

POLITECNICO DI BARI

I FACOLTA' DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

**TESI DI LAUREA IN
SISTEMI INFORMATIVI**

*SISTEMA PER LA VERIFICA AUTOMATICA
DELLE PROPRIETA' DI SITI WEB DINAMICI
CON L'USO DI METODI FORMALI*

Relatori:

Chiar.mo Prof. Eugenio Di Sciascio

Chiar.mo Prof. Giacomo Piscitelli

Correlatrice:

Chiar.ma Dott.ssa Marina Mongiello

Laureando:
Graziano Nardelli

ANNO ACCADEMICO 2001 – 2002

Ai miei genitori

Indice

<u>Capitolo 1: Applicazioni web e metodi di verifica</u>	1
1.1 Metodi di verifica di un'applicazione web	2
1.2 Grado di fiducia di un'applicazione	4
1.3 Alcune caratteristiche delle applicazioni web	5
1.3.1 Pagine statiche e pagine dinamiche	5
1.3.1.1 Dinamicità lato server	6
1.3.2 Protocollo HTTP	10
1.3.2.1 Form	11
1.3.2.2 Metodo Post	11
1.3.2.3 Metodo Get	11
1.4 Organizzazione del metodo di verifica delle applicazioni web	13
<u>Capitolo 2: Model checking</u>	15
2.1 Introduzione	16
2.2 Model Checking	17
2.3 Un model checker simbolico: NuSMV	18
2.4 Logica CTL	21
2.4.1 Semantica della logica CTL	23
2.4.2 Il model checker SMV (Symbolic Model Verifier)	27
<u>Capitolo 3: Struttura interna del progetto</u>	39
3.1 Modello a stati finiti di un sito web	40
3.2 Modello a stati finiti usato nelle verifiche offline	42
3.3 Modello a stati finiti usato nelle verifiche online	46
3.4 Determinazione e utilizzo dei modelli a stati finiti	48
3.5 Codice in ingresso a SMV dei modelli realizzati	50
3.5.1 Codice in ingresso a SMV dei modelli offline	50
3.5.2 Codice in ingresso a SMV dei modelli online	52
3.6 Specifiche in logica CTL utilizzate nelle verifiche automatiche	53

3.7 Verifiche automatiche eseguite per siti che si basano sul commercio elettronico _____	63
3.7.1 Struttura del commercio elettronico con home page personale _____	64
3.7.2 Struttura del commercio elettronico senza home page personale _____	65

Capitolo 4: Descrizione del progetto _____ 71

4.1 Introduzione _____	72
4.2 Installazione Software _____	72
4.3 Utilizzo del software _____	73
4.3.1 Parsing dalla scheda principale _____	74
4.3.2 Parsing dalla scheda mirror _____	75
4.4 Differenza tra parsing offline e parsing online _____	76
4.5 Come eseguire il parsing _____	77
4.6 Risultati del parsing di un sito _____	78
4.7 Salvataggio dei dati _____	80
4.8 Determinazione del codice SMV _____	80
4.9 Verifiche CTL _____	82
4.9.1 Varie verifiche (su <i>SMV offline</i>) _____	83
4.9.2 Verifica di Raggiungibilità delle pagine (su <i>SMV offline</i>) _____	85
4.9.3 Verifica di Connessione delle pagine (su <i>SMV offline</i>) _____	85
4.9.4 Presenza degli oggetti in tutte le pagine o in una determinata pagina (su <i>SMV offline</i>) _____	86
4.9.5 Provenienza oggetto (su <i>SMV offline</i>) _____	88
4.9.6 Pattern di Specifica (su <i>SMV offline</i>) _____	88
4.9.7 Dimensione e data (su <i>SMV offline</i>) _____	89
4.9.8 Varie verifiche (su <i>SMV online</i>) _____	90
4.9.9 Verifica di Raggiungibilità delle pagine (su <i>SMV online</i>) _____	92
4.9.10 Connessione di una determinata pagina (su <i>SMV online</i>) _____	93
4.9.11 Gestione manuale _____	94
4.10 Come vengono visualizzati i risultati delle verifiche CTL _____	95
4.11 Altre Verifiche _____	96
4.11.1 File e URL assenti e presenti _____	97
4.11.2 File non utili _____	98
4.11.3 Raggiungibilità delle pagine in modo globale _____	99
4.11.4 Contenuto delle pagine online rispetto a quelle offline _____	100

4.11.5	Verifica delle ancore	101
4.12	Commercio Elettronico	102
4.12.1	Associazioni	102
4.12.1.1	Associazioni Pagine	103
4.12.1.2	Associazioni Password	104
4.12.2	Verifiche	105
4.12.2.1	Verifiche offline	105
4.12.2.2	Verifiche con le password	106
4.13	Help online	107
4.14	Modifica ed esecuzione dell'ultimo Codice SMV	108

Capitolo 5: Esempi di verifiche applicati al sito CDMarket 109

5.1	Introduzione	110
5.2	Descrizione di CDMarket	110
5.3	Verifica di CDMarket con DaWeb	112
5.3.1	Varie verifiche senza CTL fatte per l'esempio CDMARKET	113
5.3.2	Verifiche Offline fatte per l'esempio CDMarket	116
5.3.3	Verifiche Online fatte per l'esempio CDMARKET	119
5.3.4	Verifiche di siti basati sul commercio elettronico per CDMarket	123

Appendice A: Descrizione interna del progetto 128

A.1	Introduzione	129
A.2	Descrizione delle singole classi	129
A.3	Diagrammi appartenenti al linguaggio UML	137
A.3.1	Collegamento tra frame	137
A.3.2	Classi per determinare i Moduli SMV	138
A.3.3	Classi per eseguire il parsing online	139
A.3.4	Classi per eseguire il parsing offline	140
A.4	Analisi di DaWeb con i diagrammi DFD	141

<u>APPENDICE B: Schemi per pattern di specifica per CTL e loro utilizzo</u>	147
B.1 Introduzione	148
B.2 Pattern di specifica proposti da Dwyer	150
B.2.1 Assenza	151
B.2.2 Esistenza	151
B.2.3 Esistenza limitata	152
B.2.4 Universalità	152
B.2.5 Precedenza	152
B.2.6 Risposta	153
B.2.7 Catena di Precedenza	153
B.2.8 Catena di Risposta	154
B.2.9 Pattern di catena vincolata	155
B.3 Utilizzo degli schemi nel progetto	155
B.4 Significato delle varie estensioni nel progetto	158
<u>Appendice C: Codice SMV offline e codice SMV online</u>	162
C.1 Introduzione	163
C.2 Codice SMV offline	163
C.3 Codice SMV online	169
<u>Conclusioni</u>	182
Descrizione sintetica delle caratteristiche del progetto	182
Suggerimenti per ampliare le funzionalità del progetto	182

Introduzione

Descrizione progetto

In questa tesi si affronta il problema della verifica delle proprietà di siti web sia statici che dinamici, mediante l'uso di **metodi formali** che garantiscano la correttezza del progetto sulla base della verifica matematica del modello del sistema. Inoltre, questi metodi, consentono di definire il progetto di un sistema in termini di specifiche precise e non ambigue fornendo la base per un'analisi sistematica. Tra i metodi formali di maggiore rilevanza, il metodo scelto è quello del **model checking**, in quanto, rispetto ad altri metodi presenti sul commercio, permette di gestire le verifiche in maniera del tutto automatica.

Installazione di DaWeb.

Il Cd allegato presenta una struttura abbastanza semplice per installare **DaWeb** e tutti i suoi componenti; infatti è presente una schermata, che si avvia automaticamente nel momento in cui si inserisce il CD, che ci fa da guida su ciò che si vuole fare.

Prima di eseguire l'installazione di *DaWeb*, occorre installare, se non è già presente nel sistema, il *JDK1.3* (si richiede che il *JDK1.3* sia allocato in *c:/jdk1.3*). Inoltre, sempre se non è presente nel sistema, bisogna installare anche il *Personal Web Server (PWS)* se si vogliono analizzare le pagine *asp* e l'*easyPHP* (pacchetto contenente il server *apache* e *mysql*) se si vogliono analizzare le pagine *php*. (È possibile analizzare anche altri tipi di pagine dinamiche ma in questo caso è indispensabile avere a disposizione altri tipi server, diversi dal *PWS* e dal server *apache*, capace di elaborarli).

All'interno del CD sono presenti anche il sorgente Java di *DaWeb*, la tesi in formato elettronico e anche tutti i programmi freeware e shareware utilizzati per lo sviluppo di tutto il lavoro.

Metodo di realizzazione del progetto

DaWeb è stato realizzato utilizzando il linguaggio di programmazione Java. È stato utilizzato questo linguaggio perché è orientato agli oggetti e possiede numerose funzioni utili per eseguire interazioni con la rete. È stato scelto questo linguaggio anche perché è multiplatforma. Ciò è fondamentale per questa applicazione visto che viene spesso allocata in un server che per motivi commerciali quasi sempre ha un sistema operativo diverso da Windows [1].

Il progetto è suddiviso in molte classi (dove ogni classe è contenuta in un file *.class*) che interagiscono tra loro. Queste classi sono commentate in appendice A.

Sempre in Appendice A sono stati realizzati dei diagrammi di analisi (con il linguaggio UML) utili per comprendere meglio sia le interazioni delle classi, sia la suddivisione dei compiti.

Capitolo 1

APPLICAZIONI WEB E METODI DI VERIFICA

1.1 Metodi di verifica di un'applicazione web

Realizzare un'applicazione web, è un'operazione che ormai tutti possono compiere. Per le applicazioni statiche il procedimento è molto elementare; basta ormai pensare che ci sono molti programmi applicativi che realizzano pagine in modo automatico, procedendo in maniera grafica in un modo molto simile alla scrittura di un documento con un text editor. Per le applicazioni dinamiche le cose sono un po' più complicate; infatti si devono necessariamente realizzare degli script che dovranno essere eseguiti da un server opportuno. Entrambe le applicazioni possono essere comunque realizzate da qualsiasi persona. Questi possono realizzare qualsiasi cosa e metterla in rete. I numerosi grandi pregi di tutto questo hanno un grande difetto: oltre ad esserci in rete progetti professionali, sicuramente ci sono anche progetti di scarso contenuto e anche poco funzionali.

Per realizzare un progetto professionale, un web designer, deve avere molta esperienza, in diversi campi. Deve infatti curare la grafica, gestire le interfacce e i collegamenti tra le pagine, creare degli script ad hoc, strutturare un database; deve anche capire quali sono le scelte migliori da fare nei vari settori tenendo anche conto delle esigenze commerciali.

Visto la grande varietà di cose da fare, per realizzare un'applicazione web, spesso risulta inevitabile suddividere il lavoro tra più persone.

Anche se un'applicazione web nasce spesso da un'idea, la sua struttura viene quasi sempre realizzata basandosi su uno schema dipendente dal tipo di applicazione che si vuole svolgere. Nello schema ci sono anche delle regole che devono essere rispettate, dall'applicazione che si sta realizzando, in modo da determinare alla fine qualcosa di professionale. (Questo schema spesso risulta personalizzato dal web designer, in base alla propria esperienza e alle proprie idee).

Sia chi realizza un'applicazione web professionale che chi realizza un'applicazione poco professionale, deve assolutamente eseguire alla fine delle verifiche.

La verifica è indispensabile per sostenere il controllo di qualità della progettazione di sistema.

L'affidabilità di elaborazione dell'informazione è un punto chiave nel processo di progettazione di sistema. Il sistema di verifica potrebbe essere considerato il momento più importante dell'intero progetto perché solo in questo modo si può realmente sapere se il proprio prodotto finale sarà o no corrispondente alle aspettative del futuro acquirente. In molti progetti, infatti, gran parte del tempo e degli sforzi sono spesi per la verifica piuttosto che per la costruzione! Ciò perché se il sistema di verifica è efficiente la casa produttrice acquista prestigio in quanto è in grado di fornire le prestazioni e la sicurezza richiesta; in caso contrario potrebbe addirittura entrare in gioco il futuro dell'azienda stessa visto che nel mondo d'oggi è difficile e ci vuole tempo per diventare una casa produttrice affermata e di qualità ma ci vuole poco per perdere la fiducia dei propri consumatori.

Alcune tecniche di verifiche sono la simulazione e il testing.

La **simulazione** è basata su un modello che descrive i possibili comportamenti del sistema. Si tratta di un modello eseguibile: infatti uno strumento software - simulatore - può determinare i comportamenti del sistema sulla base di alcuni scenari che possono essere generati, ad esempio, in modo casuale. La simulazione può fornire, quindi, una prima rapida verifica della qualità del progetto, ma non è adatta a riscontrare errori; essa può, infatti, garantire l'analisi solo di alcuni comportamenti del sistema.

Il **testing** richiede l'implementazione di un prototipo funzionante del sistema, a cui fornire opportuni input di test ed osservare la reazione del sistema per poter stabilire se è conforme alle aspettative. Sostanzialmente, il testing si basa su assunzioni analoghe a quelle della simulazione con la differenza che la simulazione è basata su un modello del sistema, mentre il test su un'implementazione reale. Pur essendo molto utilizzato nella pratica, il testing si basa su un approccio euristico; inoltre analizza solo un sottoinsieme di tutti i possibili comportamenti del sistema, pertanto non può garantire l'assenza di errori, ma solo individuarne l'eventuale presenza. Ciò è tanto più vero quanto più aumenta la complessità del progetto.

Poiché tali metodi sono eseguiti mediante un simulatore, nel caso della simulazione e su un prototipo, nel caso del testing, risultano essere anche molto costosi. Negli anni '80 è stata introdotta una nuova tecnica per risolvere il problema del "design validation" che si basa sull'approccio della verifica formale. Diversi approcci sono stati

proposti per implementare questo nuovo metodo, ed uno di questi è il **model checking** che sarà descritto dettagliatamente nel secondo capitolo. [9]

1.2 **Grado di fiducia di un'applicazione**

L'impatto di sistemi software o hardware sulla società e quindi sul mercato, è strettamente legato al livello di fiducia che può essere loro dato.

Il fatto di avere fiducia in un'applicazione spesso è legato al suo buon funzionamento ed alla sua correttezza: noi, infatti, ci fidiamo di un sistema software o hardware se crediamo che tale sistema faccia sempre quello per cui è stato progettato, nel modo in cui è stato stabilito.

Il grado di fiducia richiesto da un'applicazione spesso varia a seconda del rischio che si corre nel caso di un suo eventuale funzionamento scorretto. Molte applicazioni, come ad esempio editor o fogli elettronici, non richiedono molta fiducia nella loro correttezza. Altre applicazioni hanno invece bisogno di un alto grado di fiducia. Sono infatti le applicazioni classificate come sistemi *safety-critical*, ovvero quei sistemi il cui funzionamento scorretto può mettere in gioco delle vite umane, o causare gravi danni all'ambiente, o infermità alle persone. Esempi tipici sono i sistemi che gestiscono il traffico aereo o il controllo delle ferrovie.

Un'altra area di applicazioni di alta criticità che si sta diffondendo sempre più rapidamente con lo sviluppo del commercio elettronico, è quella dei sistemi *money-critical*. Si tratta di sistemi il cui funzionamento scorretto può causare gravi perdite di denaro.

In fase di sviluppo di un progetto, eseguire verifiche, permette di affermare il rispetto di alcuni parametri di qualità, facendo aumentare quindi il grado di fiducia. [8]

1.3 Alcune caratteristiche delle applicazioni web

Dopo aver descritto l'importanza della verifica nei diversi sistemi, sarà considerato adesso come affrontare le verifiche alle applicazioni web. Prima però di entrare nel dettaglio, occorre descrivere alcuni aspetti, di queste applicazioni, per capire inoltre alcune soluzioni adoperate.

1.3.1 Pagine statiche e pagine dinamiche

Nello schema tipico di interazione del World Wide Web, il **web client** (ad esempio il browser) fa richiesta di una URL ad un **web server**. Tipicamente, tale URL corrisponde ad un file contenuto nella memoria di massa del computer su cui è in esecuzione il web server. Tale file può essere di due tipi:

1. Il file corrisponde ad una risorsa “statica”, nel senso che non contiene programmi, ma un insieme di dati (ad esempio, documenti HTML, immagini, suoni, filmati) che vengono “aperti” dal browser, o da applicazioni collegate al browser. Pertanto, il web server esegue semplicemente la trasmissione di tale file al web client. Questo prende il nome di modello a server statico (fig. 1-1).

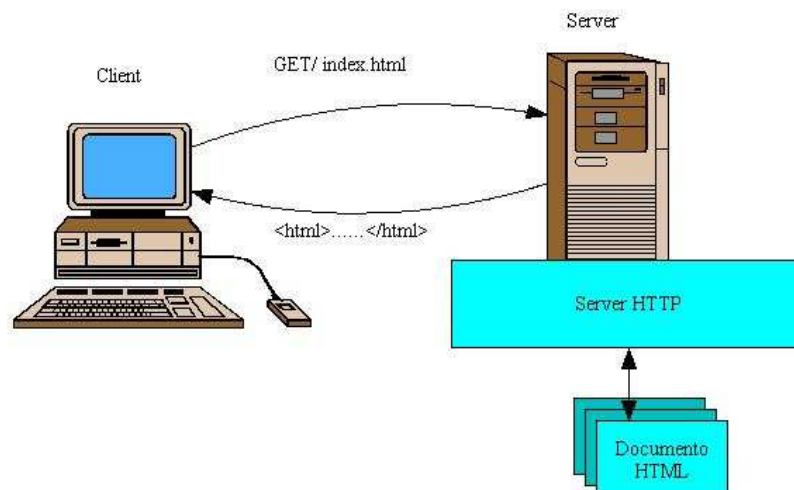


Figura 1-1: Modello del server che invia le pagine statiche

2. Il file contiene un programma (o più programmi). Tale programma può essere destinato ad essere eseguito dal web client oppure dal web server stesso. Si distinguono pertanto due casi:
 - ***dinamicità lato client***: il file rappresenta un programma (o contiene al suo interno uno o più programmi) che, una volta trasmesso, deve essere eseguito dal web client (browser). Anche in tal caso, il web server effettua semplicemente la trasmissione di tale file verso il web client (i due tipi di programmi che possono essere collegati ad un documento html sono gli applet e gli script). Il modello fisico, anche in questo caso, è a server statico (fig. 1-1)
 - ***dinamicità lato server***: il file contiene un programma che viene eseguito dal web server all'atto della richiesta del client, la cui esecuzione provoca la generazione, da parte del server, di una risorsa (tipicamente un documento HTML) che viene spedita al client come risposta alla sua richiesta. La risorsa viene quindi *generata dinamicamente* dal web server al momento della richiesta del web client. (fig. 1-2)

Quest'ultimo punto sarà approfondito nel prossimo paragrafo.

1.3.1.1 Dinamicità lato server

Nella generazione dinamica di pagine web, il web server può essere visto come un "ponte" (gateway) tra una sorgente di informazione e l'utente di tale informazione. In pratica, il web server in tali casi diventa un'entità "attiva" che, in risposta alla richiesta di un web client, recupera una serie di informazioni da una sorgente informativa (in questo caso si parla anche di n-tier) e quindi genera uno o più documenti ipertestuali che riportano tali informazioni.

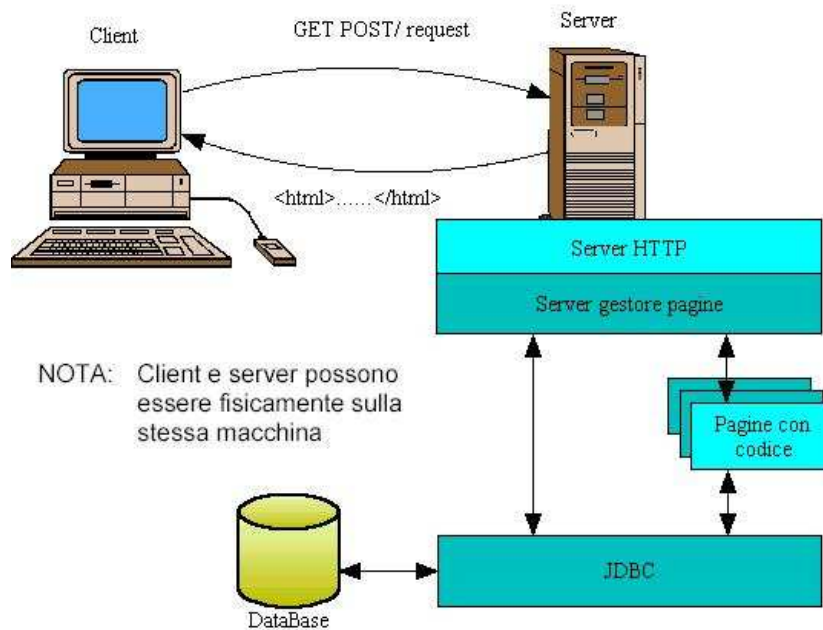


Figura 1–2: Modello del server che genera le pagine dinamiche

L'utilizzo della generazione dinamica dei documenti HTML (e più in generale, di risorse web) è *necessario* in tutte le applicazioni dove l'informazione non è statica oppure non è "pre-formattabile". Ad esempio:

- sorgenti in cui l'informazione cambia rapidamente: news, meteo, basi di dati transazionali (cioè archivi di dati contenenti informazioni che cambiano in tempo reale), ecc.;
- applicazioni in cui (pre-)esiste una grande mole di informazioni non- HTML che si vuole rendere accessibile via web;
- applicazioni in cui l'informazione è selezionata e presentata in base alla richiesta dell'utente: la tipica applicazione di questo tipo è quella dei motori di ricerca, in

cui la pagina web da presentare dipende dalla interrogazione (query) che l'utente ha fatto al motore di ricerca;

- applicazioni in cui l'informazione è personalizzata in base al comportamento dell'utente: questo tipo di applicazioni, chiamate "siti web adattativi" (dall'inglese *adaptive web sites*), è utilizzata soprattutto nelle applicazioni di commercio elettronico, in cui si cerca di capire che tipo di utente sta accedendo al sito web, per potergli proporre gli articoli in vendita (o la pubblicità) più vicini ai suoi interessi;
- applicazioni in cui l'utente trasmette informazioni al sistema (di solito attraverso moduli): ad esempio, siti che richiedono la registrazione dell'utente, siti di Internet provider, ecc. In tali casi, è spesso necessario da parte del server generare pagine web il cui contenuto dipende dall'informazione che l'utente ha trasmesso al server (ad esempio, pagine di riepilogo delle informazioni trasmesse dall'utente, che vengono presentate all'utente a scopo di conferma).

In tali applicazioni, si chiede al web server che riceve una richiesta da un client, di *generare* pagine web (documenti HTML) al momento della richiesta, utilizzando opportuni programmi che, in base alla richiesta ricevuta e/o allo stato della sorgente informativa a cui il web server è collegato, generano uno o più documenti HTML che vengono inviati come risposta alla richiesta del client.

In base al tipo (linguaggio di programmazione) di programmi utilizzati, si identificano le seguenti principali tecnologie utilizzate per la generazione dinamica di documenti sul web:

- *CGI (Common Gateway Interface)*. I programmi CGI (chiamati anche CGI-BIN) sono una particolare classe di programmi scritti nel linguaggio C. La richiesta,

da parte di un client, di una URL corrispondente ad un programma CGI causa l'esecuzione, sul web server, di tale programma, che risiede sul server in versione compilata nel linguaggio macchina della CPU del computer su cui è in esecuzione il web server: ricordiamo infatti che il C è un linguaggio compilato. L'esecuzione del CGI ha come risultato la generazione di un documento (o più documenti) HTML. Storicamente, quella dei programmi CGI è stata la prima tecnologia apparsa nelle applicazioni web per la generazione dinamica di risorse: attualmente, tale tecnologia appare superata da quelle più recenti basate, ad esempio, su JAVA.

- *Java Servlet.* è una particolare classe di programmi JAVA. I servlet possono essere considerati la versione “lato server” degli applet: la parola servlet è infatti l'abbreviazione di “server applet”. La richiesta, da parte di un client, di una URL corrispondente ad un JAVA servlet causa l'esecuzione, sul web server, di tale programma, che, come nel caso degli applet, è in versione compilata, cioè espresso in bytecode JAVA. L'esecuzione del servlet ha come risultato la generazione di un documento (o più documenti) HTML.
- *ASP (Active Server Page).* A differenza dei precedenti, ASP è un linguaggio di scripting, che pertanto *non* viene compilato, ma interpretato dal web server. In particolare, ASP permette di scrivere documenti HTML contenenti script, cioè programmi, scritti nel linguaggio Visual Basic. La differenza con gli script JavaScript visti in precedenza è che in ASP gli script vengono eseguiti sul web server e non sul web client (browser). Il caricamento di un documento HTML contenente script ASP sul server comporta l'esecuzione degli script contenuti nel documento: l'esecuzione di tali script “completa” il documento HTML iniziale con parti del documento generate dinamicamente. L'utente pertanto riceve un normale documento HTML senza script ASP;

- *JSP (Java Server Page)*. Come il precedente, anche questo è un linguaggio di scripting, tuttavia, a differenza di ASP, l'esecuzione di uno script JSP non ha come risultato la generazione di un documento HTML, ma la generazione di un programma Java Servlet: a sua volta, l'esecuzione di tale programma servlet, come visto in precedenza, ha per effetto la generazione del documento HTML. Pertanto, lo scopo del linguaggio JSP è quello di semplificare la generazione dei programmi JAVA Servlet
- *PHP (Personal Home Page)*. Chiamato a volte, erroneamente, "Pre-Hypertext Processor", anche questo linguaggio è un linguaggio di scripting per il server. Tale linguaggio è molto semplice da apprendere e può essere utilizzato in particolare per interfacciare il web server con una base di dati.

Per far comunicare il client con il server ci deve essere un protocollo di comunicazione. Questo è il protocollo http: hypertext transfer protocol.

1.3.2 Protocollo HTTP

Introduciamo adesso alcune caratteristiche del protocollo http, utili per mandare informazioni dal client al server e viceversa. Le informazioni che partono dal client, che prendono il nome di richieste, vengono analizzate dal server la quale produce i risultati che vengono alla fine rispediti al client che ha fatto la richiesta.

Dal client, le informazioni vengono mandate al server sfruttando il tag form del codice html.

1.3.2.1 Form

Il form è il principale strumento di interazione a disposizione dell'HTML standard.

Mediante i tag `<FORM>` `</FORM>` viene creata una sezione della pagina in cui si posizionano particolari controlli atti all'inserimento di valori o informazioni da parte di chi sta usufruendo della pagina HTML.

L'URL cui inviare i dati viene specificato mediante l'attributo `ACTION`. Mediante l'attributo `METHOD`, invece si stabilisce il modo con cui le informazioni vengono inviate al server per l'elaborazione. `METHOD` può assumere uno dei seguenti valori: `POST` oppure `GET`.

1.3.2.2 Metodo Post

Con questo metodo, i dati vengono dapprima reperiti dal form ed impacchettati in un blocco di informazioni e poi spediti, in una seconda trasmissione, all'indirizzo riportato nella specifica `ACTION`.

Il Client, pertanto effettua, in due trasmissioni separate, l'invio di due *blocchi di informazione* al Server. Nel primo invia una serie di informazioni, atte ad identificare i particolari della trasmissione ed il browser utilizzato dal Client, denominate anche *variabili di richiesta HTTP*. Nel secondo invia un file di caratteri contenente le informazioni vere e proprie reperite dal form ed impacchettate.

Il server utilizzerà alcune delle informazioni inviate nel primo blocco per conoscere la dimensione del secondo e reperire correttamente i dati da questo.

1.3.2.3 Metodo Get

Con questo metodo, tutto avviene in un'unica trasmissione, ed i dati vengono inviati all'indirizzo specificato accodandoli alla URL, che assume la forma seguente:

`http://.../directory/pgm.est?nome1=valore1&...&nomen=valoren`

Quindi al Server arriverà un unico *blocco di informazioni*, contenente le variabili di richiesta HTTP e l'URL nella forma appena descritta.

I controlli disponibili per l'inserimento dei dati sono quelli classici di un'interfaccia WIMP, campi di testo, radio button, caselle di selezione e menù a discesa.

Le istruzioni atte a specificare il tipo di controllo e le sue caratteristiche si possono distinguere in due categorie, quelle di inserimento di singoli valori e quelle di selezione tra più argomenti. Per le prime il tag è `<INPUT>`, dotato degli attributi TYPE, NAME e VALUE, per le ultime si usa il blocco `<SELECT> </SELECT>`, all'interno del quale si posizionano le singole alternative identificate del tag `<OPTION>`.

Un esempio di utilizzo dei form è il seguente:

```
<form method="post" action="file.asp">
<p><b>Nome</b>: <input type="text" size="11" name="T1">
<b>Cognome</b>:<input type="text" size="9" name="T2"> </p>
<p><b>Città</b>: <select name="D1" size="1">
<option selected>Udine </option>
<option>Trieste </option>
<option>Gorizia </option>
<option>Pordenone </option>
</select> </p>
<p><b>Età</b>:</p>
<p>meno di 30: <input type="radio" name="R1" value="V1"> tra
i 31 e i 50: <input type="radio" checked name="R1" value="V3">
oltre i 51: <input type="radio" name="R1" value="V4"> </p>
<p><b>Argomenti di interesse:</b></p>
<p><input type="checkbox" checked name="C1" value="ON">
Scienza</p>
<p><input type="checkbox " checked name="C2" value="ON"> Sport</p>
<p><input type="checkbox " name="C3"> Letteratura</p>
<p><input type="checkbox " name="C4"> Altro</p>
</form>
```

1.4 **Organizzazione del metodo di verifica delle applicazioni web**

Per eseguire le verifiche di un'applicazione web bisogna analizzare il contenuto delle pagine al suo interno in modo da capire i vari collegamenti e le strutture presenti. Per le applicazioni dinamiche il codice html analizzato dal browser di un client, come detto nei paragrafi precedenti, è diverso dal contenuto del file presente sul server a cui appartiene. Infatti il codice che entra nel browser, è creato dal server in base alla richiesta di un client che oltre al nome della pagina che si vuole analizzare fornisce una serie di parametri. Quindi analizzare le pagine create dal server in base ai parametri forniti da un client è differente dall'analizzare il solo contenuto dei file presenti sullo stesso server. Analizzare entrambe le cose risulta utile poi, confrontando i risultati (che rappresentano l'insieme dei link, ancore, action, immagini, ect, per ogni pagina trovata), per considerare molti più aspetti sulle applicazioni dinamiche. Per analizzare entrambe le cose, risulta evidente far eseguire le verifiche sul server, infatti solo da lì è possibile accedere ai file, dal file system (in tal caso si parla di analisi offline) e ai risultati prodotti in base ai parametri dall'indirizzo http://localhost (in tal caso si parla di analisi online).

Fatto questo è possibile affermare il seguente **principio**:

I risultati ottenuti dall'analisi online risultano
una **restrizione** di quelli ottenuti dall' analisi offline.

Le pagine che vengono effettivamente visualizzate nel browser del client sono le pagine risultanti dal server in base ai parametri, cioè quelle pagine di cui viene eseguita l'analisi online. I risultati dell'analisi offline (sempre nel caso di applicazioni dinamiche) possono essere considerati come l'unione di tutti i possibili risultati delle analisi online con tutti i possibili parametri inseribili. Si può anche dire che uno dei risultati dell'analisi offline **potrebbe** capitare come risultato dell'analisi online.

C'è da dire però che dall'analisi online si possono determinare dei risultati provenienti da database, non presenti pertanto nell'analisi offline (dalla quale chiaramente non si accede a nessun database) (fig 1-3)

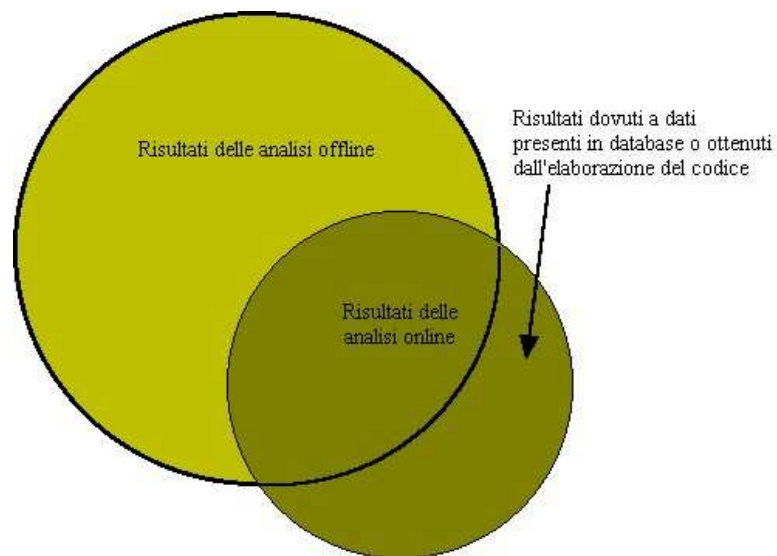


Figura 1–3: Rappresentazione dei risultati delle verifiche offline e delle verifiche online

Se ciò accadesse, cadrebbe il principio affermato precedentemente. Per evitare questo, i risultati provenienti dal database vengono inglobati anche nei risultati dell'analisi offline, in un secondo momento.

Capitolo 2

MODEL CHECKING

2.1 Introduzione

Un modo per verificare la correttezza e il buon funzionamento di un modello lo si ottiene usando i metodi formali.

Questi metodi di verifica si basano su due approcci principali: il primo consiste nel *theorem proving automatico*; il secondo invece, più recente, è quello del *model checking*.

Nel primo caso, sia il sistema che le proprietà che si vogliono verificare devono essere descritti in una logica appropriata. Viene poi costruita una dimostrazione del fatto che la descrizione del sistema implica una descrizione della proprietà. Questo approccio è molto flessibile e potente, ma, quanto più espressiva è la logica, tanto più risulta difficile costruire le dimostrazioni. Viene, quindi, richiesto spesso l'intervento esterno dell'utente il quale deve suggerire lemmi, o passi intermedi di prova, affinché il sistema riesca a suggerire una dimostrazione.

Nel secondo caso si necessita della descrizione del sistema sotto forma di macchina a stati finiti e di proprietà espresse in una logica proposizionale temporale. Viene poi usata un'efficiente procedura di ricerca per determinare in modo automatico, se le specifiche sono soddisfatte dal grafo di transizione degli stati.

Il maggior vantaggio di questo secondo metodo è di essere completamente automatico: in genere l'utente deve solo fornire una rappresentazione ad alto livello del sistema e delle proprietà da verificare. L'algoritmo di model checking termina con la risposta "vero", se il modello soddisfa la proprietà, o fornendo un controesempio che mostra perché la proprietà non è verificata. I controesempi sono molto utili, poiché aiutano a comprendere anche piccoli errori in fase di progettazione di sistemi complessi.

2.2 Model Checking

Il model checking è una tecnica automatica ideata per la progettazione e la verifica di sistemi reattivi a stati finiti, come ad esempio protocolli di comunicazione o sistemi *safety o money critical*.

Nel model checking le specifiche sono espresse in una logica proposizionale temporale che fornisce un linguaggio semplice e conciso che permette, comunque, di esprimere molte proprietà utili.

Nelle logiche temporali, ai comuni operatori di logica proposizionale (cioè congiunzione \wedge , negazione \neg , disgiunzione \vee , implicazione \rightarrow , eccetera), si aggiungono operatori temporali (quale ad esempio “in uno stato futuro” F), che permettono di esprimere proprietà che cambiano nel tempo. Ad esempio, può essere espressa una proprietà del tipo “se adesso è vera p , in qualche istante futuro sarà vera q (in simboli $p \rightarrow Fq$).

Il *temporal logic model checking* fu sviluppato nei primi anni '80 da Clarke ed Emerson. Essi hanno anche fornito un algoritmo in grado di eseguire verifiche automatiche di formule temporali per una logica chiamata *Computation Tree Logic* (CTL) [8]. L'algoritmo proposto era abbastanza efficiente, eseguito in un tempo polinomiale nella lunghezza della formula da verificare e nel modello del sistema. Clarke, Emerson e Sista migliorarono in seguito il loro algoritmo fornendone una nuova versione in grado di essere eseguito in un tempo lineare nel prodotto delle dimensioni della formula e del modello. Questo algoritmo è stato implementato ed ha avuto un grande successo nella verifica di protocolli di rete e piccoli circuiti sequenziali. Il suo impiego fu tuttavia limitato dal problema dell'esplosione degli stati che si verifica facilmente nel caso in cui il sistema presenti più componenti che possono cambiare in parallelo. Una soluzione per ovviare a questo problema è stata quella di utilizzare una rappresentazione simbolica per il grafo di transizione degli stati basata sul *diagramma di decisioni binarie ordinate* (OBDD) proposta qualche anno dopo da McMillan [8]. Tale tecnica consiste nel codificare ogni stato con l'assegnazione di un valore booleano; la relazione di transizione può, quindi, essere espressa da una formula booleana composta da due variabili: una che codifica il vecchio stato e l'altra che codifica il nuovo. Questa formula è rappresentata, dunque, da un diagramma di decisione binaria. Sono stati sviluppati algoritmi efficienti per lavorare con gli OBDD e le rappresentazioni che ne risultano sono spesso molto compatte. La relazione di transizione di una macchina a stati finiti, espressa da una formula booleana, è quindi convertita in un OBDD. L'algoritmo di model checking simbolico è basato su una

procedura che calcola l'OBDD rappresentante l'insieme degli stati che soddisfano la proprietà desiderata.

Attualmente sono disponibili molti tool efficienti che implementano il model checking, alcuni dei quali sono attualmente in uso nell'industria e nel campo della ricerca (tra questi ricordiamo *SPIN*, *SMV* e *NuSMV*). [10]

2.3 Un model checker simbolico: NuSMV

In questo paragrafo sarà fatta una breve descrizione della struttura di NuSMV, model checker simbolico sviluppato negli anni scorsi dalla Carnegie Mellon University (CMU) e dall'istituto per la ricerca scientifica e tecnologica (IRST).

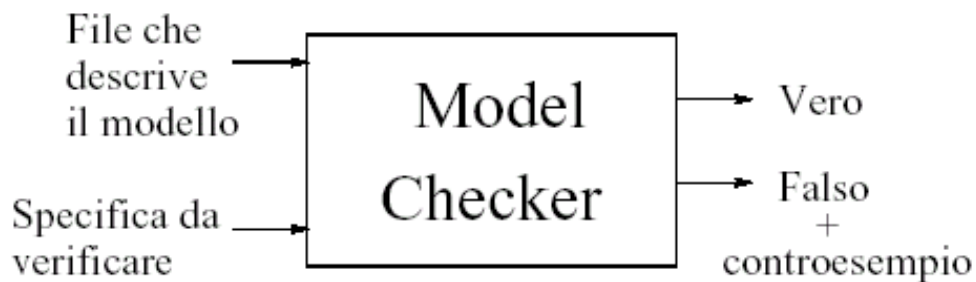


Figura 2-1: Schema generale di NuSMV

Nella figura 2-1 è rappresentato il comportamento generale di un model checker: fornendo in input il modello del sistema da verificare e una specifica espressa in linguaggio logico viene restituito un valore vero se la verifica è verificata altrimenti viene fornito un controesempio.

Nel caso particolare di NuSMV, il file che describe il modello deve essere descritto in un linguaggio particolare, chiamato *SMV-language*, atto a descrivere sistemi a stati finiti. I soli tipi di dati consentiti sono i booleani, i sottoinsiemi limitati di interi e dati enumerati in modo simbolico. È inoltre consentita la definizione di array su tali tipi di dati.

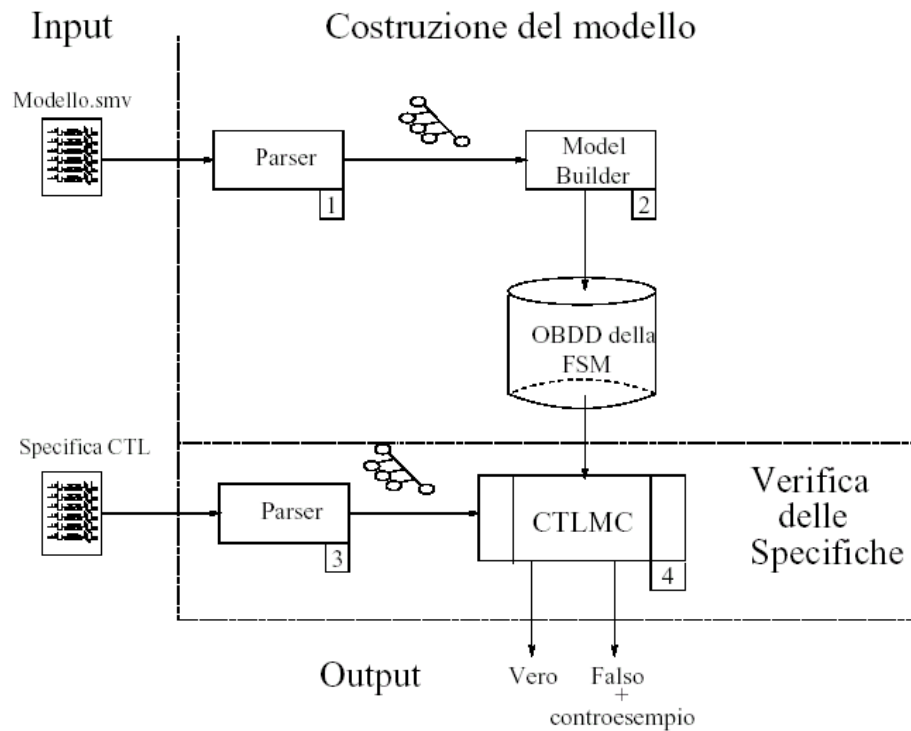


Figura 2-2: Schema interno di NuSMV

In NuSMV il processo di verifica può essere scomposto in quattro passi principali (v. fig. 2-2).

1. Il primo passo è quello della lettura del file input e della costruzione di una rappresentazione interna del modello in esso descritto. Questo passo è associato al comando *read_model* che prende come input il nome del file che contiene la descrizione del modello. Se questo passo va a buon fine, NuSMV costruisce al suo interno una rappresentazione del file letto e la conserva in una variabile globale chiamata *parse_tree*.
2. Il passo successivo consiste nella costruzione del modello, a cui si arriva attraverso più fasi. Vediamole:

Innanzitutto viene eseguito il comando *flatten_hierarchy* le cui funzioni fanno in modo che la descrizione gerarchica di un modello viene

trasformata in una descrizione piatta in cui risultano separate tutte le informazioni necessarie per costruire l'automa. Si tratta dei nomi delle variabili con rispettivo dominio di una rappresentazione simbolica degli stati iniziali e della relazione di transizione: tutte le informazioni vengono inserite in apposite variabili globali.

Viene poi eseguito il comando *build_variables* che effettua la codifica delle variabili scalari in variabili booleane.

A questo punto la funzione associata al comando *build_model* usa la codifica delle variabili eseguita all'esecuzione del comando *build_variables* per trasformare in OBDD le strutture costruite in fase di *flatten_hierarchy*.

Solo una volta che sono stati eseguiti correttamente tutti i comandi precedenti, è il momento di eseguire *compute_fairness* al quale spetta il compito di valutare gli eventuali vincoli, chiamati *fairness constraints*, che possono essere presenti nel file di input. Si tratta di formule CTL che restringono il modello all'insieme degli stati in cui esse sono verificate.

A questo punto attraverso il comando *compute_reachable*, è possibile calcolare l'insieme degli stati raggiungibili. Questo è utile nel momento in cui si vuole semplificare la verifica delle specifiche restringendo la ricerca ai soli stati raggiungibili.

3. È possibile iniziare la verifica della formula CTL attraverso il comando *check_spec* a cui può essere fornito come argomento una formula da verificare. Se richiamato senza alcun argomento cerca le eventuali formule CTL presenti nel file di input. In entrambi i casi, come prima cosa viene costruita una rappresentazione interna della specifica.
4. Il comando *check_spec* chiama poi l'algoritmo di CTL model checking simbolico che prende l'OBDD della macchina a stati finiti e la specifica

e ritorna vero o falso a seconda che sia verificata o meno nel modello. Nel caso particolare in cui una specifica non è verificata, viene fornito come controesempio un cammino in cui la formula non è verificata. [12]

2.4 Logica CTL

La *Computation Tree Logic* è una logica proposizionale che consente di specificare le proprietà di un sistema che nel tempo può evolvere in diversi modi.

Un sistema di questo tipo viene modellato per mezzo di un albero infinito in cui ogni nodo rappresenta uno stato in cui si può trovare il sistema, ogni arco una possibile transizione di stato ed ogni cammino una possibile computazione cioè un possibile comportamento del sistema stesso.

In questo contesto la CTL è una logica di tipo “*branching time*”, ovvero, una logica in cui le proprietà di un sistema nel tempo si possono evolvere in modi diversi.

Per descrivere questa logica si fa ricorso alla notazione Backus-Naur Form:

$$\phi ::= \perp \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid AX \phi \mid EX \phi \mid AG \phi \mid EG \phi \mid AF \phi \mid EF \phi \mid A[\phi U \phi] \mid E[\phi U \phi]$$

dove p rappresenta una proposizione CTL.

I simboli \perp (proposizione mai verificata) e \top (proposizione sempre verificata) rappresentano proposizioni CTL; $\neg\phi$ rappresenta la proposizione complementare a ϕ ; i simboli \wedge e \vee rappresentano rispettivamente i segni di congiunzione e disgiunzione logica, \rightarrow rappresenta il segno di implicazione logica. I connettivi AX, EX, AG, EG, AF, EF, AU ed EU sono chiamati connettivi temporali e sono costituiti da una coppia di simboli. I simboli A ed E significano rispettivamente “lungo tutti i percorsi” e “lungo almeno un percorso”. I simboli X, F, G e U significano rispettivamente “stato futuro”,

“qualche stato futuro”, “ogni stato futuro”, “finché”. AU e EU sono connettivi temporali binari mentre gli altri sono unari. Tali simboli vanno sempre in coppia e mai separati.

Vi sono delle priorità obbligatorie per i connettivi della logica CTL. I connettivi unari hanno alta priorità. Seguono, in priorità, gli operatori \wedge e \vee ; dopodiché vi sono \rightarrow , AU e EU. Si utilizzano le parentesi per modificare le priorità.

Vediamo alcuni esempi di proposizioni ben definite, cioè corrette. Si suppone che p , q e r sono proposizioni elementari. Le seguenti sono proposizioni CTL ben definite:

- EG r ;
- AG ($q \rightarrow$ EG r);
- A [r U q];
- EF E [r U q];
- A [p U EF r].

Non sono proposizioni CTL ben definite le seguenti:

- FG r poiché F e G non vanno mai separati da A oppure E;
- A \neg G \neg p ;
- F [r U q];
- AEF r .

