non venga verificato, ovvero non sia immediatamente disponbile l'informazione perché già memorizzata nella cache, si procede allora proponendo l'alternativa alla visualizzazione fintanto che non sarà completata la funzione in questione. In questo caso è stata scelta la frase significativa *'Attendere la risposta del server...'*.

Si passa poi a creare un oggetto attraverso l'esecuzione della funzione `sack()`, che imposta tutti i parametri necessari a caratterizzare una chiamata di funzione di tipo Ajax.

Il tutto si completa con tre successive chiamate a funzione. La prima consente di individuare il file, ovvero la pagina ASP di appoggio da utilizzare per recuperare l'informazione corretta. La seconda invece permette di preparare l'esecuzione di una funzione, specificata in questo caso dal terzo parametro passato, quando viene dato il segnale di completamento e si esegue la funzione che visualizza a video il contenuto recuperato. Infine viene eseguita la vera e propria funzione Ajax.

4.3 Considerazioni sul costo computazionale dal punto di vista delle interrogazioni

Al fine di dimostrare direttamente i benefici introdotti dall'adozione dell tecnologie Ajax contestualmente alla reingegnerizzazione del Manifesto degli Studi, si ritiene molto importante effettuare un confronto tra il costo computazionale della versione iniziale e delle sua rielaborazione. Tale costo, però, verrà effettuato da un punto di vista che si considera di rilievo rispetto alla totalità delle operazioni da eseguire, ovvero quello delle interrogazioni del database della Facoltà. Questa scelta è giustificata dal fatto che l'onere maggiore, all'interno del codice della pagina, è dovuto al reperimento delle informazioni specifiche relative agli anni di corso di un dato corso di studi, e non alla realizzazione di stampe a video o assegnazioni a variabili. Si considereranno pertanto solamente la presenza di interrogazioni e costruzioni cicliche relative ad esse. È chiaro che queste approssimazioni rendano il calcolo impreciso rispetto al vero costo computazionale, ma dal momento che tali elementi ricoprono un ruolo più importante di tutti gli altri, il risultato del calcolo del costo computazionale si porta nella stessa direzione che si avrebbe se si effettuasse quello completo.

Si sottolinea inoltre che durante l'approfondimento del calcolo del costo delle operazioni da effettuare nel caso della versione rielaborata si considereranno solamente le interrogazioni che sono effettuate fino alla prima risposta HTTP ricevuta, ovvero fino al momento in cui il browser può cominciare ad usufruire dei contenuti visualizzandoli a video. In questa ipotesi è però chiaro che non si tiene conto delle operazioni effettuate in modo asincrono con Ajax, dal momento che il termine di paragone della performance risulta essere l'intervallo di tempo che permette di effettuare la visualizzazione a video, che, come già detto, dipende dal completamento della richiesta http effettuata.

Si affronta allora innanzitutto il calcolo sulla versione orginiaria del codice, ovvero quella proposta dalla pagina *Manifesto.asp*.

Si premette già che si trarranno conclusioni nell'ipotesi di trovarsi nel caso peggiore dell'analisi, ovvero, senza introdurre quantità numeriche definite, si considereranno tutti i risultati delle interrogazioni come non nulli. Di seguito si capirà la necessità di questa ipotesi.

