UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
**Facoltà di Ingegneria**
**Corso di Studi in Ingegneria Informatica**



tesi di laurea

# Metodi per la classificazione di features di pagine Web ai fini del reverse engineering

Anno Accademico 2005/2006

**relatore**
Ch.ma prof. Anna Rita Fasolino

**correlatore**
Ch.mo prof. Porfirio Tramontana

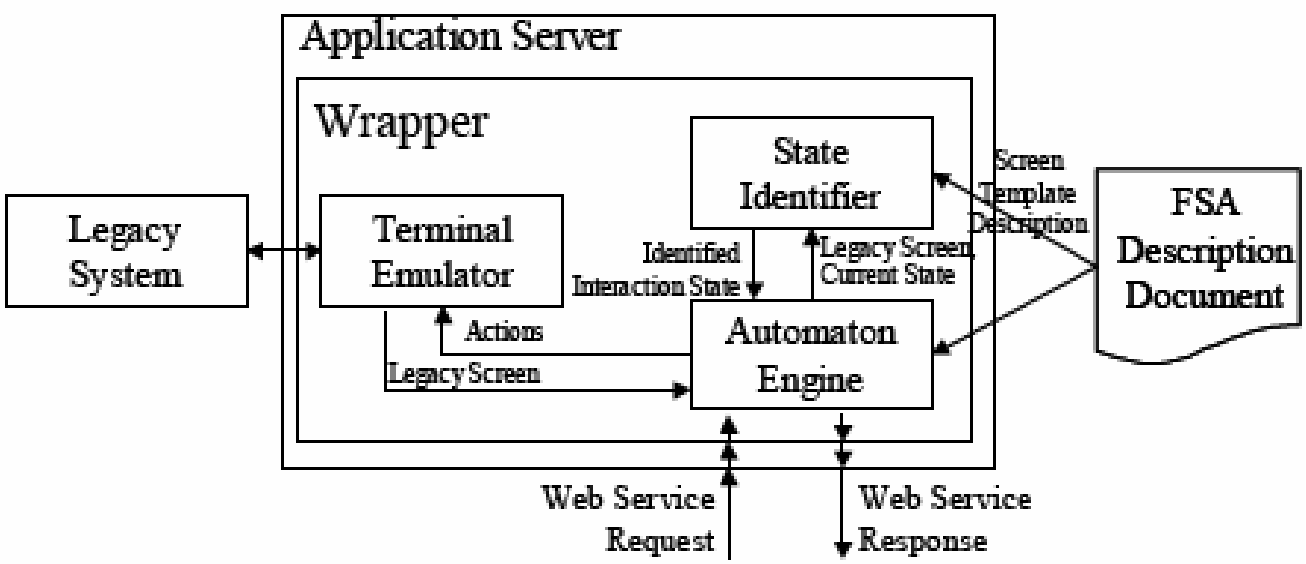
**candidato**
Fattorusso Massimiliano
Matr. 534/930

# Il contesto

• **Obiettivo**: Esportare i servizi di una Web Application per poterli accedere come un Web Service. In altre parole trasformare il paradigma interattivo in quello richiesta/risposta tipico dei Web Services

• **Soluzione adottata**: tecnica **black box di reverse engineering basata su wrapping.** Ovvero incapsulando il sistema originale con uno strato software che: mostra una nuova interfaccia nascondendo quella originale ed interagisce con il sistema legacy.

• Costruzione di un automa a stati finiti non deterministico che rappresenta il modello di interazione con la Web Application; ogni stato d'interazione dell'automa restituisce schermate HTML.



# Il problema

- E' possibile prevedere come risponde la Web Application se sollecitata da determinati input?
  Risposta: No, il comportamento di una Web Application **dipende** dal suo stato interno.

- Il Wrapper deve essere progettato in modo da stabilire **dinamicamente** quale scenario di interazione è correntemente in esecuzione nella Web Application ed eseguire il comportamento richiesto da essa.

- Difficoltà nell'analisi e nella conoscenza della Web Application, in assenza di un'appropriata documentazione;

- le BCPs (Built Client Page) costruite a partire da una server page possono essere molto diverse le une dalle altre.