2.4.1 Semantica della logica CTL

Un modello $M = (S, \rightarrow, L)$, secondo la logica CTL, è costituito da un insieme di stati S terminanti con una relazione di transizione \rightarrow (una relazione binaria in S), tale che per ogni $s \in S$ esiste almeno un $s' \in S$ per cui risulta $s \rightarrow s'$ è una funzione

$$L : S \rightarrow \mathcal{P}(\text{Atomi}).$$

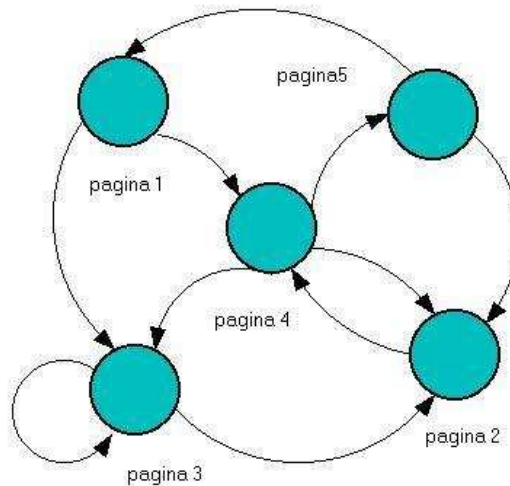


Figura 2-3: Modello secondo la logica CTL definito in base alla struttura di Kripke

Una struttura definita in tal modo è chiamata struttura di Kripke. Per esprimere tutte le informazioni di un modello finito M per CTL, si può utilizzare un grafo orientato i cui nodi (gli stati) contengono le proposizioni atomiche che sono vere in quello stato. Le transizioni sono espresse dagli archi orientati (fig. 2-3). Il requisito che per ogni stato $s \in S$ esiste almeno uno stato $s' \in S$ per cui risulta $s \rightarrow s'$ significa che nessuno stato del sistema può essere terminante. In tal caso si può inserire uno stato s_d che possiede una transizione $s \rightarrow s_d$ in modo da rendere sempre possibile la transizione in ogni istante di tempo (fig. 2-4).

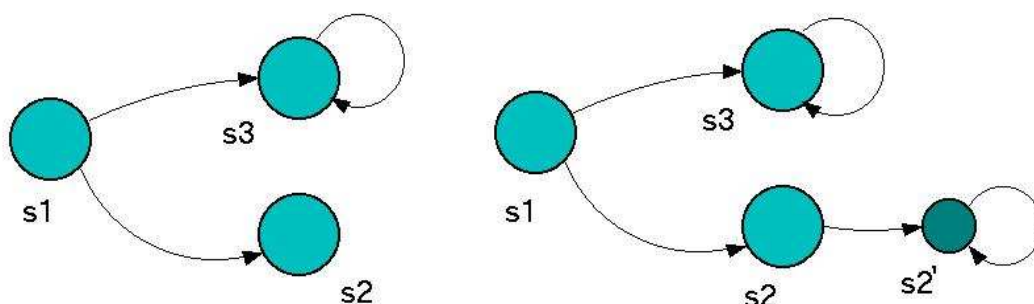


Figura 2-4: Modifica del modello a stati finiti per evitare che ogni stato non sia terminante secondo la struttura di Kripke

A questo punto risulta utile definire il significato dei connettivi CTL.

Dato un modello CTL, $M = (S, \rightarrow, L)$, se per un dato stato $s \in S$ la proposizione CTL ϕ è soddisfatta, si scrive che:

$$M, s \models \phi.$$

La relazione appena enunciata è applicabile ad ogni proposizione CTL per induzione:

1. $M, s \models \top$ e $M, s \not\models \perp$ per ogni $s \in S$.
2. $M, s \models p \Leftrightarrow p \in L(s)$.
3. $M, s \models \neg \phi \Leftrightarrow M, s \not\models \phi$.
4. $M, s \models \phi_1 \wedge \phi_2 \Leftrightarrow M, s \models \phi_1$ e $M, s \models \phi_2$.
5. $M, s \models \phi_1 \vee \phi_2 \Leftrightarrow M, s \models \phi_1$ o $M, s \models \phi_2$.
6. $M, s \models \phi_1 \rightarrow \phi_2 \Leftrightarrow M, s \not\models \phi_1$ o $M, s \models \phi_2$.
7. $M, s \models AX \phi \Leftrightarrow$ per ogni s_1 tale che $s \rightarrow s_1$ si ha $M, s_1 \models \phi$. AX significa “in ogni stato immediatamente successivo” a partire dallo stato s .
8. $M, s \models EX \phi \Leftrightarrow$ esiste almeno un s_1 tale che $s \rightarrow s_1$ si ha $M, s_1 \models \phi$. EX significa “in almeno uno stato immediatamente successivo” a partire dallo stato s . A è duale di E.
9. $M, s \models AG \phi \Leftrightarrow$ per ogni percorso $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$, dove s_1 eguaglia s , per ogni s_i , lungo il percorso si ha $M, s_i \models \phi$. AG significa “per ogni percorso di computazione che comincia in s la proprietà ϕ sussiste globalmente”. “Lungo il percorso” include lo stato iniziale s del percorso.
10. $M, s \models EG \phi \Leftrightarrow$ esiste almeno un percorso $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$, dove s_1 eguaglia s , per ogni s_i , lungo il percorso si ha $M, s_i \models \phi$. EG significa “esiste un percorso che comincia in s tale che ϕ sussiste globalmente lungo il percorso”.
11. $M, s \models AF \phi \Leftrightarrow$ per ogni percorso $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$, dove s_1 eguaglia s , esiste almeno un s_i tale che $M, s_i \models \phi$. AF significa “per ogni percorso di computazione che comincia in s , esiste almeno uno stato futuro dove la proprietà ϕ sussiste”.
12. $M, s \models EF \phi \Leftrightarrow$ esiste almeno un percorso $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$, dove s_1 eguaglia s , in cui esiste almeno un s_i tale che $M, s_i \models \phi$. EF significa “per

alcuni percorsi di computazione che cominciano in s , esiste almeno uno stato futuro dove la proprietà ϕ sussiste”.

13. $M, s \models A [\phi_1 U \phi_2] \Leftrightarrow$ per ogni percorso $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$, dove s_1 eguaglia s , tale percorso soddisfa $\phi_1 U \phi_2$, vale a dire ci sono alcuni s_i lungo il percorso, per cui risulta $M, s_i \models \phi_2$, e per ogni $j < i$, si ha che $M, s_j \models \phi_1$. In altri termini, per ogni percorso di computazione che inizia in s soddisfa ϕ finché ψ .
14. $M, s \models E [\phi_1 U \phi_2] \Leftrightarrow$ esiste almeno percorso $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$, dove s_1 eguaglia s e tale percorso soddisfa $\phi_1 U \phi_2$. In altri termini, esiste un percorso di computazione che inizia in s che soddisfa ϕ finché ψ .

Riferendoci al modello a stati finiti, un *cammino* rappresenta uno dei possibili percorsi nel grafo.

In figura 2-5 vengono rappresentati un modello a stato finiti secondo la struttura di Kripke e il modello ad albero, ad esso equivalente, in cui è più semplice individuare tutti i possibili cammini e interpretare il significato delle formule CTL.

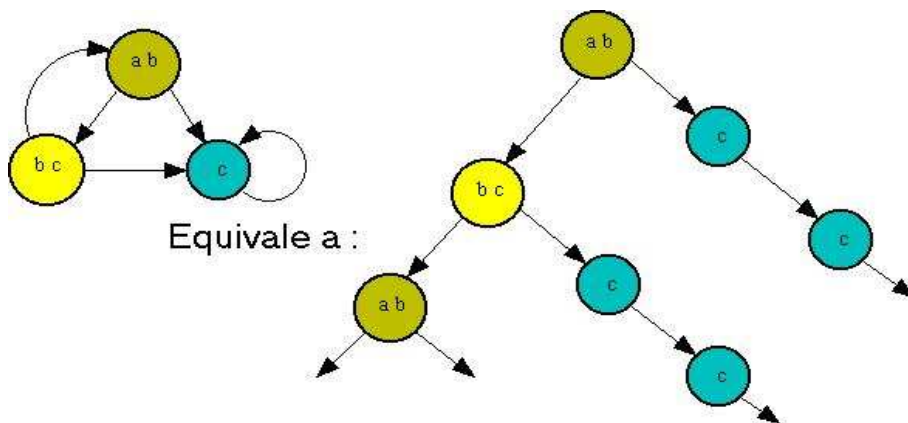


Figura 2-5: Modello a stati finiti e modello ad albero equivalente

Proprio basandoci sul questo modello ad albero saranno descritte alcune formule CTL.

- **EX p:** Esiste almeno un percorso in cui p deve essere vera nello stato successivo (exists next) (v. fig 2-6)

- **AX p**: In tutti i percorsi p deve essere vera in tutti gli stati successivi (always next) (v. fig 2-7)
- **EF p**: Esiste almeno un percorso in cui p deve essere vera in almeno uno stato (exists in the future) (v. fig 2-8)
- **AF p**: In tutti i percorsi p deve essere vera in almeno in uno stato (always in the future) (v. fig 2-9)

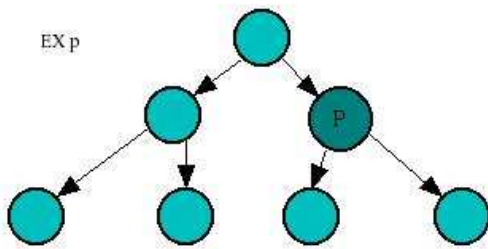


Figura 2-6: EX p

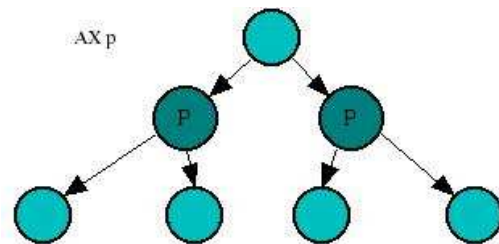


Figura 2-7: AX p

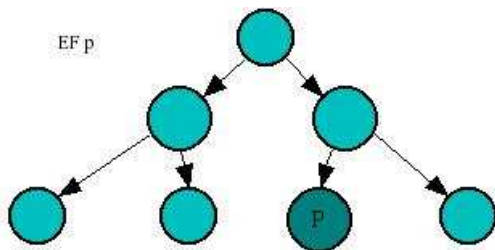


Figura 2-8: EF p

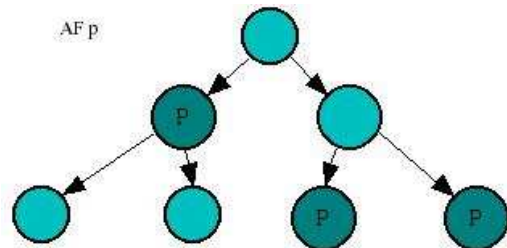


Figura 2-9: AF p

- **EG p**: Esiste almeno un percorso in cui p deve essere vera in tutti gli stati (exists globally) (v. fig. 2-10)
- **AG p**: In tutti i percorsi p deve essere vera in tutti gli stati (always globally) (v. fig. 2-11)
- **E[q U p]**: Esiste almeno un percorso in cui q deve essere vera finché si presenta un stato in cui p risulta vera (exist until) (v. fig. 2-12)
- **A[q U p]**: In tutti i percorsi q deve essere vera in tutti gli stati finché si presenta uno stato in cui p risulta vera (always until) (v. fig. 2-13)

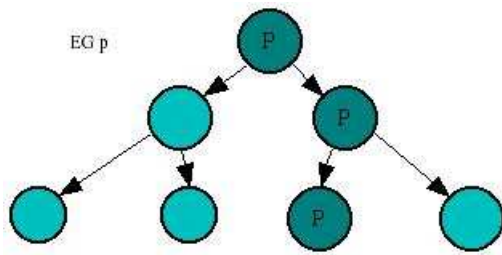


Figura 2-10: EG p

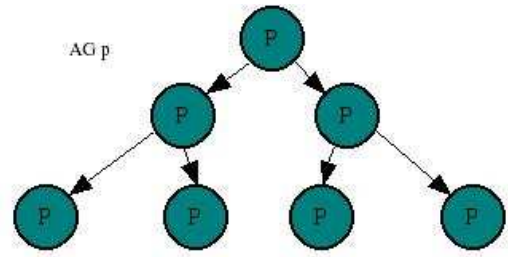


Figura 2-11: AG p

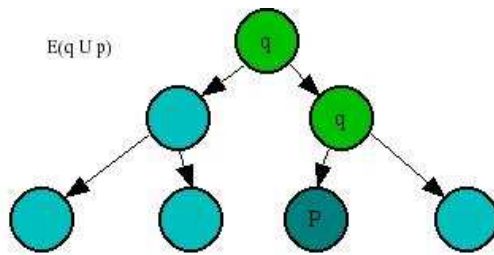


Figura 2-12: E[q U p]

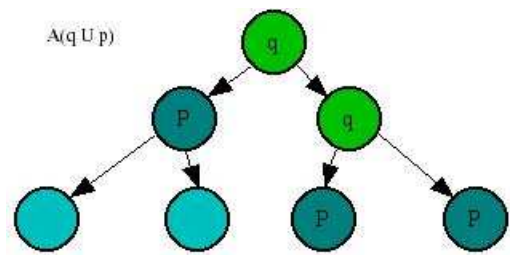


Figura 2-13: A[q U p]

2.4.2 Il model checker SMV (Symbolic Model Verifier)

Lo scopo principale del linguaggio in ingresso al sistema SMV è descrivere le relazioni di transizione di una struttura finita di Kripke, descritta in precedenza.

Prima di descrivere la sintassi e la semantica del linguaggio, consideriamo un semplice esempio che illustri i concetti base, rappresentato nella figura seguente:

```

MODULE main
VAR
    request : boolean;
    state : {ready , busy}
ASSIGN
    init(state) := ready;
    next(state) := case
    state = ready & request : busy;
    1 : {ready , busy};
    esac;
SPEC
    AG(request → AF state = busy)
    
```

Figura 2-14: Esempio di un file di input di ingresso al sistema SMV

Il file di input descrive il modello e le specifiche. Il modello è una struttura Kripke, il cui stato è definito tramite una collezione di variabili di stato. La variabile "request" è dichiarata come una variabile booleana, mentre la variabile "state" è uno scalare che può assumere un valore simbolico "ready" o "busy". Il valore di una variabile scalare è codificato dall'interprete usando una collezione di variabili booleane, così che la relazione di transizione può essere rappresentata da un OBDD. Questa codifica è invisibile all'utente. La relazione di transizione della struttura Kripke e il suo stato/i iniziale/i, sono specificati mediante assegnazioni parallele, introdotte con la parola chiave "ASSIGN". Nel suddetto programma, il valore iniziale della variabile "state" è impostato su "ready". Il nuovo valore di "state" è determinato assegnando ad esso il valore dell'espressione riportata nella figura 2-15.

Il valore di un'espressione case è determinata dalla prima espressione disponibile sul lato destro di un ":" tale che la condizione disponibile sul lato sinistro è vera. Quindi, se "state = ready & request" sia vera, allora il risultato dell'espressione è "busy", altrimenti, esso è l'insieme {ready, busy}.

```
case
    state = ready & request : busy;
    1 : {ready, busy};
esac;
```

Figura 2-15: Esempio di espressione "case"

Quando ad una variabile è assegnato un insieme, il risultato è una scelta non deterministica tra i valori che compongono l'insieme stesso. Quindi se il valore dello stato non è "ready", o "request" è falso (nel corrente stato), il valore di "state" nel nuovo stato può essere o "ready" o "busy".

Si deve notare che la variabile "request" non è stata assegnata in questo programma. Ciò lascia libero il sistema SMV di scegliere qualsiasi valore per essa, dandole le caratteristiche di un ingresso senza costrizioni al sistema.

Le specifiche del sistema appaiono come una formula in CTL sotto la parola chiave "SPEC". Il "model checker" SMV verifica che tutti i possibili stati iniziali soddisfino le specifiche. In questo caso, la specifica indica che se "request" è infinitamente vera, allora inevitabilmente il valore dello stato è "busy".

E' possibile descrivere, brevemente, sistemi sincroni o asincroni per descrivere dettagliati modelli deterministici o astratti e per utilizzare la struttura modulare di un sistema al fine di ottenere la descrizione più concisa. E' anche possibile scrivere assurdità logiche se uno desidera, usando dichiarazioni TRANS e INIT.

Una specifica SMV consiste in dichiarazioni di processi, dichiarazioni (locali e globali) di variabili, dichiarazioni di formule ed una specifica di formule che devono essere verificate. Il modulo principale è chiamato "**main**", come nel linguaggio C. La struttura globale di un file SMV è rappresentata nella figura 2-16.

L'utente può scomporre la descrizione di un sistema complesso a stati finiti in più moduli. Ognuno di essi può essere istanziato più volte e, inoltre, possono essere utilizzate variabili definite in altri moduli. I moduli possono avere dei parametri ed il loro costrutto è del tipo descritto in figura 2-17.

```
MODULE main

VAR dichiarazioni di variabili
ASSIGN assegnazioni di variabili globali
DEFINITION definizione di proprietà da essere verificata /*
opzionale /*
SPEC specifiche CTL da verificare

MODULE /* submodule 1 /*
MODULE /* submodule 2 /*
.....
```

Figura 2-16: Struttura globale di un file SMV

```
MODULE P (parametri formali)
VAR definizioni locali
ASSIGN assegnamenti initial a variabili
ASSIGN assegnamenti next a variabili
```

Figura 2-17: Costrutto di un modulo chiamato P

Viene, di seguito, descritta dettagliatamente la sintassi e la semantica del linguaggio in ingresso ad SMV iniziando dalle convenzioni lessicali.

Un "atom" nella sintassi descritta sotto può essere qualsiasi sequenza di caratteri nell'insieme {A-Z,a-z,0-9,_,-}, che inizi con un carattere alfabetico. Tutti i caratteri sono significativi in un nome. Qualsiasi stringa che inizia con due lineeette "--" e che finisce con una nuova linea è un commento. Un numero è definito come una qualsiasi sequenza di cifre. Qualsiasi altro simbolo riconosciuto dal programma d'analisi è stato chiuso tra virgolette nella sintassi espressa sotto.

L'ordine di precedenza, nella seguente figura, va dall'alto verso il basso e da sinistra verso destra.

*	,	/						
+	,	-						
mod								
=	,	<	,	>	,	<=	,	>=
!								
&								
→	,	↔						

Figura 2-18: Ordine di precedenza degli operatori

Le espressioni sono formate da variabili, costanti, collezione di operatori e possono includere congiunzioni booleane, operatori aritmetici interi ed espressioni case. La sintassi delle espressioni è rappresentata in figura 2-19.

Un "id", o identificatore, è un simbolo o un'espressione che identifica un oggetto, come una variabile o un simbolo definito. Le possibili ambiguità sono segnalate dall'interprete come un errore. L'espressione "next(x)" invia al valore dell'identificativo x che identifica un nuovo stato che può essere un valore noto in modo deterministico oppure può dipendere dal valore delle altre variabili.

La sintassi dell'espressione **case** è rappresentata in figura 2-20. Un'espressione "case" restituisce il valore della prima espressione sul lato destro quando la

corrispondente espressione sul lato sinistro è vera. Così, se `expr_a1` è vera, allora il risultato è `expr_b1`. Altrimenti, se `expr_a2` è vera, allora il risultato è `expr_b2`, etc. Se nessuna delle espressioni sul lato sinistro è vera, il risultato dell'espressione "case" è il valore numerico 1.

E' considerato un errore, per ogni espressione sul lato sinistro, restituire un valore diverso al di fuori dei veri valori 0 o 1.

La sintassi dell'espressione "set" è rappresentata in figura 2-21. Un "set" può essere definito elencando i suoi elementi all'interno delle parentesi. Gli elementi dell'insieme possono essere numeri o costanti simboliche. L'operatore "inclusion" esamina un valore per membro in un set mentre l'operatore "union" esegue l'unione di due "sets".

```
expr ::
atom                                     ;; una costante simbolica
| number                                 ;; una costante numerica
| id                                     ;; un identificatore variabile
| "!" expr                               ;; not logico
| expr1 "&" expr2                         ;; and logico
| expr1 "|" expr2                       ;; or logico
| expr1 "→ >" expr2                     ;; implicazione logica
| expr1 "↔" expr2                       ;; equivalenza logica
| expr1 "=" expr2                       ;; uguaglianza
| expr1 "<" expr2                         ;; minore di
| expr1 ">" expr2                         ;; maggiore di
| expr1 "<=" expr2                      ;; minore o uguale
| expr1 ">=" expr2                      ;; maggiore o uguale
| expr1 "+" expr2                       ;; somma intera
| expr1 "-" expr2                       ;; sottrazione intera
| expr1 "*" expr2                       ;; moltiplicazione intera
| expr1 "/" expr2                       ;; divisione intera
| expr1 "mod" expr2                    ;; resto intero
| "next" "(" "id" ")"                  ;; valore nuovo
| set_expr                             ;; un'espressione set
| case_expr                             ;; un'espressione case
```

Figura 2-19: Sintassi delle espressioni

Uno stato del modello è un'assegnazione di valori appartenenti ad un insieme di variabili. Queste variabili (ed anche istanze di moduli) sono dichiarate con la notazione rappresentata in figura 2-22.

Il tipo associato con una dichiarazione variabile può essere un boolean, uno scalare, un modulo definito dall'utente, o un array di qualcuno di questi (incluso arrays di arrays). Un tipo specifico ha la seguente sintassi rappresentata in figura 2-23.

```
case_expr ::
  "case"
    expr_a1 ":" expr_b1 ";"
    expr_a2 ":" expr_b2 ";"
    .....
    expr_an ":" expr_bn ";"
  "esac"
```

Figura 2-20: Sintassi dell'espressione "case"

```
set_expr ::
  "{" val1 "," ... "," valn "}"
| expr1 "in" expr2      ;; set predicato inclusion
| expr1 "union" expr2   ;; set union
```

Figura 2-21: Sintassi dell'espressione "set"

```
decl :: "VAR"
atom1 ":" type1 ";"
atom2 ":" type2 ";"
```

Figura 2-22: Dichiarazioni delle variabili

Una variabile di tipo boolean prende il valore numerico 0 o 1, che rappresentano rispettivamente false e true. Nel caso di una lista di valori racchiusi tra virgolette, la variabile è uno scalare che prende ognuno di questi valori. Nel caso di una dichiarazione array, l'espressione `expr1` è il più basso salto su ciò che è scritto sotto, ed `expr2` è il più alto salto. Tutte e due le espressioni devono esprimere numericamente costanti intere.

Infine, un *atom* seguito opzionalmente da una lista di espressioni in parentesi indica un'istanza di un modulo *atom*. La parola chiave *process* costringe il modulo ad essere istanziato come un processo asincrono; esso è un modulo istanziato con la parola chiave *process*.

```
type :: boolean
| "{" val1 "," ... "," valn "}"
| "array" expr1 ".." expr2 "of" type
! atom [ "(" expr1 "," expr2 "," ... exprn ")" ]
| "process" atom [ "(" expr1 "," expr2 "," ... exprn ")" ]

val :: atom | number
```

Figura 2-23: Sintassi di un tipo specifico

Una dichiarazione d'assegnamento ha la seguente forma:

```
decl :: "ASSIGN"
dest1 " := " expr1 ";"
dest2 " := " expr2 ";"
.....

dest :: atom
| "init" "(" atom ")"
| "next" "(" atom ")"
```

Figura 2-24: Forma di una dichiarazione d'assegnamento

Sul lato sinistro dell'assegnazione, *atom* indica il valore corrente di una variabile, *init(atom)* indica il suo valore iniziale e *next(atom)* indica il suo valore futuro nel nuovo stato. Se l'espressione è valutata come un set, allora l'assegnamento afferma che il lato sinistro è contenuto in quel set.

E' da considerare un errore se il valore dell'espressione non è contenuto nel range della variabile sul lato sinistro. Inoltre, è un errore assegnare ad una variabile un valore più di una volta ad un dato istante. Per essere precisi, è un errore se:

- il valore nuovo o corrente di una variabile è assegnata più di una volta in un dato processo;
- il valore iniziale di una variabile è assegnata più di una volta nel programma;
- il valore corrente e il valore iniziale di una variabile sono assegnati insieme nel programma;
- il valore corrente ed il valore futuro di una variabile sono assegnati insieme nel programma..

La relazione di transizione R del modello è un insieme di coppie stato corrente/stato futuro. In ogni caso una data coppia che si trova nel set considerato, è determinato da una valutata espressione booleana T , introdotta con la parola chiave "TRANS".

La sintassi di una dichiarazione TRANS è in figura:

```
decl :: "TRANS" expr
```

Figura 2-25: Sintassi di una dichiarazione "TRANS"

Se c'è più di una dichiarazione TRANS, la relazione di transizione è l'unione di tutte le dichiarazioni TRANS. E' considerato un errore per l'espressione dare un valore diverso da 0 o 1. L'insieme degli stati iniziali di un modello è determinato da un'espressione booleana sotto la parola chiave INIT.

La sintassi di una dichiarazione INIT è:

```
decl :: "INIT" expr
```

Figura 2-26: Sintassi di una dichiarazione "INIT"

Se c'è più di una dichiarazione "INIT", il set iniziale è l'unione di tutte le dichiarazioni "INIT".

E' considerato un errore per l'espressione contenere l'operatore "next()", o dare un valore diverso da 0 o 1.

La specifica di sistema è data come una formula nella logica temporale CTL, introdotta dalla parola chiave "SPEC".

La sintassi di questa dichiarazione è:

```
decl :: "SPEC" ctlform
```

Figura 2-27: Sintassi di una dichiarazione "SPEC"

Esempi di queste specifiche saranno mostrate nel quarto capitolo quando si discuterà della sperimentazione del software sviluppato nella tesi; in tale capitolo sarà mostrato, infatti, l'intero file con estensione "smv" in cui vi saranno tutte le specifiche, nella forma sopra illustrata, riguardanti i requisiti che il sito sotto analisi deve soddisfare.

Una formula CTL ha la seguente sintassi:

```
ctlform ::
expr                ;; un'espressione booleana
| "!" ctlform       ;; not logico
| ctlform1 "&" ctlform2    ;; and logico
| ctlform1 "|" ctlform2   ;; or logico
| ctlform1 "->" ctlform2  ;; implicazione logica
| ctlform1 "<->" ctlform2  ;; equivalenza logica
| "E" pathform      ;; quantificatore di traiettoria esistenziale
| "A" pathform      ;; quantificatore di traiettoria
universale
```

Figura 2-28: Sintassi di una formula CTL

La sintassi di una formula path è:

```
pathform ::
"X" ctlform           ;; tempo futuro
"F" ctlform           ;; alla fine
"G" ctlform           ;; in modo globale
ctlform1 "U" ctlform2 ;; finchè
```

Figura 2-29: Sintassi di una fomula "path"

L'ordine di preferenza degli operatori è rappresentato nella seguente figura (dall'alto verso il basso e da sinistra verso destra):

```
E, A, X, F, G, U
!
&
|
→ , ↔
```

Figura 2-30: Ordine di preferenza degli operatori

Possono essere usate le parentesi per raggruppare le espressioni. Inoltre se dovesse esserci più di una dichiarazione "SPEC", la specifica è l'unione di tutte le dichiarazioni SPEC.

E' considerato un errore per un'espressione in una formula CTL contenere un operatore "next()" o restituire un valore diverso da 0 o 1.

Per fare descrizioni più brevi, si può associare un simbolo ad una espressione che viene usata molte volte. La sintassi per questa dichiarazione è definita in figura 2-31.

Nel momento in cui ogni identificatore, riferito al simbolo sul lato sinistro, compare in un'espressione, esso è rimpiazzato dall'espressione sul lato destro. L'espressione sul lato destro è sempre valutata nel suo contesto originale.

Non sono permessi e sono considerati come un errore fare definizioni cicliche; però sono permessi riferimenti diretti a definire simboli.

```
decl :: "DEFINE"  
atom1 " := " expr1 ";"  
atom2 " := " expr2 ";"  
...  
atomn " := " exprn ";"
```

Figura 2-31: Sintassi di una dichiarazione "DEFINE"

Un modulo è una collezione incapsulata di dichiarazioni. Una volta definito, può essere riusato molte volte quando necessario. I moduli possono anche essere parametrizzati, così che ogni istanza di un modulo può riferirsi a differenti valori dati. Un modulo può contenere istanze di altri moduli, consentendo una gerarchia strutturale per essere costruita. La sintassi di un modulo è:

```
module ::  
  "MODULE" atom [ "(" atom1 "," atom2 "," ... atomn ")" ]  
  decl1  
  decl2  
  ...  
  decln
```

figura 2-32: Sintassi di un modulo

Il simbolo "atom" immediatamente seguente la parola chiave "MODULE" è il nome associato al modulo. Un'istanza di un modulo è creato usando la dichiarazione "VAR". Questa dichiarazione fornisce un nome per l'istanza, ed anche una lista di parametri attuali, che sono assegnati ai parametri formali nella definizione del modulo. Un parametro attuale può anche essere un'espressione.

Si noti che la semantica per istanziare un modulo può anche essere assimilabile al richiamo per riferimento.

Viene considerato un errore se il numero dei parametri attuali è diverso dal numero dei parametri formali.

In conclusione si può riassumere la rappresentazione della sintassi di un programma SMV nel seguente modo:

```
program ::  
module1  
module2  
.....  
modulen
```

Figura 2-33: Sintassi di un programma SMV

Deve esserci un modulo con il nome "**main**" e nessun parametro formale. Il modulo "main" è il primo ad essere valutato dall'interprete. [8] [9]

Capitolo 3

STRUTTURA INTERNA DEL PROGETTO

3.1 Modello a stati finiti di un sito web

Il modello utilizzato per descrivere un sito, ai fini della verifica mediante il model checking, è quello a stati finiti. Nel caso in cui il sito offra un servizio interattivo si può parlare di un'applicazione web ed è noto come, attraverso strumenti di analisi, è possibile definire un modello a stati finiti in cui ciascuno stato è rappresentato dallo stato corrente dell'applicazione stessa (da una pagina).

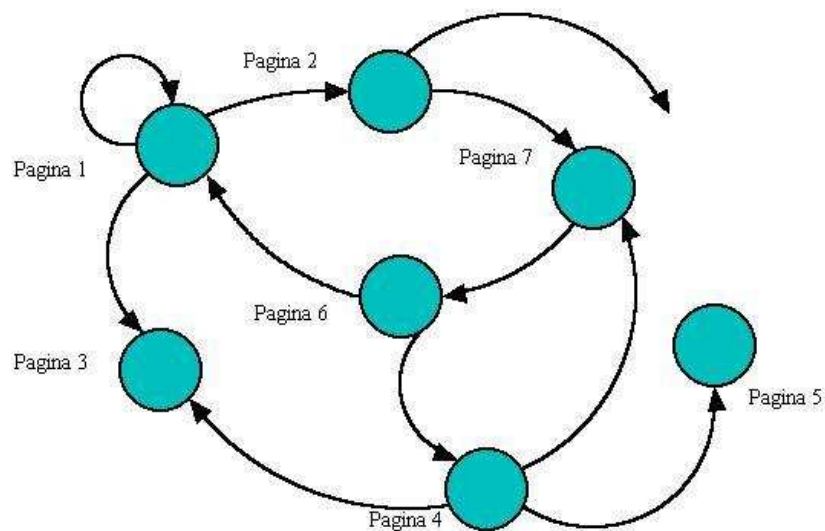


Figura 3-1: Modello semplificato di un sito web: i nodi rappresentano le pagine web mentre gli archi rappresentano i collegamenti tra le pagine

In figura 3-1 viene rappresentato un modello a stati finiti di un semplice sito. Nel modello, i nodi rappresentano le pagine web mentre gli archi rappresentano i collegamenti tra le pagine. In questo modello a causa di pagine del sito che non contengono link (che producono stati terminanti) e a causa di una pagina che contiene un link a una pagina inesistente (che produce un arco che non si collega a nessuno stato) non viene rispettato il modello di Kripke. Per evitare questo bisogna affinare il modello a stati finiti considerando come nodo non solo lo stato pagina ma anche altri tipi di stati.

In figura 3-2, oltre allo stato pagina è stato considerato lo stato link. Con questo modello si possono considerare aspetti che nel modello di figura 3-1 non possono essere valutati.

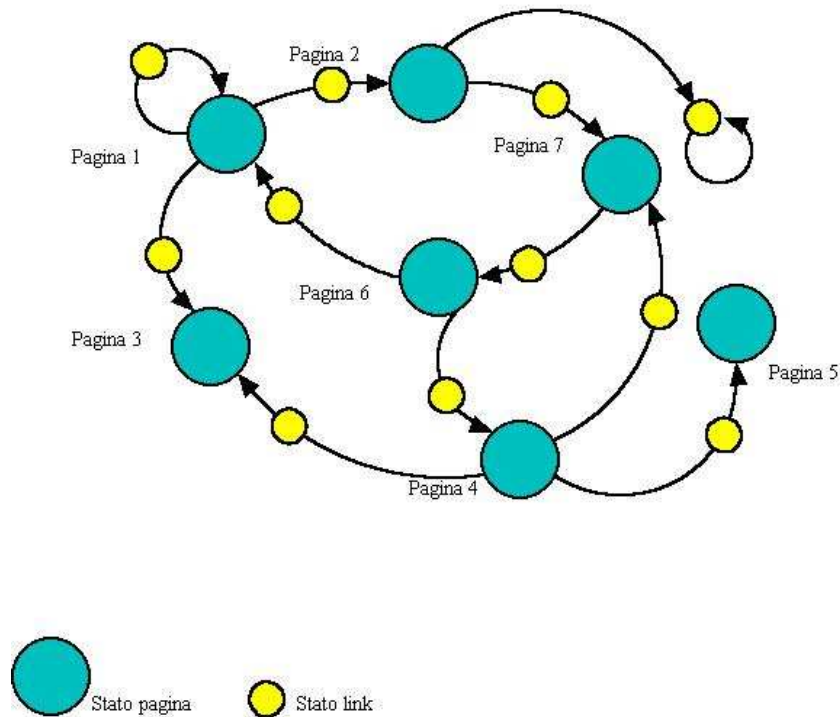


Figura 3-2: Modello a stati finiti di un sito web dove il nodo è rappresentato o dalla pagina o dal link

Considerando come nodo, oltre allo stato pagina, anche lo stato link, è possibile fornire una descrizione più accurata per verificare diverse proprietà relative ai link stessi, poi, è possibile verificare siti che presentano dei collegamenti che non portano a nessuna pagina (a causa dell'assenza di tale pagina nella directory virtuale). In tal caso, questo modello, fa sì che il sistema rimanga nello stato *link* in analogia con la struttura di Kripke, che non ammette stati terminanti, in cui non vi è alcuna transizione.

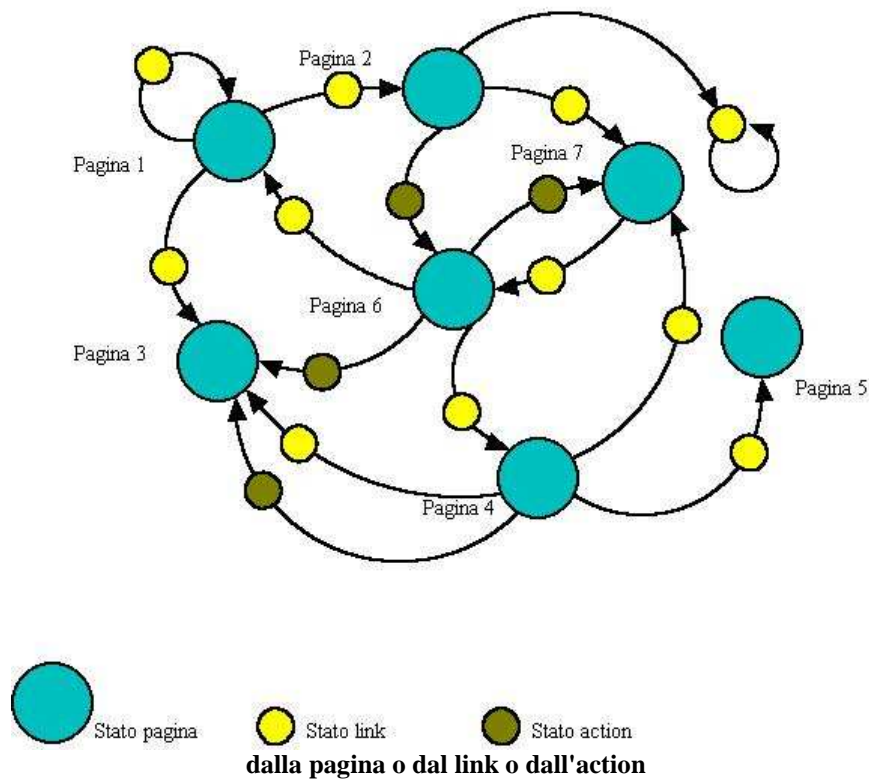
Non è stato comunque risolto il problema delle pagine che non contengono link, che essendo rappresentati con stati terminanti, non fanno rispettare ancora al modello, se queste si presentano, la struttura di Kripke.

Comunque questo modello a stati finiti è la base da cui bisogna partire per determinare i modelli a stati finiti effettivamente utilizzati per le verifiche online e le verifiche offline che saranno discusse in seguito.

3.2 Modello a stati finiti usato nelle verifiche offline

Nelle verifiche offline, vengono analizzate pagine *asp*, *php*, *jsp*, ect. nel loro contenuto, cioè entrando nei loro file dal file system del server. Questi tipi di pagine oltre a interagire tra di loro tramite link, interagiscono anche tramite i form.

La struttura del modello a stati finiti, descritta precedentemente (figura 3-2), visto che tiene conto solo delle interazioni tra le pagine con i link, nel momento in cui si eseguono verifiche delle pagine descritte sopra, possono essere mancanti di alcune informazioni. Una soluzione che si può adottare potrebbe essere quella di considerare le interazioni eseguite con i form uguali alle interazioni eseguite con i link. In questo caso però quando si presenta un collegamento interrotto, non si riesce a capire se la causa è dovuta a un link o a un form. Per risolvere questo problema bisogna modificare ulteriormente il modello a stati finiti aggiungendo un ulteriore tipo di nodo che rappresenta lo stato **action** che indica l'interazione tra le pagine avvenuta tramite form (v. fig. 3-3)



Volendo anche distinguere la differenza tra ancore e semplici link, è possibile ancora affinare il modello aggiungendo un altro tipo di nodo che rappresenta lo stato *ancora* che indica l'interazione tra le pagine con link che presentano un'ancora (figura 3-4).

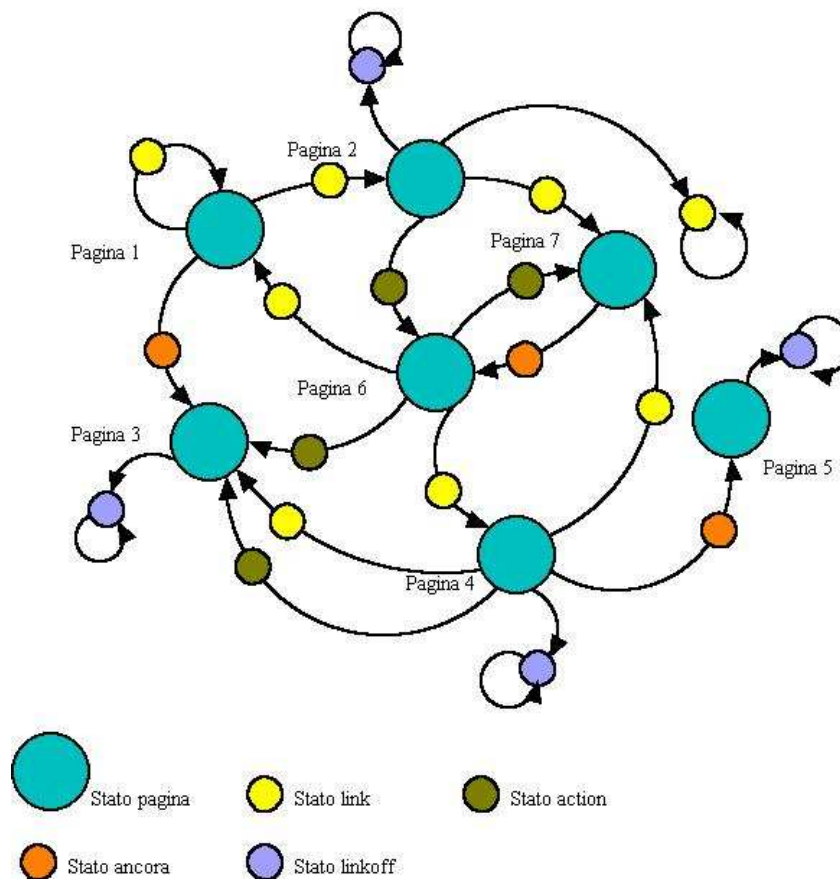


Figura 3-4: Modello a stati finiti dove il nodo è rappresentato dallo stato pagina, dallo stato action o dallo stato ancora

Nel modello è stato anche aggiunto lo stato *linkoff*. In questo stato si entra quando una pagina non presenta al suo interno dei link oppure quando una pagina successiva è esterna (in questo caso viene evitata la presenza di stati terminanti e quindi, anche quando nel sito ci sono pagine che non contengono link, viene rispettata la struttura di Kripte).

Da questo modello è possibile determinare tutte le verifiche offline. In realtà dal punto di vista pratico il modello conclusivo adottato non è quello di figura 3-4 ma è quello mostrato nella figura 3-5.

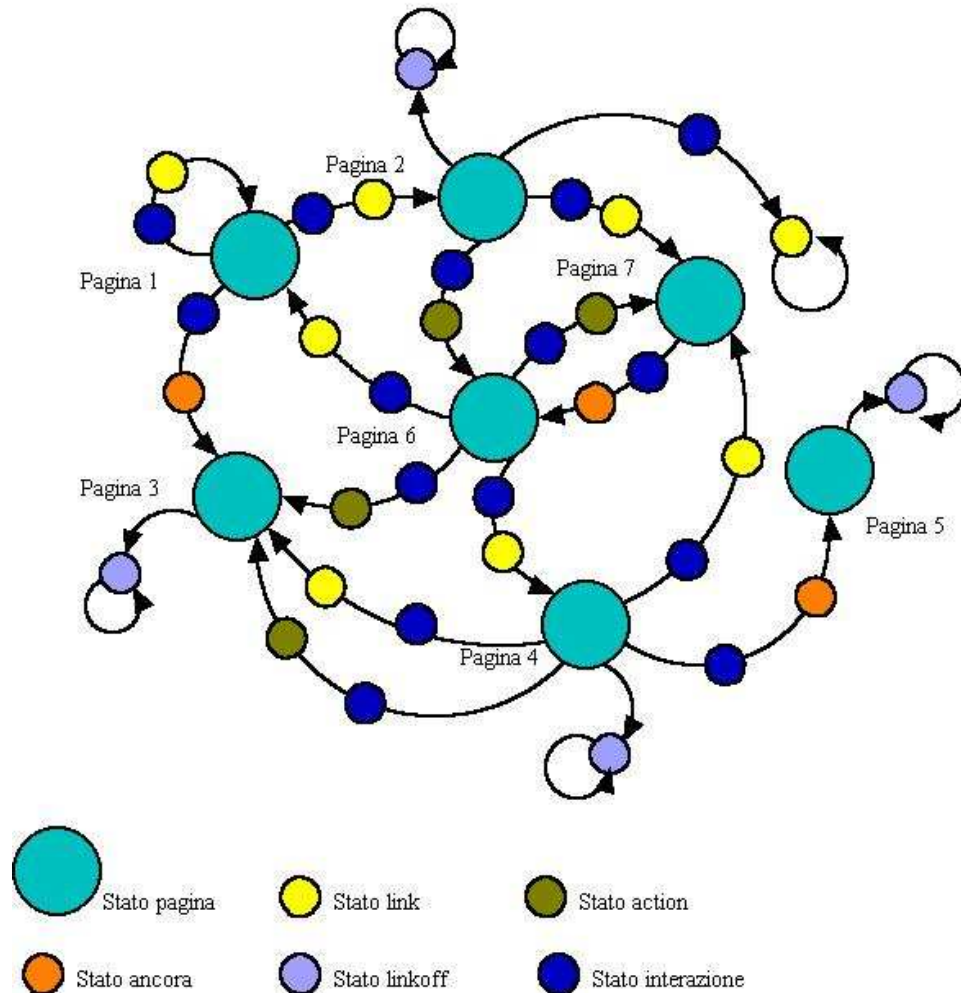


Figura 3-5: Modello a stati finiti, effettivamente utilizzato dalle verifiche offline, dove è stato aggiunto lo stato interazione

In questo nuovo modello è stato aggiunto lo stato *interazione*. Questo nuovo tipo di stato si presenta sempre in tutti i tipi di interazione tra le pagine e anticipa sempre lo stato che indica la causa dell'interazione.

Questo tipo di modello come detto prima è stato realizzato solo per motivi pratici; in effetti i risultati che possono essere determinati con questo modello, potevano essere determinati anche con il modello presentato in figura 3-4.

L'algoritmo che determina il modello, infatti, nel momento in cui rileva un'interazione con un'altra pagina (indipendentemente dal modo in cui avviene), dallo stato *pagina* passa allo stato *interazione*. È proprio nello stato *interazione* che viene valutato il tipo e qui viene deciso quindi se andare nello stato *link*, nello stato *ancora* oppure nello stato *action*. In questi nuovi stati se la pagina esaminata è esistente nella directory virtuale si passa nuovamente allo stato *pagina* altrimenti si rimane bloccati (con un loop) al loro interno.

Quest'algoritmo viene riassunto nella figura 3-6.