Dopo un centinaio di linee di codice si incontra già una interrogazione particolare, ovvero quella che seleziona tutte le informazioni nominative del corso di studi individuato. Successivamente si seleziona il primo curriculum da visualizzare. Si seleziona quindi il nome dell'orientamento e le eventuali modalità di laurea, attraverso una ulteriore interrogazione. Seguendo l'ordine imposto dal codice, risulta chiaro che si effettuano dapprima tutte le selezioni sull'orientamento comune o unico, per poi lasciare spazio agli orientamenti presenti alternativi ad esso. Le prossime interrogazioni sono tutte contenute all'interno di un ciclo for, le cui condizioni dipendono dagli anni di corso presenti. Intanto è importante notare che viene subito effettuata una selezione sull'identificativo dell'insegnamento, in base però ad una condizione imposta in un if, che ne determina una esclusione vicendevole. Viene poi effettuata una ulteriore interrogazione, nel caso in cui l'ultima descritta non dia un risultato nullo. A seguito di questa selezione ne viene eseguita un'altra, la cui sintassi però dipende ancora una volta dal valore dell'anno di corso interessato dall'evoluzione del ciclo. Il risultato di quest'ultima interrogazione diventa condizione di un ciclo while, dentro al quale si esegue sicuramente una selezione per volta. Prima di terminare il contenuto di uno solo delle iterazioni dettate dal ciclo for esterno, si effettua una ulteriore interrogazione sulle note dell'anno di corso appena individuato. Si visualizzano allora le note dell'orientamento appena completato.

Si selezionano ora gli orientamenti differenti da quello comune o unico, mediante l'utilizzo di una nuova interrogazione. Il valore restituito da questa operazione condiziona la reiterazione del ciclo che la segue, definito da un ciclo `while`. Si porranno delle ipotesi successivamente su questo valore dipendente per contestualizzare il confronto. Entrati nel ciclo si incontra subito una ulteriore iterazione `for`, che è dovuta al numero di anni di corso presenti nel corso di studi specificato. Tale ciclo risulta veramente articolato al suo interno, innestando ricorsivamente delle iterazioni sempre controllate dai valori assunti

dalla selezione a cui ciascuna fa riferimento al livello superiore. Viene proposta qui di seguito una espressione letterale che permette di visualizzare la complessità computazionale, sotto le ipotesi inizialmente dichiarate di analisi nel caso peggiore.

```
S + S + S + for(S + S + S + while(S) + S) + S + S +
   + while(for(S + S + while(S + while(S)) + S))
```

che si può semplificare in questa forma:

```
5*S + for(4*S + while(S)) + while(for(3*S + while(S + while(S))))
```

dove è da intendersi nel modo seguente: `S` indica una interrogazione, una `select`; `for` indica che tutto il suo contenuto viene eseguito dentro un ciclo `for`; `while` che il suo contenuto viene eseguito all'interno di una iterazione di tipo `while`.

Passiamo ora all'analisi della nuova versione del Manifesto degli Studi, ovvero quella realizzata con la pagina *ManifestoAJAX.asp*.

Si incontra subito una prima interrogazione, dal cui contenuto non nullo dipendono altra due selezioni e un ciclo `while`. Quest'ultimo è condizionato dal valore della interrogazione che contiene. È da evidenziare comunque il fatto che la selezione interna viene effettuata una sola volta, perché dopo tale esecuzione il ciclo si interrompe.

Proseguendo nell'analisi, entrando già nella sezione relativa agli elementi costitutivi della visualizzazione a schede, si incontra una prima selezione, che individua le informazioni nominative del corso di studi cercato. Nell'ipotesi ora che tale selezione non produce un risultato nullo, si procede ora a interrogare il database per recuperare il primo curriculum da visualizzare nella tab di default. Successivamente si effettua la selezione del nome dell'orientamento ed, eventualmente, delle modalità di laurea. Segue quindi una selezione dell'identificativo dell'insegnamento. È necessario allora selezionare i periodi dell'anno accademico in questione e le relative date di di inizio e di fine. Il risultato di questa interrogazione viene sfruttato per individuare i nomi e i dati degli insegnamenti presenti all'interno di ogni periodo. Allo stesso modo si individuano i relativi docenti che tengono

questi insegnamenti, con una apposita interrogazione. Si provvede allora ad individuare le note dell'orientamento appena stampato.