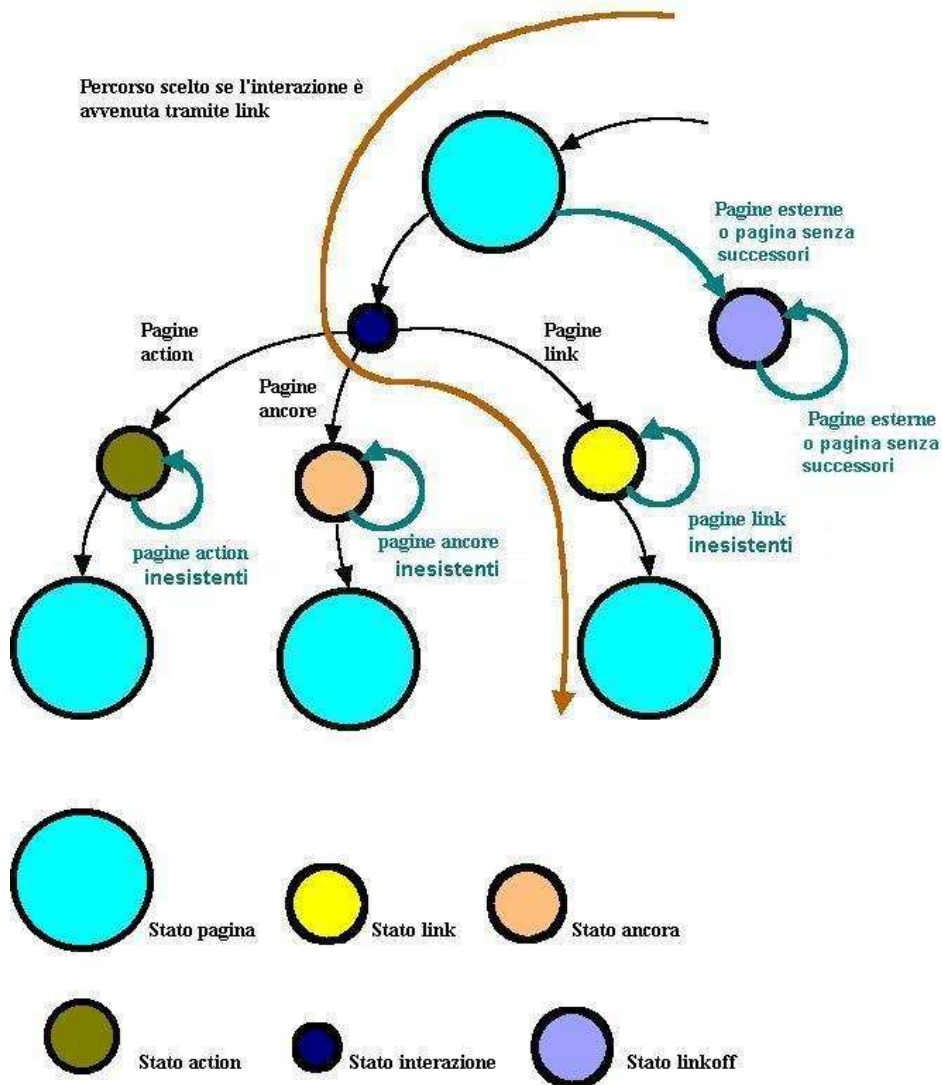


Figura 3-6: Schema dell'algoritmo utilizzato per determinare il modello offline

3.3 Modello a stati finiti usato nelle verifiche online

Nel momento in cui bisogna determinare un modello delle pagine dinamiche elaborate dal server, quello visto nel paragrafo precedente non va bene. Nelle pagine dinamiche infatti il modello a stati finiti di un sito web non è statico, ma dipende dai parametri inseriti in fase di apertura di una pagina. Questi parametri possono essere inseriti o tramite un form oppure dopo un punto interrogativo all'interno di un link. È evidente che con differenti parametri, partendo dalla stessa pagina dinamica, il server determina differenti pagine in corrispondenza dei parametri inseriti.

Un semplice sito che presenta pagine dinamiche è quello che gestisce la posta elettronica di diversi utenti. In tal caso, in corrispondenza di una password inserita, viene determinata la home page personale di un utente. È possibile quindi determinare tante home page quanti sono gli utenti che si sono registrati sotto quel gestore di e-mail. Questo non significa che nel server che gestisce l'e-mail, per visualizzare tutte queste home page, ci siano tanti file quanti sono gli utenti registrati; in realtà esiste un solo file che produce diversi risultati in base ai parametri inseriti (in questo caso in base alle password).

Per rappresentare questa situazione complessa in un modello a stati finiti bisogna ragionare in maniera differente rispetto a come fatto nel paragrafo precedente.

Mentre percorrendo un semplice link, il modello a stati finiti è simile a quello visto in figura 3-3, la situazione cambia, invece, nel momento in cui bisogna percorrere il collegamento avvenuto tramite form. In effetti bisogna considerare come diversi parametri inseriti tramite lo stesso form producano risultati differenti.

Il modello a stati finiti avrà quindi una struttura simile a quella mostrata in figura 3-7.

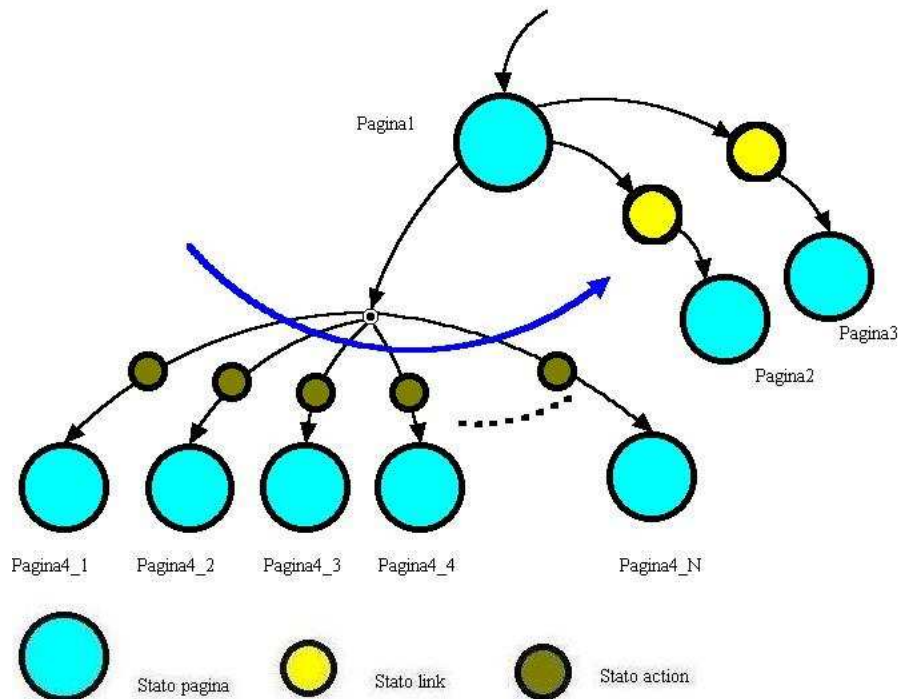


Figura 3-7: Struttura del modello a stati finiti online

Il numero N delle pagine ricavate dal processo server in base alla scelta dei parametri, può assumere un valore altissimo (si pensi al numero di utenti registrati che, stando all'esempio precedente, posseggono una home page nel sito che gestisce l'e-mail). Nelle nostre verifiche non è possibile considerare tutti i possibili N stati, determinati in base a tutti i possibili parametri inseribili, ma saranno considerati un numero arbitrario di stati fissato, nel parsing online, dal numero di volte che è stato cliccato il tasto *DiNuovo* da un form dopo aver inserito parametri differenti. In tal caso si scelgono solo alcuni dei possibili valori dei parametri.

C'è da dire che nel modello a stati finiti, l'analisi delle ancore non viene eseguito, visto che tutte le verifiche sulle ancore possono essere eseguite basandosi sul modello a stati finiti descritto nel paragrafo precedente.

Nella figura 3-8 è visualizzato un modello a stati finiti di un piccolo sito dinamico. In questo caso c'è da notare che anche un piccolissimo sito dinamico presenta una struttura abbastanza complicata.

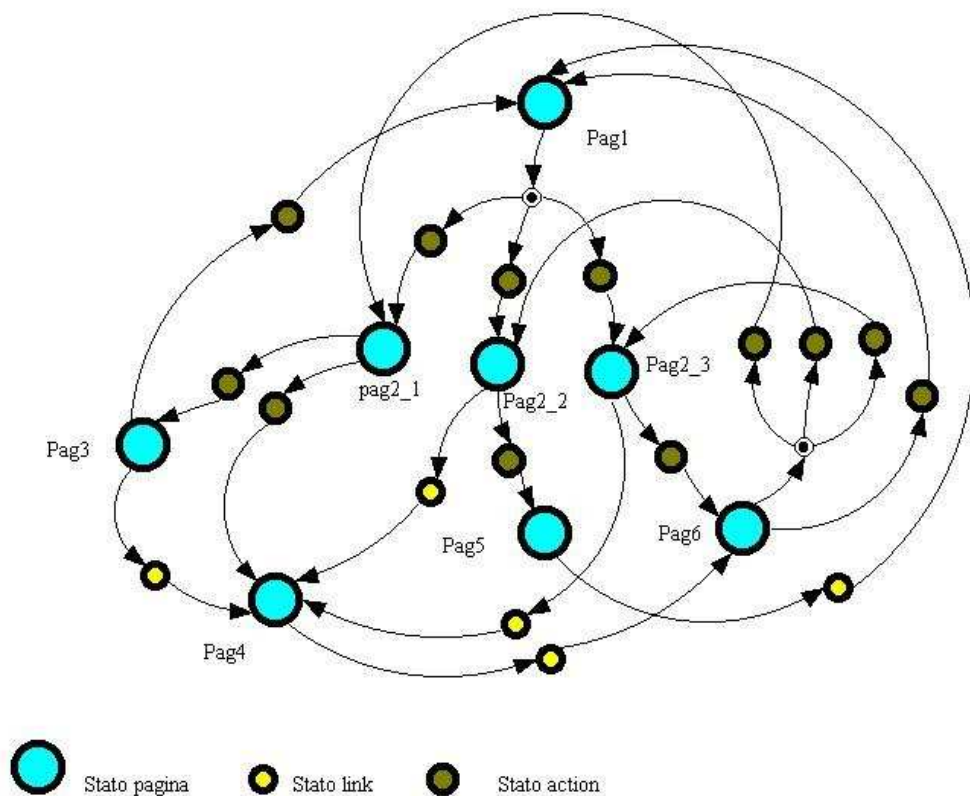


Figura 3-8: Modello a stati finiti per verifiche online

3.4 Determinazione e utilizzo dei modelli a stati finiti

Dopo aver descritto i modelli a stati finiti utili per le verifiche online e le verifiche offline, sarà visto adesso come vengono determinati.

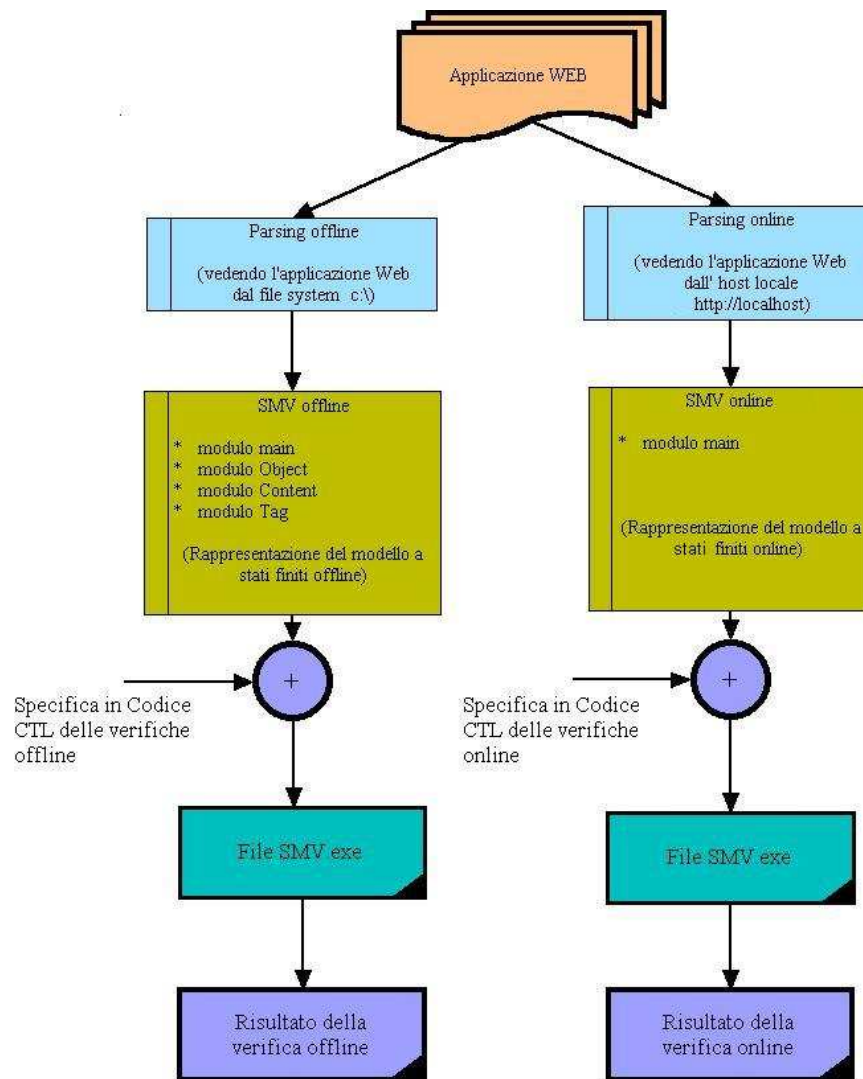


Figura 3-9: Algoritmo principale del sistema

Questi modelli necessitano di varie informazioni che bisogna catturare dalla struttura delle pagine del sito. Ci sarà pertanto un algoritmo capace di analizzare le pagine del sito, prelevando in modo opportuno le informazioni strutturate presenti all'interno. Questo algoritmo prende il nome di **parsing** e si divide in:

- **parsing offline:** Utile per poter poi eseguire le verifiche offline.
- **parsing online:** Utile per poter poi eseguire le verifiche online

Le informazioni ricavate da questi parsing vengono immagazzinate in vettori. Ci sarà poi un altro algoritmo che prendendo queste informazioni ne ricava i modelli a stati finiti (descritti ampiamente nei paragrafi precedenti) rappresentati rispettivamente nel codice *SMV offline* e nel codice *SMV online* la cui trattazione inizia dal paragrafo successivo (v. fig. 3-9). Questo codice, scritto all'interno di un file, insieme alle specifiche scritte in CTL (scritte alla fine dello stesso file) viene inviato al programma SMV che crea un file di testo dentro il quale è presente il risultato della verifica.

3.5 Codice SMV dei modelli realizzati

3.5.1 Codice SMV dei modelli offline

Il codice *SMV offline*, è costituito da diversi moduli.

Il modulo *main* è costituito da una parte dichiarativa e da una d'assegnamento. Le dichiarazioni sono relative alle seguenti variabili:

- *State*: definisce i possibili stati:
 - *page*: una pagina qualsiasi o un processo lato server
 - *link*: stato transitorio che si presenta nel momento in cui si vuole seguire un passaggio di stato. Questo stato si presenta sempre indipendentemente se il passaggio di stato è avvenuto tramite semplice link, tramite ancora oppure tramite action del form.
 - *ancora*: stato che si presenta dopo lo stato link nell'ipotesi che il passaggio di stato avvenga tramite ancora
 - *linkpg*: stato che si presenta dopo lo stato link nell'ipotesi che il passaggio di stato avvenga tramite semplice collegamento
 - *action*: stato che si presenta dopo lo stato link nell'ipotesi che il passaggio di stato avvenga tramite action di un form

- *Paging*: definisce tutte le possibili pagine (statiche o dinamiche). Sono incluse anche le pagine che non sono realmente collegate, a causa di un URL non corretto, nel riferimento del link o errori d'altro tipo. I valori di *Paging* sono i nomi dei file delle pagine del sito, comprendenti il percorso relativo e un prefisso che indica la sua provenienza (ad esempio *linkpg:/cdmarket/index.htm* indica che il file interno *cdmarket/index.htm* proviene da un semplice link), e gli URL nel caso di collegamenti esterni al sito (ad esempio *http://www.libero.it*).

Oltre alle variabili, nel modulo *main*, vengono istanziati altri moduli; essi sono:

1. Il modulo *content* che opera sul parametro *Paging* (che è la variabile precedentemente descritta) ed ha le seguenti variabili:
 - *internal* (variabile booleana) che definisce se una pagina è interna oppure no;
 - *home* (variabile booleana) che indica se una pagina è l'home page oppure no;
 - *dynamic* (variabile booleana) che indica se la pagina è dinamica o statica;
 - *date* che indica la data dell'ultima modifica (con formato *aaaammgg* dove *a* sta per anno, *m* per mese e *g* per giorno);
 - *weight* che indica il peso in byte di una pagina;
2. Il modulo *object* che opera sul parametro *Paging* (che è la variabile precedentemente descritta) e ha le variabili:
 - *Image1*, *image2*, *ecc*, (variabili booleane) che indicano la presenza o meno di una particolare immagine o oggetto (inteso come file non Html, uno script lato client, un applet, ecc.) in una data pagina; tali nomi di variabili nella reale implementazione sono sostituiti dai nomi dei file degli oggetti e immagini a cui si riferiscono;

3. Il modulo *type* che ha le costanti:

- *Image1*, *image2*, *ecc* che indicano il tipo di un oggetto, assumono i valori *IMG*, *EMBED*, ecc. e cioè i nomi dei tag che li richiamano all'interno di una pagina oppure *linkToObject* nel caso di oggetti collegati mediante il tag A (ad esempio, indirizzi di posta elettronica). Per i nomi di tali costanti si veda il caso relativo al modulo *object*.

3.5.2 Codice in ingresso a SMV dei modelli online

Il codice del modello online è caratterizzato dal solo modulo *main*, che è costituito anche in questo caso da una parte dichiarativa e da una d'assegnamento.

Le dichiarazioni sono relative alle seguenti variabili:

- *State*: definisce i possibili stati:
 - *Page*: una pagina qualsiasi o un processo lato server
 - *link*: stato che si presenta dopo lo stato *link* nell'ipotesi che il passaggio di stato avvenga tramite semplice collegamento, tramite action di un form e anche tramite ancora
- *Paging*: definisce tutte le possibili pagine (statiche o dinamiche). Sono incluse anche le pagine che non sono realmente collegate, a causa di un URL non corretto, nel riferimento del link o errori d'altro tipo. I valori di *Paging* sono i nomi dei file delle pagine del sito comprendente il percorso relativo (ad esempio *cdmarket/index.htm*) e gli URL nel caso di collegamenti esterni al sito (ad esempio *http://www.libero.it*).

- *Var1* e *Var2* definiscono le variabili inviate al server o con il metodo post o con il metodo get, (nell'implementazione reale al posto di essi vi saranno i nomi delle variabili effettive). I valori assunti da esse sono i valori delle variabili inviate al server web. Questi valori sono quelli che vengono inseriti negli appositi form presenti nelle pagine web, prima che si clicchi il tasto submit presente nel form stesso. Nel progetto che determina questo codice smv, i valori delle variabili vengono inseriti nei vari form che compaiono durante il parsing online.

3.6 Specifiche in logica CTL utilizzate nelle verifiche automatiche

Saranno descritte adesso le specifiche CTL che verranno inserite in fondo ai codici SMV trattati nelle varie verifiche automatiche presenti in *DaWeb*.

Queste specifiche vengono determinate dopo che, nelle varie schede del progetto, si compilano i rispettivi campi.

Le verifiche automatiche possibili da fare sono le seguenti:

● Schede Offline:

- Nella scheda *Varie verifiche offline* viene verificata:

- La raggiungibilità della home page in più stati:

Verifica se da ogni pagina è possibile ritornare, seguendo uno dei qualsiasi percorsi, alla pagina iniziale del sito.

Un esempio di specifica CTL è:

$$AG(State=page \rightarrow EF Paging=linkpg_D_Zindex_Qhtm)$$

- La raggiungibilità della home page in un unico stato:

Verifica se ogni pagina possiede un collegamento diretto alla pagina iniziale del sito.

Un esempio di specifica CTL è:

$$AG(State=page \ \& \ !Paging=linkpg_D_Zindex_Qhtm \ -> \ EX(EX(EX \ Paging=linkpg_D_Zindex_Qhtm)))$$

- La consistenza del link:

Verifica se tutti i link del sito conducono a delle pagine esistenti (per pagine esistenti si tiene conto sia delle pagine interne presenti nel file system del server e anche di quelle esterne effettivamente presenti in rete).

La specifica CTL utilizzata è:

$$AG(State=linkpg \ -> \ EX \ (State=page \))$$

- La consistenza delle ancore:

Verifica se tutti i link con ancora del sito conducono a delle pagine esistenti (per pagine esistenti si tiene conto sia delle pagine interne presenti nel file system del server e anche di quelle esterne effettivamente presenti in rete).

La specifica CTL utilizzata è:

$$AG(State=ancora \ -> \ EX \ (State=page))$$

- La consistenza della action:

Verifica se tutti i form presenti nelle pagine del sito conducono a delle pagine esistenti (per pagine esistenti si tiene conto sia delle pagine interne presenti nel file system del server che anche delle pagine esterne effettivamente presenti in rete).

La specifica CTL è:

$AG(State=action \rightarrow EX(State=page))$

- Se tutte le pagine sono interne:

Verifica se tutte i collegamenti all'interno delle pagine del sito, conducono a delle pagine interne.

La specifica CTL utilizzata è:

$AG(c.internal)$

- Se tutte le pagine sono statiche:

Verifica se tutte le pagine all'interno del sito hanno le estensione delle pagine statiche, cioè verifica se le pagine all'interno del sito vengono elaborate o no dal server.

La specifica CTL utilizzata è:

$AG(!c.dynamic)$

- Nella scheda *Raggiungibilità offline* viene verificata:

- Se è possibile raggiungere una pagina precisa, partendo da una determinata pagina, considerando che il percorso può compiere più stati.

Un esempio di specifica CTL è:

$AG(Paging=linkpg_D_Zguest_Qhtm \rightarrow EF$

$Paging=action_D_Zautenticazione_Qasp)$

- Se è possibile raggiungere una determinata pagina, percorrendo da un'altra determinata pagina, considerando che il percorso può compiere un unico stato. Questa verifica può anche essere descritta

dicendo che viene verificata se in una determinata pagina esiste il collegamento ad un'altra determinata pagina.

Un esempio di specifica CTL è:

$$AG(Paging=linkpg_D_Zguest_Qhtm \rightarrow EX(EX(EX \\ Paging=action_D_Zautenticazione_Qasp)))$$

- Nella scheda *Connessione offline* viene verificato:
 - Se una determinata pagina è connessa con ritorno diretto. Per la connessione di una pagina si intende se dalla pagina iniziale del sito è possibile raggiungerla e se poi da questa pagina è possibile ritornare seguendo un percorso diretto (cioè senza passare da altre pagine) alla pagina iniziale.

Un esempio di specifica CTL è:

$$AG(Paging=linkpg_D_Zindex_Qhtm \rightarrow EF Paging=linkpg_D_Zindex_Qhtm) \\ \& AG(Paging=linkpg_D_Zindex_Qhtm \rightarrow EX(EX(EX \\ Paging=linkpg_D_Zindex_Qhtm)))$$

- Se una determinata pagina è connessa con ritorno indiretto. (Il significato di connessione è stato detto nel punto precedente)

Un esempio di specifica CTL è:

$$AG(Paging=linkpg_D_Zindex_Qhtm \rightarrow EF Paging=linkpg_D_Zindex_Qhtm) \\ \& AG(Paging=linkpg_D_Zindex_Qhtm \rightarrow EF Paging= \\ linkpg_D_Zindex_Qhtm)$$

- Nella scheda *Presenza Oggetto Globale* viene verificata se

In tutto le pagine del sito è presente un determinato oggetto (per oggetto si intende: un'immagine, uno sfondo, un applet, un indirizzo e-mail... ect).

Un esempio di specifica CTL è:

AG(o.backgr_D_Znote_Qgif=1)

- Nella scheda *Presenza Oggetto relativa alla pagina* viene verificata se:

In una determinata pagina del sito è presente un determinato oggetto.

Un esempio di specifica CTL è:

EF(Paging=linkpg_D_Zreg_Qasp & o.backgr_D_Znote_Qgif=1)

- Nella scheda *Provenienza Oggetto* viene verificata se:

All'interno del sito, un determinato oggetto proviene da un determinato tag scelto i tag tra: img, body, embed, applet, LinkToObject. (LinkToObject rappresentano gli oggetti provenienti dal tag A).

Un esempio di specifica CTL è:

t.backgr_D_Znote_Qgif=IMG

- Nella scheda *Data e dimensioni* viene verificata se:

- All'interno del sito tutte le pagine hanno una dimensione inferiore a un determinato valore (espresso in byte). (Questa verifica può risultare utile nel momento in cui si vuole verificare se tutte le pagine all'interno di un sito hanno una dimensione tale da non far attendere molto tempo a un utente nel momento in cui vengono caricate).

Un esempio di specifica CTL è:

AG(!c.weight>200)

- All'interno del sito tutte le pagine hanno una dimensione superiore a un determinato valore (espresso in byte).

Un esempio di specifica CTL è:

$AG(!c.weight < 5425)$

- All'interno del sito tutte le pagine hanno una data di ultima modifica più alta rispetto a un determinato valore.

Un esempio di specifica CTL è:

$AG(!c.date < 19990808)$

- All'interno del sito tutte le pagine hanno una data di ultima modifica più bassa rispetto a un determinato valore. (Questa verifica può risultare utile nel momento in cui si vuole verificare se tutte le pagine all'interno del sito hanno sono state o meno aggiornate).

Un esempio di specifica CTL è:

$AG(!c.date > 20020628)$

- Nella scheda *Pattern di specifica* viene verificata:

(il valore messo al posto di P può essere o il nome di un oggetto presente in una pagina oppure il nome di una pagina stessa, mentre il valore messo al posto di S,R e Q può essere solo il nome di una pagina)

- L'assenza dell'oggetto P globale
- L'assenza dell'oggetto P prima della pagina R
- L'assenza dell'oggetto P dopo la pagina Q
- L'assenza dell'oggetto P tra le pagine R e Q
- L'assenza dell'oggetto P dopo la pagina Q, finché si presenta la pagina R.
- La presenza dell'oggetto P globale
- La presenza dell'oggetto P prima della pagina R
- La presenza di un determinato oggetto dopo la pagina Q
- La presenza di un determinato oggetto tra la pagina R e la pagina Q

- La presenza di un determinato oggetto dopo la pagina Q, finché si presenta la pagina R.

- L'esistenza dell'oggetto P globale
- L'esistenza dell'oggetto P prima della pagina R
- L'esistenza di un determinato oggetto dopo la pagina Q
- L'esistenza di un determinato oggetto tra la pagina R e la pagina Q
- L'esistenza di un determinato oggetto dopo la pagina Q, finché si presenta la pagina R.

- L'assenza della pagina P globale
- L'assenza della pagina P prima della pagina R
- L'assenza della pagina P dopo la pagina Q
- L'assenza della pagina P tra la pagina R e la pagina Q
- L'assenza della pagina P dopo la pagina Q, finché si presenta la pagina R.

- La presenza della pagina P globale
- La presenza della pagina P prima della pagina R
- La presenza della pagina P dopo la pagina Q
- La presenza della pagina P tra la pagina R e la pagina Q
- La presenza della pagina P dopo la pagina Q finché si presenta la pagina R.

- L'esistenza della pagina P globale
- L'esistenza della pagina P prima della pagina R
- L'esistenza della pagina P dopo la pagina Q
- L'esistenza della pagina P tra la pagina R e la pagina Q
- L'esistenza della pagina P dopo la pagina Q, finché si presenta la pagina R.

- La precedenza della pagina P dalla pagina S globale
- La precedenza della pagina P dalla pagina S prima della pagina R
- La precedenza della pagina P dalla pagina S dopo la pagina Q

- La precedenza della pagina P dalla pagina S tra la pagina R e la pagina Q
- La precedenza della pagina P dalla pagina S dopo la pagina Q, finché si presenta la pagina R.

- La risposta della pagina P dalla pagina S globale
- La risposta della pagina P dalla pagina S prima della pagina R
- La risposta della pagina P dalla pagina S dopo la pagina Q
- La risposta della pagina P dalla pagina S tra la pagina R e la pagina Q
- La risposta della pagina P dalla pagina S dopo la pagina Q, finché si presenta la pagina R.

Una trattazione completa dei pattern di specifica viene fatta in appendice B

● Schede online

- Nella scheda *Varie verifiche online* viene verificata:

- La raggiungibilità della home page in più stati:

Verifica se da ogni pagina che presenta come parametro, un determinato valore fissato, è possibile ritornare, seguendo uno dei qualsiasi percorsi, alla pagina iniziale del sito.

Un esempio di specifica CTL è:

$$AG(State=page \ \& \ cli=pippo \ \rightarrow \ EF \ (Paging=index_Qhtm_8))$$

- La raggiungibilità della home page in un unico stato:

Verifica se ogni pagina del sito, che presenta come parametro un determinato valore fissato, possieda un collegamento diretto alla pagina iniziale del sito.

Un esempio di specifica CTL è:

$$AG(State=page \ \& \ cli=pippo \ \& \ !Paging=index_Qhtm_8 \\ \rightarrow \ EX(EX \ (Paging=index_Qhtm_8)))$$

- La consistenza dei link:

Verifica se tutti i link delle pagine del sito che presentano come parametro un determinato valore fissato conducono a delle pagine esistenti (per pagine esistenti si tiene conto delle pagine interne presenti nel file system del server e anche delle pagine esterne effettivamente presenti in rete).

Un esempio di specifica CTL è:

$$AG(State=link \ \& \ cli=pippo \ \rightarrow \ EX \ (State=page))$$

- Nella scheda *Raggiungibilità online* viene verificata:
 - Se è possibile raggiungere una pagina precisa, partendo da una determinata pagina, considerando che il percorso può compiere più stati, con il vincolo che la pagina di partenza e la pagina di destinazione presentino come parametro un determinato valore fissato oppure un valore nullo.

Un esempio di specifica CTL è:

$$EF(Paging=autenticazione_Qasp_15 \ \& \ (cli=nrllgrz \ | \ cli=null)) \ \& \\ AG(Paging=autenticazione_Qasp_15 \ \& \ (cli=nrllgrz \ | \\ cli=null) \ \rightarrow \ EF \ (Paging=preventivo_Qasp_24 \ \& \\ (cli=nrllgrz \ | \ cli=null)))$$

- Se è possibile raggiungere una pagina precisa, passando da un'altra determinata pagina, considerando che il percorso può compiere un unico stato, con il vincolo che la pagina di partenza

e la pagina di destinazione presentano come parametro un determinato valore fissato oppure un valore nullo.

Un esempio di specifica CTL è:

$$\begin{aligned} &EF(\text{Paging}=\text{autenticazione_Qasp_15} \ \& \ (\text{cli}=\text{nrllgrz} \ | \ \text{cli}=\text{null})) \ \& \\ &AG(\text{Paging}=\text{autenticazione_Qasp_15} \ \& \ (\text{cli}=\text{nrllgrz} \ | \\ &\text{cli}=\text{null}) \ \rightarrow \ EX(EX(\text{Paging}=\text{preventivo_Qasp_24} \ \& \\ &(\text{cli}=\text{nrllgrz} \ | \ \text{cli}=\text{null})))) \end{aligned}$$

- Nella scheda *Connessione online* viene verificata:
 - Se una determinata pagina è connessa con ritorno diretto. Per connessione si intende che dalla pagina iniziale del sito è possibile raggiungere una precisa pagina e poi da questa ritornare seguendo un percorso diretto (cioè senza passare da altre pagine) alla pagina iniziale. A differenza della stessa verifica vista nella scheda offline, questa verifica viene fatta sul modello sul codice *SMV online*.

Un esempio di specifica CTL è:

$$\begin{aligned} &AG(\text{Paging}=\text{index_Qhtm_8} \ \rightarrow \ EF \ \text{Paging}=\text{autenticazione_Qasp_12}) \\ &\ \& \ AG(\text{Paging}=\text{autenticazione_Qasp_12} \ \rightarrow \ EX(EX \\ &\ \text{Paging}=\text{index_Qhtm_8})) \end{aligned}$$

- Se una determinata pagina è connessa con ritorno indiretto (Il significato di connessione è stato detto nel punto precedente). A differenza della stessa verifica vista nella scheda offline, questa verifica viene fatta sul modello sul codice *SMV online*.

Un esempio di specifica CTL è:

$$AG(\text{Paging}=\text{index_Qhtm_8} \ \rightarrow \ EF \ \text{Paging}=\text{autenticazione_Qasp_12})$$

*& AG(Paging=autenticazione_Qasp_12 -> EF
Paging=index_Qhtm_8).*

3.7 Verifiche automatiche eseguite per siti che si basano sul commercio elettronico

Nei siti, oltre a dover rispettare alcune caratteristiche di funzionalità generale, bisogna tener conto di certi criteri dipendenti dalla categoria a cui appartengono. Criteri che se non rispettati, determinano un lavoro non professionale.

Per esempio, per un sito bancario bisogna considerare una struttura abbastanza standardizzata, da poter permettere l'interazione cliente-banca (ogni utente in tal caso deve possedere una home page personale), con gestione della tracciatura degli ordini, con la possibilità della registrazione degli utenti attraverso l'utilizzo di varie forme di comunicazione ed inoltre dovrebbe consentire le interazioni tra i dipendenti della banca dando anche la possibilità di gestire il loro lavoro.

Un altro esempio è quello di un sito basato sul commercio elettronico. Da varie analisi fatte su questi tipi di siti, in rete, è possibile affermare che le strutture fisiche di questi siti, si possono suddividere in due categoriche. Queste categorie possono essere nominate nel seguente modo:

- Struttura di un sito basato sul commercio elettronico con home page personale
- Struttura di un sito basato sul commercio elettronico senza home page personale

Le strutture di entrambe le categorie, hanno sempre lo scaffale (da cui scegliere il prodotto), il carrello, la pagina di conferma acquisto e la pagina di registrazione, ma mentre in una categoria la registrazione viene fatta all'inizio e si ha a disposizione una propria home page personale in cui è possibile entrare immediatamente, nel secondo caso la registrazione si esegue prima di confermare l'acquisto. In tal caso non c'è bisogno che ogni utente possieda una home page personale e la registrazione serve solo per fornire le informazioni utili per la spedizione dei prodotti acquistati.

3.7.1 Struttura di un sito basato sul commercio elettronico con home page personale

La struttura di un sito basato sul commercio elettronico con home page personale è rappresentata nel diagramma a stati finiti mostrato nella figura seguente.

(In questo diagramma lo stato può rappresentare o una pagina o un insieme di pagine).

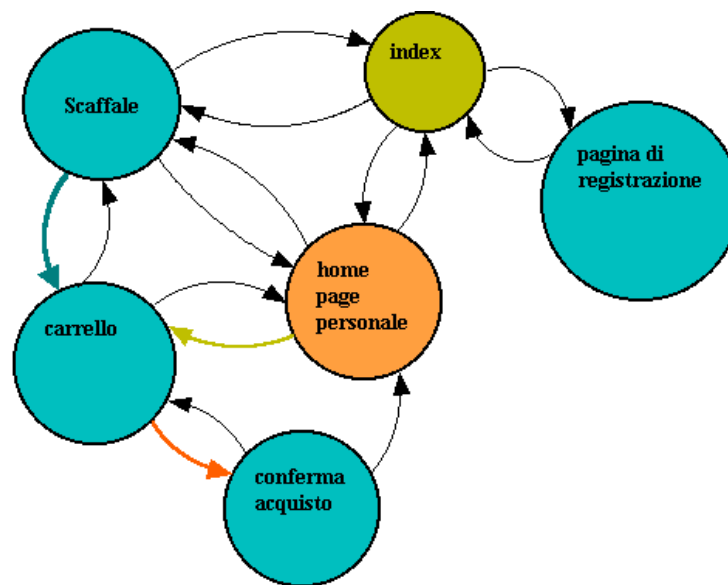


Figura 3–10: Struttura di un sito basato sul commercio elettronico con home page personale

In questo diagramma si nota facilmente come la pagina di registrazione sia collegata direttamente alla pagina *index* e come dalla pagina *index* si può accedere al *home page personale* (questo accesso avviene sempre tramite un form in cui si inserisce l'*userid* e la *password*). Dal *home page personale* poi è possibile gestire tutte le funzionalità di navigazione e di acquisto messe a disposizione dal sito. È possibile, quindi, eseguire la ricerca dei prodotti dallo *scaffale*, inserirli nel *carrello* e acquistarli di conseguenza (nello stato di *conferma acquisto* è presente la modalità di pagamento e l'inserimento di altri dati personali nel momento in cui si vuole spedire i prodotti acquistati a un destinatario diverso dalla persona che si è registrata). Dall'*index* è possibile anche navigare tra i prodotti dello *scaffale* ma in questo caso non è possibile acquistarli. Una

cosa importante da notare in questo modello a stati finiti è l'unico collegamento presente dallo stato *carrello* allo stato *conferma acquisto* e il collegamento dallo stato *scaffale* allo stato *carrello* (dal *home page personale* spesso è anche possibile entrare nel *carrello*, in modo da visualizzare in che stato si trova).

3.7.2 Struttura di un sito basato sul commercio elettronico senza home page personale

La struttura di un sito basato sul commercio elettronico senza home page personale è rappresentata nel diagramma a stati finiti mostrato nella figura seguente. (Anche in questo diagramma lo stato può rappresentare o una pagina o un insieme di pagine).

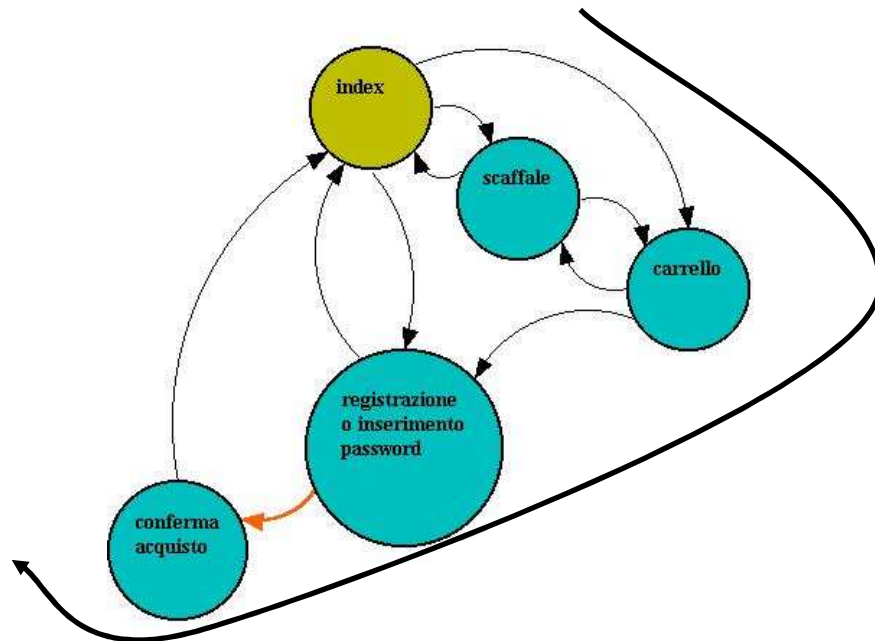


Figura 3-11: Struttura di un sito basato sul commercio elettronico senza home page personale

In questo diagramma, dalla pagina *index* è possibile accedere facilmente allo *scaffale*, è quindi possibile navigare tra i prodotti con la possibilità poi di inserirli all'interno di un *carrello* che all'inizio non ha un'identità (cioè non ha un proprietario). I prodotti presenti nel *carrello* possono essere poi acquistati, ma prima di fare questo, dal *carrello* si deve per forza andare nello stato *registrazione o conferma password* dalla quale è possibile inserire i dati personali, utili per poter poi spedire il prodotto. Se

questi dati sono stati già inseriti in un precedente momento, in questo stato è sufficiente inserire una *userid* e una *password*. Una volta inseriti questi valori è possibile confermare l'acquisto entrando nello stato *conferma acquisto* dal quale è possibile decidere anche la modalità di pagamento.

Una cosa importante da notare in questo modello a stati finiti è l'unico collegamento presente dallo stato *registrazione o inserimento password* allo stato *conferma acquisto* ed il collegamento dallo stato *scaffale* allo stato *carrello*. In base a queste caratteristiche è possibile eseguire delle verifiche utili per capire se un sito basato sul commercio elettronico rispetta una delle due strutture descritte precedentemente. Queste verifiche sono utili, in fase di progettazione, per capire se il sito viene correttamente realizzato secondo uno dei due modelli. Queste verifiche possono essere eseguite, anche in questo caso, con l'utilizzo di metodi formali, come fatto per verificare le proprietà dei siti web statici e dinamici descritti precedentemente.

Nel form principale di *DaWeb* è presente proprio una scheda che si chiama *Commercio Elettronico* dentro la quale è possibile fare le verifiche su modelli di siti basati sul commercio elettronico che possiedono una home page personale. Queste verifiche sono le seguenti:

- Nella scheda *verifica offline* viene verificato:
 1. Se la home page personale viene raggiunta direttamente dalla pagina *index*; cioè se nella pagina *index* ci sia un form dal quale, inserendo una *userid* e *password*, è possibile accedere alla home page personale.

Un esempio di specifica CTL è:

$$AG(\text{Paging}=\text{linkpg_D_Zcdmarket_Zindex_Qhtm} \rightarrow EX(EX(EX(\text{Paging}=\text{action_D_Zcdmarket_Zselezione_Qasp}))).$$

2. Se la pagina di registrazione è collegata direttamente alla pagina *index*. Questa non è una verifica importante ma viene rispettata in molti siti professionali in cui è possibile registrare i propri clienti

Un esempio di specifica CTL è:

$$AG(\text{Paging}=\text{linkpg_D_Zindex_Qhtm} \rightarrow EX(EX(EX(\text{Paging}=\text{linkpg_D_Zreg_Qasp}))))$$

3. Se la pagina di acquisto confermato, viene anticipato sempre dal carrello. Questa verifica deve essere sempre rispettata da parte di un sito basato sul commercio elettronico perché è ovvio che prima di confermare un acquisto, i prodotti devono essere inseriti in un carrello. Il carrello infatti è una pagina nella quale è possibile inserire all'interno i prodotti scelti e solo alla fine dal carrello, è possibile poterli acquistare.

Un esempio di specifica CTL è:

$$\begin{aligned} & !E!(\text{Paging}=\text{action_D_Zpreventivo_Qasp}) U \\ & ((\text{Paging}=\text{action_D_Zacquisto_Qasp}) \& \\ & !(\text{Paging}=\text{action_D_Zpreventivo_Qasp})) \end{aligned}$$

4. Se il carrello viene anticipato solo dalla pagina dei prodotti. Anche questa verifica deve essere rispettata in un sito basato sul commercio elettronico. Infatti i dati inseriti nel carrello saranno presi dalla pagina dei prodotti, quindi, è evidente che questa pagina la deve anticipare. Molto spesso però dalla home page personale è possibile entrare nella pagine che identifica il carrello, per visualizzare il suo contenuto. In questo caso anche se il risultato della verifica è falso, non è detto che la struttura sia errata.

Un esempio di specifica CTL è:

$$\begin{aligned} & !E!(\text{Paging}=\text{action_D_Zselezione1_Qasp}) U \\ & ((\text{Paging}=\text{action_D_Zpreventivo_Qasp}) \& \\ & !(\text{Paging}=\text{action_D_Zselezione1_Qasp})). \end{aligned}$$

5. Se il carrello viene raggiunto direttamente dalla pagine dei prodotti. Questa verifica sicuramente risulta vera se risulta vera

la verifica precedente, quando però tale verifica non è vera, il che significa che il carrello viene anticipato non solo dalla pagina dei prodotti, questa verifica risulta utile per vedere se tra le pagine che richiamano il carrello è presente la pagina dei prodotti

Un esempio di specifica CTL è:

$$AG(\text{Paging}=\text{action_D_Zcdmarket_Zselezione1_Qasp} \rightarrow EX(EX(EX \text{Paging}=\text{action_D_Zcdmarket_Zpreventivo_Qasp})))$$

6. Se il carrello viene raggiunto direttamente dalla pagine dei prodotti e dalla home page personale. Quando la verifica presentata nel punto numero quattro è falsa, ed è vera la verifica del punto precedente, allora, visto che questo implica che, oltre alla pagina dei prodotti, ci sono altre pagine che anticipano il carrello, ha senso eseguire questa verifica per vedere se tra queste pagine è presente la home page personale.

Un esempio di specifica CTL è:

$$AG(\text{Paging}=\text{action_D_Zcdmarket_Zselezione1_Qasp} \rightarrow EX(EX(EX \text{Paging}=\text{action_D_Zcdmarket_Zpreventivo_Qasp}))) \& \\ AG(\text{Paging}=\text{action_D_Zcdmarket_Zselezione_Qasp} \rightarrow EX(EX(EX \text{Paging}=\text{action_D_Zcdmarket_Zpreventivo_Qasp})))$$

- Nella scheda *Verifica con password* viene verificato:

1. Se la home page personale viene visualizzata da tutti quelli che posseggono le password inserite in un'apposita lista (che si trova nella scheda associazioni password). In questo modo si verifica

se è possibile raggiungere una home page con una password non autorizzata oppure senza inserire la password.

Un esempio di specifica CTL è:

```
( EF(Paging=cdmarket_Zselezione_Qasp_12 & cli=nrllgrz) |
  EF(Paging=cdmarket_Zselezione_Qasp_14 & cli=nrllgrz) |
  EF(Paging=cdmarket_Zselezione_Qasp_16 & cli=nrllgrz) |
  EF(Paging=cdmarket_Zselezione_Qasp_25 & cli=nrllgrz) |
  EF(Paging=cdmarket_Zselezione_Qasp_29 & cli=nrllgrz))
& ( EF(Paging=cdmarket_Zselezione_Qasp_12 & cli=felice)
  | EF(Paging=cdmarket_Zselezione_Qasp_14 & cli=felice) |
  EF(Paging=cdmarket_Zselezione_Qasp_16 & cli=felice) |
  EF(Paging=cdmarket_Zselezione_Qasp_25 & cli=felice) |
  EF(Paging=cdmarket_Zselezione_Qasp_29 & cli=felice)) &
( EF(Paging=cdmarket_Zselezione_Qasp_12 & cli=nrllgrz)
  | EF(Paging=cdmarket_Zselezione_Qasp_14 &
  cli=nrllgrz) | EF(Paging=cdmarket_Zselezione_Qasp_16
  & cli=nrllgrz) |
  EF(Paging=cdmarket_Zselezione_Qasp_25 & cli=nrllgrz) |
  EF(Paging=cdmarket_Zselezione_Qasp_29 & cli=nrllgrz))
& ( EF(Paging=cdmarket_Zselezione_Qasp_12 & cli=felice)
  | EF(Paging=cdmarket_Zselezione_Qasp_14 & cli=felice) |
  EF(Paging=cdmarket_Zselezione_Qasp_16 & cli=felice) |
  EF(Paging=cdmarket_Zselezione_Qasp_25 & cli=felice) |
  EF(Paging=cdmarket_Zselezione_Qasp_29 & cli=felice))
```

2. Se la il carrello viene visualizzato da tutti quelli che posseggono le password inserite in un'apposita lista (che si trova nella scheda associazioni password). In questo modo si verifica se è possibile raggiungere una home page con una password non autorizzata oppure senza inserire la password.

Un esempio di specifica CTL è:

```
( EF(Paging=cdmarket_Zpreventivo_Qasp_22 & cli=nrdllgrz) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_3 & cli=nrdllgrz) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_4 & cli=nrdllgrz) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_23 & cli=nrdllgrz)
  | EF(Paging=cdmarket_Zpreventivo_Qasp_5 &
  cli=nrdllgrz)) & (
  EF(Paging=cdmarket_Zpreventivo_Qasp_22 & cli=felice) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_3 & cli=felice) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_4 & cli=felice) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_23 & cli=felice) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_5 & cli=felice)) &
  ( EF(Paging=cdmarket_Zpreventivo_Qasp_22 &
  cli=nrdllgrz) | EF(Paging=cdmarket_Zpreventivo_Qasp_3
  & cli=nrdllgrz) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_4 & cli=nrdllgrz) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_23 & cli=nrdllgrz)
  | EF(Paging=cdmarket_Zpreventivo_Qasp_5 &
  cli=nrdllgrz)) & (
  EF(Paging=cdmarket_Zpreventivo_Qasp_22 & cli=felice) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_3 & cli=felice) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_4 & cli=felice) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_23 & cli=felice) |
  EF(Paging=cdmarket_Zpreventivo_Qasp_5 & cli=felice)).
```

Capitolo 4

DESCRIZIONE DEL PROGETTO

4.1 Introduzione

Saranno descritti adesso, i vari passi che devono essere eseguiti all'interno di *DaWeb*, per determinare i vari risultati.

DaWeb permette di analizzare sia siti statici che siti dinamici. È possibile, quindi, analizzare oltre le pagine html anche le pagine asp, php, jsp ect. Questo software viene eseguito su un server, lì dove è presente il sito che si vuole analizzare. Per analizzare queste pagine, però, occorre che il server, su cui viene eseguito tale software, riesca ad elaborare tali tipi di file. Per esempio, per elaborare le pagine asp occorre possedere su piattaforma windows, il *PWS* oppure l'*IIS*, mentre per elaborare le pagine php è possibile procedere, per esempio, con l'installazione di un server *apache* (la modalità di installazione su piattaforma windows del server *apache* e del *PWS* è presente nel cd allegato).

4.2 Installazione Software

Inserito il cd allegato, compare automaticamente la seguente schermata che facilita l'installazione delle parti utili per il funzionamento del software, presentando anche la relativa documentazione.



Figura 4-1: Schermata introduttiva che di presenta nel momento in cui si inserisce il CD

4.3 Utilizzo del software

Una volta installato il software, è possibile procedere nel suo utilizzo.

Il form principale di *DaWeb* è presente nella figura seguente:



Figura 4-2: Form principale del software *DaWeb*

Questo form è composto da diverse schede organizzate in maniera logica, dalla quale è possibile svolgere i vari compiti, utili per eseguire le verifiche che saranno discusse in seguito.

La prima cosa da fare è determinare il parsing di un'applicazione web. In tal caso è possibile procedere in due modi differenti:

1. Dalla scheda principale:

Nell'ipotesi in cui l'applicazione web (indipendentemente che sia statica o dinamica), presente sulla stessa macchina su cui si fa eseguire *DaWeb*, non è stata ottenuta tramite un programma di mirror.

2. Dalla scheda mirror:

Nell'ipotesi in cui l'applicazione web, presente sulla macchina in cui si fa eseguire *DaWeb*, è stata ottenuta tramite un programma di mirror (in particolare, il programma di mirror utilizzato prende il nome di Offline Explorer 2.4 ed è possibile installarlo, nella versione shareware, direttamente dallo stesso *DaWeb*).

4.3.1 Parsing dalla scheda principale

Dalla lista a comparsa (presente sotto la scritta *inserisci la directory virtuale* che prende anche il nome di *ComboBox*) è possibile scegliere o inserire la path in cui sono presenti tutte le directory virtuali. All'inizio dopo l'installazione tale lista è vuota. Occorre scrivere pertanto al suo interno l'indirizzo della suddetta path (se si è installato il *PWS* con le impostazioni di default, bisogna inserire in tale *ComboBox*: `c:/inetpub/wwwroot/`).

Dopo aver scritto la path, se si clicca sul tasto *inserisci*, la path inserita rimane permanentemente nella lista anche dopo aver chiuso il programma.

Una volta inserito nel *ComboBox* l'indirizzo della path in cui si trovano tutte le directory virtuali, dal tasto *seleziona* è possibile entrare nelle varie directory virtuali e selezionare il file *index* del sito (cioè quel file che viene avviato come prima pagina di un sito).

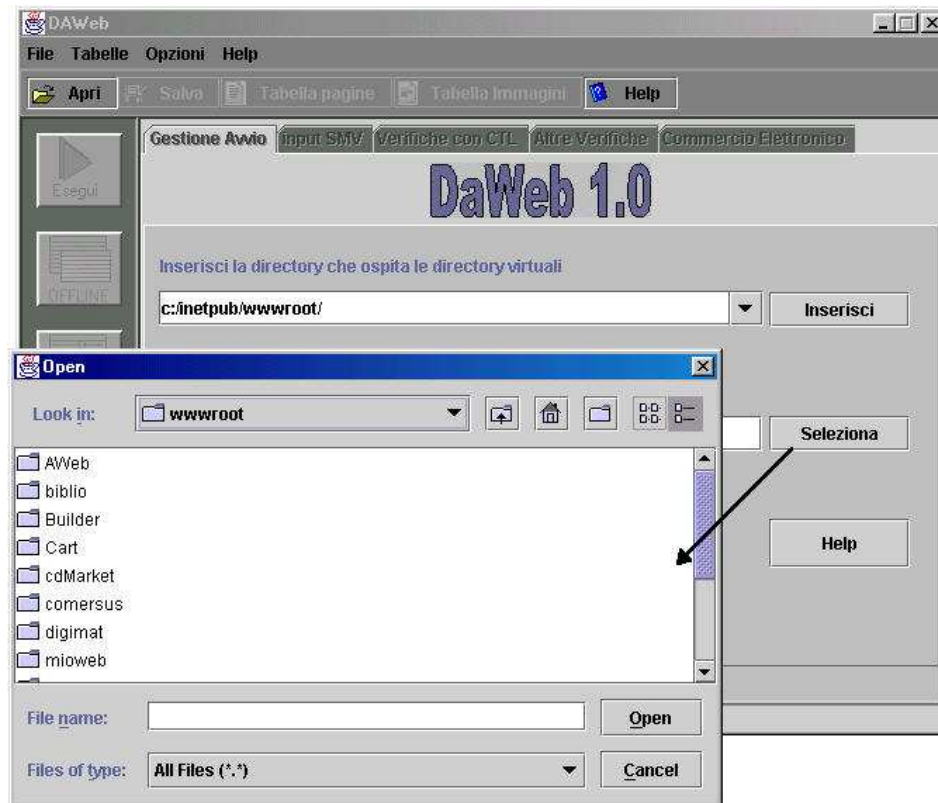


Figura 4-3: Metodo per ricercare l'*index* dalla directory virtuale

Dopo aver selezionato l'*index* (la cui path viene inserito nel campo testo vicino al tasto *seleziona*) è possibile cliccare sul tasto *esegui* presente nella colonna a sinistra.

Il *CheckBox* presente in fondo alla pagina permette di eseguire anche la verifica online. Questa verifica però, si può fare nell'ipotesi in cui è possibile accedere all'indirizzo: <http://localhost/> (inoltre occorre caricare nel server la stessa path di quella inserita nel *ComboBox* presente nella schermata e come detto in precedenza tale server deve essere in grado di elaborare il tipo di pagine che si presentano nell'applicazione web che si sta verificando).

4.3.2 Parsing dalla scheda mirror

I file determinati dal programma mirror *Offline Explorer* vengono salvati in cartelle che prendono lo stesso nome dell'URL di provenienza. Quindi molto spesso i

nomi di queste cartelle iniziano con `www`. Realizzando il parsing dalla scheda principale, il contenuto delle cartelle che iniziano con `www` vengono viste come pagine esterne. La scheda `mirror` è stata creata proprio per eliminare questo problema. In questa scheda è stato eliminata anche la possibilità di eseguire il parsing online; in effetti la struttura online delle pagine ricavate dal programma `mirror` è simile alla struttura offline, quindi eseguire il parsing online non avrebbe senso.

A parte tutto questo, il modo di eseguire il parsing dalla scheda `mirror`, è simile a quello visto dalla scheda principale.

4.4 Differenza tra parsing offline e parsing online

Visto che il software *DaWeb* viene eseguito su un server, è possibile elaborare sia le pagine dal file system, prendendo le pagine dal percorso assoluto che individua la directory virtuale, sia le pagine risultanti dal processo server dall' URL `http://localhost/...` che si collega alla directory virtuale.

Le caratteristiche del server permettono di capire quali sono le pagine che possono essere analizzate. Infatti come detto anche in precedenza, se si usa un server su piattaforma windows su cui è installato il *PWS* oppure l'*IIS*, è possibile analizzare le pagine *asp*, mentre se si utilizzano altri server (ad esempio *apache* o server che si appoggiano su *servlet*) è possibile analizzare pagine *php*, o pagine *jsp*, ect.

In questo modo, è possibile, sia valutare il file associato alle pagine dinamiche nel suo contenuto (come insieme di script che vengono eseguiti lato server e codice html) e il risultato html determinato dal server della stessa pagina dinamica ottenuta in base a dei parametri inseriti nei form che la richiamano. Il primo caso prende il nome di *parsing offline* mentre il secondo prende il nome di *parsing online*.

Nel parsing offline vengono analizzati anche i vari script lato server e lato client, ottenendo come risultato anche l'insieme dei probabili link che si possono determinare dall'esecuzione del codice.

4.5 Come eseguire il parsing

Mentre l'esecuzione del parsing offline è immediato nel momento in cui si clicca sul tasto esegui, l'esecuzione del parsing online risulta più complessa da fare. Infatti durante il parsing online occorre inserire, nei vari form che si presentano, gli opportuni parametri. Questi parametri sono gli stessi di quelli che vengono inseriti nei form delle pagine visualizzate su un browser e sono indispensabili per il server per determinare la pagina successiva (le pagine dinamiche infatti dipendono dai parametri). La struttura del form che si determina durante la fase di parsing online è la seguente:



	Type	Name	Value
1	password	cli	pippo
2	submit		INVIA

Non Invio Invio DiNuovo Browser

Figura 4-4: form che compare durante il parsing online in cui è possibile inserire i parametri

La parte da compilare è quella che si trova sotto la colonna di nome *Value*.

Sarà ora descritto questo form:

- Il titolo rappresenta il nome della pagina web che presenta il form in questione.
- Il nome della pagina presente immediatamente sotto il titolo rappresenta la pagina di destinazione in cui si stanno inviando i parametri.
- Il tasto *Invio* invia i parametri alla pagina successiva e fa continuare il processo di parsing.

- Il tasto *DiNuovo* invia i parametri alla pagina successiva e ripete nuovamente l'esecuzione della stesso form in modo da poter inserire parametri differenti.
- Il tasto *Non Invio* dà la possibilità all'operatore di non inviare i dati alla pagina successiva e quindi dà la possibilità di non andare avanti lungo un cammino. Questo risulta utile quando il parsing ritorna su un percorso già eseguito in precedenza.
- Il tasto *browser* permette di far visualizzare in Explorer la pagina in questione che presenta il form che si sta compilando.

I form vengono visualizzati in maniera opportuna durante il parsing online, non sempre tutte le volte che viene trovato un form nella pagina che si sta analizzando; infatti viene visualizzato se non è stato mai visualizzato precedentemente e presenta dei campi che devono essere compilati.

4.6 Risultati del parsing di un sito

Una volta cliccato sul tasto *esegui* e completato l'algoritmo di parsing, si possono valutare i risultati prodotti. I risultati di questi parsing vengono riassunti nelle varie tabelle che possono essere evidenziate cliccando sui rispettivi tasti *online* (v. fig 4.6) e *offline* (v. fig 4.5) presenti nella colonnina a sinistra del form principale oppure selezionando una delle seguenti tabelle presenti nel menù *tabelle* (v. fig 4.7):

- Tabella delle immagini (sia offline che online)
- Tabella delle e-mail (sia offline che online)
- Tabella delle pagine interne (sia offline che online)
- Tabella delle pagine esterne (sia offline che online)
- Tabella delle pagine dinamiche
- Tabella delle pagine statiche
- Tabella dei link a file (sia offline che online)
- Tabella delle ancore
- Tabella dei segnalibri

- Tabella degli applet
- Tabella dei form
- Tabella degli elementi all'interno dei form

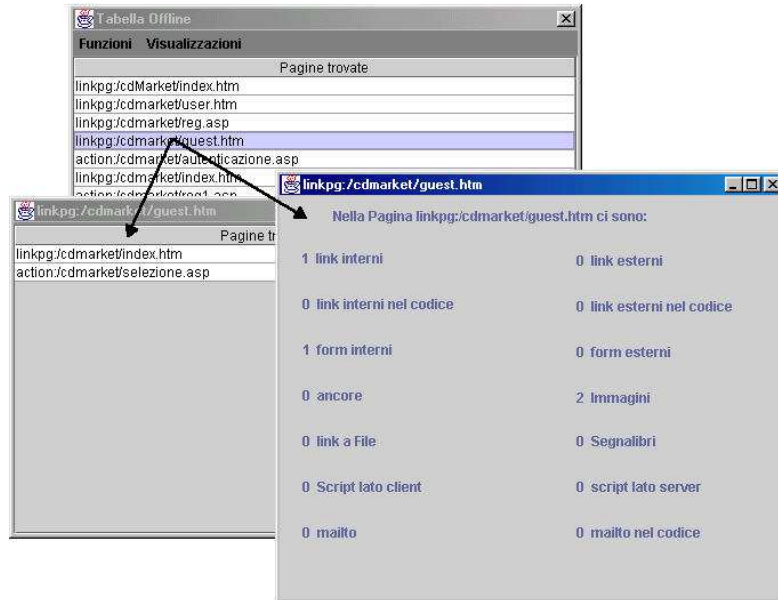


Figura 4-5: Tabelle che si determinano dal parsing offline

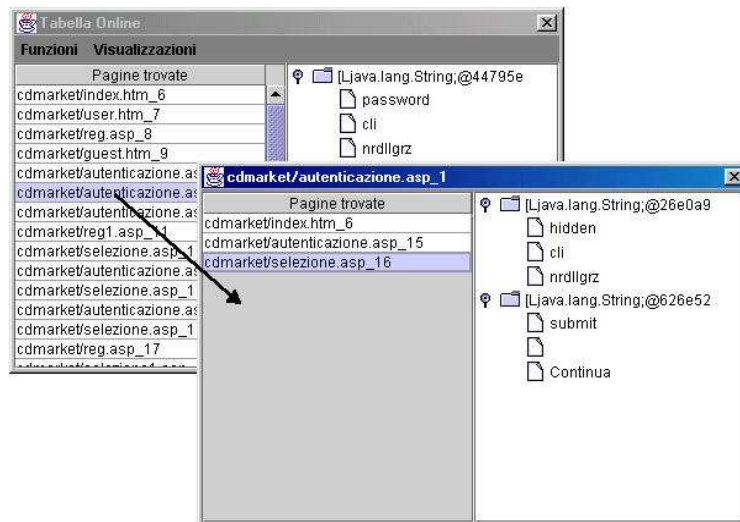


Figura 4-6: Tabelle che si determinano dal parsing online

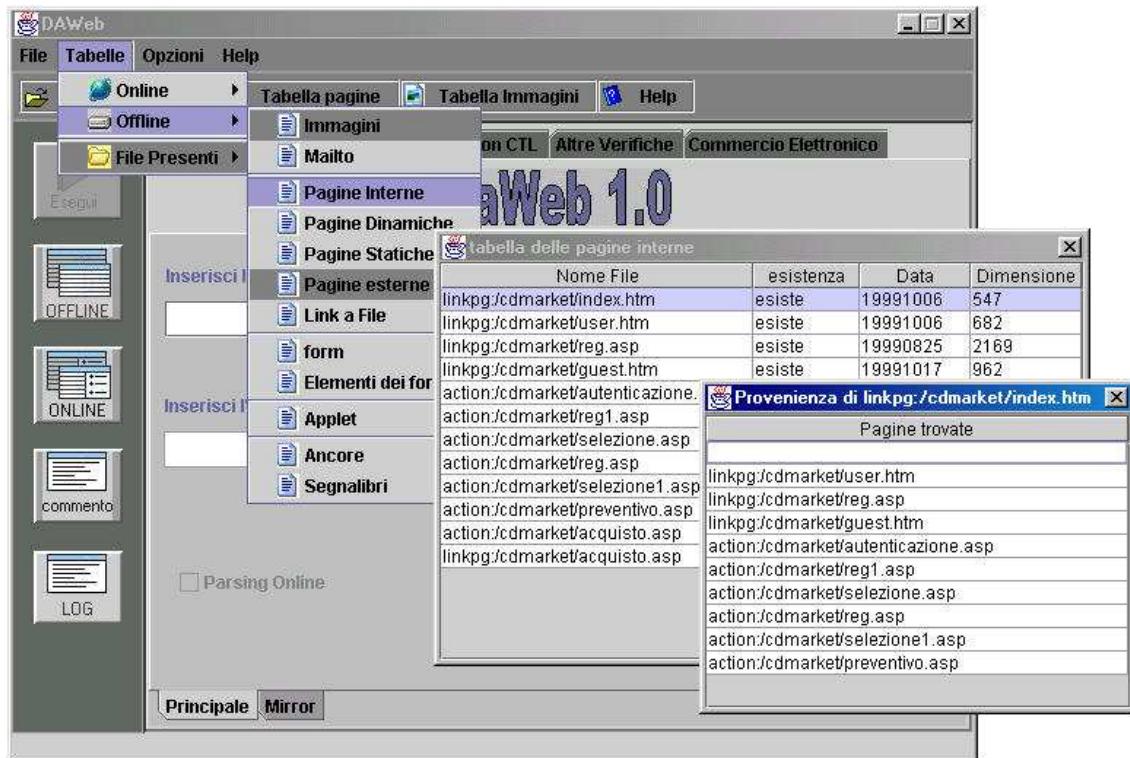


Figura 4-7: Tabella delle pagine dinamiche presenti nel sito analizzato

4.7 Salvataggio dei dati

La procedura di parsing, per siti di non piccole dimensioni, non viene svolto in tempi ridotti, soprattutto quando viene deciso di fare anche il parsing online, in cui bisogna inserire i parametri nei form che compaiono durante. Per evitare di fare questo ogni volta, è possibile salvare tutti i dati del parsing in un file *.vof*. In un momento successivo, se si vuole verificare qualcosa su dati già determinati e salvati, non occorre ripetere il parsing, ma è possibile aprire il file *.vof*.

4.8 Determinazione del codice SMV

Dopo avere eseguito il parsing, o aperto un file *.vof*, è possibile determinare il codice SMV, con il quale viene rappresentato il modello a stati finiti del sito che si è

analizzato. In realtà vengono determinati: il codice *SMV offline* e il codice *SMV online* (quest'ultimo codice viene determinato se è presente il parsing online). Questi codici hanno una struttura diversa. Naturalmente il primo prende i dati dal parsing offline mentre il secondo prende i dati dal parsing online. Inoltre, mentre nel primo caso il modello a stati finiti è più accurato per quanto riguarda ancora e action, dentro il quale possono anche essere inseriti i vari moduli che tengono conto degli oggetti (immagini, mailto, form, input, embed ect) contenuti nella varie pagine, delle dimensioni, delle date di ultima modifica dei file del sito e dei tag di provenienza, nel secondo caso il parsing è più accurato per quanto riguarda l'avanzamento delle pagine in base ai parametri inseriti. Quindi, con il CTL, mentre nel primo caso è possibile eseguire una verifica accurata offline delle caratteristiche del sito, nel secondo caso è possibile verificare le caratteristiche del sito tenendo anche in considerazioni eventuali parametri inseriti nella fase del parsing online. In figura 4-8 è presente la scheda del form principale dalla quale è possibile determinare il codice *SMV offline* e online (se presente).

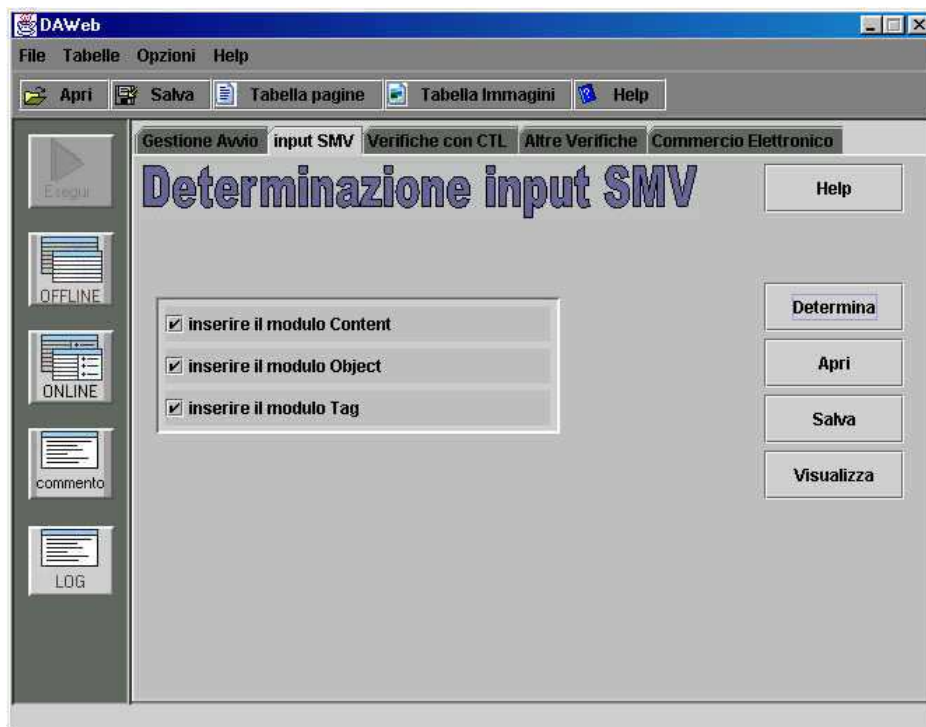


Figura 4-8: Scheda da cui è possibile determinare e visualizzare i codici SMV del sito analizzato

Da questa scheda, con il tasto *salva*, è possibile salvare in un file *.smv* sia il codice *SMV offline* che il codice *SMV online*. Nello stesso modo, con il tasto *apri*, è possibile aprire sia il codice *SMV online* che il codice *SMV offline*. Aprendo questi file in questo modo però può capitare che i dati ricavati dal parsing sono diversi dal codice presente. In tal caso non sarà quindi possibile, o non avrà senso eseguire in automatico le verifiche CTL che saranno descritte in seguito. Sarà comunque possibile eseguire le verifiche inserendo le specifiche CTL manualmente.

4.9 Verifiche CTL

Una volta determinato il codice SMV, che rappresenta il modello a stati finiti del sito analizzato, è possibile, eseguire le varie verifiche utilizzando le specifiche CTL che vengono inserite alla fine del codice SMV.

In realtà, come detto in precedenza, i codici SMV determinati sono due: quello online e quello offline. Pertanto in base alla verifica, si deve inserire la specifica in fondo a uno dei due codici.

Per le verifiche che si basano sulle caratteristiche generali del sito, si devono inserire le specifiche in fondo al codice *SMV offline* mentre per le verifiche che si basano sull'andamento del sito in base ai parametri si devono inserire le specifiche in fondo al codice *SMV online*.

La creazione delle specifiche CTL può essere fatta sia in maniera manuale (in tal caso bisogna conoscere sia come è fatto il modello sia la sintassi del codice CTL) che in maniera automatica scegliendo, dal form principale, la scheda associata alla verifica che si vuole fare e compilare i vari campi al suo interno in modo opportuno.

A questo punto saranno descritte le varie schede del form principale dalla quale è possibile eseguire le verifiche automatiche.

4.9.1 Varie verifiche (su SMV offline)

Facendo riferimento alla figura 4-9, è semplice capire come eseguire le verifiche presenti nella lista a comparsa presente all'interno della scheda *varie*.

Una volta scelta la voce dalla lista, basta cliccare sul tasto *esegui* presente all'interno della stessa scheda

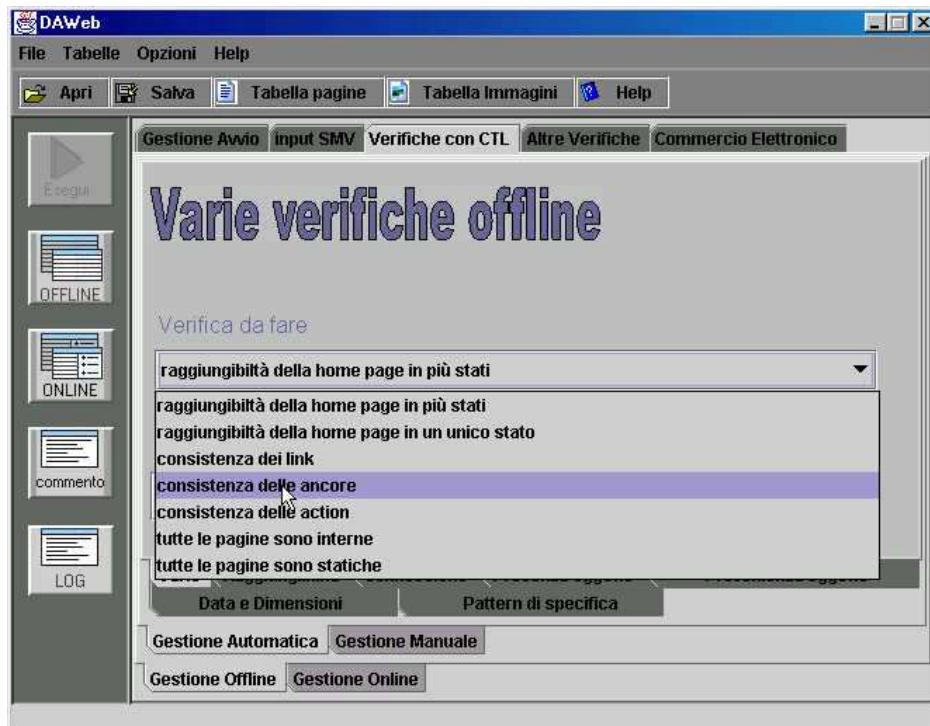


Figura 4–9: Varie verifiche che vengono fatte basandosi sul codice SMV offline

Le varie verifiche che si possono eseguire in questa scheda sono:

- *Raggiungibilità della home page in più stati*

Verifica se è possibile, da ogni pagina del sito, la raggiungibilità della home page, anche se questa viene raggiunta in maniera non diretta cioè passando prima da altre pagine.

- *Raggiungibilità della home page in un unico stato*

Verifica se è possibile, da ogni pagina del sito, la raggiungibilità della home page, nell'ipotesi che venga raggiunta in maniera diretta. In altre parole si deve verificare se all'interno di ogni pagina ci sia un collegamento alla home page del sito.

- *Consistenza dei link*

Verifica se i *link*, di ogni pagina del sito, conducono sempre a pagine o file esistenti sia interni che esterni al sito.

- *Consistenza delle ancore*

Verifica se le *ancore*, di ogni pagina del sito, conducono sempre a pagine o file esistenti sia interni che esterni al sito.

- *Consistenza delle action*

Verifica se le *action*, di ogni pagina del sito, conducono sempre a pagine o file esistenti sia interni che esterni al sito.

- *Verifica che tutte le pagine sono interne*

Verifica se i link di ogni pagina del sito sono fatti a pagine interne del sito. Se almeno un link è fatto a una pagina esterna, il risultato della verifica è falso

- *Verifica che tutte le pagine sono statiche*

Verifica se qualche pagina interna del sito viene elaborata dal server. Se ciò succede almeno per una pagina, il risultato della verifica è falso.

4.9.2 Verifica di Raggiungibilità delle pagine (su SMV offline)

Facendo riferimento alla figura 4-10, è semplice capire come eseguire la verifica di raggiungibilità delle pagine. Per eseguire questa verifica, dalle lista a comparsa bisogna scegliere la pagina di partenza e la pagina di destinazione e bisogna inoltre scegliere se la raggiungibilità debba avvenire o meno attraverso il passaggio da altre pagine.



Figura 4–10: Verifica di raggiungibilità delle pagine (si basa sul codice SMV offline)

4.9.3 Verifica di Connessione delle pagine (su SMV offline)

Facendo riferimento alla figura 4-11, si può capire come eseguire la verifica di connessione di una determinata pagina.

Per connessione si intende la verifica simultanea della raggiungibilità di una determinata pagina dalla pagina *index* e del ritorno da questa pagina di nuovo alla

pagina *index*. Questo ritorno inoltre può essere deciso se farlo direttamente oppure attraverso il passaggio di altre pagine.

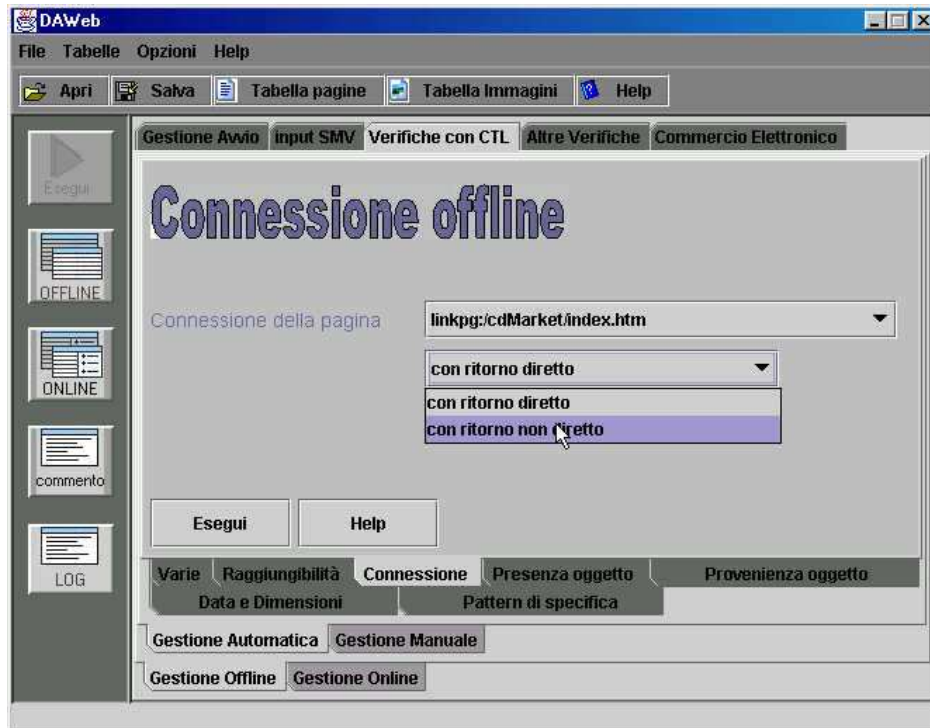


Figura 4-11: Verifica di connessione di una determinata pagina (si basa sul codice SMV offline)

4.9.4 Presenza degli oggetti in tutte le pagine o in una determinata pagina (su SMV offline)

Un'altra verifica che è possibile fare permette di valutare se un determinato oggetto si trova in tutte le pagine o in una determinata pagina. Per fare questo è semplice procedere attraverso le due schede del form principale mostrate rispettivamente in figura 4-12 e in figura 4-13.

Mentre nel primo caso bisogna selezionare solo l'oggetto presente nel sito che si vuole verificare, nel secondo caso bisogna selezionare sia l'oggetto che la pagina.



Figura 4-12: Verifica della presenza di un oggetto in tutte le pagine

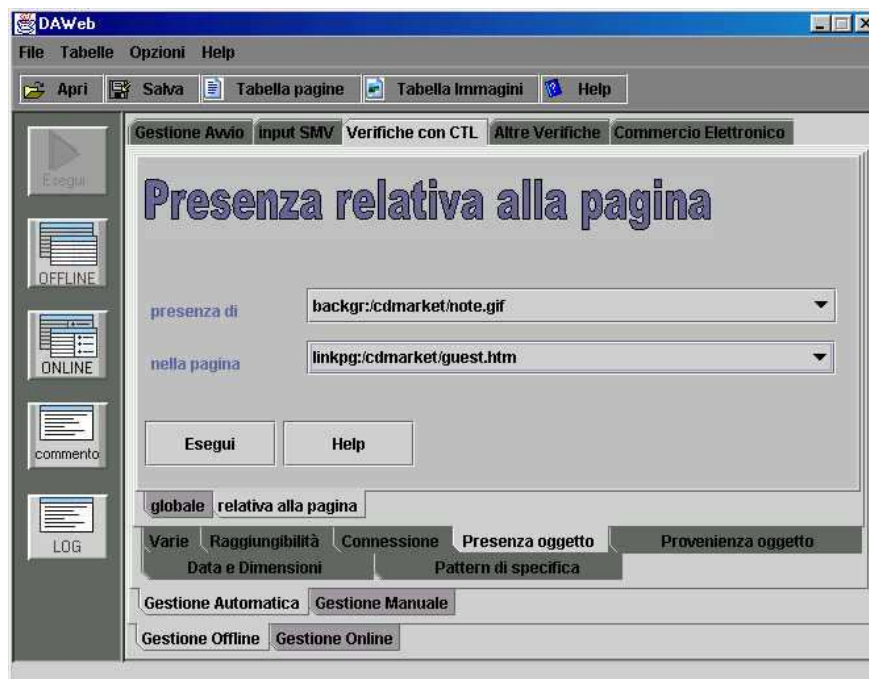


Figura 4-13: Verifica della presenza di un oggetto in una determinata pagina

4.9.5 Provenienza oggetto (su SMV offline)

Un'altra verifica che è possibile fare permette di valutare se un determinato oggetto è stato preso, all'interno delle pagine html da uno dei seguenti tag: *LinkToObject* (che rappresenta il tag A), *APPLET*, *IMG*, *BODY*, *EMBED*, *FORM* o *INPUT*.

Per fare questo è semplice procedere attraverso la schede del form principale mostrata in figura 4-14.



Figura 4–14: Verifica della provenienza di un oggetto

4.9.6 Pattern di Specifica (su SMV offline)

Nella scheda di figura 4-15 è possibile eseguire alcune verifiche utilizzando i pattern di specifica.

I pattern di specifica sono delle formule predefinite e studiate a priori per eseguire verifiche più complesse.

La descrizione dettagliata dei pattern di specifica e il suo utilizzo nel progetto è presente in appendice B.

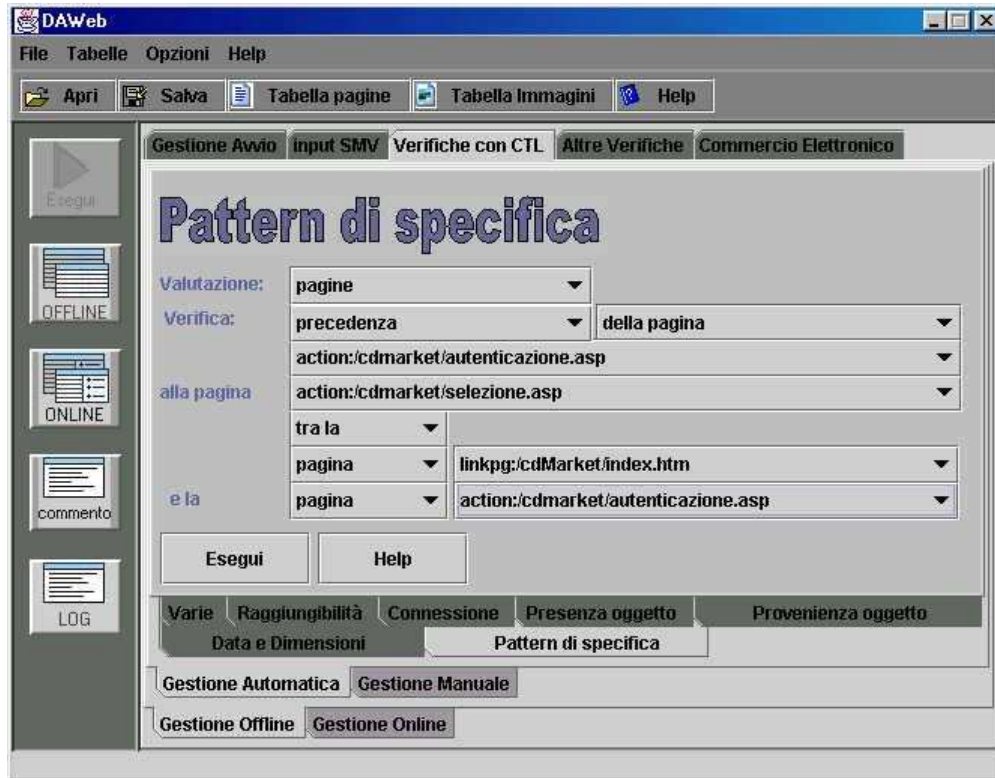


Figura 4-15: Scheda in cui è possibile eseguire le verifiche utilizzando i pattern di specifica

4.9.7 Dimensione e data (su SMV offline)

Un'altra verifica che è possibile fare permette di valutare se le pagine hanno una dimensione o una data superiore o inferiore a una prefissata dimensione o data. Questa data o dimensione è possibile impostarla a mano oppure è possibile sceglierla tra le date o le dimensioni prese dalla valutazione delle pagine all'interno del sito (il formato della data è *aaaammgg* dove *a* sta per anno, *m* per mese e *g* per giorno mentre la dimensione è espressa in byte). Tutto questo è possibile farlo nella scheda del form principale, presente nella figura 4-16.

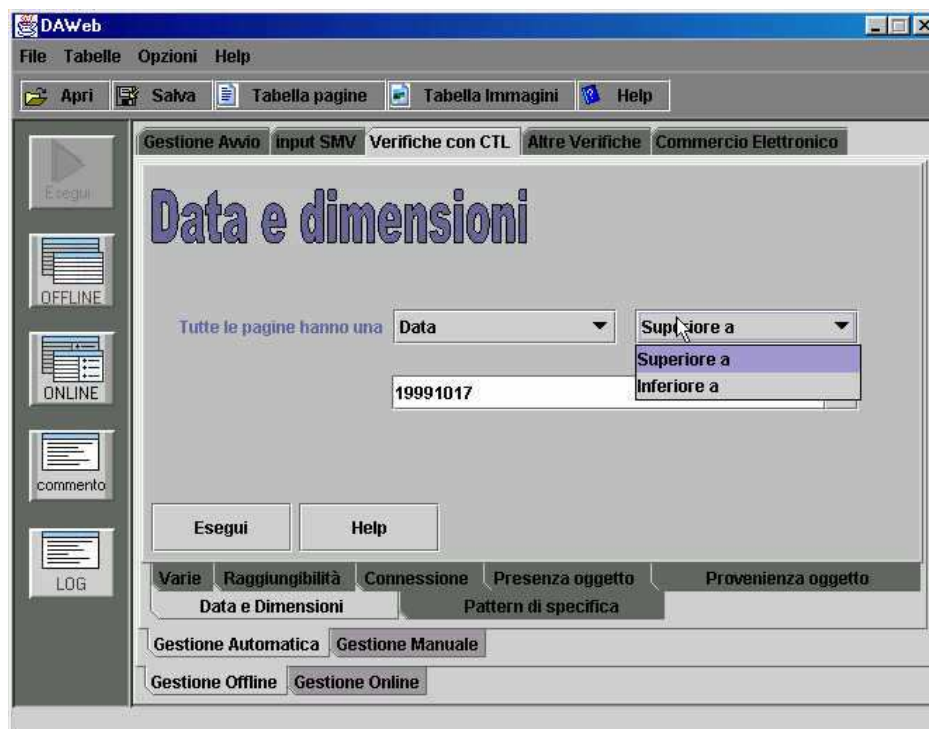


Figura 4–16: Scheda in cui è possibile eseguire le verifiche tenendo conto di date di ultima modifica e dimensioni delle pagine

4.9.8 Varie verifiche (su *SMV online*)

Utilizzando adesso l'*SMV online*, è possibile eseguire alcune verifiche già fatte utilizzando l'*SMV offline*, dove in questo caso però, si impone anche il vincolo dei parametri (che spesso è la password dell'utente) come mostrato in figura 4-17.

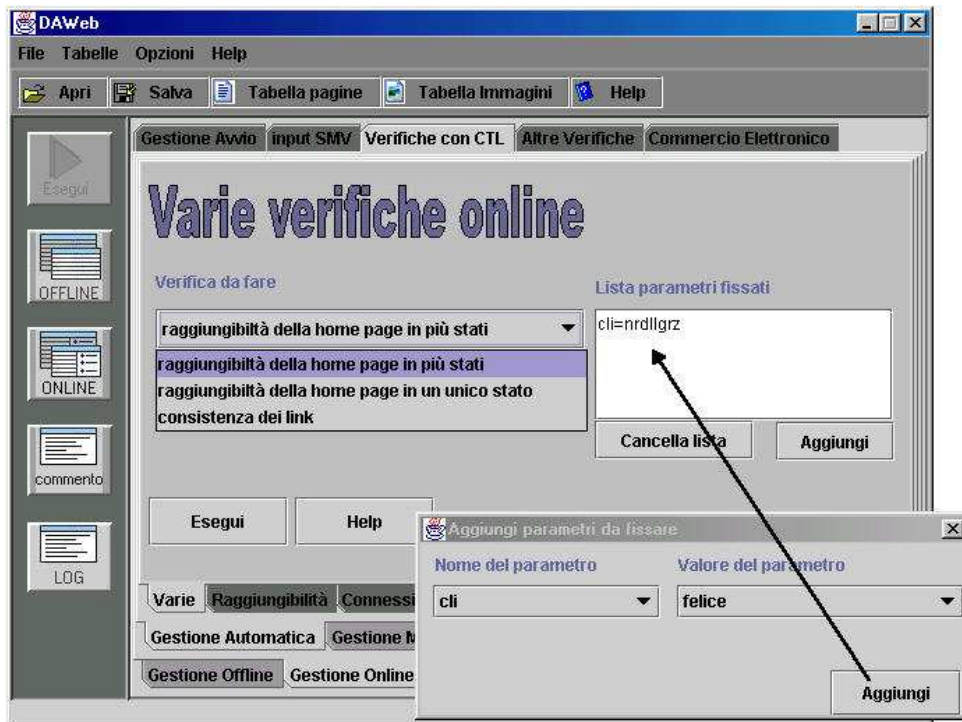


Figura 4–17: Varie verifiche fatte con l'imposizione di un parametro

Fissando alcuni parametri, in questa scheda si dà la possibilità di far eseguire le verifiche, su una restrizione del modello a stati finiti del sito i cui stati verificano i parametri fissati.

Quindi per esempio, fissando una password è possibile eseguire la verifica scelta solo alle pagine riferite a quel determinato utente che presentano tale password.

Le varie verifiche che possono essere eseguite in questo caso sono le seguenti:

- Raggiungibilità della home page in più stati
- Raggiungibilità della home page in un unico stato
- Consistenza dei link (in cui ingloba action e ancore)

4.9.9 Verifica di Raggiungibilità delle pagine (su *SMV online*)

Utilizzando il codice *SMV online* è possibile eseguire la verifica di raggiungibilità delle pagine basandosi anche sulla variazione dei parametri.

Una volta fissato il parametro nel modo indicato dalla figura 4-18, viene verificato se la pagina di partenza è stata determinata in base a tale parametro e una volta verificato questo si verifica se il percorso eseguito verso la pagina di destinazione, presenta sempre lo stesso parametro.

Tutto questo permette di verificare per esempio se è possibile che un utente con una determinata password (che in questo esempio rappresenta il parametro fissato) riesce a entrare nelle pagine accessibili con un'altra password (che sono le pagine che solo un altro utente può raggiungere).

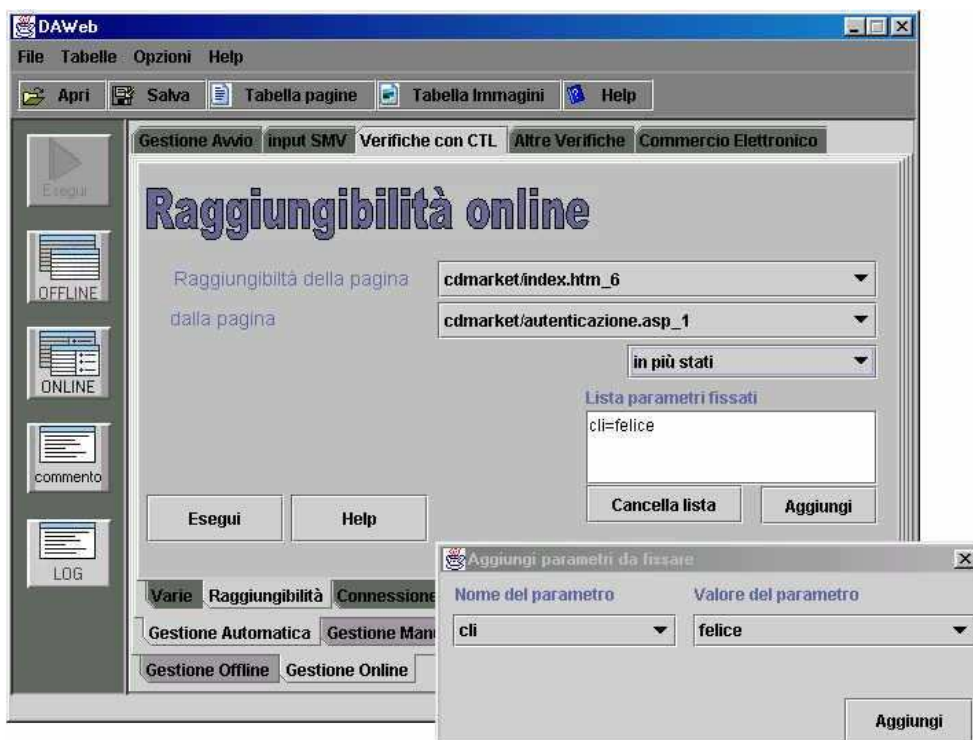


Figura 4-18: Verifica di raggiungibilità fatta con l'imposizione di un parametro

4.9.10 Connessione di una determinata pagina (su *SMV online*)

Questo tipo di verifica è la stessa di quella fatta in precedenza utilizzando il codice *SMV offline* però in questo caso si fa riferimento alla restrizione delle pagine ottenute in base ai parametri fissati, durante il parsing online. In tal caso è semplice procedere in questa verifica ed è possibile farla, entrando nella scheda Connessione della Gestione online del form principale. Questa scheda è illustrata nella figura 4-19.