La seconda parte del codice procede nella selezione di tutte le informazioni degli orientamenti differenti da quello comune o unico. Da questo punto in poi si eseguono le stesse interrogazioni appena descritte per l'orientamento comune.

```
S + S + S + while(S)* + S + S + S + S + S + S + S + S +
                + while(S + S + S + S + S + S)
```

Ora, potendo approssimare `while( S )*` con `S`, dal momento che con questa notazione particolare si indica l'esecuzione per una sola e unica volta dell'interrogazione che contiene, è possibile semplificare quest'ultima espressione come

```
12*S + while ( 6*S )
```

che è di sicuro più semplice di quella espressa nel caso della versione originaria del Manifesto degli Studi.

Mettendo ora a confronto ravvicinato le due espressioni

```
5*S + for(4*S + while(S)) + while(for(3*S + while(S + while(S))))
```

```
12*S + while ( 6*S )
```

si può osservare come compaia un primo termine, che è evidenziato in rosso, che rappresenta un ciclo `for`, e come risulti molto più articolato il contenuto del termine `while` comune ad entrambe le espressioni. In particolare è evidente che il contenuto nella seconda versione di questo termine ha un andamento lineare, mentre nel primo caso esso dipende direttamente da un ciclo in esso innestato, oltre che essere appesantito da un ulteriore ciclo `while` necessariamente da eseguire.

Dal confronto del costo computazionale, nonostante le approssimazioni e l'ipotesi del percorso peggiore da studiare, è chiaro che la versione rielaborata con le tecnologie Ajax risulta avere una risposta molto più pronta all'atto della richiesta al server. La richiesta

HTTP avanzata dal client viene infatti esaurita in un tempo percettibilmente minore, le cui operazioni de eseguire sono identificabili nelle espressioni suddette. Ciò significa che la visualizzazione delle informazioni nelle due versioni comincerà in istanti differenti. Ipotizzando infatti che interrogazioni identiche vengano eseguite in intervalli temporali identici, è chiaro che la il browser potrà permettere, nel caso della versione del Manifesto degli Studi rielaborata, una visualizzazione dei contenuti molto più veloce.

4.4 Rielaborazione grafica

Per completare l'illustrazione di tutte le operazioni effettuate per mettere in atto la reingegnerizzazione del Manifesto degli Studi, è importante descrivere anche le novità introdotte sotto l'aspetto grafico.

Si è già descritta precedentemente la struttura che è stata implementata per realizzare la visualizzazione a schede a transizione scorrevole, ma si vuole ora accennare solamente ad alcuni altri importanti parametri che concorrono alla riorganizzazione grafica della pagina.

Con questa ulteriore intenzione si vuole far focalizzare l'attenzione sia al nuovo foglio di stile realizzato che alle funzioni che permettono la dinamicità dello scorrimento dei contenuti. Per quanto riguarda il primo elemento dell'approfondimento, si parlerà del foglio di stile `manifestoAjax.css`, mentre nel secondo caso degli script `inizializza.js`, `JQfunction.js` e `JQScrollTo.js`. In particolare si premette già che verranno analizzati solamente il foglio di stile e il primo degli script elencati, dal momento che i successivi concorrono solamente ad eseguire tutta la serie di operazioni necessarie a realizzare l'espediente grafico dello scorrimento, ma che soprattutto sono disponibili in rete in un apposito pacchetto per consentire uno sfruttamento immediato senza la necessità di parametrizzarle. Tale infatti è stato l'utilizzo che se ne è fatto all'interno della rielaborazione prodotta.

4.4.1 Foglio di stile `manifestoAjax.css`

Si ritiene utile proporre innanzitutto una descrizione del foglio di stile introdotto appositamente per operare la formattazione degli elementi propri della visualizzazione a schede a transizione scorrevole adottata.

Occorre indubbiamente definire il significato e l'importanza dei fogli di stile, denominati CSS, ovvero *Common Style Sheets*. Essi sono files che racchiudono tutte le formattazioni grafiche di una determinata serie di elementi contenuti all'interno di una pagina. Sono utili nel caso di una gestione dinamica dei contenuti perché permettono di richiamare le impostazioni di formattazione attribuendo delle classi specifiche, volutamente create all'interno dei fogli di stile, a qualsiasi elemento venga creato nella pagina ASP.