Ricordiamo che per connessione si intende la verifica simultanea della raggiungibilità di una determinata pagina dalla pagina *index* e del ritorno da questa pagina di nuovo alla pagina *index*. Questo ritorno, inoltre, può essere fatto o direttamente oppure attraverso altre pagine.

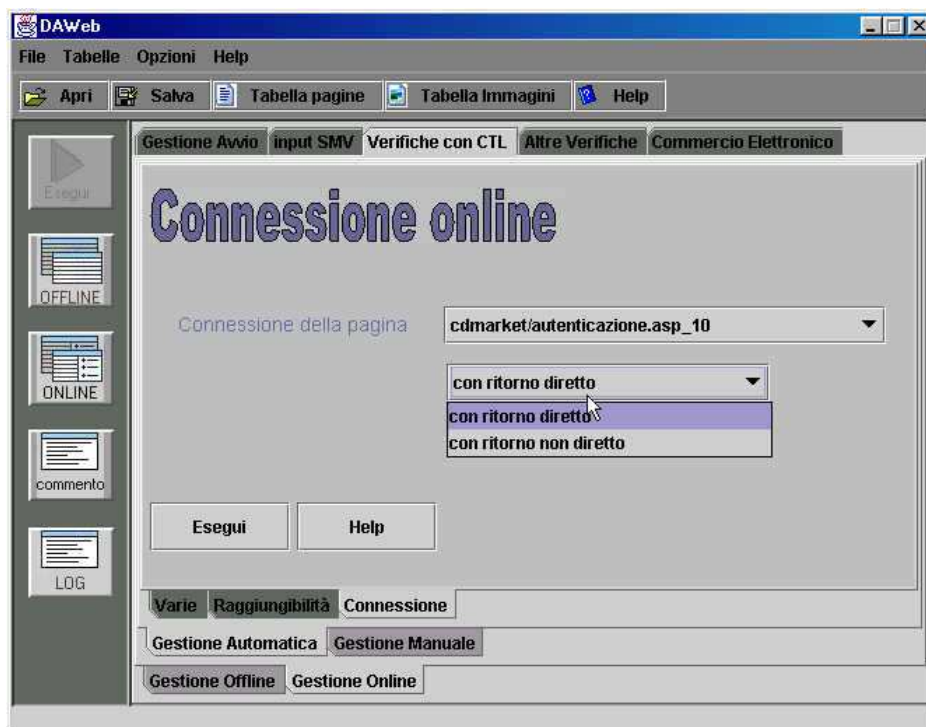


Figura 4–19: Verifica di connessione fatta basandosi sul codice *SMV online*

4.9.11 Gestione manuale

Sono state viste finora come alcune specifiche possono essere eseguite automaticamente, attraverso l'impostazione di opportune voci prese da delle liste a comparsa (*ComboBox*) presenti nelle varie schede.

Oltre questo è possibile eseguire anche delle verifiche basandosi su un codice CTL scritto manualmente. Per scrivere questo codice, bisogna però conoscere bene il modello SMV realizzato per descrivere il sito e la sintassi del codice CTL.

In questo caso è necessario anche capire se bisogna utilizzare il codice *SMV online* o il codice *SMV offline*. Per scrivere manualmente il codice CTL, per verifiche basate su codice *SMV offline*, si deve entrare nella scheda visualizzata in figura 4-20, mentre per scrivere manualmente il codice CTL, per verifiche basate su codice *SMV online*, bisogna invece entrare nella scheda di figura 4-21. In entrambe le schede è possibile utilizzare la tabella dei connettivi utile per comporre le formule CTL.

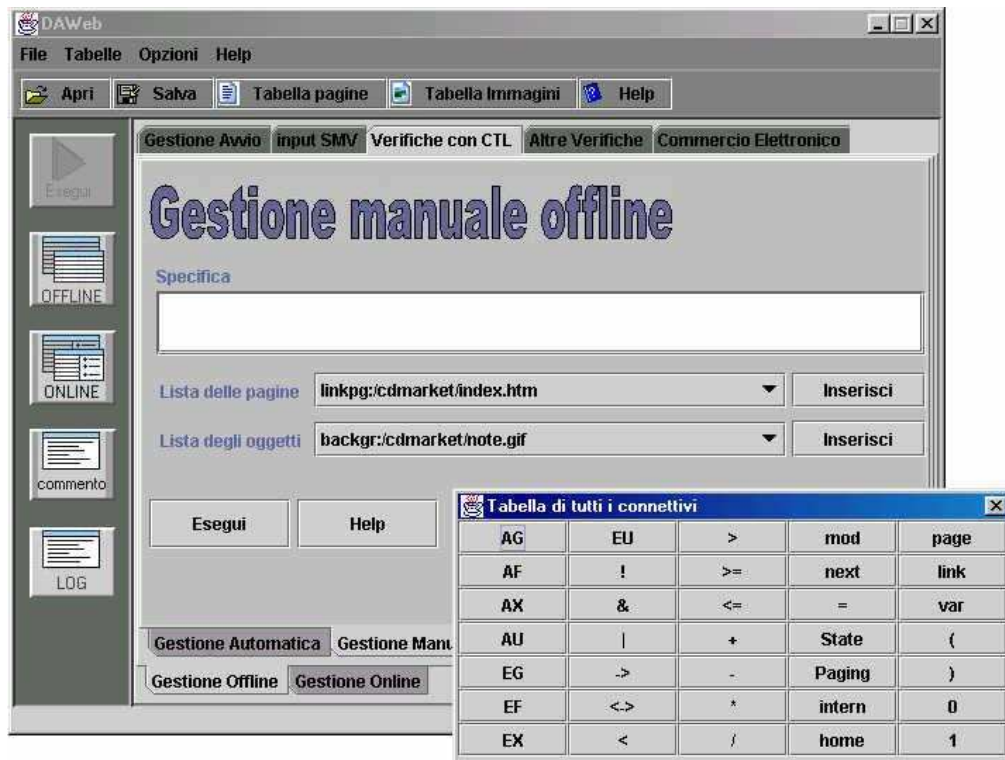


Figura 4-20: Gestione manuale offline



Figura 4–21: Gestione manuale online

4.10 Come vengono visualizzati i risultati delle verifiche CTL

Cliccando sul tasto *esegui* presente nelle varie schede mostrate nel paragrafo precedente, viene visualizzato un form all'interno del quale appare il risultato determinato dal programma SMV . Questo form è composto da due schede. Nella prima è possibile visualizzare l'output generato dal SMV, mentre nella seconda scheda è possibile valutare l'input al SMV che ha determinato tale l'output. Le due schede di questo form sono visualizzati rispettivamente in figura 4-22 e in figura 4-23.

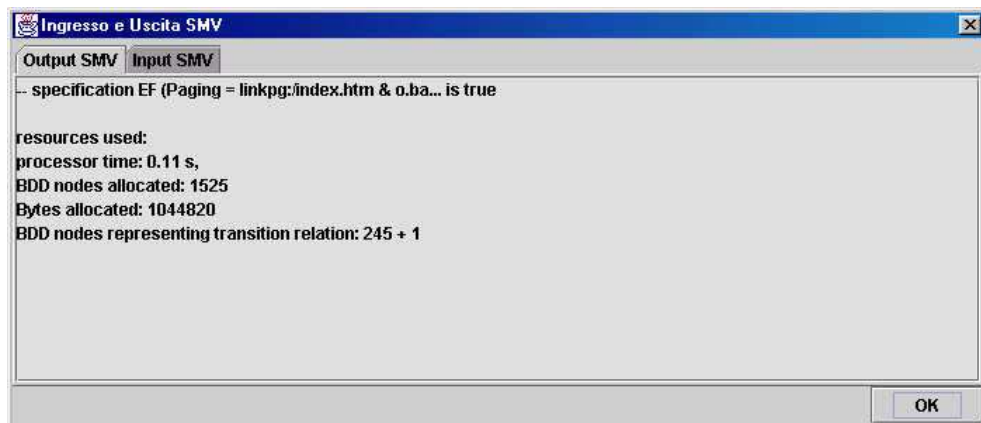


Figura 4–22: Output generato dall' SMV

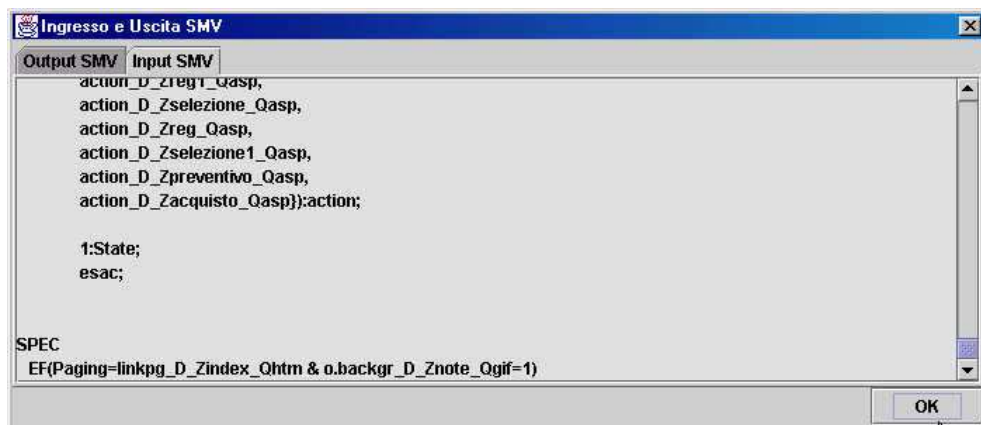


Figura 4–23: Input inviato all' SMV che ha determinato l'output visualizzato nella prima scheda

4.11 Altre Verifiche

DaWeb permette inoltre di eseguire altre verifiche che non si basano sul codice SMV ma si basano sui dati ottenuti direttamente dal parsing online e dal parsing offline.

4.11.1 File e URL assenti e presenti

Dalla scheda illustrata in figura 4-24 è possibile verificare se tutti i file richiamati dalle pagine del sito sono effettivamente presenti nella directory virtuale; inoltre se viene richiamato un URL esterno, viene verificato se tale URL esiste. Tutto questo viene visualizzato in una lista e grazie all'aiuto dei colori, viene eseguita tale verifica. Infatti vengono colorate con il verde tutte le pagine utilizzate dal sito verificato, presenti nella propria directory virtuale, vengono colorate con il blu tutte le URL richiamate dall'interno delle pagine del sito effettivamente presenti in rete, mentre vengono colorate con il rosso tutte le pagine interne o le URL non presenti rispettivamente nella directory virtuale e in rete.

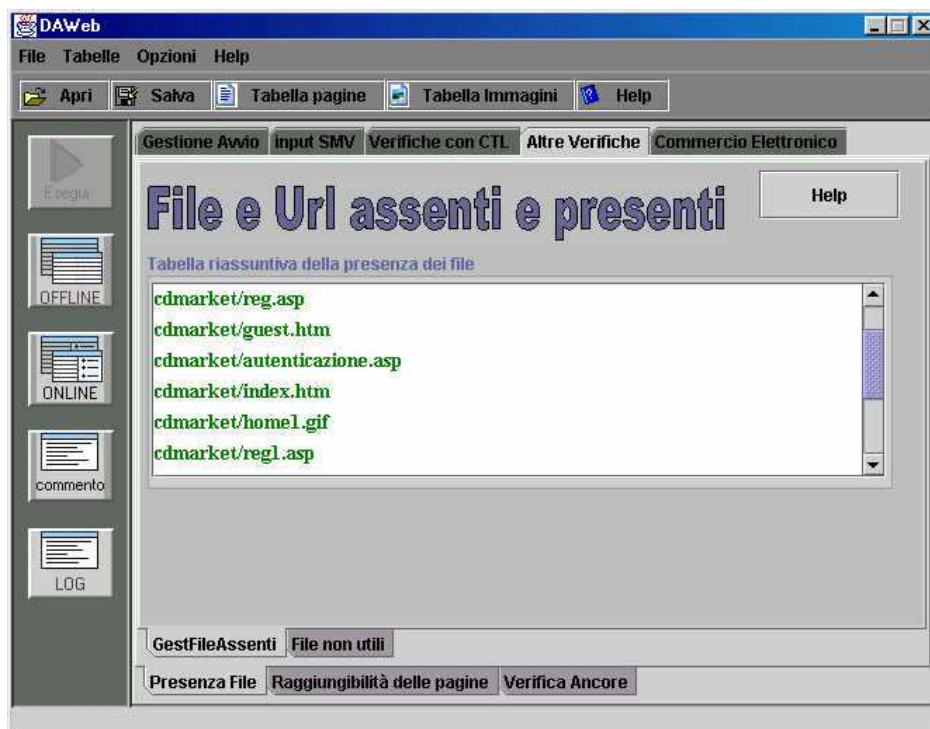


Figura 4-24: Verifica della presenza dei file all'interno della directory virtuale e verifica dell'esistenza delle pagine esterne

4.11.2 File non utili

In questa scheda è possibile verificare se alcuni file presenti nella directory virtuale non vengono per niente utilizzati dal sito e quindi all'interno di tale directory occupano solo spazio. Questa verifica viene eseguita utilizzando come nel caso precedente una lista dove, in questo caso, sono presenti tutti i file della directory virtuale e sempre grazie all'aiuto dei colori è possibile valutare l'esito. Infatti vengono indicate con il verde i file presenti nella directory virtuale effettivamente utilizzati dal sito, con il rosso quei file che sono presenti nella directory virtuale che non vengono utilizzati dal sito e che quindi occupano solo spazio e infine con il nero quei file presenti sempre nella directory virtuale che non possono essere determinati nel processo di parsing, quindi non è possibile sapere se tali file sono utili o meno. Tutto questo è possibile visualizzarlo nella figura 4-25.

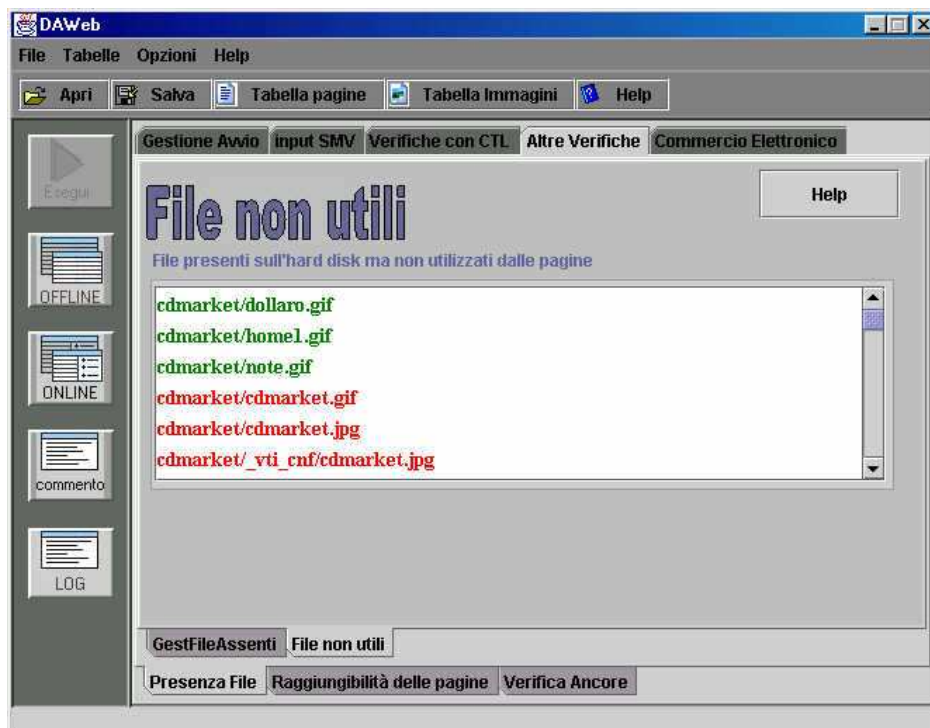


Figura 4-25: File non utili presenti nella directory virtuale

4.11.3 Raggiungibilità delle pagine in modo globale

Un'altra verifica che è possibile fare è quella della raggiungibilità globale delle pagine in base ai parametri inseriti. I parametri inseriti, infatti, nella verifica online possono essere tali da non permettere di poter entrare in alcune pagine del sito.

Questa verifica è possibile farla prendendo in considerazione il concetto che nel parsing online si ottiene una restrizione della pagine del parsing offline e questa restrizione dipende dai parametri.

La scheda utilizzata per eseguire questa verifica è presente nella figura seguente.



Figura 4–26: Lista delle pagine non raggiungibili in base ai parametri

4.11.4 Contenuto delle pagine online rispetto a quelle offline

Sempre in questo ambito un'altra verifica che è possibile fare si basa nel valutare per ogni pagina, se i parametri inseriti possono nascondere alcuni link o form all'interno di una pagina dinamica. Tutto questo viene visualizzato in una tabella e sempre grazie ai colori viene valutata la verifica.

All'interno della scheda, da delle liste a comparsa è possibile scegliere la pagina che si vuole verificare. Una volta scelta la pagina, nella tabella presente in seguito, vengono visualizzate righe di colore verde, se queste corrispondono a link effettivamente presenti nella pagina dinamica scelta, mentre vengono visualizzate righe di colore rosso, se queste corrispondono a link che, a causa di qualche script lato server, invece non compaiono (tale link infatti è stato determinato nel parsing offline).

La scheda che permette di fare tutto questo è visualizzato in figura 4-27.



Figura 4-27: Lista dei contenuti di ogni pagina dinamica

4.11.5 Verifica delle ancore

Oltre ad eseguire la verifica delle ancore, fatte con il codice CTL sul codice SMV, viene eseguita in maniera indipendente un'altra verifica delle ancore che permette di determinare un risultato più chiaro. Infatti mentre con il codice CTL si può sapere o meno se sono state verificate le ancore in maniera complessiva (consistenza delle ancore) in questo caso si può fare una valutazione sulla singola ancora.

Per valutare questo, anche in questo caso viene utilizzata una tabella nella quale, nell'ipotesi in cui nel sito ci sono ancore, vengono visualizzate con colori differenti, tutti i link con le ancore presenti. Se il link con l'ancora è di colore verde significa che il collegamento è eseguito correttamente, se è di colore rosso significa che il collegamento presenta qualche errore.

La scheda in cui è possibile eseguire questa verifica è presente nella figura 4-28.

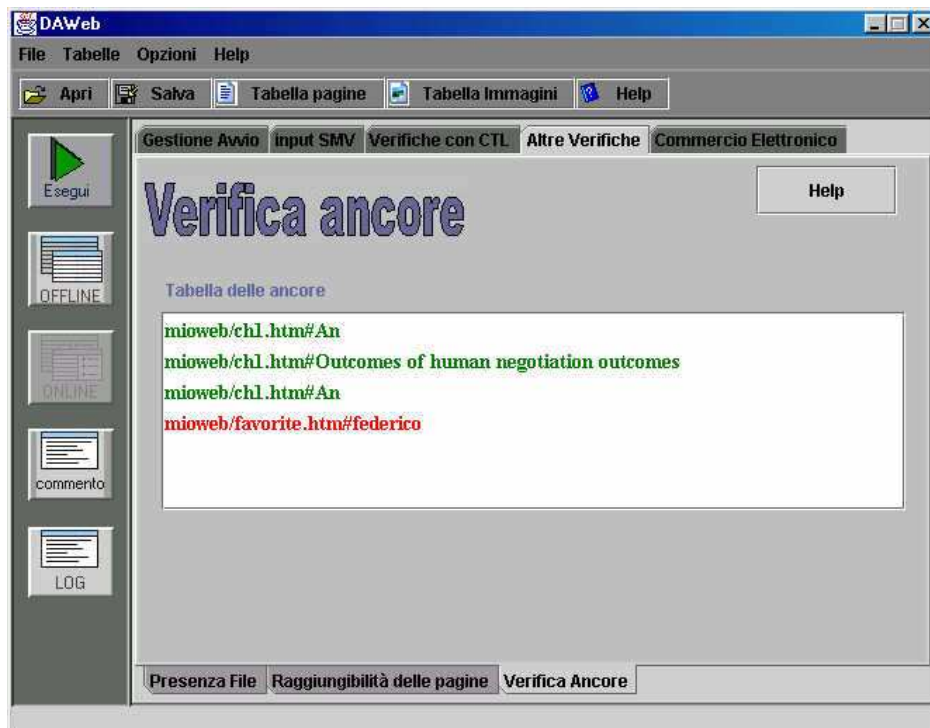


Figura 4-28: Tabella che valuta le ancore nel sito

4.12 Commercio Elettronico

In *DaWeb* è presente una scheda che si chiama *Commercio Elettronico*. In questa scheda è possibile fare delle verifiche che possono essere utili per un sito basato appunto sul commercio elettronico; verifiche che, se non rispettate, deducono che il progetto di un sito basato sul commercio elettronico non rispetta alcune funzionalità importanti per un buon utilizzo.

Sarà illustrato adesso tutto ciò che si trova nella scheda *Commercio elettronico* evidenziando il modo di procedere e come devono essere utilizzati i risultati ottenuti.

4.12.1 Associazioni

Prima di procedere alle verifiche sulle proprietà di un sito del commercio elettronico con *DaWeb*, bisogna fornire alcune informazioni aggiuntive, utili ad identificare alcuni stati fondamentali e alcune variabili utilizzati nel sito.

Per fare questo nella scheda delle associazioni, sono presenti altre due schede che sono: *Associazioni Pagine* e *Associazioni Password*.

4.12.1.1 Associazioni Pagine

La scheda Associazioni Pagine è quella presente in figura 4-29.

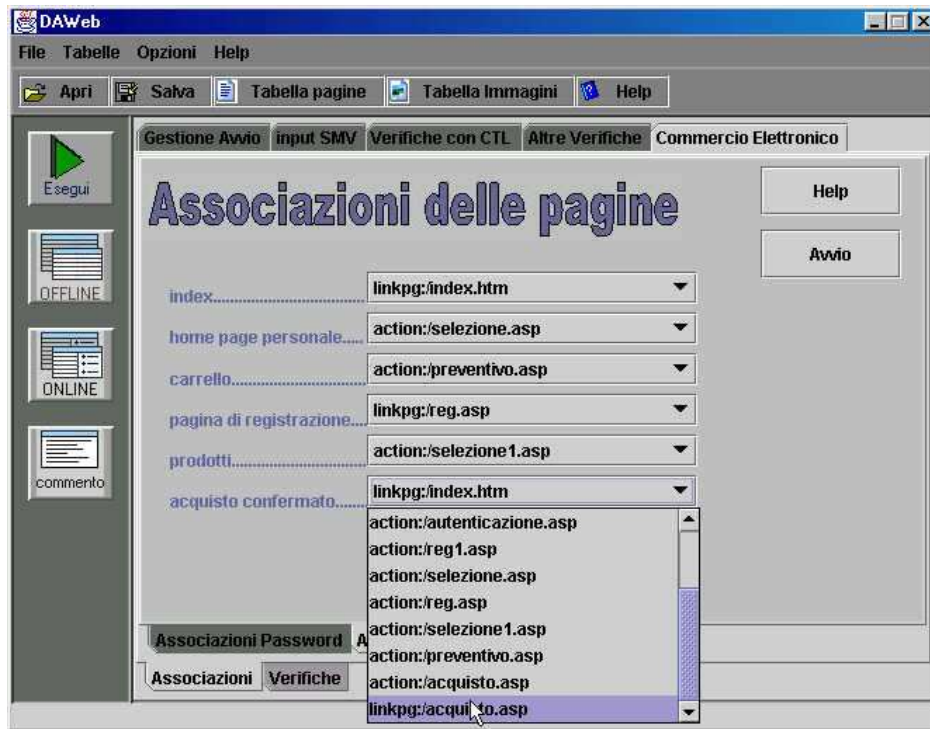


Figura 4-29: Scheda in cui è possibile fare le associazioni delle pagine di un sito di commercio elettronico

Dalla figura si nota chiaramente in che modo fare le associazioni delle pagine e il suo significato.

Questo procedimento risulta ovvio farlo perché ogni sito basato sul commercio elettronico rinomina i file utilizzati (che rappresentano le pagine) in maniera del tutto arbitraria, quindi *DaWeb* non riconosce lo stato, se queste informazioni non vengono date prima dell'inizio delle verifiche. Proprio per questo, solo dopo avere fatto tutte le associazioni, è possibile, ed ha senso eseguire le verifiche offline sul commercio elettronico.

4.12.1.2 Associazioni Password

Tra le verifiche possibili in questa scheda, ci sono alcune che richiedono informazioni sul nome del parametro che rappresenta la password e i valori di tutte (o alcune) le password inserite durante il parsing online.

Durante il parsing online, visto che chi procede alle verifiche del sistema ha accesso a tutte le password, non ha problemi nell'inserire tutte quelle che ritiene opportune (sia valide che non valide).

La scheda Associazioni Password è quella presente in figura 4-30.



Figura 4–30: Scheda in cui è possibile fare le associazioni delle password di un sito del commercio elettronico

4.12.2 Verifiche

Dopo avere eseguito le associazioni è possibile eseguire le verifiche. Queste verifiche si dividono in due parti: *Verifiche Offline* e *Verifiche sulle password*. Mentre nel primo caso si verificano i collegamenti delle pagine all'interno del sito, nel secondo caso si verifica se l'accesso ad alcune pagine avviene solo con le password consentite (cioè con le password inserite nella scheda associazioni delle password).

4.12.2.1 Verifiche offline

La scheda che permette di eseguire queste verifiche è illustrata in figura 4-31

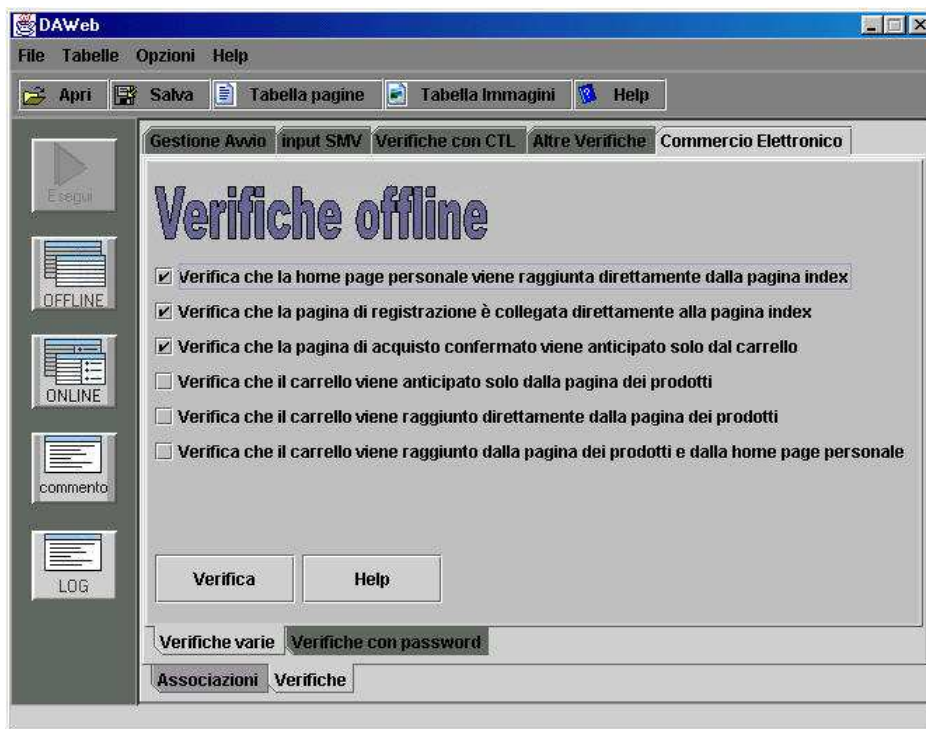


Figura 4–31: Altre verifiche offline utili per siti sul commercio elettronico

Le verifiche presenti in questa scheda ovviamente riguardano il commercio elettronico sono:

- Verifica che l'home page personale viene raggiunta direttamente dalla pagina *index*

- Verifica che la pagina di registrazione sia collegata direttamente alla pagina *index*
- Verifica che la pagina di acquisto confermato venga anticipato solo dal carrello
- Verifica che il carrello venga anticipato solo dalla pagina dei prodotti
- Verifica che il carrello venga raggiunto direttamente dalla pagina dei prodotti
- Verifica che il carrello venga raggiunto direttamente dalla pagina dei prodotti e dalla home page personale

Queste verifiche si basano sul codice *SMV offline* e utilizzano alcune specifiche CTL prese dai pattern di specifica.

4.12.2.2 Verifiche con le password

La scheda che permette di eseguire queste verifiche è illustrata in figura 4-32



Figura 4-32: Altre verifiche online utili per i siti basati sul commercio elettronico

Le verifiche presenti in questa scheda, ovviamente riguardanti il commercio elettronico, sono:

- Verifica che la home page viene raggiunta solo da chi è entrato con una delle password presenti nella lista della scheda *associazioni password*.
- Verifica che il carrello viene visualizzato solo dal chi è entrato con una delle password presenti nella lista della scheda *associazioni password*.

4.13 Help online

In tutte le schede presenti in *DaWeb*, è presente il tasto *Help*. Cliccando su uno di questi tasti, si apre *l'acrobat reader* e viene visualizzata la documentazione relativa alla scheda da cui si è cliccato.

Tutto questo funziona nel momento in cui è installato nel sistema *l'acrobat reader*. In realtà è possibile utilizzare un qualsiasi software che legge i file *pdf*.

La posizione di questo software sarà indicata all'interno di *DaWeb* utilizzando il form che è possibile aprire cliccando nel menù *opzioni*, sulla voce *acrobat reader*.

Il form viene visualizzato in figura 4-33

Da questo form è possibile installare, nell'ipotesi in cui non lo fosse già, *l'acrobat reader 5* (quest'ultima funzionalità va bene solo nei sistemi windows).

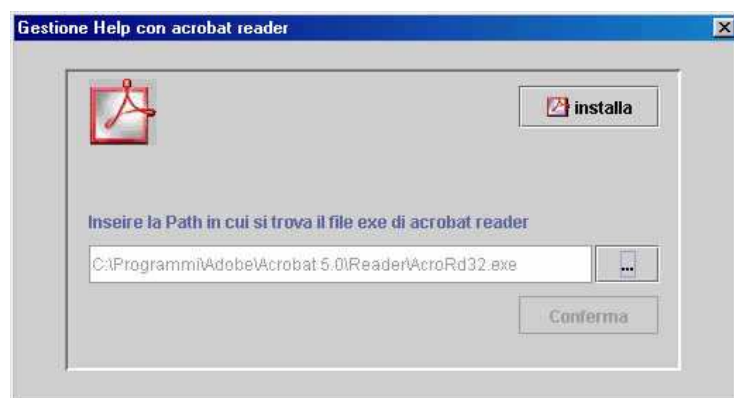


Figura 4–33: Form utile per indicare la posizione del software che legge i file *pdf*, e permette inoltre di installare *acrobat reader* (nei sistemi windows)

4.14 Modifica ed esecuzione dell'ultimo Codice SMV

In *DaWeb* inoltre è possibile, oltre a gestire manualmente la specifica scritta in CTL, è possibile anche modificare il codice SMV.

Tutto questo risulta utile per chi conosce la sintassi del codice SMV a idealizzare alcuni casi non previsti da *DaWeb*.

Tutto questo si può fare grazie alla presenza di un form che è possibile aprire cliccando sul tasto *Gestione ultimo SMV* presente nel menù *opzioni*. Questo form è possibile visualizzarlo in figura 4-34.

All'interno del form sarà visualizzato il Codice SMV che *DaWeb* ha utilizzato quando si è eseguita l'ultima verifica.

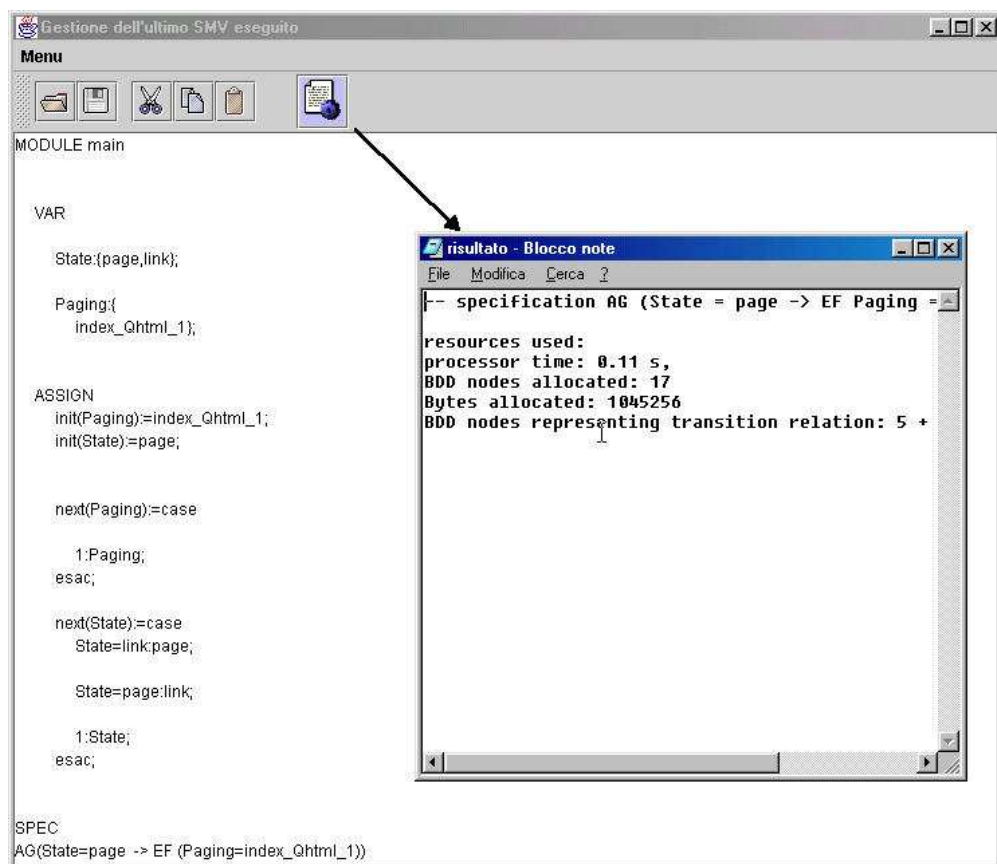


Figura 4-34: Form utile per modificare ed eseguire l'ultimo SMV eseguito da *DaWeb*

Capitolo 5

ESEMPI DI VERIFICHE APPLICATI AL SITO *CDMarket*

5.1 Introduzione

CDMarket è un piccolo sito che gestisce il commercio di CD audio, appartiene quindi alla categoria dei siti basati sul commercio elettronico. Con questo sito si possono eseguire tutte le funzionalità di *DaWeb* cioè in poche parole è possibile eseguire le varie verifiche offline, online, oltre alle verifiche fatte sul commercio elettronico e le altre verifiche eseguite senza l'utilizzo del codice SMV.

5.2 Descrizione di CDMarket

Dalla pagina *index* di *CDMarket* è possibile andare alla pagina *registrazione* da cui è possibile registrarsi, alla pagina *visitatore* da dove è possibile visionare solo i prodotti e alla pagina di *autenticazione password* da dove è possibile inserire la password utile per entrare nella home page personale mostrata nella figura seguente.

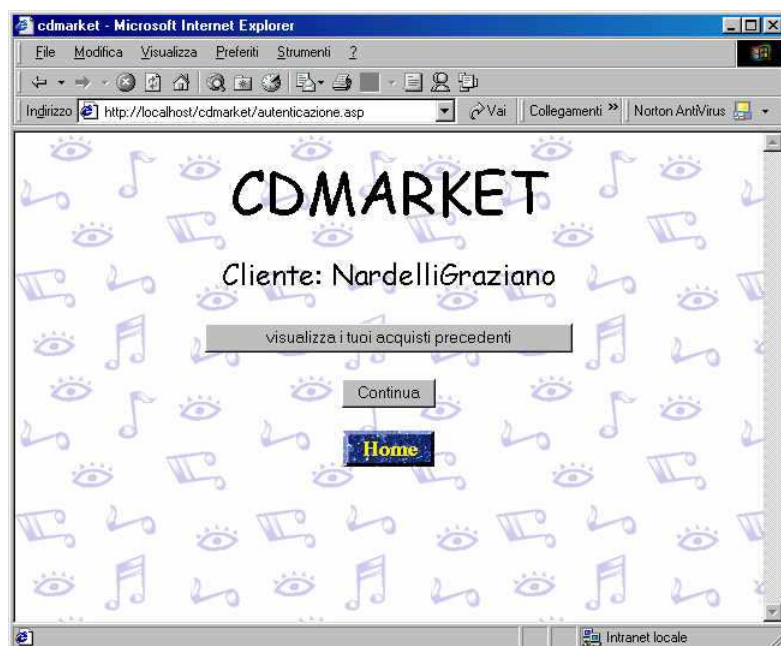


Figura 5-1 : Pagina di autenticazione di CDMarket

Cliccando sul tasto continua, è possibile entrare nella pagina *selezione.asp* (presente nella figura seguente) dalla quale è possibile scegliere i cd che si vogliono, poi, acquistare.

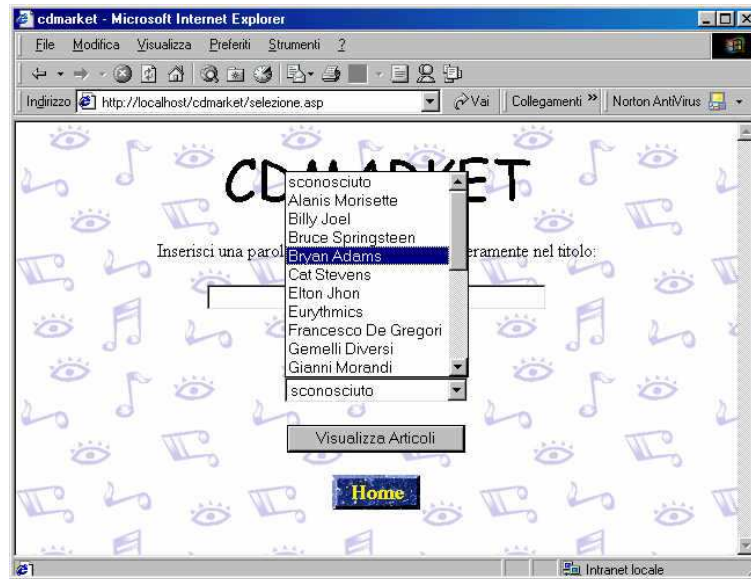


Figura 5-2 : Pagina selezione.asp do CDMarket

Una volta scelto il cd (o i cd appartenenti al cantante selezionato) cliccando sul tasto *Visualizza Articoli* è possibile entrare nella pagina in cui è presente una tabella che riassume le caratteristiche dell'articolo tra cui il nome del cd, l'autore e il prezzo di listino. Dalla stessa pagina è poi possibile inserire il valore numerico che identifica la quantità dei cd che si vogliono acquistare (figura 5-3).

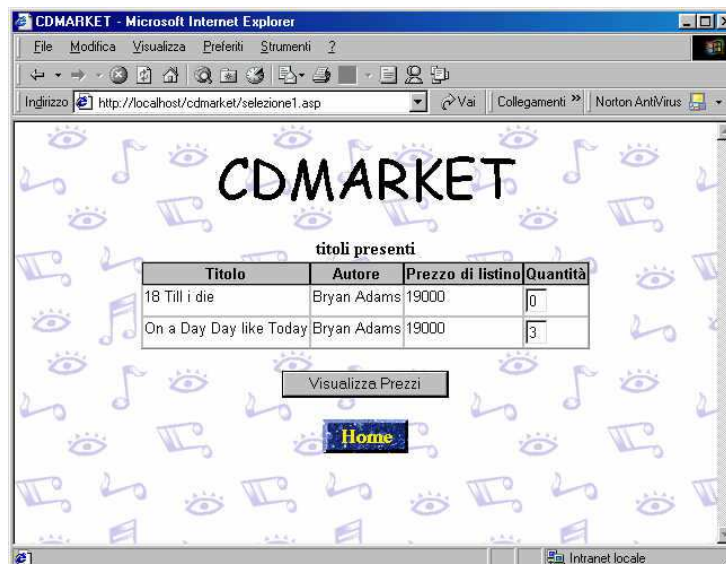
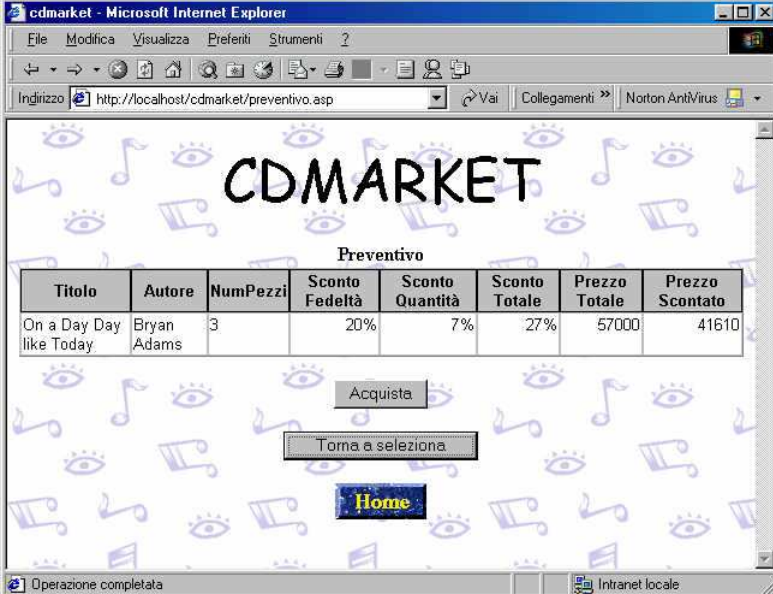


Figura 5-3: Pagina selezione1.asp di CDMarket da cui è possibile scegliere la quantità dei prodotti da acquistare

A questo punto, una volta scelto le quantità dei cd che si vogliono acquistare, è possibile, cliccando sul tasto *Visualizza Prezzi* entrare nella pagina *preventivo.asp* (che rappresenta il carrello). In questa pagina è presente la tabella in cui sono elencati gli articoli che sono stati inseriti contenente inoltre i valori della quantità dei pezzi, lo sconto di qualità, lo sconto fedeltà, lo sconto totale, il prezzo totale e il prezzo scontato (figura 5-4).



The screenshot shows a web browser window titled "cdmarket - Microsoft Internet Explorer". The address bar displays "http://localhost/cdmarket/preventivo.asp". The main content area has a decorative background with musical notes and eyes. At the top, the word "CDMARKET" is written in large letters. Below it, the word "Preventivo" is centered. A table lists the items in the cart:

Titolo	Autore	NumPezzi	Sconto Fedeltà	Sconto Quantità	Sconto Totale	Prezzo Totale	Prezzo Scontato
On a Day Day like Today	Bryan Adams	3	20%	7%	27%	57000	41610

Below the table, there are three buttons: "Acquista", "Torna a selezione", and "Home". The status bar at the bottom indicates "Operazione completata" and "Intranet locale".

Figura 5-4: Pagina che visualizza il preventivo di CDMarket (che ha le funzioni anche di carrello) da cui è possibile confermare l'acquisto

Cliccando sul tasto *acquista* è possibile entrare nella pagina che conferma l'acquisto. In questa nuova pagina volontariamente si è eseguito un solo link verso se stesso. Questo è un errore che non permette poi di poter ritornare indietro verso altre pagine, di conseguenza si rimane bloccati. Questo tipo di errore è stato inserito per vedere come *DaWeb* lo identifica nelle sue verifiche.

5.3 Verifica di CDMarket con DaWeb

Prima di iniziare le verifiche con *DaWeb*, bisogna eseguire il parsing del sito. Per eseguire tutte le verifiche sopra citate, è necessario eseguire sia il parsing offline che il parsing online. Visto che si deve eseguire anche quest'ultimo parsing, il

procedimento, come detto nei capitoli precedenti, non è immediato, infatti, durante, si devono compilare le varie schermate che si presentano in corrispondenza dei form all'interno della pagina che in quel momento si sta esaminando.

Per l'esempio CDMarket, inizialmente, è stato determinato un parsing online, senza inserire nelle schermate che si presentano alcuna informazione. Può capitare, soprattutto in questo caso, di non poter accedere ad alcune pagine del sito, infatti è stato detto in precedenza che il parsing online è una restrizione del parsing offline, quindi può contenere un numero inferiori di stati.

5.3.1 Varie verifiche senza CTL fatte per l'esempio CDMARKET

Dopo aver fatto il parsing di *CDMarket* sia offline che online (non inserendo nessun informazione nei parametri) saranno eseguite inizialmente alcune delle varie verifiche eseguite senza l'utilizzo del SMV e le specifiche CTL (cioè saranno eseguite le verifiche presenti all'interno della scheda *Altre Verifiche*). Precisamente sarà eseguita la *raggiungibilità globale* e alcuni casi della *raggiungibilità relativa*.



Figura 5-5: Pagine non raggiungibili nell'esempio di CDMARKET, nel caso in cui non venga inserito nessun parametro nel parsing online

Nella figura 5-5 è visualizzata la schermata in cui è possibile eseguire la *Raggiungibilità globale*. In questa schermata è presente una lista in cui sono presenti le seguenti pagine:

- *CDMarket/preventivo.asp*
- *CDMarket/acquisto.asp*

Queste due pagine non possono essere raggiunte se si è scelto di non inserire i parametri. Questo risultato è logico che si presenti, in quanto, non inserendo i parametri, non è possibile entrare in una delle tante home page personali. Infatti per entrarci occorre inserire, durante il parsing online, la password del cliente proprietario della home page (in questo caso è possibile entrare solo nella home page del visitatore). Non entrando nella home page, non è possibile fare acquisti, quindi non deve essere giustamente possibile entrare nella pagina *preventivo.asp* (che nel nostro esempio rappresenta il carrello) e nella pagina *acquisto.asp* che rappresenta quella pagina da dove è possibile confermare l'acquisto.

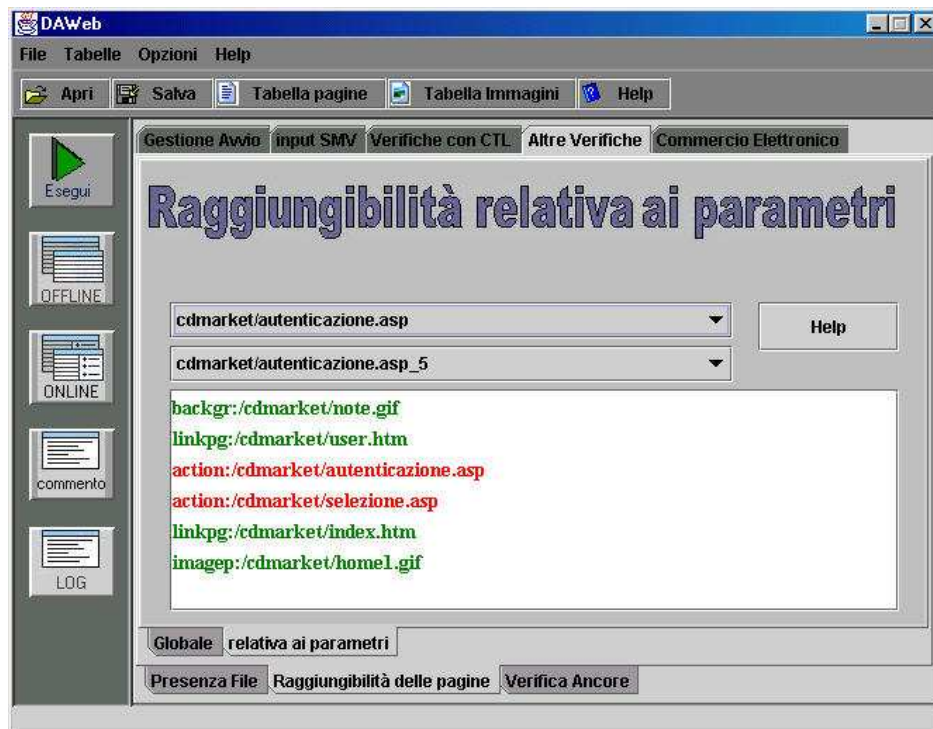


Figura 5-6: Raggiungibilità delle pagine, dalla pagina autenticazione.asp relativa ai parametri nulli inseriti durante il parsing online

Nella *raggiungibilità relativa ai parametri* è possibile, grazie ai colori, capire quali link o oggetti vengono nascosti, in una pagina in relazione ai parametri inseriti durante il parsing online.

In figura 5-6 e nella figura 5-7 vengono visualizzati i risultati di quest'analisi per le pagine *autenticazione.asp* e *reg1.asp*.

Nella verifica della pagina *autenticazione.asp* si nota, nella figura 5-6 che con i parametri inseriti nulli, vengono nascosti i form che richiamano la stessa pagina *autenticazione.asp* e la pagina *selezione.asp*.

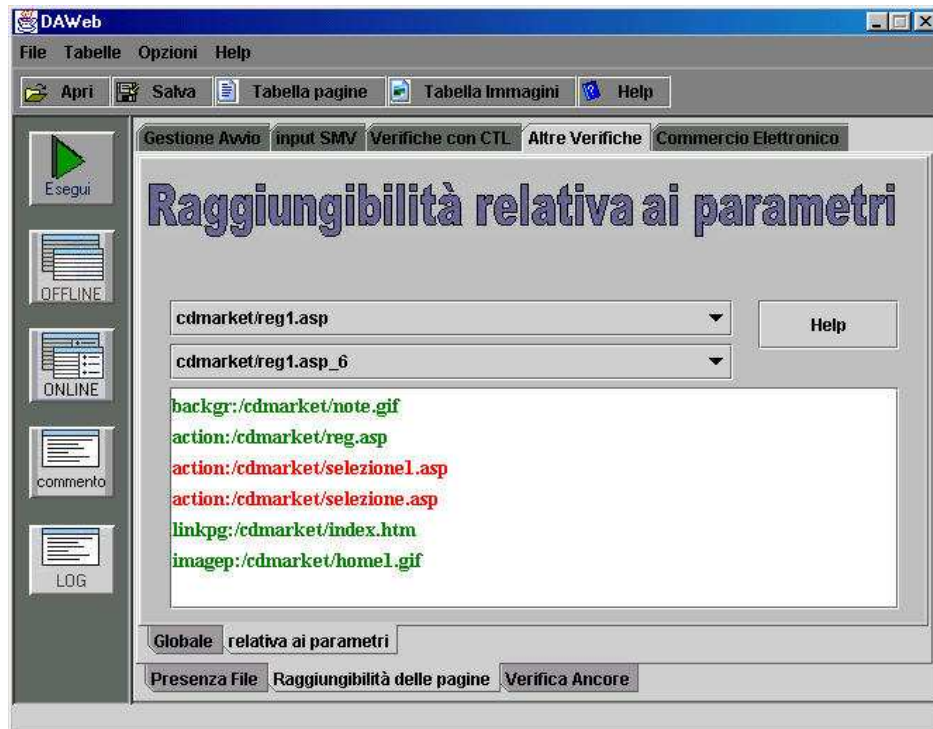


Figura 5-7 : Raggiungibilità delle pagine, dalla pagina *reg1.asp* relativa ai parametri nulli inseriti durante il parsing online

Nella verifica della pagina *reg1.asp* si nota, nella figura 5-7 che con i parametri inseriti nulli, vengono nascosti i form che richiamano la pagina *selezione.asp* e la pagina *selezione1.asp*.

5.3.2 Verifiche Offline fatte per l'esempio CDMarket

Saranno eseguite adesso e anche commentate, le verifiche offline fatte per il sito CDMarket. Prima di fare questo, nella scheda *input SMV*, dopo aver selezionato tutti i *CheckBox*, bisogna cliccare sul tasto *Determina*.

Questo permette di determinare il codice *SMV offline* (e anche online in questo caso) utile per eseguire le verifiche offline, visto che queste si basano proprio su questo codice con l'inserimento in fondo delle specifiche scritte in CTL.

Nella scheda *varie verifiche offline* è possibile verificare:

- *La raggiungibilità della home page in più stati*

Il risultato ottenuto è falso, perché volutamente è stata realizzata la pagina acquisto, senza link. Pertanto, nel momento in cui si entra in tale pagina, non è più possibile (non utilizzando i tasti di navigazione del browser) ritornare alla home page del sito.

- *La consistenza dei link*

Il risultato ottenuto è vero, questo significa che se nell'intero sito, ci sono dei link, questi conducono sempre a pagine esistenti.

- *La consistenza delle ancore*

L'intero sito non contiene ancore, pertanto il risultato ottenuto è vero.

- *La consistenza delle action*

Il risultato ottenuto è vero, questo significa che se nell'intero sito, ci sono dei form, questi conducono sempre a pagine esistenti.

- *Se tutte le pagine sono interne*

Il risultato ottenuto è vero, in effetti non esiste all'interno, nessun link che conduce a un sito esterno.

- *Se tutte le pagine sono statiche*

Il risultato ottenuto è falso, in effetti quasi tutte le pagine del sito sono pagine dinamiche.

Dalla scheda *raggiungibilità* è possibile verificare per esempio se è possibile raggiungere la pagina *reg.asp* dalla pagina *acquisto.asp* in più stati. Il risultato di questa verifica è falso. In effetti dalla pagina *acquisto.asp* non è possibile ritornare indietro; di conseguenza non è possibile raggiungere un'ulteriore pagina qualunque essa sia.

Anche la verifica di *connessione* con ritorno indiretto della pagina *acquisto.asp* dà risultato falso proprio perché per definizione di connessione, tale pagina non è connessa. Infatti una pagina è connessa se dalla home page è possibile raggiungere tale pagina e poi da quest'ultima è possibile ritornare di nuovo alla home page. La pagina *acquisto.asp* non è connessa perché, per come è stato detto anche prima, non è possibile da tale pagina raggiungere una qualsiasi altra pagina.

Entrando nella scheda *presenza oggetto*, è possibile verificare per esempio se lo sfondo *note.gif* è presente in tutte le pagine. Il risultato della verifica è falso perché la pagina *acquisto.asp* presenta come sfondo non l'immagine *note.gif* ma l'immagine *dollaro.gif*. Se poi viene eseguita la verifica che impone la presenza dell'immagine *note.gif* nella pagina *reg.asp* il risultato è vero.

Dalla scheda *provenienza dell'oggetto* è possibile verificare da quale tag del codice html è stato preso un determinato oggetto. Un esempio che è possibile fare permette di vedere se l'immagine *note.gif* è stato preso dal tag *body*. Questa verifica risulta vera.

Dalla scheda *data e dimensioni* è possibile verificare se le pagine del sito sono state aggiornate. Per fare questo, per esempio, è possibile imporre che le date di ultima modifica delle pagine siano anteriori alla data del primo gennaio 2000. Questa verifica risulta falsa in quanto ci sono più pagine che hanno una data di ultima modifica più recente di tale data.

Molte verifiche possibili da fare con i *pattern di specifica*, per questo tipo di sito non hanno senso, in quanto come mostrato in appendice B, alcune verifiche sono

significative se vengono eseguite su siti che hanno determinate caratteristiche. Infatti mentre l'estensione *globale* ha senso per tutti i tipi di siti, le altre (*prima di*, *dopo di*, *tra le* e *dopo finché*) hanno senso solo per i siti, i cui stati possono essere raggruppati in insiemi e questi insiemi possono comunicare tra di loro solo tramite un unico percorso (vedasi appendice B).

Un esempio con estensione *globale* che è possibile eseguire è quello mostrato in figura 5-8. In questo esempio viene verificata la precedenza della pagina *preventivo.asp* alla pagina *acquisto.asp* in maniera globale. Il risultato di questa verifica è vera in quanto sempre la pagina *preventivo.asp* anticipa la pagina *acquisto.asp*. Questa verifica è esprimibile anche nel seguente modo: la pagina *acquisto.asp* è la causa della pagina *preventivo.asp*. Visto che la pagina *preventivo.asp* rappresenta il carrello del sito basato sul commercio elettronico e *acquisto.asp* rappresenta la pagina di conferma acquisto, la precedenza risulta allora ovvia. In effetti, la possibilità di confermare l'acquisto può essere fatta solo dal carrello. Visto che quindi questa verifica risulta vera in tutti i siti basati sul commercio elettronico, sarà ripresa nella scheda *commercio elettronico*.

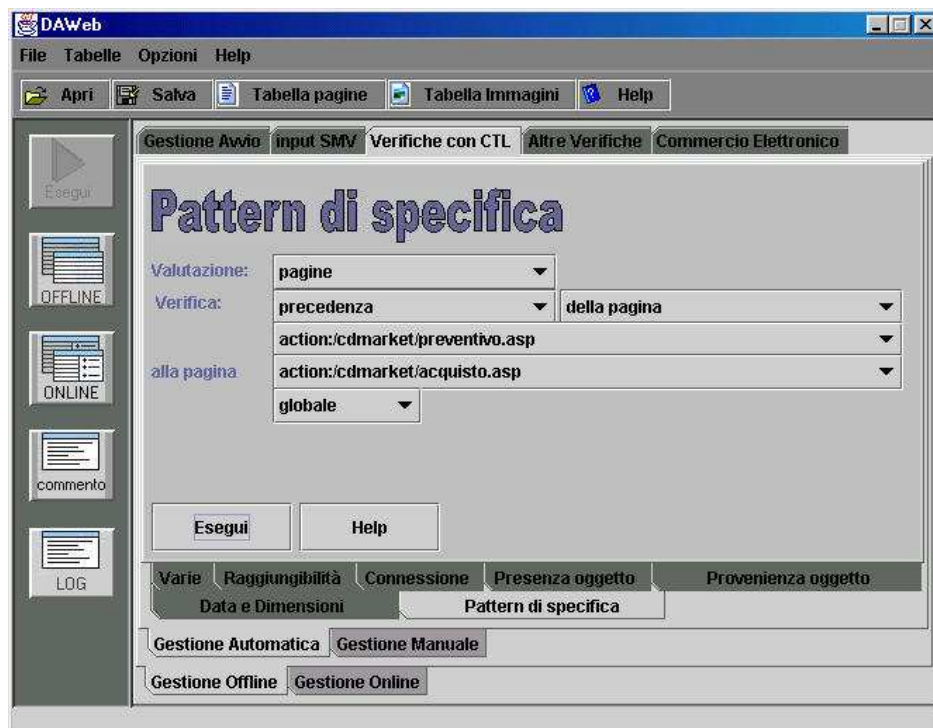


Figura 5-8: Un esempio di utilizzo dei pattern di specifica per il sito CDMarket

5.3.3 Verifiche Online fatte per l'esempio CDMARKET

Saranno eseguite e commentate adesso le verifiche in cui entrano in gioco i parametri. Pertanto occorre utilizzare il codice *SMV online* a sua volta ottenuto dal parsing online.

Per ottenere risultati significativi, occorre che il codice SMV si basi su dati di un parsing online determinato con un'adeguata varietà di parametri inseriti dall'utente. Pertanto, in questo caso, non va bene il parsing online determinato nel corso dell'esempio che è stato utile per eseguire le varie verifiche senza l'utilizzo del codice CTL. Risulta necessario rifare il parsing inserendo in maniera opportuna i parametri nei vari frame che si presentano mano a mano, in modo da ottenere un diagramma a stati finiti simile a quello mostrato nella figura 3-7 con più informazioni possibili.

Come primo esempio saranno eseguite le varie verifiche online, gestite dalla scheda mostrata in figura 5-9

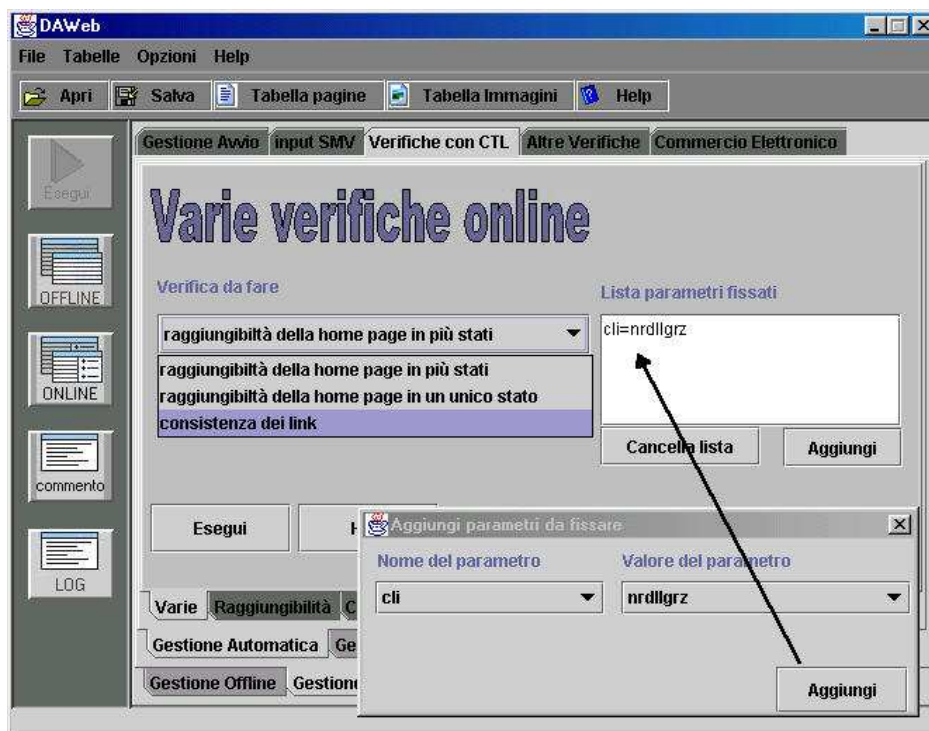


Figura 5-9: Schermata in cui è possibile eseguire le varie verifiche online

Nell'esempio CDMarket, in questa scheda, fissando il parametro *cli=nrdllgrz*, dove cli rappresenta il parametro della password di accesso da parte degli utenti alla propria home page, viene eseguita la verifica selezionata solo per gli stati che verificano *cli=nrdllgrz* cioè per gli stati interni che rappresentano le pagine che solo chi possiede tale password può visualizzare.

La verifica della raggiungibilità della home page in più stati ottenuta partendo dagli stati che verificano *cli=nrdllgrz* da un risultato falso. In effetti anche online, nel momento in cui si accede alla pagina *acquisto.asp*, non è possibile andare più in un'altra pagina non è quindi nemmeno possibile ritornare alla home page.

La verifica della *consistenza dei link* in questo caso è positiva; questo significa che ogni link presente in ogni pagina conduce sempre ad una pagina esistente.



Figura 5–10: Raggiungibilità della pagina selezione1.asp dalla pagina autenticazione.asp avendo fisso il parametro cli=nrdllgrz

Entrando nella scheda *Raggiungibilità online* è possibile verificare, per esempio, se un utente con una determinata password (che in questo esempio rappresenta la password fissata) riesce a entrare nelle pagine che presentano un'altra password (che sono le pagine che solo un altro utente può raggiungere).

In figura 5-10 è possibile notare come viene impostata la verifica della raggiungibilità della pagina *selesione1.asp_19* dalla pagina *autenticazione.asp_10* in più stati avendo fisso il parametro *cli=nrllgrz*. Questa verifica dà come risultato un valore vero. In effetti è possibile notare questo, analizzando la tabella online che compare nel momento in cui si clicca sul tasto online presente nella colonnina a sinistra del form principale di *DaWeb*. Da questa tabella, infatti, si può capire come dalla pagina *autenticazione.asp_10* è possibile raggiungere, sempre avendo come parametro *cli=nrllgrz* la pagina *selezione1.asp_19*.



Figura 5-11: Raggiungibilità della pagina *preventivo.asp* dalla pagina *autenticazione.asp* avendo fisso il parametro *cli=nrllgrz*

In figura 5-11 è possibile notare invece come viene impostata la verifica della raggiungibilità della pagina *preventivo.asp_23* dalla pagina *autenticazione.asp_10* in più stati avendo fisso il parametro *cli=nrdllgrz*. Questa verifica da come risultato un valore falso. È possibile avere conferma di ciò, analizzando sempre la tabella online. Da questa tabella, infatti, si può capire come dalla pagina *autenticazione.asp_10* non è possibile raggiungere, sempre avendo come parametro *cli=nrdllgrz* la pagina *preventivo.asp_23*. Quest'ultima infatti poteva essere raggiungibile se si partiva da una pagina che presentava come parametro *cli=felice* e non *cli=nrdllgrz*.

Entrando nella scheda *Connessione online* è possibile verificare se una determinata pagina è connessa. A differenza della connessione offline analizzata precedentemente che si basava sul codice *SMV offline*, la verifica della connessione online viene eseguita utilizzando il codice *SMV online*; vengono quindi considerati gli stati che dipendono dai parametri inseriti durante il parsing online che risultano strutturati in maniera diversa rispetto agli stati determinati durante il parsing offline. I risultati della connessione in questo caso, possono essere differenti rispetto al caso della connessione offline. Comunque c'è da dire che se nella connessione offline una determinata pagina non era connessa, nella connessione online, tutte le pagine ottenute partendo da tale pagina, a sua volta non sono connesse. È possibile notare ciò valutando la connessione online di tutte le pagine provenienti da *acquisto.asp* nell'esempio di CDMarket. Tali valutazioni portano, infatti, a risultati falsi. Questo significa che le pagine analizzate non sono connesse cioè non viene verificato che dalla home page vengono raggiunte queste pagine e poi da queste sia possibile ritornare di nuovo alla home page.

5.3.4 Verifiche di siti basati sul commercio elettronico per CDMarket

CDMarket è un piccolo esempio di commercio elettronico, quindi deve rispettare alcune caratteristiche importanti. Alcune verifiche di queste caratteristiche si possono fare con *DaWeb* entrando nella scheda *Commercio elettronico*.

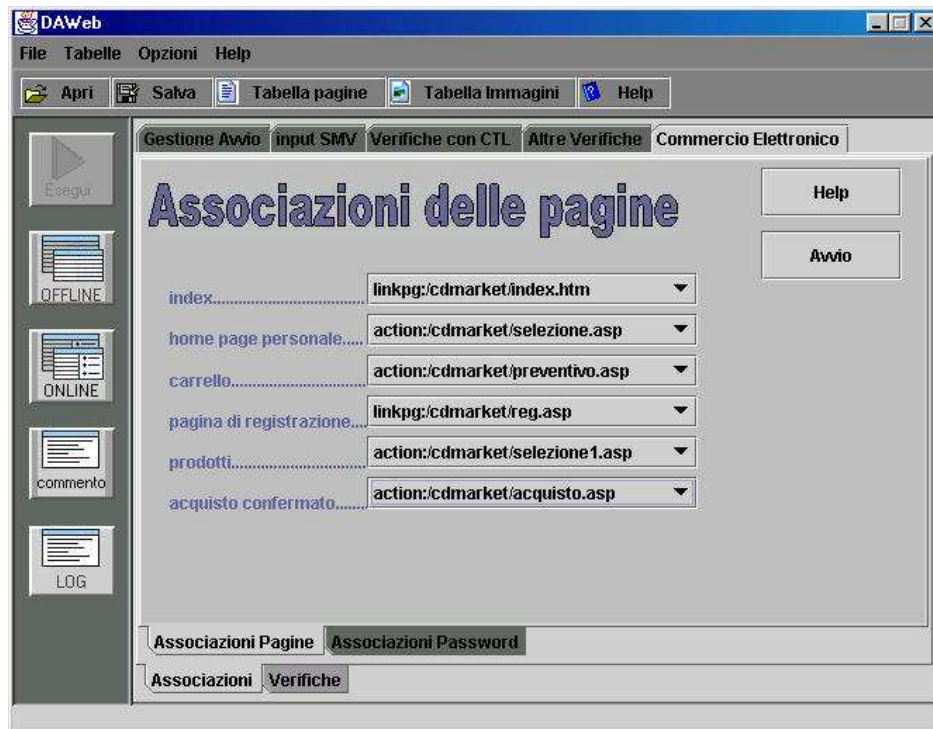


Figura 5–12: Associazioni delle pagine utile per identificare gli stati di CDMarket a DaWeb

Come prima cosa, in questa scheda, bisogna fare le associazioni delle pagine. Queste associazioni vengono fatte per dare la possibilità a *DaWeb* di capire come si chiamano determinati stati, in modo da associarli un comportamento e in base a questo fare poi le verifiche.

L'associazione delle pagine eseguite per *CDMarket* è presente in figura 5-12.

Facendo queste associazioni è possibile già eseguire le verifiche offline presenti nella scheda verifiche del commercio elettronico.

Queste verifiche sono:

- *verifica che l'home page personale venga raggiunta direttamente dalla pagina index*

Questa verifica non è indispensabile per il buon funzionamento di un sito basato sul commercio elettronico, ma è una richiesta che spesso

viene verificata nei siti professionali, per velocizzare, da parte dei clienti, l'accesso alla propria home page personale. Questa verifica risulta falsa per CDMarket, quindi in tale applicazione non si accede direttamente alla home page dalla pagina *index*.

- *Verifica che la pagina di registrazione è collegata direttamente alla pagina index.*

Anche questa non è una richiesta indispensabile per il funzionamento del sito basato sul commercio elettronico, ma è una richiesta che spesso viene verificata nei siti professionali, per facilitare la possibilità di registrazione da parte dei clienti. Questa verifica risulta vera per CDMarket.

- *Verifica che la pagina di acquisto confermato venga anticipata solo dal carrello.*

In un sito basato sul commercio elettronico, è sempre verificato che la pagina di conferma acquisto, deve essere sempre e solo richiamata dal carrello. Questa verifica risulta vera per CDMarket.

- *Verifica che il carrello venga anticipato solo dalla pagina dei prodotti*

È possibile verificare in questo caso, se il carrello viene anticipato solo dalla pagina dei prodotti. Il risultato falso di questa verifica porterebbe a dire che la pagina dei prodotti non anticipa per niente il carrello oppure che, oltre alla pagina dei prodotti, sicuramente è presente un'altra pagina che anticipa sempre lo stesso carrello (per capire quale delle due affermazioni viene verificata occorre eseguire la prossima verifica).

Il risultato della verifica per *CDMarket* è vero. (Le verifiche successive non hanno importanza, in questo caso, visto che già si conoscono i risultati in quanto è evidente che la successiva verifica da risultato vero e l'altra ancora da un risultato falso).

- *Verifica che il carrello venga raggiunto direttamente dalla pagina dei prodotti*

È possibile verificare in questo caso, se la pagina del carrello è anticipata dalla pagina dei prodotti. Questa verifica ha senso farla solo nel caso in cui la verifica precedente risulta falsa, infatti se fosse vera, sarebbe sempre vera anche questa verifica.

Per *CDMarket* come detto nel punto precedente questa verifica è vera.

- *Verifica che il carrello venga raggiunto dalla pagina dei prodotti e dalla home page personale*

Se la verifica precedente è vera e quella precedente ancora è falsa, ha senso eseguire questa verifica, per vedere se oltre alla pagina dei prodotti, visto che c'è sicuramente almeno un'altra pagina che anticipa il carrello, anche la home page personale anticipa lo stesso carrello. Per *CDMarket*, questa verifica, per ciò che è stato detto nei punti precedenti risulta ovviamente falsa.

I risultati ottenuti portano a dire che *CDMarket* verifica alcune caratteristiche fondamentali per un sito basato sul commercio elettronico.

Altre verifiche che si possono fare si basano sull'utilizzo della password. Per farle, occorre prima eseguire le associazioni delle password. In queste associazioni, bisogna capire quale è la variabile che rappresenta la password utente e bisogna selezionare, poi, tutti o alcuni dei valori che sono stati attribuiti a tale variabile durante il parsing online.



Figura 5–13 : In questa scheda si devono inserire le password che permettono di avere un accesso alla home page

In figura 5-13 si nota come vengono eseguite le associazioni delle password. In tal caso si sceglie prima il nome del parametro che rappresenta la password che in *CDMarket* rappresenta *cli*, dopo di che, tra tutti i valori vengono scelte le password: *nrDllgrz*, *felice* e *visitatore*.

Facendo queste associazioni è possibile eseguire la seguente verifica:

Verifica che la home page viene raggiunta con tutte le password inserite (nella lista presente nella scheda: associazioni password).

Il risultato di questa verifica, con le associazione delle password fatta nel modo descritto (v. fig. 5-13), porta ad un risultato vero. Nelle associazioni delle password è stata inserito anche il valore *visitatore*, questo valore non è una password utente ma è la password scelta dal sistema nel momento in cui si vuole entrare nel sito come un semplice visitatore. Anche se poi con la stessa

password, non è possibile entrare nel carrello e quindi non è possibile eseguire l'acquisto dei prodotti (non è possibile entrare quindi nella pagina *conferma acquisto*).

Sempre tenendo in considerazione le password, è possibile anche fare la seguente verifica:

Verifica se la pagina del carrello può essere visualizzata solo da chi ha una delle password inserite (nella lista presente nella scheda: associazioni password).

In tal caso il risultato della verifica è falso, visto che, con le associazioni delle password fatte, un semplice visitatore non può entrare nel carrello. Se si toglie dalla lista delle password il valore *visitatore*, il risultato della verifica diventa vero.

Quindi in CDMarket, non è possibile entrare, per un utente che non abbia la propria password, nella pagina del carrello.

Appendice A

DESCRIZIONE INTERNA DEL PROGETTO

A.1 Introduzione

Saranno descritte adesso le caratteristiche interne del progetto. Tale descrizione consiste nel commentare il funzionamento di ogni classe implementata e nel capire come queste classi sono collegate insieme in modo da determinare il progetto finale.

Tutto questo è facilitato grazie all'utilizzo di alcuni diagrammi che rispettano il linguaggio UML (*Unified Modelling Language*).

A.2 Descrizione delle singole classi

Saranno descritte ora i modi in cui agiscono le singole classi:

- **AcrobatReader.java**

Questa classe istanziata dal frame principale, determina un frame, dalla quale è possibile identificare la posizione di *acrobat reader*. Tutto questo serve per visualizzare l'help all'interno del progetto; infatti, si fa utilizzo di file *pdf* che possono essere aperti appunto con *acrobat reader*. Questo non riduce la portabilità, infatti, al posto di *acrobat reader* si può identificare la posizione di un qualsiasi altro programma capace di leggere i file *pdf*. Dal form inoltre è possibile installare *acrobat reader* nell'ipotesi in cui tale software ancora non è presente nel sistema (quest'ultima funzionalità va bene solo su piattaforma windows).

- **ApriFrameForm.java**

Questa classe anticipa la classe *FrameForm* (che rappresenta un form) e serve per far diventare statica la sua l'apertura fissando inoltre la dimensione di tale form.

- **AssegnazioneVariabili.java**

Determina, prendendo informazioni dal vettore *VSTotale* (vettore statico in cui è presente il risultato delle pagine interne del parsing online), due vettori utili per poter scegliere nelle varie schede del form principale, il rispettivo parametro da fissare. Il primo vettore presenta quindi la lista di tutti i nomi delle variabili

presenti all'interno del progetto, mentre nel secondo vettore ci sono altri vettori al cui interno ci sono i valori che può assumere una determinata variabile.

- **CalculatorFrame.java**

Permette di visualizzare la tabella dei connettivi, cioè l'insieme dei tag utili per comporre il codice CTL.

Questa tabella risulta utile per facilitare la costruzione del codice nella gestione manuale offline e online.

- **crealInputStream.java**

Questa classe viene utilizzata durante il parsing online con lo scopo di determinare, partendo dagli elementi del *VettoreTotale*, l'inputstream indipendentemente se la pagina successiva è una pagina dinamica oppure No. Questo è possibile farlo utilizzando il metodo *DetInputStream* presente in questa classe. In questo metodo viene verificato se la pagina è statica oppure dinamica e in base a questo viene deciso il modo in cui viene determinato l'inputstream. Infatti per le pagine statiche l'inputstream viene determinato utilizzando semplicemente il metodo della classe *file* presente nel package *Java.io* mentre per le pagine dinamiche, per determinare l'*inputstream* (che dipende anche dai parametri inseriti) viene utilizzata la classe *writeURL* descritta in seguito.

- **DaWeb.java**

È la classe di apertura del progetto contenente il metodo *main*, che serve solo per far partire *DaWebFrame* che è la vera classe principale del progetto.

DaWebBase.java

È la classe base di quasi tutte le classi di *DaWeb* e contiene molti metodi generali utilizzati dalle varie classi.

- **DaWebFrame.java**

È la classe principale del progetto.

Il suo scopo è quello di far visualizzare il frame principale del progetto. Tale frame rappresenta l'interfaccia utente utile per eseguire tutte le analisi e le verifiche.

- **DaWebFrame_AboutBox.java**

Frame utile per about del progetto.

- **DeterminazioneListaFile.java**

Determina ciò che si trova all' interno della directory virtuale e nelle sue sottocartelle. Tutte queste informazioni vengono inserite in questi tre vettori:

- *ListaPagine* : Lista in cui si trovano tutte le pagine interne
- *ListaImmagini* : Lista in cui si trovano tutte le immagini
- *ListaAltro* : lista in cui si trovano file diversi da immagini e da pagine

All'interno di questi vettori sono contenute altre informazioni quali la dimensioni e la data di ultima modifica di ogni file trovato.

- **FiltroVettore.java**

Determina, una volta avuto dal costruttore un vettore di stringhe, un vettore contenente le voci del vettore di partenza che al loro interno presentano (oppure non presentano) una determinata stringa prefissata .

- **eseguiSMV.java**

Presenta un metodo che ha un facile accesso al programma SMV che sta alla base delle verifiche con CTL eseguite.

- **FormTabellaOffline.java**

Rappresenta il form che identifica la tabella dei risultati offline, che è possibile visualizzare cliccando sul tasto offline presente nella colonnina a sinistra del form principale. Cliccando ulteriormente su una delle righe di questa tabella, nella quale si trova il nome di una determinata pagina, è possibile aprire un'altra tabella, che viene visualizzata sempre utilizzando la classe *FormTabellaOffline*, che identifica i contenuti di tale pagina.

- **FormTabellaPrincipale.java**

Rappresenta il form che identifica la tabella dei risultati online, che è possibile visualizzare cliccando sul tasto online presente nella colonna a sinistra del form principale. Cliccando ulteriormente su una delle righe di questa tabella, nella quale si trova il nome di una determinata pagina, è possibile aprire un'altra tabella, che viene visualizzata sempre utilizzando la classe *FormTabellaPrincipale*, che identifica i contenuti di tale pagina.

La tabella determinata da questa classe rispetto la tabella determinata dalla classe *FormTabellaOffline* da la possibilità di poter mostrare anche i parametri inseriti durante il parsing online, associati a ciascuna pagina.

- **FrameAggiungiParametro.java**

Facendo uso della classe *AssegnazioniVariabili*, questa classe determina il frame che viene aperto dal frame principale, nelle verifiche online, lì dove si vuole fissare un parametro.

- **FrameCommento.java**

Con questo frame è possibile inserire il commento a un progetto già eseguito. Questo frame può essere aperto cliccando il tasto *commento* presente nella colonna a sinistra del form principale.

In questo commento nel momento in cui si salvano i dati del parsing in un file *.vof*, vengono indicate alcune informazioni di come è stato eseguito il parsing e viene inserita inoltre anche la data in cui è avvenuto il salvataggio.

- **FrameForm.java**

Questo frame si apre nel momento in cui si devono inserire dei parametri in un form trovato in una pagina, utile per poter aprire la successiva pagina (nel parsing online).

- **FrameRiassunto.java**

Questo frame è possibile visualizzarlo cliccando sulle righe della tabella offline che è possibile aprire cliccando sul tasto *offline* presente nella colonna a sinistra del form principale, nell'ipotesi in cui anche è selezionato il relativo *CheckBox* presente sempre nella tabella offline. In questo frame viene indicato il

riassunto della pagina analizzata, fornendo tutte le quantità numeriche del contenuto della pagina.

- **FrameRisultato.java**

È un semplice frame in cui è possibile mostrare sia l'input che va nel programma SMV e il relativo output che viene determinato. Questo Frame viene richiamato in tutte le verifiche fatte con il codice CTL sia online che offline.

- **FrameVisualizzaInputSMV.java**

È un semplice frame che viene utilizzato per visualizzare il codice SMV nella scheda *InputSMV*.

- **htmlTokenOffline.java**

Questa classe è il motore del parsing offline. In effetti determina tutti i vettori che contengono al loro interno le informazioni del parsing. Analizzando la stringa *Contenuto* che contiene il file html, si trovano, tramite tutti i tag, le informazioni interne della pagina.

Il metodo principale è *DaFile()*. Questo metodo valuta la pagina *i*-esima della tabella *VettoreTotaleOffline* e permette se ci sono pagine interne nuove, di incrementare la tabella *VettoreTotaleOffline*. Una volta completata l'analisi della pagina *i*-esima, si passa all'analisi della pagina successiva della tabella *VettoreTotaleOffline*. In tal caso viene richiamato in maniera ricorsiva il metodo *DaFile()*.

- **htmlTokenizer.java**

Questa classe è il motore del parsing online.

In effetti determina tutti i vettori che contengono al loro interno le informazioni del parsing. Analizzando l'inputstream che contiene il file html, si trovano, tramite tutti i tag, le informazioni interne della pagina.

Il problema si complica nel momento in cui si presenta un form; in tal caso infatti bisogna aprire il form in cui è possibile inserire i dati.

Il metodo principale è *DaFile()*. Questo metodo valuta la pagina *i*-esima della tabella *VettoreTotale*, apre i form per inserire i parametri (se ci sono) e permette di incrementare la tabella *VettoreTotale*. Una volta completata l'analisi della

pagina *i*-esima, si passa all'analisi della pagina successiva della tabella *VettoreTotale*. In tal caso viene richiamato in maniera ricorsiva il metodo *DaFile()*. C'è da precisare comunque che bisogna determinare prima l'inputstream. Questo inputstream viene generato grazie all'utilizzo della classe *Creainputstream*.

- **LogDaWeb**

È utile per interagire con il file *LogDaWeb.txt* aggiungendo una riga che contiene alcune informazioni che si ottengono nel corso dell'utilizzo di *DaWeb*. Per ogni riga inserita viene anche inserito la data in cui queste informazioni si sono ricavate.

- **ModuloContent.java**

Presenta i metodi utili per determinare il modulo *content* del codice SMV, utile per dare informazioni sulle caratteristiche delle pagina offline.

- **ModuloMain.java**

Presenta i metodi utili per determinare il modulo *main* del codice SMV con dati presi dal parsing offline.

- **ModuloMainOnline.java**

Presenta i metodi utili per determinare il modulo *main* del codice SMV con dati presi dal parsing online.

- **ModuloObject.java**

Presenta i metodi utili per determinare il modulo *Object* del codice SMV, utile per dare informazioni sulle oggetti contenuti nelle pagine.

- **ModuloTag.java**

Presenta i metodi utili per determinare il modulo *Tag* del codice SMV, utile per eseguire la verifica di provenienza dei vari oggetti

- **PathAssoluto.java**

Contiene dei metodi utili per determinare la path assoluta o la path relativa di link individuato in una pagina dal parsing offline

- **PathAssolutoOnline.java**

Contiene dei metodi utili per determinare la path assoluta o la path relativa di un link individuato in una pagina dal parsing online

- **SalvaApri.java**

Permette di salvare la struttura di un vettore al cui interno c'è qualsiasi tipo di oggetto, al interno di un file. Questa classe è utilizzata per salvare tutti i vettori determinati dal parsing, all'interno di un file *.vof*.

La stessa classe viene utilizzata nel momento in cui si vuole aprire un file *.vof*

- **TabellaImmagini.java**

Questo form permette di visualizzare tutte le tabelle riassuntive che si trovano sotto la voce tabella presente nel menù a discesa del form principale.

- **TabellaPagine.java**

In questa classe vengono determinati i vettori *VettoreTotale* e *VettoreAssociato*. Il metodo *ImportaPagine*, presente nella classe, gestisce il vettore in ingresso che può rappresentare o l'insieme dei form oppure l'insieme dei link di una pagina. Tale Vettore viene inserito in maniera opportuna nel *VettoreAssociato* e vengono inseriti i suoi elementi anche nel *VettoreTotale* nell'ipotesi che tali elementi non sono già presenti al suo interno.

VettoreTotale e *VettoreAssociato* sono gli input della classe *VettoriStrutturati* che determina i risultati finale del parsing online.

- **TextEditor.java**

Rappresenta un semplice TextEditor che gestisce il codice SMV. All'interno del TextEditor è infatti presente un tasto che, se premuto, esegue il codice SMV editato al suo interno.

Nel momento in cui si apre questa classe, all'interno è già presente l'ultimo codice SMV utilizzato da *DaWeb* nel corso delle sue verifiche. Questa classe utilizza anche le classi:

- *NotepadInterazione.java*: utile per aprire il notepad
- *InterazioneSMV.java*: utile per eseguire il programma SMV nel momento in si preme il tasto descritto sopra.

- **TokenFile.java**

Sono presenti alcuni metodi utili per l'interazione con i file.

- **VettoriStrutturati.java**

Determina i vettori *VSTotale* e *VSAssociato* che rappresentano i vettori risultanti del parsing online, che rispetto ai vettori *VettoreTotale* e *VettoreAssociato* hanno una struttura migliore per creare il codice SMV e per essere visualizzati nella tabella *online* (visualizzabile tramite il tasto presente nella colonna a sinistra del form principale).

- **writeURL.java**

Permette di determinare il risultato di una pagina dinamica. Infatti, avendo come input il nome della pagina di destinazione e i parametri di input, è possibile determinare un inputstream che permette di determinare un file dentro il quale si trova la pagina html risultante.

A.3 Diagrammi appartenenti al linguaggio UML

A questo punto, sarà rappresentato il modo in cui le classi sono connesse per determinare il progetto finale.

La dimensione del progetto non permette di far visualizzare il tutto in un unico diagramma. Il progetto è quindi suddiviso in vari diagrammi in base alle diverse funzionalità.

A.3.1 Collegamento tra frame

Nel diagramma di figura A-1 si presenta il collegamento delle classi che rappresentano i frame del progetto.

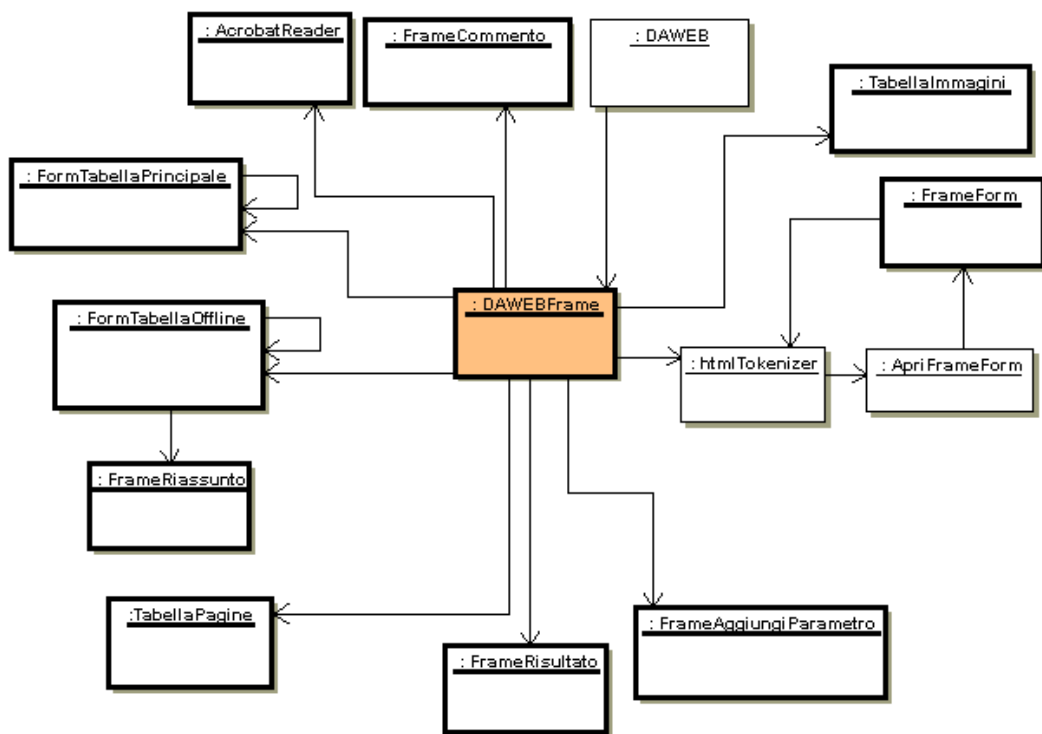


Figura A-1 : Diagramma dei collegamenti dei frame che formano *Da Web*

A.3.2 Classi per determinare i Moduli SMV

Nel diagramma di figura A-2 è visualizzato come le classi che determinano i moduli SMV vengono collegati tra di loro.