Prima di approfondire il foglio di stile `manifestoAjax.css`, è necessario premettere che è stato fondamentale modificare il foglio di stile `imposta.css`, che risulta però comune a qualsiasi pagina del portale della Facoltà, nella parte relativa alla formattazione dell'elemento `content`, ripulendolo di qualsiasi specifica precedente.

```
#content{
}
```

Tale azione è stata dettata dalla volontà di adottare la visualizzazione a schede, ma soprattutto dalla specifica struttura divisionale che le realizza. In assenza di questa, infatti, non sarebbe stata necessaria alcuna modifica.
A questo punto è possibile presentare una parte significativa del contenuto del foglio di stile realizzato.
La parte di cui si parla è quella relativa all'elemento `mask`, che ha un ruolo fondamentale nell'attuazione dello scorrimento delle transizioni tra schede.

```
#mask {
      width:625px;
      overflow:hidden;
      margin-left: 120px;
      background: white;
      vertical-align: top;
      text-align:justify;
      border:2px solid Grey;
}
```

Si deve porre l'attenzione sull'attributo `overflow:hidden` che consente di poter utilizzare la `mask` come se fosse una finestra fissa al centro dello schermo, che limita la visibilità di una sola parte del contenuto che in realtà possiede. Ricordando infatti la figura

4.3 di questo capitolo, è chiara l'associazione che si è fatta tra la maschera e una finestra a scorrimento, che permette una visualizzazione parziale in un particolare istante.

4.4.2 *Script* `inizializza.js` *e funzione* `inizializzaPanel<%=nPanel%>()`

Si completa l'approfondimento con l'illustrazione del codice da cui dipende il funzionamento a transizione a scorrimento dei contenuti degli elementi associati alle tab.

La funzione `inizializzaPanel<%=nPanel%>()` viene chiamata all'inizio del codice e poi successivamente come terzo parametro della chiamata alla funzione Ajax, permettendo, all'atto del completamento della richiesta HTTP, o rispettivamente del completamento della stessa chiamata Ajax, di impostare l'altezza della `mask` in modo corretto, ovvero della stessa altezza della totalità dei contenuti relativi ad un anno di corso, e di gestire prontamente l'altezza della stessa nel momento in cui si passa da una tab all'altra.

Un passaggio importante, ad esempio ponendoci nel primo caso che incontriamo nel tag `<head>`, di tale funzione è il seguente

```
$('#mask').css({ 'height': $('#panel-1').height() });
```

che mette in atto l'impostazione dell'altezza del primo sottoelemento in base alla quantità dei suoi contenuti. È importante sottolineare che tale elemento è quello che viene realizzato per default, come già spiegato, secondo la volontà di determinate scelte progettuali.

Un ulteriore comando da sottolineare è il seguente:

```
$('#mask').scrollTo($(this).attr('href'), 800);
```

che permette di chiamare la funzione che si adopererà a realizzare la transizione a scorrimento, che qui non si affronta. Il valore numerico indicato è il tempo in millesimi che si deve impigare per completare la transizione.

Lo script .js contiene semplicemente i codici della funzione appena descritta, adattata però ad ogni iterazione.

Senza l'utilizzo di tale funzione, né all'inizio del codice, né al termine di ogni chiamata Ajax, non si otterrebbe affatto un risultato graficamente gradevole.

Capitolo V

Conclusioni e Progetti futuri

L'attività sperimentale svolta ha avuto come obiettivo la reingegnerizzazione del Manifesto degli Studi della Facoltà di Ingegneria dell'Università di Modena e Reggio Emilia attraverso l'introduzione e la corretta implementazione delle tecnologie AJAX, al fine di migliorarne le capacità prestazionali.