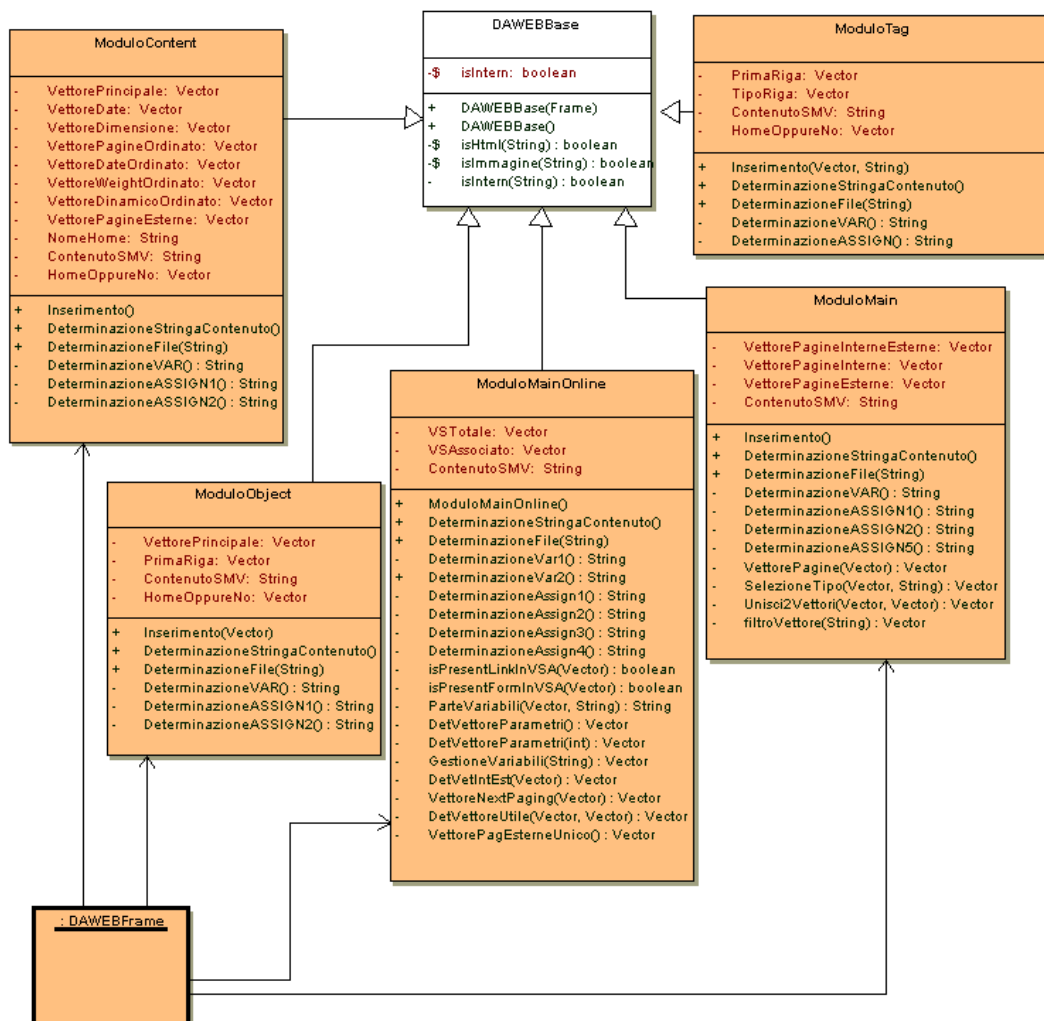


Figura A-2: Diagramma che mette in evidenza le caratteristiche delle classi che determinano i moduli SMV

A.3.3 Classi per eseguire il parsing online

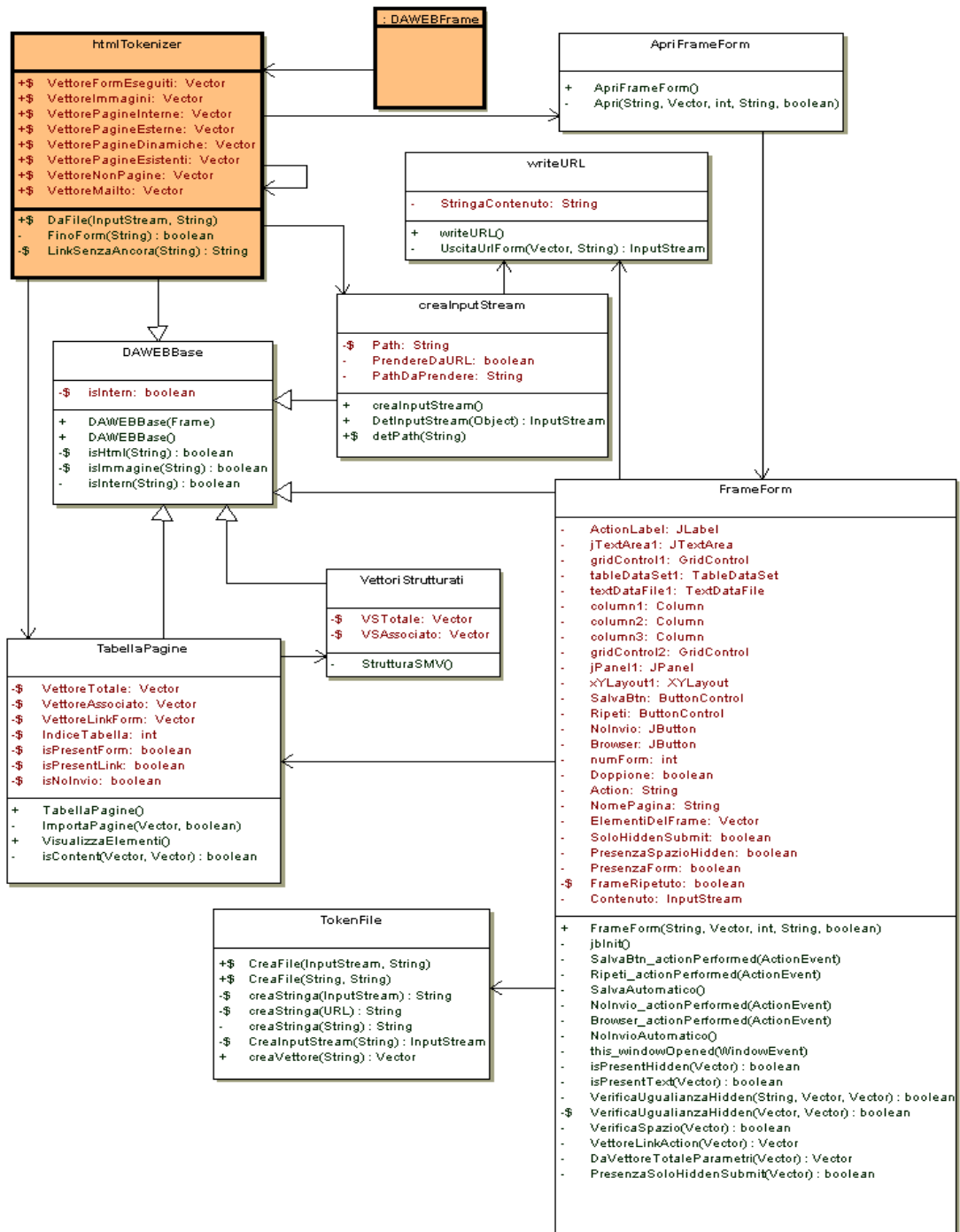


Figura A-3: Classi per eseguire il parsing online

A.3.4 Classi per eseguire il parsing offline

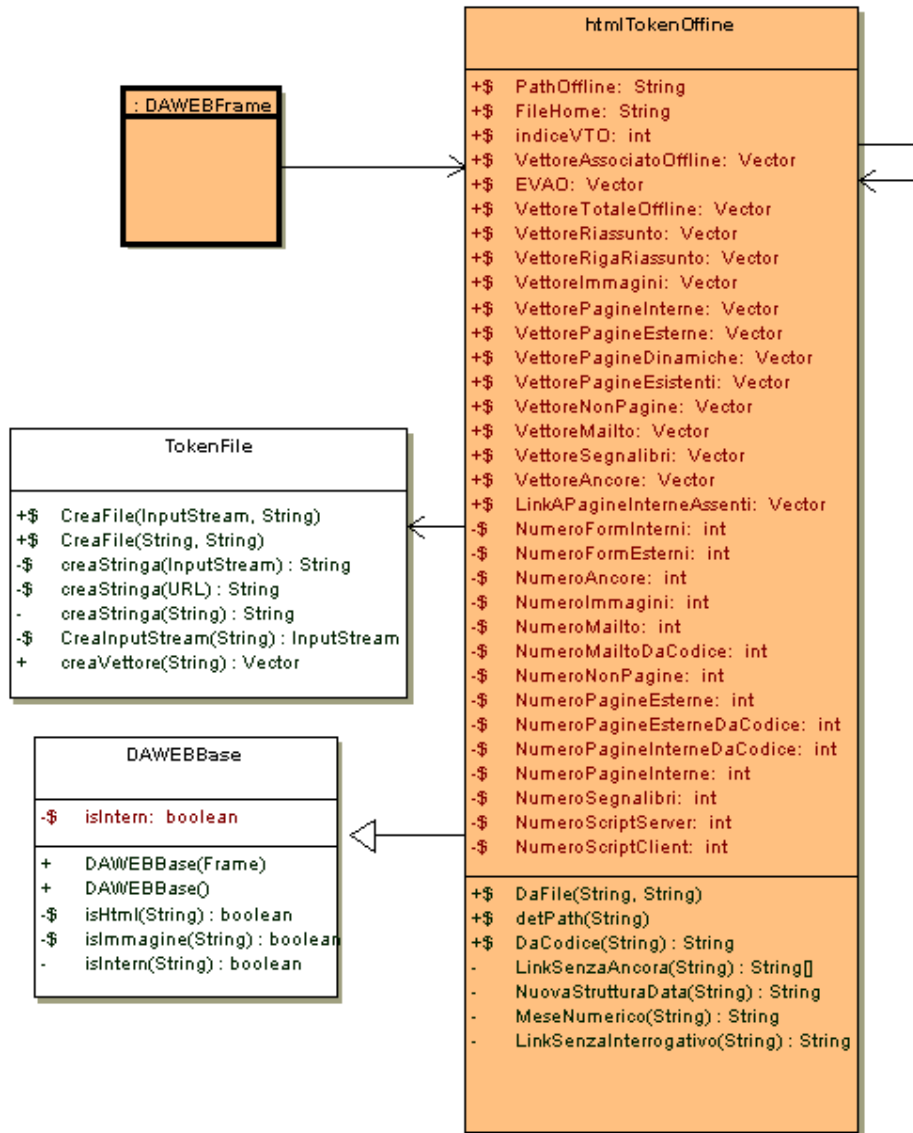


Figura A-4: Classi per eseguire il parsing offline

A.4 Analisi di DaWeb con i diagrammi DFD

Un altro diagramma utile per analizzare il software è il diagramma DFD. Questo diagramma si basa sull'*esplosione delle bolle*. Viene rappresentato quindi un problema che all'inizio sembra molto generico, ma poi diventa molto specifico, nel momento in cui sono analizzate le *bolle* ad una profondità sempre maggiore.

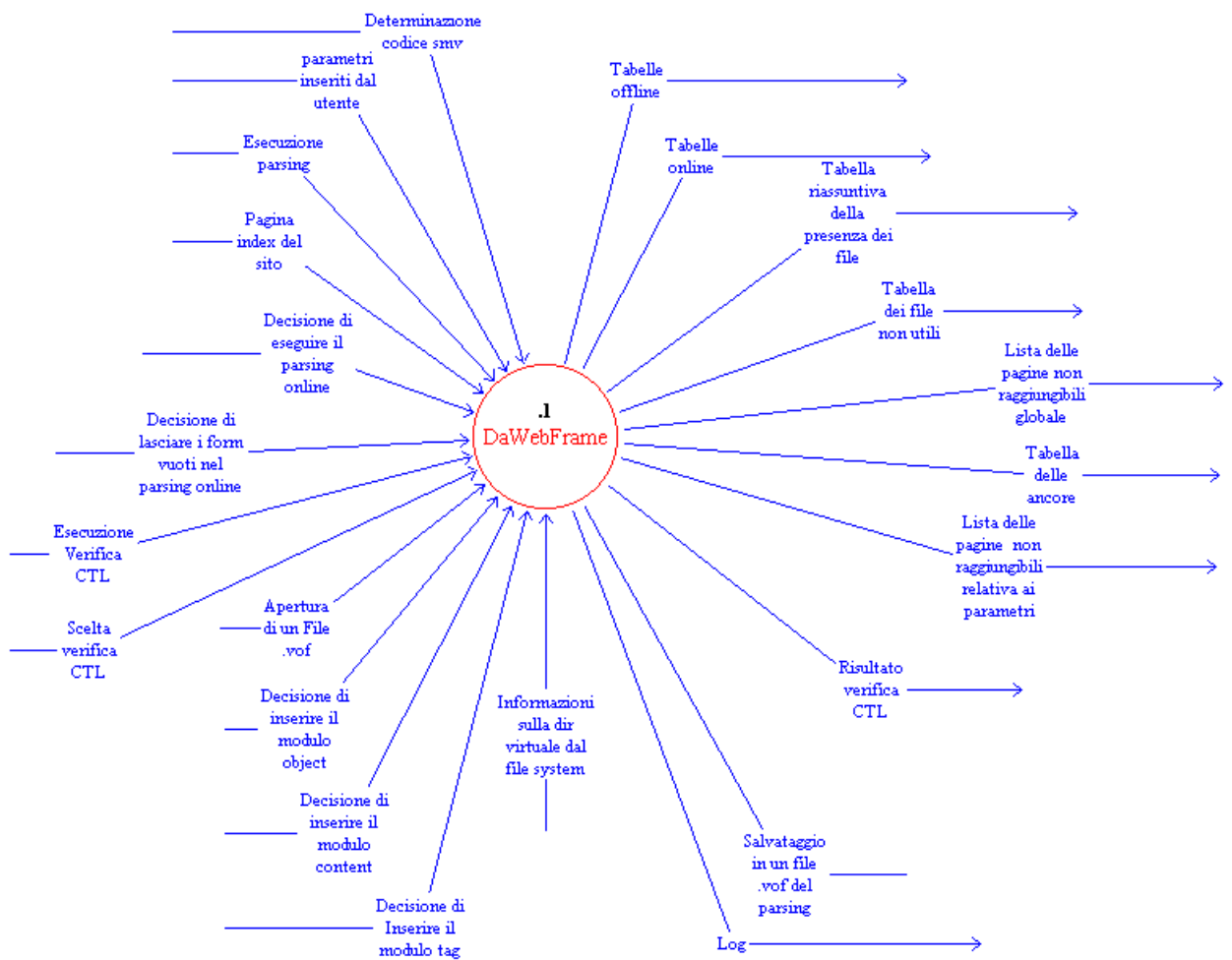


Figura A-5: Stato 1

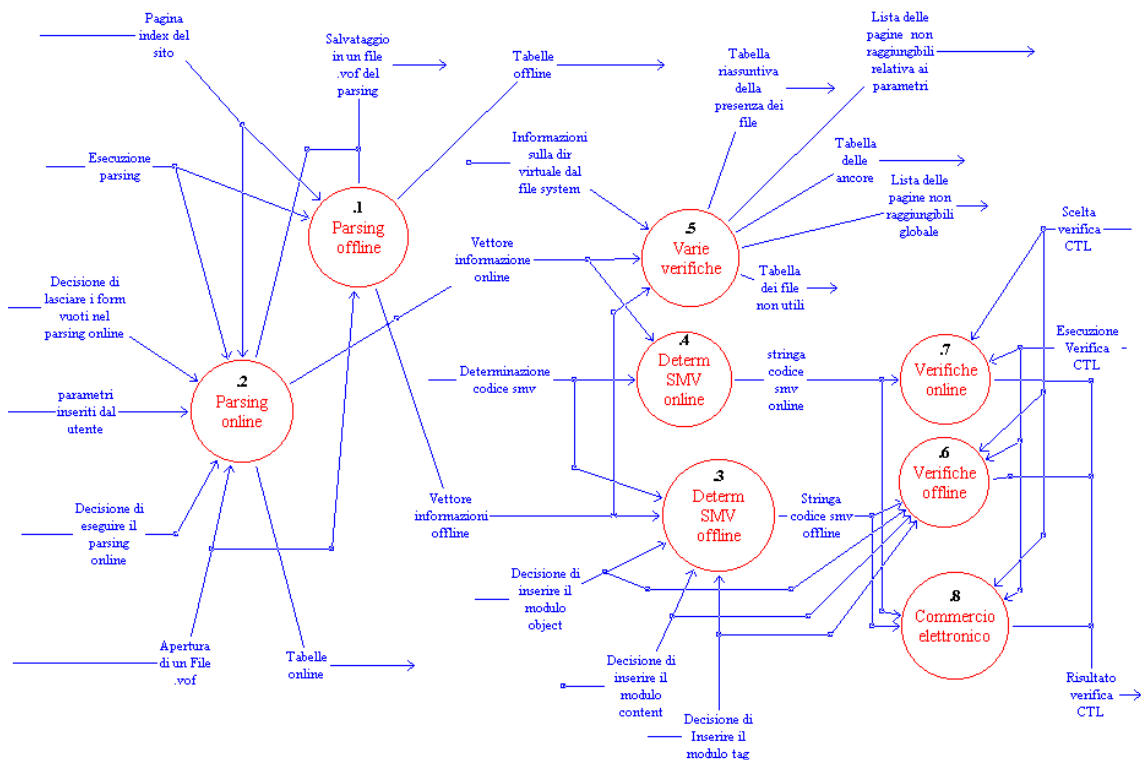


Figura A-6: Stato 1.1

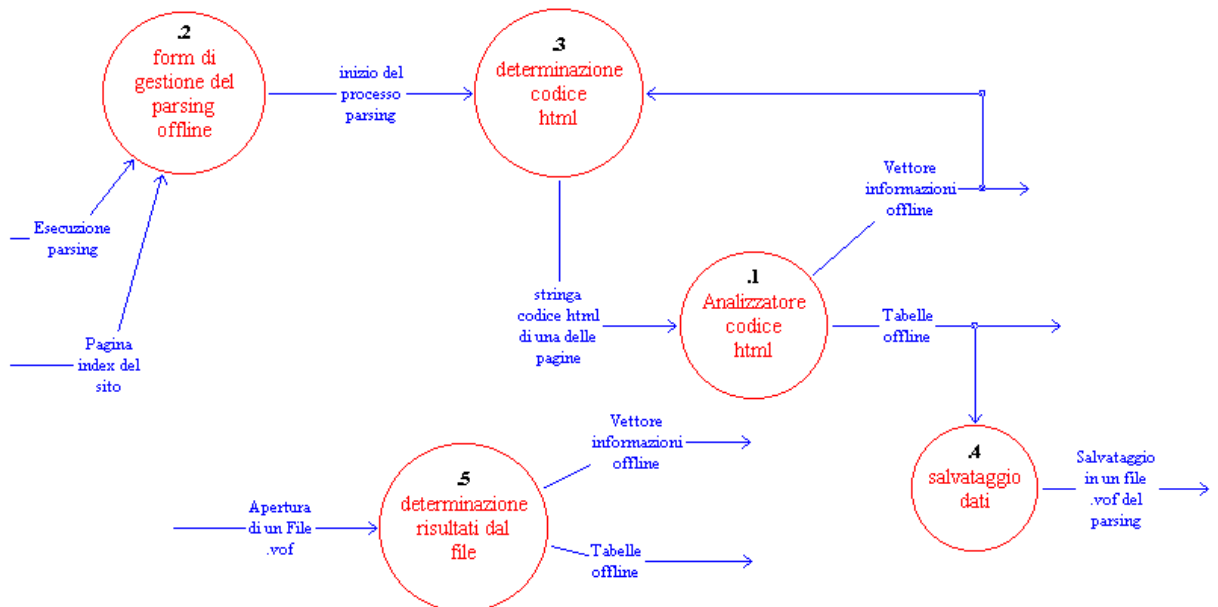


Figura A-7: Stato 1.1.1

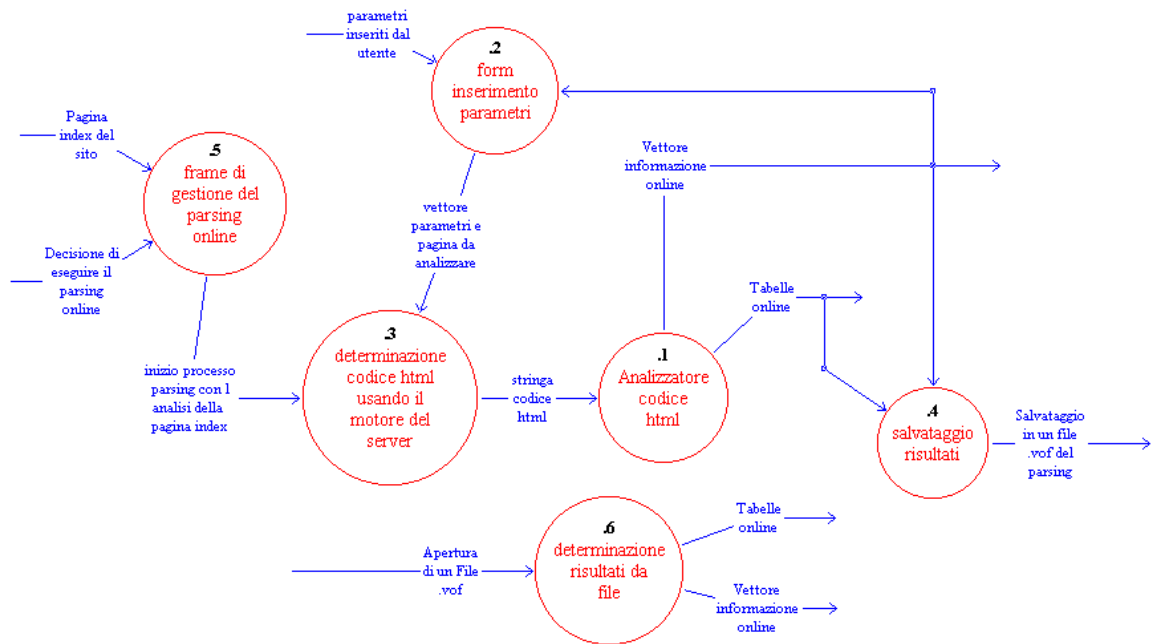


Figura A-8: Stato 1.1.2

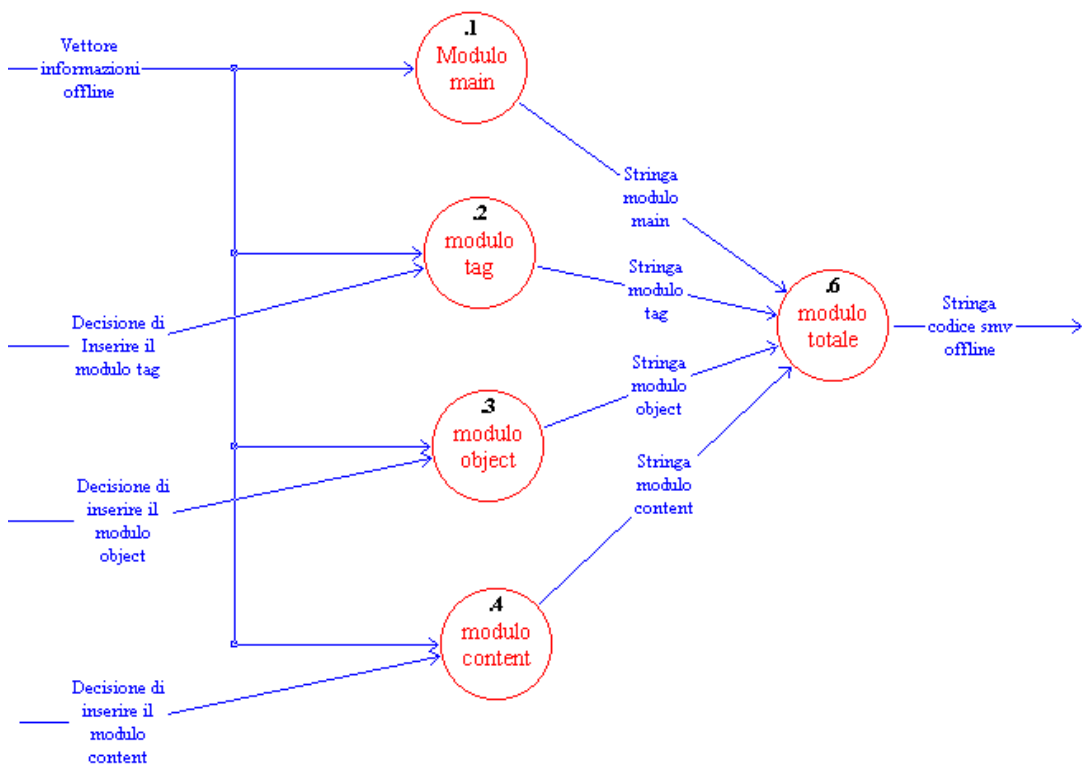


Figura A-9: Stato 1.1.3

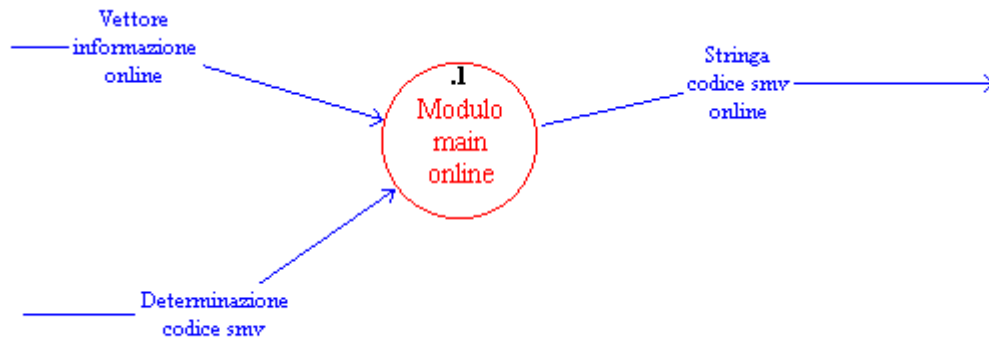


Figura A-10: Stato 1.1.4

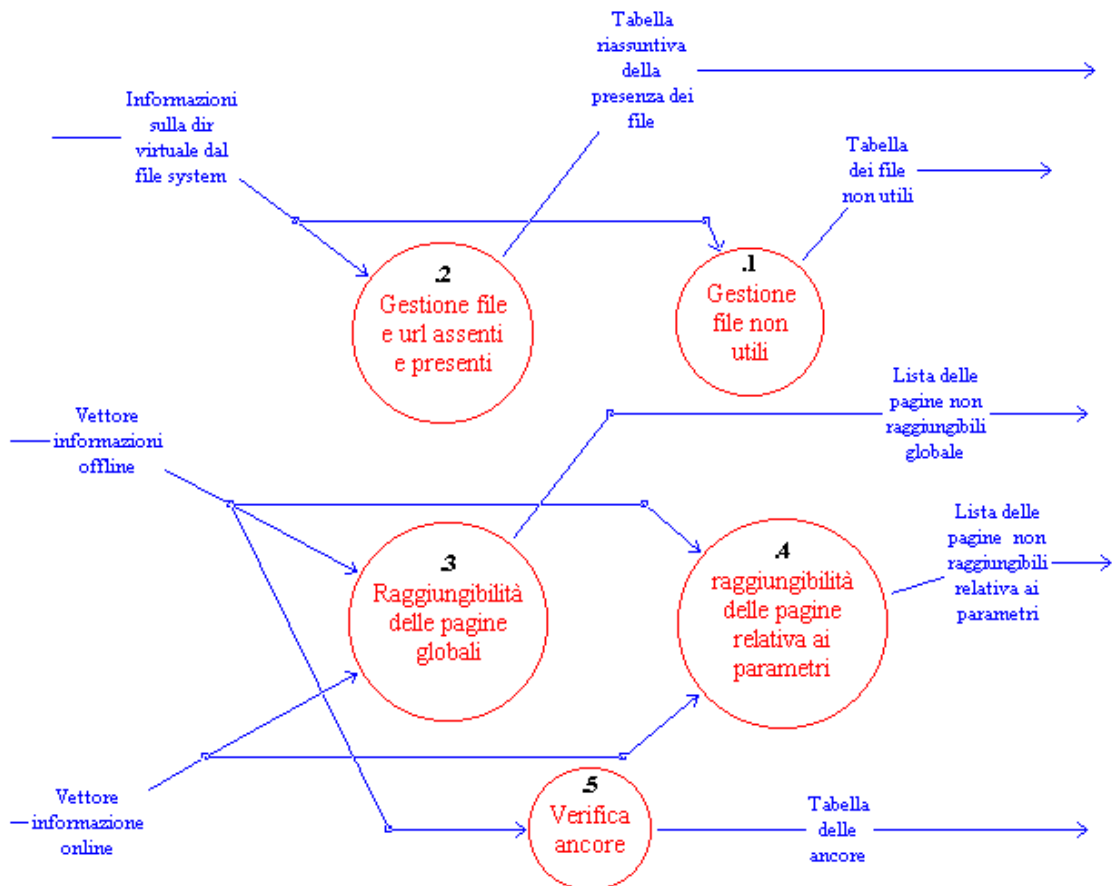


Figura A-11: Stato 1.1.5

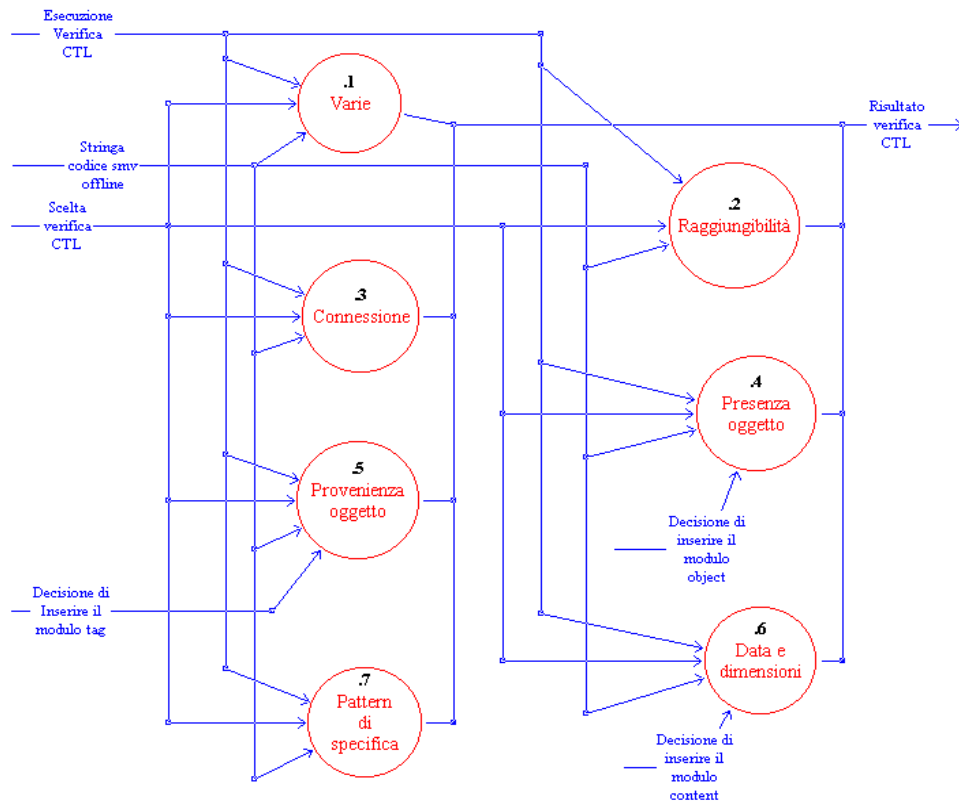


Figura A-12: Stato 1.1.6

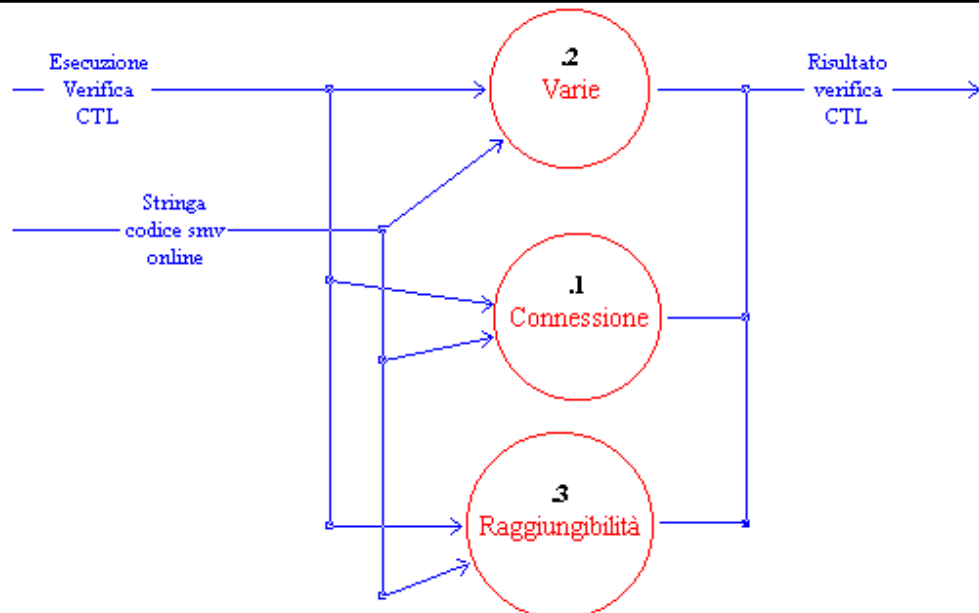


Figura A-13: Stato 1.1.7

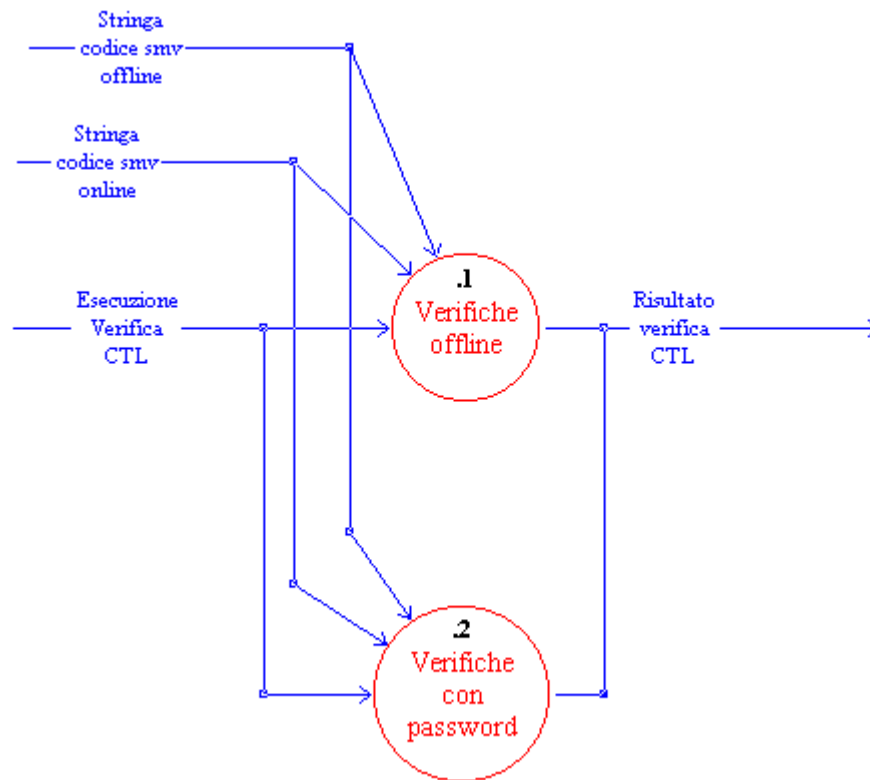


Figura A-14: Stato 1.1.8

Appendice B

SCHEMI PER I PATTERN DI SPECIFICA PER CTL E LORO UTILIZZO

B.1 Introduzione

Una collezione di semplici pattern di specifica può essere definita per assistere i professionisti nella descrizione schematica dei comportamenti dei sistemi, nel proprio formalismo di scelta migliorando il passaggio di questi metodi formali ad una diffusione pratica.

I pattern di progetto furono introdotti come mezzo di influenza dell'esperienza di progettisti esperti di sistemi. Di seguito si considerano i pattern di specifica proposti da Dwyer.

Un pattern di specifica è una descrizione generalizzata di un requisito comune sulle sequenze possibili stato/evento di un modello a stati finiti di un sistema. Un pattern comprende un nome, una precisa proposizione all'interno del pattern (ad esempio la struttura del comportamento descritto), la schematizzazione in comuni formalismi di specifica (CTL e altre logiche).

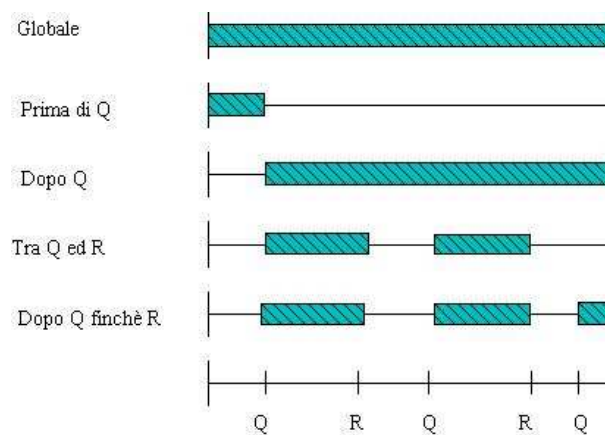


Figura B-1: Le cinque estensioni dei pattern di specifica

Ciascun pattern possiede un'estensione (scope) che è l'estensione dell'esecuzione su cui deve valere la specifica definita dal pattern. Esistono cinque tipi di estensione: **globale** (l'intera esecuzione del programma), **prima** (l'esecuzione prima

di un dato stato/evento), **dopo** (l'esecuzione dopo un dato stato/evento), **in mezzo a** (ogni parte di esecuzione tra un dato stato/evento) e **dopo – finché** (simile all'estensione precedente ad eccezione del fatto che la parte di esecuzione continua anche se il secondo stato/evento non si presenta) (v. fig. A-1).

Le informazioni nei pattern possono essere presentate in diversi modi. Un'organizzazione si basa sul classificare i pattern in termini di tipi di comportamento del sistema che descrivono.

- I pattern di occorrenza descrivono l'occorrenza di un dato stato/evento durante l'esecuzione del sistema.
- I pattern di ordine descrivono l'ordine relativo in cui stati/eventi multipli si presentano durante l'esecuzione del sistema.

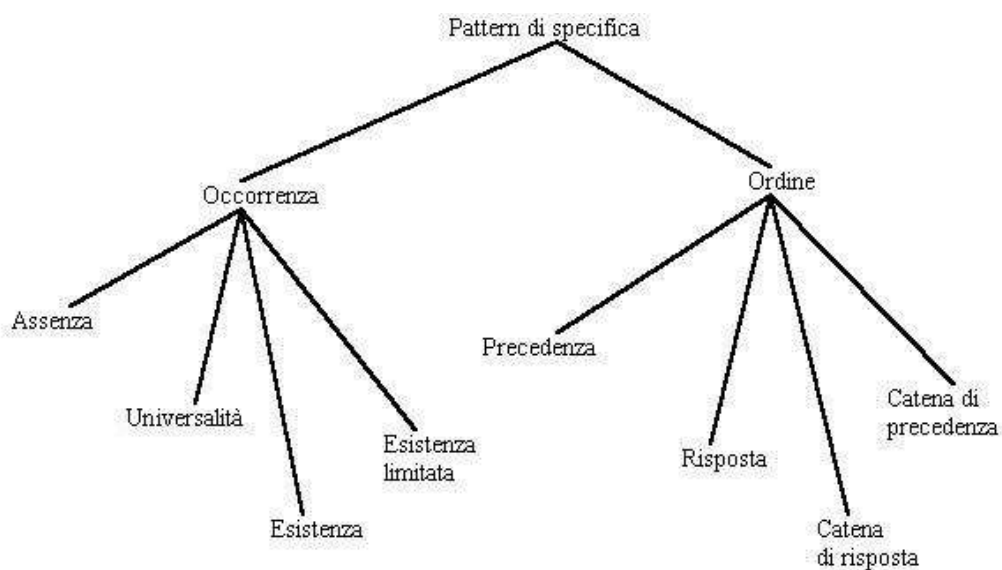


Figura B-1: Classificazione dei pattern, basata sul comportamento del sistema che descrivono

B.2 Pattern di specifica proposti da Dwyer

Molti degli schemi fanno uso dell'operatore "finché debole" (W) che è in relazione all'operatore finché (U) mediante le seguenti equivalenze:

$$\begin{aligned} A[x \ W \ y] &= \neg E[\neg y \ U \ (\neg x \ \& \ \neg y)] \\ E[x \ U \ y] &= \neg A[\neg y \ W \ (\neg x \ \& \ \neg y)] \end{aligned}$$

Questo operatore è un utile variante dell'operatore finché (U) e spesso rende lo schema più semplice da capire.

Si ricordi che $A [pUq]$ è vera in uno stato se, lungo tutti i percorsi da quello stato, q è vera da qualche parte lungo il percorso e p è vera dallo stato presente fino allo stato in cui q è vera. Ciò significa che un percorso in cui p è permanentemente vera e q non lo è mai, non soddisfa $p \ U \ q$.

Comunque, l'intuizione su finché suggerisce che si potrebbero accettare percorsi in cui q non sussista mai, supposto che p sia permanentemente vera. Ad esempio, nel sistema dell'ascensore si potrebbe voler verificare se la luce di indicazione rimane accesa finché l'ascensore arriva. Intuitivamente, un percorso in cui l'ascensore non arrivi mai e l'indicatore è sempre acceso non è una violazione di questa specifica, per cui non lo si esprimerebbe come $A [\text{accesso} \ U \ \text{arriva}]$.

Si introduce quindi l'operatore "finché debole", che si indica con W. La proposizione $A [p \ W \ q]$ è vera in ogni stato se, lungo tutti i percorsi da quello stato, p è vera dallo stato presente fino al primo stato in cui q è vera, se ce n'è uno. In particolare se non esiste uno stato in cui q è vera su un percorso, allora p deve sussistere per ogni stato di quel percorso.

L'operatore finché debole può essere definito in termini dell'ordinario finché, come segue:

$$E [p \ W \ q] \equiv E [p \ U \ q] \vee EG \ p$$

Per AW, si ha

$$A[p W q] \equiv \neg E[\neg q U \neg (p \vee q)]$$

La precedente specifica relativa all'ascensore allora si legge come A [accesso W arriva].

Dopo aver descritto il significato dell'operatore W (*finchè debole*) saranno descritte, le schematizzazioni per i pattern di specifica nella logica ad albero di computazione (CTL) proposti da Dwyer, che utilizza molto spesso questo operatore.

B.2.1 Assenza

P è falsa :

Globalmente	$AG(\neg P)$
Prima di R	$A[(\neg P \mid AG(\neg R)) W R]$
Dopo Q	$AG(Q \rightarrow AG(\neg P))$
Tra Q e R	$AG(Q \ \& \ \neg R \rightarrow A[(\neg P \mid AG(\neg R)) W R])$
Dopo Q finché R	$AG(Q \ \& \ \neg R \rightarrow A[\neg P W R])$

B.2.2 Esistenza

P diventa vera :

Globalmente	$AF(P)$
Prima di R	$A[\neg R W (P \ \& \ \neg R)]$
Dopo Q	$A[\neg Q W (Q \ \& \ AF(P))]$
Tra Q e R	$AG(Q \ \& \ \neg R \rightarrow A[\neg R W (P \ \& \ \neg R)])$
Dopo Q finché R	$AG(Q \ \& \ \neg R \rightarrow A[\neg R U (P \ \& \ \neg R)])$

B.2.3 Esistenza limitata

In questo schema si illustra un'istanza del pattern di esistenza limitata, dove il limite è rappresentato da massimo due stati designati. Altri limiti possono essere specificati da variazioni su questo schema.

Transizioni a stati che soddisfano P avvengono al massimo due volte :

Globalmente	$\neg EF(\neg P \ \& \ EX(P \ \& \ EF(\neg P \ \& \ EX(P \ \& \ EF(\neg P \ \& \ EX(P))))))$
Prima di R	$\neg E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ \neg R))]))])]$
Dopo Q	$\neg E[\neg Q \ U \ (Q \ \& \ EF(\neg P \ \& \ EX(P \ \& \ EF(\neg P \ \& \ EX(P \ \& \ EF(\neg P \ \& \ EX(P)))))))]$
Tra Q e R	$AG(Q \ \rightarrow \ \neg E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ \neg R \ \& \ EF(R))]))])])]$
Dopo Q finché R	$AG(Q \ \rightarrow \ \neg E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ E[\neg R \ U \ (\neg P \ \& \ \neg R \ \& \ EX(P \ \& \ \neg R))]))])]$

B.2.4 Universalità

P è vera :

Globalmente	$AG(P)$
Prima di R	$A[(P \ \ AG(\neg R)) \ W \ R]$
Dopo Q	$AG(Q \ \rightarrow \ AG(P))$
Tra Q e R	$AG(Q \ \& \ \neg R \ \rightarrow \ A[(P \ \ AG(\neg R)) \ W \ R])$
Dopo Q finché R	$AG(Q \ \& \ \neg R \ \rightarrow \ A[P \ W \ R])$

B.2.5 Precedenza

S precede P :

Globalmente	$A[!P \ W \ S]$
Prima di R	$A[(!P \ \ AG(!R)) \ W \ (S \ \ R)]$
Dopo Q	$A[!Q \ W \ (Q \ \& \ A[!P \ W \ S])]$
Tra Q e R	$AG(Q \ \& \ !R \ \rightarrow \ A[(!P \ \ AG(!R)) \ W \ (S \ \ R)])$
Dopo Q finché R	$AG(Q \ \& \ !R \ \rightarrow \ A[!P \ W \ (S \ \ R)])$

B.2.6 Risposta

s risponde a P :

Globalmente	$AG(P \ \rightarrow \ AF(S))$
Prima di R	$A[((P \ \rightarrow \ A[!R \ U \ (S \ \& \ !R)]) \ \ AG(!R)) \ W \ R]$
Dopo Q	$A[!Q \ W \ (Q \ \& \ AG(P \ \rightarrow \ AF(S)))]$
Tra Q e R	$AG(Q \ \& \ !R \ \rightarrow \ A[((P \ \rightarrow \ A[!R \ U \ (S \ \& \ !R)]) \ \ AG(!R)) \ W \ R])$
Dopo Q finché R	$AG(Q \ \& \ !R \ \rightarrow \ A[(P \ \rightarrow \ A[!R \ U \ (S \ \& \ !R)]) \ W \ R])$

B.2.7 Catena di Precedenza

Questo illustra la catena di precedenza una causa due effetti.

P precede S, T:

Globalmente	$!E[!P \ U \ (S \ \& \ !P \ \& \ EX(EF(T)))]$
Prima di R	$!E[(!P \ \& \ !R) \ U \ (S \ \& \ !P \ \& \ !R \ \& \ EX(E[!R \ U \ (T \ \& \ !R)])))]$
Dopo Q	$!E[!Q \ U \ (Q \ \& \ E[!P \ U \ (S \ \& \ !P \ \& \ EX(EF(T)))])]$
Tra Q e R	$AG(Q \ \rightarrow \ !E[(!P \ \& \ !R) \ U \ (S \ \& \ !P \ \& \ !R \ \& \ EX(E[!R \ U \ (T \ \& \ !R \ \& \ EF(R))]))])$
Dopo Q finché R	$AG(Q \ \rightarrow \ !E[(!P \ \& \ !R) \ U \ (S \ \& \ !P \ \& \ !R \ \& \ EX(E[!R \ U \ (T \ \& \ !R)])))]$

Questo illustra la catena di precedenza due cause un effetto.

S, T precede P:

Globalmente	$\neg E[\neg S \cup P] \ \& \ \neg E[\neg P \cup (S \ \& \ \neg P \ \& \ EX(E[\neg T \cup (P \ \& \ \neg T)]))]$
Prima di R	$\neg E[(\neg S \ \& \ \neg R) \cup (P \ \& \ \neg R)] \ \& \ \neg E[(\neg P \ \& \ \neg R) \cup (S \ \& \ \neg P \ \& \ \neg R \ \& \ EX(E[(\neg T \ \& \ \neg R) \cup (P \ \& \ \neg T \ \& \ \neg R)]))]$
Dopo Q	$\neg E[\neg Q \cup (Q \ \& \ E[\neg S \cup P] \ \& \ E[\neg P \cup (S \ \& \ \neg P \ \& \ EX(E[\neg T \cup (P \ \& \ \neg T)]))])]$
Tra Q e R	$AG(Q \rightarrow \neg E[(\neg S \ \& \ \neg R) \cup (P \ \& \ \neg R \ \& \ EF(R))] \ \& \ \neg E[(\neg P \ \& \ \neg R) \cup (S \ \& \ \neg P \ \& \ \neg R \ \& \ EX(E[(\neg T \ \& \ \neg R) \cup (P \ \& \ \neg T \ \& \ \neg R \ \& \ EF(R)]))])]$
Dopo Q finché R	$AG(Q \rightarrow \neg E[(\neg S \ \& \ \neg R) \cup (P \ \& \ \neg R)] \ \& \ \neg E[(\neg P \ \& \ \neg R) \cup (S \ \& \ \neg P \ \& \ \neg R \ \& \ EX(E[(\neg T \ \& \ \neg R) \cup (P \ \& \ \neg T \ \& \ \neg R)]))])]$

B.2.8 Catena di Risposta

Questo illustra la catena di risposta stimoli 1-2.

S, T risponde a P:

Globalmente	$AG(P \rightarrow AF(S \ \& \ AX(AF(T))))$
Prima di R	$\neg E[\neg R \cup (P \ \& \ \neg R \ \& \ (E[\neg S \cup R] \ \ E[\neg R \cup (S \ \& \ \neg R \ \& \ EX(E[\neg T \cup R]))]))]$
Dopo Q	$\neg E[\neg Q \cup (Q \ \& \ EF(P \ \& \ (EG(\neg S) \ \ EF(S \ \& \ EX(EG(\neg T)))))])]$
Tra Q e R	$AG(Q \rightarrow \neg E[\neg R \cup (P \ \& \ \neg R \ \& \ (E[\neg S \cup R] \ \ E[\neg R \cup (S \ \& \ \neg R \ \& \ EX(E[\neg T \cup R]))]))])]$
Dopo Q finché R	$AG(Q \rightarrow \neg E[\neg R \cup (P \ \& \ \neg R \ \& \ (E[\neg S \cup R] \ \ EG(\neg S \ \& \ \neg R) \ \ E[\neg R \cup (S \ \& \ \neg R \ \& \ EX(E[\neg T \cup R] \ \ EG(\neg T \ \& \ \neg R)]))]))])]$

Questo illustra la catena di risposta stimoli 2-1.