Uno dei requisiti fondamentali ha riguardato la trasparenza della nuova versione rispetto al sistema attuale, e il mantenimento di questo è stato dimostrato all'interno dei capitoli II e IV. Si è evidenziato inoltre come la pagina ottenuta dalla rielaborazione sia perfettamente sovrapponibile alla precedente, senza incorrere in modifiche dell'intero sistema, e come sia altrettanto efficace nel presentare gli stessi contenuti. Nel capitolo IV è stata inoltre descritta la rielaborazione grafica effettuata, che consente innanzitutto di sfruttare al meglio le funzionalità del recupero asincrono delle informazioni con AJAX, e inoltre di rendere più appetibile e navigabile la pagina web del Manifesto degli Studi. Tale nuova veste grafica presenta ora una visualizzazione dei contenuti a schede a transizione scorrevole. Sempre all'interno del capitolo IV è stata infine approfondita l'innovativa funzione che si rende responsabile del recupero asincrono dei contenuti, i cui fondamenti teorici sono stati dapprima esposti nell'intero capitolo III.

L’attività progettuale svolta mi ha innanzitutto permesso di conoscere più da vicino una delle tante realtà del mondo dell’informatica, restandone affascinato, ma ha anche prodotto un risultato concreto e al passo con i tempi che potrebbe benissimo essere sfruttato come punto di partenza per una reingegnerizzazione su più fronti del portale della Facoltà di Ingegneria.

Bibliografia

[1] AJAX: A New Approach to Web Applications
http://www.adaptivepath.com/publications/essays/archives/000385.php

[2] W3C – World Wide Web Consortium
http://www.w3c.org

[3] Visual Studio Web Developer 2008 Express Edition
http://www.microsoft.com/express/vwd/

[4] MSDNAA
http://www.msdnaa.net

[5] Wikipedia
http://it.wikipedia.org/

[6] Guida ASP
http://www.acerbi.re.it/corsoasp/default.asp

[7] Guida AJAX
http://javascript.html.it/guide/leggi/95/guida-ajax/

[8] Tutorial per realizzare un menu a tabs scorrevoli con jquery
http://www.queness.com/post/274/jquery-sliding-tab-menu-for-sidebar-tutorial

[9] Pacchetto JQScrollTo
http://flesler.blogspot.com/2007/10/jqueryscrollto.html

[10] Pacchetto per la gestione del contenuto dinamico con Ajax
http://www.dhtmlgoodies.com/index.html?whichScript=ajax-dynamic-content

[11] Tutorial per realizzare un menu a tab attraverso i fogli di stile
http://css.html.it/articoli/leggi/418/un-menu-a-tabs-con-i-css/

[12] Tutorial per la realizzazione di tab a contenuto con transizione scorrevole
http://javascript.html.it/script/vedi/5306/sliding-tabs/

Ringraziamenti

Questa tesi segna formalmente la conclusione di un periodo della mia vita piuttosto particolare. Con la sua discussione si chiude la mia avventura a Modena cominciata tre anni fa, e si apre già però una nuova esperienza lavorativa e di studio. Il lavoro che sta dietro a questa tesi mi ha insegnato molto, perché mi ha permesso di scoprire più da vicino una parte del mondo dell'informatica, perché mi ha insegnato veramente cosa significa programmare, perché mi ha insegnato la soddisfazione nel realizzare qualcosa che funziona e piace, ma soprattutto perché mi ha permesso di conoscere e imparare. Mi è piaciuto molto lavorare a questa tesi, e mi ha confermato che ho scelto bene la strada che sto percorrendo.

Credo di poter definire con certezza che questa è stata la mia prima vera esperienza lavorativa. Ora che si è conclusa, mi accorgo che ha richiesto molto da me, ma che non ha coinvolto solo me stesso.

È quindi giusto e doveroso ringraziare tutte quelle persone che ho avuto il piacere di conoscere e da cui ho avuto modo di imparare qualcosa, sia a livello umano che in ambito professionale, perché l'esperienza che ho fatto sia in questi anni che in quest'ultimo periodo mi ha migliorato e cresciuto.

Ma prima di tutto ringrazio la mia famiglia.