P risponde a S, T:

Globalmente	<code>!EF(S & EX(EF(T & EG(!P))))</code>
Prima di R	<code>!E[!R U (S & !R & EX(E[!R U (T & !R & E[!P U R])))]</code>
Dopo Q	<code>!E[!Q U (Q & EF(S & EX(EF(T & EG(!P)))))]</code>
Tra Q e R	<code>AG(Q -> !E[!R U (S & !R & EX(E[!R U (T & !R & E[!P U R])))]</code>
Dopo Q finché R	<code>AG(Q -> !E[!R U (S & !R & EX(E[!R U (T & !R & (E[!P U R] EG(!P & !R)))))]</code>

B.2.9 Pattern di catena vincolata

Questa è una catena di risposta vincolata 1-2 da una singola proposizione.

S, T senza Z risponde a P:

Globalmente	<code>AG(P -> AF(S & !Z & AX(A[!Z U T])))</code>
Prima di R	<code>!E[!R U (P & !R & (E[!S U R] E[!R U (S & !R & (E[!T U Z] EX(E[!T U R])))])))]</code>
Dopo Q	<code>!E[!Q U (Q & EF(P & (EG(!S) EF(S & (E[!T U Z] EX(EG(!T)))))))]</code>
Tra Q e R	<code>AG(Q -> !E[!R U (P & !R & (E[!S U R] E[!R U (S & !R & (E[!T U Z] EX(E[!T U R])))])))]</code>
Dopo Q finché R	<code>AG(Q -> !E[!R U (P & !R & (E[!S U R] EG(!S & !R) E[!R U (S & !R & (E[!T U Z] EX(E[!T U R] EG(!T & !R)))))])))]</code>

B.3 Utilizzo degli schemi nel progetto

Nel paragrafo 3.5 sono stati descritte le varie verifiche che si possono compiere in automatico dal progetto. Tra queste verifiche, sono state elencate anche tutte le

verifiche che possono essere eseguite utilizzando i pattern di specifica cioè le formule CTL presenti in questi schemi.

Non tutte le formule sono state utilizzate, infatti per il modello a stati finiti utilizzato per rappresentare un sito web, alcune non hanno significato.

Sono state utilizzate solo le formule appartenenti ai gruppi:

- Assenza (utilizzato per la verifica sugli oggetti e sulle pagine)
- Esistenza (utilizzato per la verifica sugli oggetti e sulle pagine)
- Universalità (utilizzato per la verifica sugli oggetti e sulle pagine)
- Precedenza (utilizzato per la verifica sulle pagine)
- Risposta (utilizzato per la verifica sulle pagine)

Riferendoci a vari schemi utilizzati, il valore messo al posto di P può essere o il nome di un oggetto presente in una pagina oppure il nome di una pagina stessa, mentre il valore messo al posto di S,R e Q può essere solo il nome di una pagina.

Le varie verifiche fatte sono le seguenti:

1. L'assenza dell'oggetto P globale
 2. L'assenza dell'oggetto P prima della pagina R
 3. L'assenza dell'oggetto P dopo la pagina Q
 4. L'assenza dell'oggetto P tra le pagine R e Q
 5. L'assenza dell'oggetto P dopo la pagina Q, finché si verifica la pagina R.
-
1. La presenza dell'oggetto P globale
 2. La presenza dell'oggetto P prima della pagina R
 3. La presenza di un determinato oggetto dopo la pagina Q
 4. La presenza di un determinato oggetto tra la pagina R e la pagina Q
 5. La presenza di un determinato oggetto dopo la pagina Q, finché si verifica la pagina R.
-
1. L'esistenza dell'oggetto P globale
 2. L'esistenza dell'oggetto P prima della pagina R

3. L'esistenza di un determinato oggetto dopo la pagina Q
4. L'esistenza di un determinato oggetto tra la pagina R e la pagina Q
5. L'esistenza di un determinato oggetto dopo la pagina Q finche si verifica la pagina R.

1. L'assenza della pagina P globale
2. L'assenza della pagina P prima della pagina R
3. L'assenza della pagina P dopo la pagina Q
4. L'assenza della pagina P tra la pagina R e la pagina Q
5. L'assenza della pagina P dopo la pagina Q, finche si verifica la pagina R.

1. La presenza della pagina P globale
2. La presenza della pagina P prima della pagina R
3. La presenza della pagina P dopo la pagina Q
4. La presenza della pagina P tra la pagina R e la pagina Q
5. La presenza della pagina P dopo la pagina Q, finche si verifica la pagina R

1. L'esistenza della pagina P globale
2. L'esistenza della pagina P prima della pagina R
3. L'esistenza della pagina P dopo la pagina Q
4. L'esistenza della pagina P tra la pagina R e la pagina Q
5. L'esistenza della pagina P dopo la pagina Q, finche si verifica la pagina R

1. La precedenza della pagina P dalla pagina S globale
2. La precedenza della pagina P dalla pagina S prima della pagina R
3. La precedenza della pagina P dalla pagina S dopo la pagina Q
4. La precedenza della pagina P dalla pagina S tra la pagina R e la pagina Q
5. La precedenza della pagina P dalla pagina S dopo la pagina Q, finche si verifica la pagina R

1. La risposta della pagina P dalla pagina S globale
2. La risposta della pagina P dalla pagina S prima della pagina R
3. La risposta della pagina P dalla pagina S dopo la pagina Q
4. La risposta della pagina P dalla pagina S tra la pagina R e la pagina Q

5. La risposta della pagina P dalla pagina S dopo la pagina Q, finché si verifica la pagina R.

B.4 Significato delle varie estensioni nel progetto

- **Globale**

Quando si richiede di fare una verifica globale, si vuole prendere in considerazione tutte le pagine del sito trovate durante il parsing.

- **Prima della pagina R**

Quando si richiede di fare un'operazione prima della pagina R, si vuole prendere in considerazione un determinato sottoinsieme di tutte le pagine trovate durante il parsing, con il vincolo che tale sottoinsieme di pagine, anticipa, nel modello a stati finiti, la pagina R.

Questo tipo di verifica risulta utile per siti che hanno un modello a stati finiti simile a quello mostrato nella figura B-3

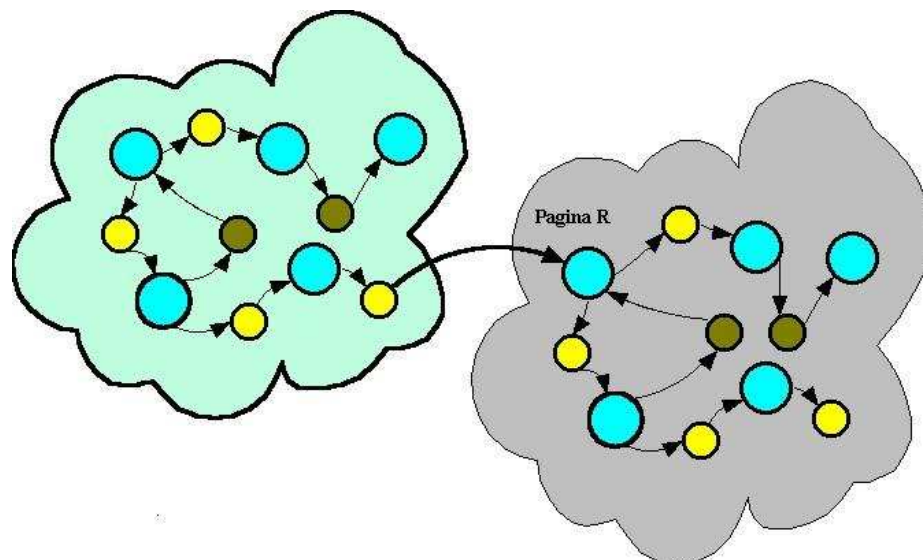


Figura B-3: Esempio di modello a stati finiti che indica quale è il sottoinsieme che bisogna prendere in considerazione nelle verifiche fatte prima della pagina R. Questo sottoinsieme è quello indicato con la nuvoletta verde

In base al modello di tale figura, le verifiche di questo tipo, prendono in considerazione solo il sottoinsieme indicato con la nuvoletta verde.

- **Dopo la pagina Q**

Quando si richiede di fare un'operazione dopo la pagina Q, si vuole prendere in considerazione una determinato sottoinsieme di tutte le pagine trovate durante il parsing, con il vincolo che tale sottoinsieme di pagine, posticipa, nel modello a stati finiti, la pagina Q.

Questo tipo di verifica risulta utile per siti che hanno un modello a stati finiti simile a quello mostrato nella figura B-4

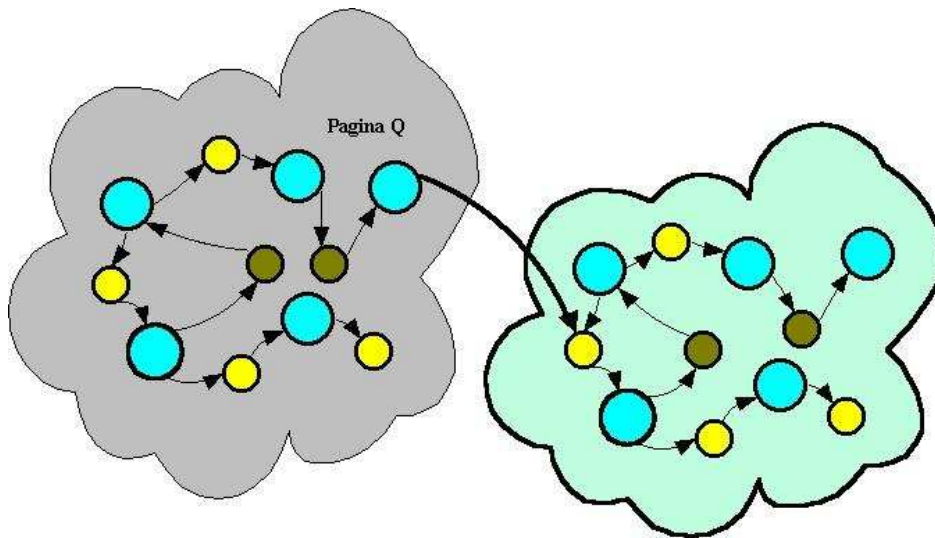


Figura B-4: Esempio di modello a stati finiti che indica quale è il sottoinsieme che bisogna prendere in considerazione nelle verifiche fatte dopo la pagina Q. Questo sottoinsieme è quello indicato con la nuvoletta verde

In base al modello di tale figura, le verifiche di questo tipo, prendono in considerazione solo il sottoinsieme indicato con la nuvoletta verde.

- **Tra la pagina R e la pagina Q**

Quando si richiede di fare un'operazione tra la pagina R e la pagina Q, si vuole prendere in considerazione un determinato sottoinsieme di tutte le pagine trovate durante il parsing, con il vincolo che tale sottoinsieme di pagine, anticipa nel modello a stati finiti, la pagina Q e posticipa la pagina R. (se non è presente la pagina R tale sottoinsieme non può essere determinato quindi in questo caso risulta nullo).

Questo tipo di verifica risulta utile per siti che hanno un modello a stati finiti simile a quello mostrato nella figura B-5.

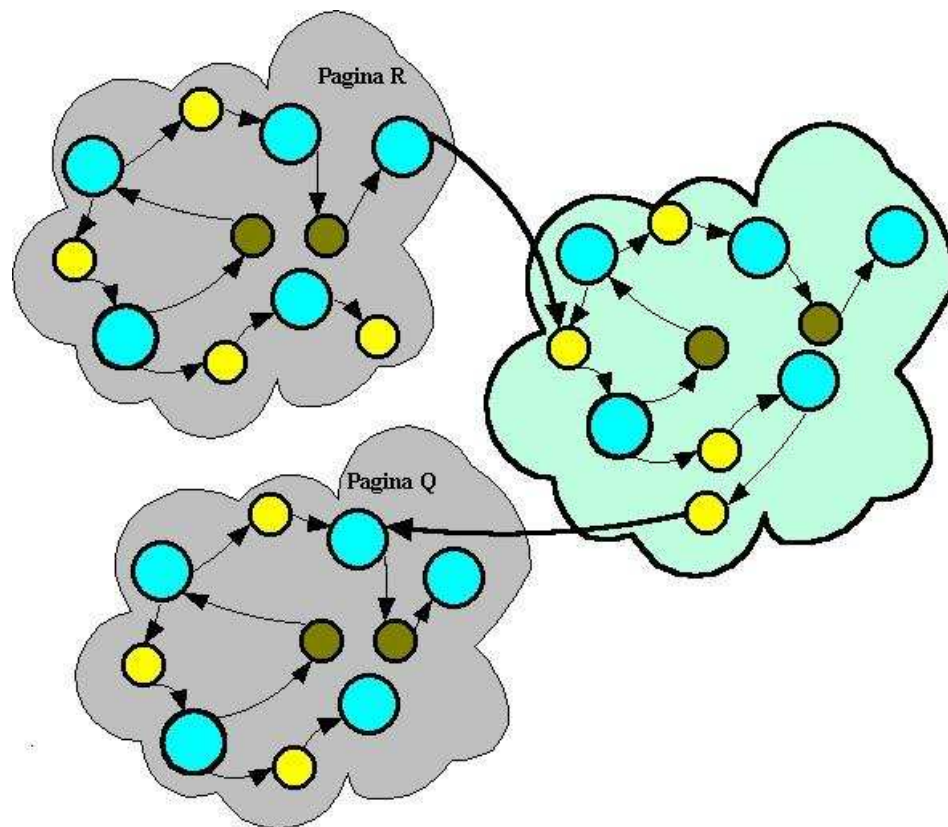


Figura B-5: Esempio di modello a stati finiti che indica quale è il sottoinsieme che bisogna prendere in considerazione nelle verifiche fatte tra la pagina R e la pagina Q. Questo sottoinsieme è quello indicato con la nuvoletta verde

- **Dopo la pagina R finché si verifica la pagina Q**

Quando infine si richiede di fare un'operazione dopo la pagina R finché si verifica la pagina Q, si vuole prendere in considerazione un determinato sottoinsieme di tutte le pagine trovate durante il parsing, con il vincolo che tale sottoinsieme di pagine, nel modello, posticipa la pagina R e si conclude nel momento in cui si presenta la pagina Q (a differenza del caso precedente il sottoinsieme si presenta anche se la pagina Q non esiste).

Per un modello a stati finiti che presenta la pagina Q, il sottoinsieme, a cui vengono eseguite le verifiche, risulta uguale a quello mostrato in figura B-5 mentre per i modelli a stati finiti che non presenta la pagina Q, il sottoinsieme, a cui vengono eseguite le verifiche, risulta uguale a quello mostrato in figura B-4.

Appendice C

**CODICE SMV OFFLINE
E CODICE SMV ONLINE**

C.1 Introduzione

In questa appendice saranno riportati il codice *SMV offline* e online determinati automaticamente da *DaWeb* rispettivamente dopo il parsing offline e il parsing online dell'esempio *CdMarket*.

C.2 Codice *SMV offline*

```
MODULE content(Paging)
  VAR
    internal:boolean;
    home:boolean;
    dynamic:boolean;
    date:{ 19991006,19990825,19991017,19990808,20020619,20020628,20020622 };
    weight:{ 547,682,2169,962,5425,3283,1592,4178,5534,465 };

  ASSIGN
    init(internal):=1;
    init(home):=1;
    init(dynamic):=0;
    init(weight):=547;
    init(date):=19991006;
  next(weight):=case
    next(Paging) = linkpg_D_Zcdmarket_Zindex_Qhtm : 547;
    next(Paging) = linkpg_D_Zcdmarket_Zuser_Qhtm : 682;
    next(Paging) = linkpg_D_Zcdmarket_Zreg_Qasp : 2169;
    next(Paging) = linkpg_D_Zcdmarket_Zguest_Qhtm : 962;
    next(Paging) = action_D_Zcdmarket_Zautenticazione_Qasp : 5425;
    next(Paging) = action_D_Zcdmarket_Zreg1_Qasp : 3283;
    next(Paging) = action_D_Zcdmarket_Zselezione_Qasp : 1592;
    next(Paging) = action_D_Zcdmarket_Zreg_Qasp : 2169;
    next(Paging) = action_D_Zcdmarket_Zselezione1_Qasp : 4178;
    next(Paging) = action_D_Zcdmarket_Zpreventivo_Qasp : 5534;
    next(Paging) = action_D_Zcdmarket_Zacquisto_Qasp : 465;
    next(Paging) = linkpg_D_Zcdmarket_Zacquisto_Qasp : 465;
  1:weight;
  esac;

  next(date):=case
    next(Paging) = linkpg_D_Zcdmarket_Zindex_Qhtm : 19991006;
    next(Paging) = linkpg_D_Zcdmarket_Zuser_Qhtm : 19991006;
```

```
next(Paging) = linkpg_D_Zcdmarket_Zreg_Qasp : 19990825;
next(Paging) = linkpg_D_Zcdmarket_Zguest_Qhtm : 19991017;
next(Paging) = action_D_Zcdmarket_Zautenticazione_Qasp : 19990808;
next(Paging) = action_D_Zcdmarket_Zreg1_Qasp : 19991006;
next(Paging) = action_D_Zcdmarket_Zselezione_Qasp : 20020619;
next(Paging) = action_D_Zcdmarket_Zreg_Qasp : 19990825;
next(Paging) = action_D_Zcdmarket_Zselezione1_Qasp : 19990808;
next(Paging) = action_D_Zcdmarket_Zpreventivo_Qasp : 20020628;
next(Paging) = action_D_Zcdmarket_Zacquisto_Qasp : 20020622;
next(Paging) = linkpg_D_Zcdmarket_Zacquisto_Qasp : 20020622;
1:date;
esac;
```

```
next(dynamic):=case
  next(Paging) = linkpg_D_Zcdmarket_Zindex_Qhtm : 0;
  next(Paging) = linkpg_D_Zcdmarket_Zuser_Qhtm : 0;
  next(Paging) = linkpg_D_Zcdmarket_Zreg_Qasp : 1;
  next(Paging) = linkpg_D_Zcdmarket_Zguest_Qhtm : 0;
  next(Paging) = action_D_Zcdmarket_Zautenticazione_Qasp : 1;
  next(Paging) = action_D_Zcdmarket_Zreg1_Qasp : 1;
  next(Paging) = action_D_Zcdmarket_Zselezione_Qasp : 1;
  next(Paging) = action_D_Zcdmarket_Zreg_Qasp : 1;
  next(Paging) = action_D_Zcdmarket_Zselezione1_Qasp : 1;
  next(Paging) = action_D_Zcdmarket_Zpreventivo_Qasp : 1;
  next(Paging) = action_D_Zcdmarket_Zacquisto_Qasp : 1;
  next(Paging) = linkpg_D_Zcdmarket_Zacquisto_Qasp : 1;
1:0;
esac;
```

```
next(internal):=case
  1:1;
esac;
```

```
next(home):=case
  next(Paging) =linkpg_D_Zcdmarket_Zindex_Qhtm:1;
  1:0;
esac;
```

MODULE object(Paging)

```
VAR
  backgr_D_Zcdmarket_Znote_Qgif:boolean;
  imagep_D_Zcdmarket_Zhome1_Qgif:boolean;
  backgr_D_Zcdmarket_Zdollaro_Qgif:boolean;
```

ASSIGN

```
init(backgr_D_Zcdmarket_Znote_Qgif):=1;
init(imagep_D_Zcdmarket_Zhome1_Qgif):=0;
init(backgr_D_Zcdmarket_Zdollaro_Qgif):=0;
```

```
next(backgr_D_Zcdmarket_Znote_Qgif):=case
  next(Paging) in {linkpg_D_Zcdmarket_Zindex_Qhtm,
    linkpg_D_Zcdmarket_Zuser_Qhtm,
    linkpg_D_Zcdmarket_Zreg_Qasp,
    linkpg_D_Zcdmarket_Zguest_Qhtm,
    action_D_Zcdmarket_Zautenticazione_Qasp,
    action_D_Zcdmarket_Zreg1_Qasp,
    action_D_Zcdmarket_Zselezione_Qasp,
    action_D_Zcdmarket_Zreg_Qasp,
    action_D_Zcdmarket_Zselezione1_Qasp,
    action_D_Zcdmarket_Zpreventivo_Qasp}:1;

    1:0;
esac;
```

```
next(imagep_D_Zcdmarket_Zhome1_Qgif):=case
  next(Paging) in {linkpg_D_Zcdmarket_Zuser_Qhtm,
    linkpg_D_Zcdmarket_Zreg_Qasp,
    linkpg_D_Zcdmarket_Zguest_Qhtm,
    action_D_Zcdmarket_Zautenticazione_Qasp,
    action_D_Zcdmarket_Zreg1_Qasp,
    action_D_Zcdmarket_Zselezione_Qasp,
    action_D_Zcdmarket_Zreg_Qasp,
    action_D_Zcdmarket_Zselezione1_Qasp,
    action_D_Zcdmarket_Zpreventivo_Qasp}:1;

    1:0;
esac;
```

```
next(backgr_D_Zcdmarket_Zdollaro_Qgif):=case
  next(Paging) in {action_D_Zcdmarket_Zacquisto_Qasp,
    linkpg_D_Zcdmarket_Zacquisto_Qasp}:1;

    1:0;
esac;
```

MODULE tag

VAR

```
imagep_D_Zcdmarket_Zhome1_Qgif: {LinkToObject,IMG,BODY,EMBED,APPLET,form,input};
backgr_D_Zcdmarket_Znote_Qgif: {LinkToObject,IMG,BODY,EMBED,APPLET,form,input};
backgr_D_Zcdmarket_Zdollaro_Qgif: {LinkToObject,IMG,BODY,EMBED,APPLET,form,input};
```

ASSIGN

```
init(imagep_D_Zcdmarket_Zhome1_Qgif):=IMG;
init(backgr_D_Zcdmarket_Znote_Qgif):=BODY;
init(backgr_D_Zcdmarket_Zdollaro_Qgif):=BODY;
```

MODULE main

VAR

```
State: { page,link,linkpg,ancora,action};
```

```
Paging: { linkpg_D_Zcdmarket_Zindex_Qhtm,
          linkpg_D_Zcdmarket_Zuser_Qhtm,
          linkpg_D_Zcdmarket_Zreg_Qasp,
          linkpg_D_Zcdmarket_Zguest_Qhtm,
          action_D_Zcdmarket_Zautenticazione_Qasp,
          action_D_Zcdmarket_Zreg1_Qasp,
          action_D_Zcdmarket_Zselezione_Qasp,
          action_D_Zcdmarket_Zreg_Qasp,
          action_D_Zcdmarket_Zselezione1_Qasp,
          action_D_Zcdmarket_Zpreventivo_Qasp,
          action_D_Zcdmarket_Zacquisto_Qasp,
          linkpg_D_Zcdmarket_Zacquisto_Qasp};
```

```
c:content(Paging);
```

```
o:object(Paging);
```

```
t:tag;
```

ASSIGN

```
init(State):=page;
init(Paging):=linkpg_D_Zcdmarket_Zindex_Qhtm;
```

```
next(Paging):=case
  State=linkpg & Paging = linkpg_D_Zcdmarket_Zindex_Qhtm:{
    linkpg_D_Zcdmarket_Zuser_Qhtm,
    linkpg_D_Zcdmarket_Zreg_Qasp,
    linkpg_D_Zcdmarket_Zguest_Qhtm};

  State=linkpg & Paging = linkpg_D_Zcdmarket_Zuser_Qhtm:{
    action_D_Zcdmarket_Zautenticazione_Qasp,
    linkpg_D_Zcdmarket_Zindex_Qhtm};

  State=linkpg & Paging = linkpg_D_Zcdmarket_Zreg_Qasp:{
    action_D_Zcdmarket_Zreg1_Qasp,
    linkpg_D_Zcdmarket_Zindex_Qhtm};

  State=linkpg & Paging = linkpg_D_Zcdmarket_Zguest_Qhtm:{
    action_D_Zcdmarket_Zselezione_Qasp,
    linkpg_D_Zcdmarket_Zindex_Qhtm};

  State=linkpg & Paging = linkpg_D_Zcdmarket_Zindex_Qhtm:{
    linkpg_D_Zcdmarket_Zuser_Qhtm,
    linkpg_D_Zcdmarket_Zreg_Qasp,
    linkpg_D_Zcdmarket_Zguest_Qhtm};

  State=linkpg & Paging = linkpg_D_Zcdmarket_Zacquisto_Qasp:{
    D_Zcdmarket_Zacquisto_Qasp};

  State=action & Paging = action_D_Zcdmarket_Zautenticazione_Qasp:{
    linkpg_D_Zcdmarket_Zuser_Qhtm,
    action_D_Zcdmarket_Zautenticazione_Qasp,
    action_D_Zcdmarket_Zselezione_Qasp,
    linkpg_D_Zcdmarket_Zindex_Qhtm};
  State=action & Paging = action_D_Zcdmarket_Zreg1_Qasp:{
    action_D_Zcdmarket_Zreg_Qasp,
    action_D_Zcdmarket_Zselezione1_Qasp,
    action_D_Zcdmarket_Zselezione_Qasp,
    linkpg_D_Zcdmarket_Zindex_Qhtm};

  State=action & Paging = action_D_Zcdmarket_Zselezione_Qasp:{
    action_D_Zcdmarket_Zselezione1_Qasp,
    linkpg_D_Zcdmarket_Zindex_Qhtm};

  State=action & Paging = action_D_Zcdmarket_Zreg_Qasp:{
    action_D_Zcdmarket_Zreg1_Qasp,
    linkpg_D_Zcdmarket_Zindex_Qhtm};

  State=action & Paging = action_D_Zcdmarket_Zselezione1_Qasp:{
```



```
    action_D_Zcdmarket_Zpreventivo_Qasp,  
    action_D_Zcdmarket_Zreg_Qasp,  
    linkpg_D_Zcdmarket_Zindex_Qhtm};  
  
    State=action & Paging = action_D_Zcdmarket_Zpreventivo_Qasp:{  
        action_D_Zcdmarket_Zacquisto_Qasp,  
        action_D_Zcdmarket_Zselezione_Qasp,  
        linkpg_D_Zcdmarket_Zindex_Qhtm};  
  
    State=action & Paging =  
    action_D_Zcdmarket_Zacquisto_Qasp:{linkpg_D_Zcdmarket_Zacquisto_Qasp};  
  
1:Paging;  
esac;  
  
next(State):=case  
    State=ancora:page;  
    State=linkpg:page;  
    State=action:page;  
  
    State=page & (Paging in {  
        linkpg_D_Zcdmarket_Zindex_Qhtm,  
        linkpg_D_Zcdmarket_Zuser_Qhtm,  
        linkpg_D_Zcdmarket_Zreg_Qasp,  
        linkpg_D_Zcdmarket_Zguest_Qhtm,  
        action_D_Zcdmarket_Zautenticazione_Qasp,  
        linkpg_D_Zcdmarket_Zindex_Qhtm,  
        action_D_Zcdmarket_Zreg1_Qasp,  
        action_D_Zcdmarket_Zselezione_Qasp,  
        action_D_Zcdmarket_Zreg_Qasp,  
        action_D_Zcdmarket_Zselezione1_Qasp,  
        action_D_Zcdmarket_Zpreventivo_Qasp,  
        action_D_Zcdmarket_Zacquisto_Qasp,  
        linkpg_D_Zcdmarket_Zacquisto_Qasp}):link;  
  
    State=link & (next(Paging) in {  
        linkpg_D_Zcdmarket_Zindex_Qhtm,  
        linkpg_D_Zcdmarket_Zuser_Qhtm,  
        linkpg_D_Zcdmarket_Zreg_Qasp,  
        linkpg_D_Zcdmarket_Zguest_Qhtm,  
        linkpg_D_Zcdmarket_Zacquisto_Qasp}):linkpg;  
  
    State=link & (next(Paging) in {  
        action_D_Zcdmarket_Zautenticazione_Qasp,  
        action_D_Zcdmarket_Zreg1_Qasp,  
        action_D_Zcdmarket_Zselezione_Qasp,  
        action_D_Zcdmarket_Zreg_Qasp,
```

```
action_D_Zcdmarket_Zselezione1_Qasp,  
action_D_Zcdmarket_Zpreventivo_Qasp,  
action_D_Zcdmarket_Zacquisto_Qasp}):action;
```

```
1:State;  
esac;
```

C.3 Codice *SMV online*

MODULE main

VAR

```
State:{page,link};
```

```
Paging:{  
  cdmarket_Zindex_Qhtm_6,  
  cdmarket_Zuser_Qhtm_7,  
  cdmarket_Zreg_Qasp_8,  
  cdmarket_Zguest_Qhtm_9,  
  cdmarket_Zautenticazione_Qasp_10,  
  cdmarket_Zautenticazione_Qasp_1,  
  cdmarket_Zautenticazione_Qasp_2,  
  cdmarket_Zreg1_Qasp_11,  
  cdmarket_Zselezione_Qasp_12,  
  cdmarket_Zautenticazione_Qasp_13,  
  cdmarket_Zselezione_Qasp_14,  
  cdmarket_Zautenticazione_Qasp_15,  
  cdmarket_Zselezione_Qasp_16,  
  cdmarket_Zreg_Qasp_17,  
  cdmarket_Zselezione1_Qasp_18,  
  cdmarket_Zselezione1_Qasp_19,  
  cdmarket_Zselezione1_Qasp_3,  
  cdmarket_Zselezione1_Qasp_20,  
  cdmarket_Zselezione1_Qasp_4,  
  cdmarket_Zreg_Qasp_21,  
  cdmarket_Zpreventivo_Qasp_22,  
  cdmarket_Zpreventivo_Qasp_5,  
  cdmarket_Zpreventivo_Qasp_23,  
  cdmarket_Zacquisto_Qasp_24,  
  cdmarket_Zselezione_Qasp_25,  
  cdmarket_Zacquisto_Qasp_26,  
  cdmarket_Zacquisto_Qasp_27,  
  cdmarket_Zacquisto_Qasp_28};
```

```
cli: { felice, nrdllgrz, nonvalido, null, visitatore };
nome: { null };
cognome: { null };
datanasc: { null };
luogonasc: { null };
cf: { null };
email: { null };
via: { null };
nciv: { null };
citta: { null };
prv: { null };
cap: { null };
tit: { null };
aut: { null, Ligabue, Litfiba, U2, Queen };
storia: { visualizza_Hi_Htuo_i_Hacquisti_Hprecedenti, null };
art: { 0004, 0005, 0008, 0003, 0006, 0014, null };
qta: { 3, 0, 4, 2, null };
sco: { 27, 28, 25, null };
```

ASSIGN

```
init(Paging) := cdmarket_Zindex_Qhtm_6;
init(State) := page;
init(cli) := null;
init(nome) := null;
init(cognome) := null;
init(datanasc) := null;
init(luogonasc) := null;
init(cf) := null;
init(email) := null;
init(via) := null;
init(nciv) := null;
init(citta) := null;
init(prov) := null;
init(cap) := null;
init(tit) := null;
init(aut) := null;
init(storia) := null;
init(art) := null;
init(qta) := null;
init(sco) := null;

next(Paging) := case
  State = link & Paging = cdmarket_Zindex_Qhtm_6: {
    cdmarket_Zuser_Qhtm_7,
```

```
cdmarket_Zreg_Qasp_8,  
cdmarket_Zguest_Qhtm_9};
```

```
State=link & Paging=cdmarket_Zuser_Qhtm_7  
& cli=felice:{  
cdmarket_Zautenticazione_Qasp_10,  
cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zuser_Qhtm_7  
& cli=nrdllgrz:{  
cdmarket_Zautenticazione_Qasp_1,  
cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zuser_Qhtm_7  
& cli=nonvalido:{  
cdmarket_Zautenticazione_Qasp_2,  
cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zreg_Qasp_8  
& nome=null  
& cognome=null  
& datanasc=null  
& luogonasc=null  
& cf=null  
& email=null  
& via=null  
& nciv=null  
& citta=null  
& prov=null  
& cap=null  
& cli=null  
& tit=null  
& aut=null:{  
cdmarket_Zreg1_Qasp_11,  
cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zguest_Qhtm_9  
& cli=visitatore:{  
cdmarket_Zselezione_Qasp_12,  
cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zautenticazione_Qasp_10  
& storia=visualizza_Hi_HtuoI_Hacquisti_Hprecedenti  
& cli=felice:{  
cdmarket_Zautenticazione_Qasp_13,  
cdmarket_Zselezione_Qasp_14,  
cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zautenticazione_Qasp_1
& storia=visualizza_Hi_HtuoI_Hacquisti_Hprecedenti
& cli=nrDllgrz:{
    cdmarket_Zautenticazione_Qasp_15,
    cdmarket_Zselezione_Qasp_16,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zautenticazione_Qasp_2:{
    cdmarket_Zuser_Qhtm_7,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zreg1_Qasp_11:{
    cdmarket_Zreg_Qasp_17,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zselezione_Qasp_12
& tit=null
& aut=null
& cli=visitatore:{
    cdmarket_Zselezione1_Qasp_18,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zautenticazione_Qasp_13
& cli=felice:{
    cdmarket_Zselezione_Qasp_14,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zselezione_Qasp_14
& tit=null
& aut=Ligabue
& cli=felice:{
    cdmarket_Zselezione1_Qasp_19,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zselezione_Qasp_14
& tit=null
& aut=Litfiba
& cli=felice:{
    cdmarket_Zselezione1_Qasp_3,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zautenticazione_Qasp_15
& cli=nrDllgrz:{
    cdmarket_Zselezione_Qasp_16,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zselezione_Qasp_16
& tit=null
& aut=U2
& cli=nrdllgrz:{
    cdmarket_Zselezione1_Qasp_20,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zselezione_Qasp_16
& tit=null
& aut=Queen
& cli=nrdllgrz:{
    cdmarket_Zselezione1_Qasp_4,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zreg_Qasp_17
& nome=null
& cognome=null
& datanasc=null
& luogonasc=null
& cf=null
& email=null
& via=null
& nciv=null
& citta=null
& prov=null
& cap=null
& cli=null
& tit=null
& aut=null:{
    cdmarket_Zreg1_Qasp_11,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zselezione1_Qasp_18
& tit=null
& aut=null:{
    cdmarket_Zreg_Qasp_21,
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zselezione1_Qasp_19
& art=0004
& qta=3
& cli=felice:{
    cdmarket_Zindex_Qhtm_6};
```

```
State=link & Paging=cdmarket_Zselezione1_Qasp_3
& art=0003
& qta=2
```

```
& cli=felice:{
  cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zselezione1_Qasp_20:{
  cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zselezione1_Qasp_4:{
  cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zreg_Qasp_21
  & nome=null
  & cognome=null
  & datanasc=null
  & luogonasc=null
  & cf=null
  & email=null
  & via=null
  & nciv=null
  & citta=null
  & prov=null
  & cap=null
  & cli=null
  & tit=null
  & aut=null:{
    cdmarket_Zreg1_Qasp_11,
    cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zpreventivo_Qasp_22
  & art=0004
  & qta=3
  & sco=27
  & cli=felice:{
    cdmarket_Zacquisto_Qasp_24,
    cdmarket_Zselezione_Qasp_25,
    cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zpreventivo_Qasp_5
  & art=0005
  & qta=4
  & sco=28
  & cli=felice:{
    cdmarket_Zacquisto_Qasp_26,
    cdmarket_Zselezione_Qasp_25,
    cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zpreventivo_Qasp_23
  & art=0003
```

```
& qta=2
& sco=25
& cli=felice:{
    cdmarket_Zacquisto_Qasp_27,
    cdmarket_Zselezione_Qasp_25,
    cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zacquisto_Qasp_24:{
    cdmarket_Zacquisto_Qasp_28};

State=link & Paging=cdmarket_Zselezione_Qasp_25
& tit=null
& aut=Ligabue
& cli=felice:{
    cdmarket_Zselezione1_Qasp_19,
    cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zselezione_Qasp_25
& tit=null
& aut=Litfiba
& cli=felice:{
    cdmarket_Zselezione1_Qasp_3,
    cdmarket_Zindex_Qhtm_6};

State=link & Paging=cdmarket_Zacquisto_Qasp_26:{
    cdmarket_Zacquisto_Qasp_28};

State=link & Paging=cdmarket_Zacquisto_Qasp_27:{
    cdmarket_Zacquisto_Qasp_28};

State=link & Paging=cdmarket_Zacquisto_Qasp_28:{
    cdmarket_Zacquisto_Qasp_28};

1:Paging;
esac;

next(State):=case
    State=link:page;

    State=page & (Paging in {
        cdmarket_Zuser_Qhtm_7,
```



```
cdmarket_Zreg_Qasp_8,  
cdmarket_Zguest_Qhtm_9,  
cdmarket_Zindex_Qhtm_6,  
cdmarket_Zautenticazione_Qasp_10,  
cdmarket_Zautenticazione_Qasp_1,  
cdmarket_Zautenticazione_Qasp_2,  
cdmarket_Zreg1_Qasp_11,  
cdmarket_Zselezione_Qasp_12,  
cdmarket_Zautenticazione_Qasp_13,  
cdmarket_Zselezione_Qasp_14,  
cdmarket_Zautenticazione_Qasp_15,  
cdmarket_Zselezione_Qasp_16,  
cdmarket_Zreg_Qasp_17,  
cdmarket_Zselezione1_Qasp_18,  
cdmarket_Zselezione1_Qasp_19,  
cdmarket_Zselezione1_Qasp_3,  
cdmarket_Zselezione1_Qasp_20,  
cdmarket_Zselezione1_Qasp_4,  
cdmarket_Zreg_Qasp_21,  
cdmarket_Zpreventivo_Qasp_22,  
cdmarket_Zpreventivo_Qasp_5,  
cdmarket_Zpreventivo_Qasp_23,  
cdmarket_Zacquisto_Qasp_24,  
cdmarket_Zselezione_Qasp_25,  
cdmarket_Zacquisto_Qasp_26,  
cdmarket_Zacquisto_Qasp_27,  
cdmarket_Zacquisto_Qasp_28}):link;
```

```
1:State;  
esac;
```

```
next(cli):=case  
  State=link & next(Paging) in {cdmarket_Zuser_Qhtm_7}:{felice ,nrllgrz  
,nonvalido};  
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};  
  State=link & next(Paging) in {cdmarket_Zguest_Qhtm_9}:{visitatore};  
  State=link & next(Paging) in {cdmarket_Zautenticazione_Qasp_10}:{felice};  
  State=link & next(Paging) in {cdmarket_Zautenticazione_Qasp_1}:{nrllgrz};  
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_12}:{visitatore};  
  State=link & next(Paging) in {cdmarket_Zautenticazione_Qasp_13}:{felice};  
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_14}:{felice};
```

```
State=link & next(Paging) in {cdmarket_Zautenticazione_Qasp_15}:{nrdllgrz};
State=link & next(Paging) in {cdmarket_Zselezione_Qasp_16}:{nrdllgrz};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
State=link & next(Paging) in {cdmarket_Zselezione1_Qasp_19}:{felice};
State=link & next(Paging) in {cdmarket_Zselezione1_Qasp_3}:{felice};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_22}:{felice};
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_5}:{felice};
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_23}:{felice};
State=link & next(Paging) in {cdmarket_Zselezione_Qasp_25}:{felice};
1:cli;
esac;
```

```
next(nome):=case
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
  1:nome;
esac;
```

```
next(cognome):=case
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
  1:cognome;
esac;
```

```
next(datanasc):=case
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
  1:datanasc;
esac;
```

```
next(luogonasc):=case
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
  1:luogonasc;
esac;
```

```
next(cf):=case
```

```
State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
1:cf;
esac;
```

```
next(email):=case
State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
1:email;
esac;
```

```
next(via):=case
State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
1:via;
esac;
```

```
next(nciv):=case
State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
1:nciv;
esac;
```

```
next(citta):=case
State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
1:citta;
esac;
```

```
next(prov):=case
State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
1:prov;
esac;
```

```
next(cap):=case
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
  1:cap;
esac;
```

```
next(tit):=case
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_12}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_14}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_16}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione1_Qasp_18}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_25}:{null};
  1:tit;
esac;
```

```
next(aut):=case
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_8}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_12}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_14}:{Ligabue ,Litfiba};
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_16}:{U2 ,Queen};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_17}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione1_Qasp_18}:{null};
  State=link & next(Paging) in {cdmarket_Zreg_Qasp_21}:{null};
  State=link & next(Paging) in {cdmarket_Zselezione_Qasp_25}:{Ligabue ,Litfiba};
  1:aut;
esac;
```

```
next(storia):=case
  State=link & next(Paging) in
{cdmarket_Zautenticazione_Qasp_10}:{visualizza_Hi_Htuo_Hacquisti_Hprecedenti};
  State=link & next(Paging) in
{cdmarket_Zautenticazione_Qasp_1}:{visualizza_Hi_Htuo_Hacquisti_Hprecedenti};
  1:storia;
esac;
```

```
next(art):=case
  State=link & next(Paging) in {cdmarket_Zselezione1_Qasp_19}:{0004 ,0005 ,0008};
  State=link & next(Paging) in {cdmarket_Zselezione1_Qasp_3}:{0003 ,0006 ,0014};
  State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_22}:{0004};

  State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_5}:{0005};
```

```
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_23}:{0003};  
l:art;  
esac;
```

```
next(qta):=case  
State=link & next(Paging) in {cdmarket_Zselezione1_Qasp_19}:{3,0,4};  
State=link & next(Paging) in {cdmarket_Zselezione1_Qasp_3}:{2,0};  
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_22}:{3};  
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_5}:{4};  
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_23}:{2};  
l:qta;  
esac;
```

```
next(sco):=case  
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_22}:{27};  
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_5}:{28};  
State=link & next(Paging) in {cdmarket_Zpreventivo_Qasp_23}:{25};  
l:sco;  
esac;
```

Conclusioni

Descrizione sintetica delle caratteristiche del progetto

La tesi propone, attraverso l'utilizzo di metodi formali, la verifica di siti web sia statici che dinamici. È possibile quindi analizzare il contenuto, oltre che delle pagine html, anche delle pagine asp, jsp e php.

Una volta analizzato il contenuto di queste pagine, il sistema realizzato determina dei modelli a stati finiti (eseguiti in codice SMV) su cui, mediante il model checking (SMV.exe) sono verificate delle proprietà fornite dall'utente in un linguaggio opportuno (basato sulla logica CTL). Queste verifiche si basano sulla struttura fisica delle pagine del sito (parsing offline) e sulla struttura logica delle pagine del sito (parsing online). Avendo a disposizione queste strutture è possibile anche prelevare altre informazioni per eseguire altre verifiche senza l'utilizzo dei metodi formali (verifiche senza l'utilizzo della logica CTL).

Si sono introdotte, inoltre, alcune verifiche su siti basati sul commercio elettronico il cui scopo è quello di verificare se vengono rispettate alcuni requisiti per il buon funzionamento.

Suggerimenti per ampliare le funzionalità del progetto

Al progetto implementato si possono ampliare le funzionalità. Infatti nella fase di parsing, oltre alle pagine che sono state analizzate, è possibile comprendere anche le pagine xml e gestire con più precisione le caratteristiche dei frame. Nella fase di verifica è possibile inserire nuove schede, in cui è possibile implementare, mediante nuove formule basate sulla logica CTL, altre verifiche automatiche; nello stesso tempo è possibile migliorare le schede già realizzate. Molto interessante sarebbe la realizzazione di un metodo che permetta l'autocomposizione di schede (che risulteranno poi permanenti) che gestiscono una verifica a cui viene associata una formula CTL

impostata dall'utente (questo deve prevedere anche l'autocomposizione di *ComboBox*, di *TextBox* di *Label* e così via).

L'argomento più vasto da poter ampliare è quello che riguarda le verifiche dei siti dinamici che si basano su determinate tipologie. Nel sistema realizzato, si è introdotto quest'argomento eseguendo le verifiche su siti basati sul commercio elettronico. Sull'idea di questa realizzazione, è possibile eseguire le verifiche su altri tipi di siti: per esempio siti bancari, siti che gestiscono le e-mail, siti che gestiscono le informazioni e così via.

Altre verifiche possibili da fare, richiedono l'analisi accurata dello script all'interno delle pagine. In questo modo si possono gestire più informazioni nel modello a stati finiti, ma nello stesso tempo si restringe la tipologia delle pagine che si possono analizzare.

Per esempio, per le pagine asp, è possibile analizzare nel codice al suo interno, le variabili *session* e *application* (utilizzando anche i cookie) ottenendo informazioni più accurate per quanto riguarda le variabili e le informazioni dell'utente (tra cui la password). Procedendo in questo modo, non si riuscirebbero però a gestire appieno le pagine php o le pagine jsp perché gestiscono queste informazioni in maniera differente: bisogna quindi analizzare singolarmente queste caratteristiche per ogni tipo di pagine.

BIBLIOGRAFIA

- [1] Java mattone dopo mattone, <http://www.java-net.tv>, il primo manuale di java in italiano rilasciato con logica open source, Massimiliano Tarquini.
- [2] Analisi e progettazione object oriented unified modelling language (UML), a cura del prof. Giacomo Piscitelli, Dipartimento di Elettrotecnica ed Elettronica Politecnico di Bari, Bari giugno 2002.
- [3] A Specification Pattern System, <http://www.cis.ksu.edu/santos/spec-patterns>, Sept 1998 by Matt Dwyer (dwyer@cis.ksu.edu)
- [4] E. Di Sciascio, F.M.Donini, M.Mongiello, G.Piscitelli. Verification of Web Sites Design by Model Cheking. In *Atti del congresso AICA 2000*, Taormina settembre 2000.
- [5] I packages del JDK, A. Cisternino, MokaByte maggio 2001 n. 50
- [6] Manuale pratico di Java, Aiello, Bettini,Dozio,Gini,Giovannini, Mancini, Molino, Morello, Puliti, Rossigni, Venditti, 2001 Hops Libri, <http://www.mokabyte.it>
- [7] Java 2 I fondamenti – seconda edizione, Mc Graw Hill, Giugno 2001,
- [8] K. L. McMillan, “The SMV system (for SMV version 2.5.4)”, 2001
- [9] Vito Egidio Mosca, “Verifica con metodologie formali delle proprietà dei siti web”, Tesi di Laurea, Politecnico di Bari, Anno Accademico 2000-2001
- [10] Leonardo Lisanti, “Verifica con metodologie formali delle proprietà dei siti web”, Tesi di Laurea, Politecnico di Bari, Anno Accademico 2001-2002
- [11] <http://www-cad.eecs.berkeley.edu/~kenmcmil>, Home page di K. L. McMillan
- [12] <http://nusmv.irst.itc.it/>, NuSMV: a new symbolic model checker