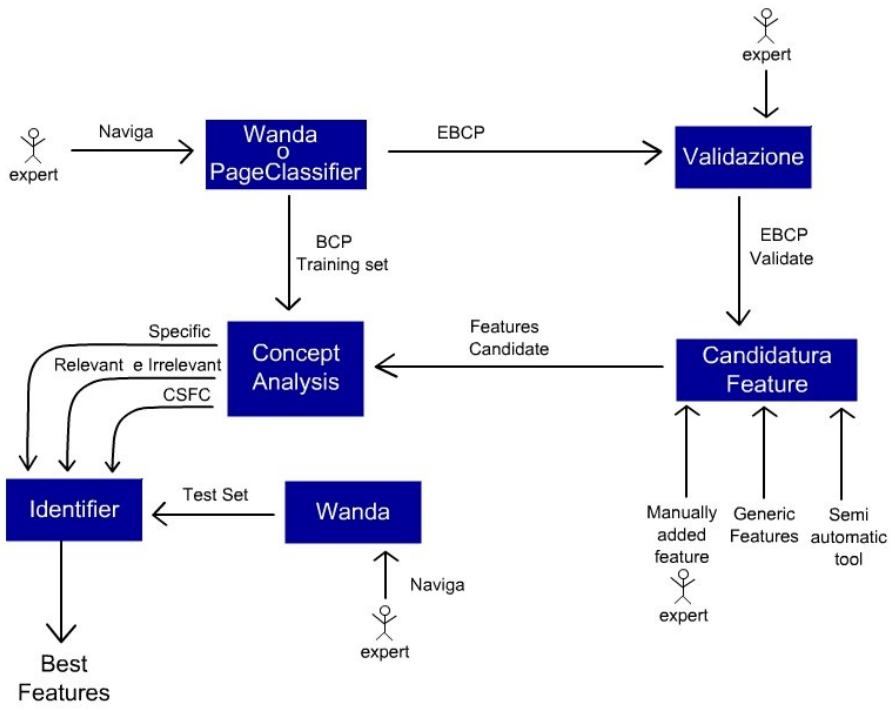
# La soluzione proposta

- Raggruppare le BCPs con caratteristiche simili in classi di equivalenza, o clusters, caratterizzate da un set di features comuni; in modo da poter definire per il wrapper un comportamento comune da adottare per tutte le BCPs appartenenti alla medesima classe di equivalenza

- Si può utilizzare una singola pagina equivalente per rappresentare una particolare classe di BCPs ed, in questo modo, ridurre lo sforzo di comprensione.

# Il contributo apportato

- Il lavoro svolto consiste nella ricerca di metodi per la classificazione delle features di pagine Web, con la finalità di rendere minimo l'insieme di features in grado di riconoscere a quali classi di equivalenza appartengono gli screen ritornati da una web application.




## Le fasi della soluzione

1. Proporre un insieme massimo di features (con generatori semi automatici di features, con un scelta empirica, usando features generiche,etc.)
2. Procurarsi un insieme di training di pagine già classificate
3. Ridurre l'insieme di features verso il minimo insieme in grado di discriminare correttamente tutte le pagine del training set.
4. Generare regole di discriminazione basandosi sull'insieme minimo di features
5. Validare le features discriminanti su un test set

# La tipologia delle features

- E' stato proposto un insieme completo di cinque tipologie di features, come indicato da Eisenbarth, Koschke e Simon in "Locating Features in Source Code".

1. ***Specific*** : attributo comune a tutti e soli gli oggetti della classe.
2. ***Relevant*** : attributo comune a tutti gli oggetti della classe ed anche a qualche altro oggetto non appartenente alla classe
3. ***Conditionally Specific*** : attributo presente solo in alcuni oggetti, i quali appartengono alla classe.
4. ***Shared*** : attributo presente in oggetti, alcuni dei quali appartengono alla classe.
5. ***Irrelevant*** : attributo non presente in nessun oggetto della classe.

Questa classificazione è di supporto alla candidatura delle features per la generazione della condizione booleana di appartenenza per ciascuna EBCP.

*Nota:*
Nel contesto in cui è stato effettuato lo studio va considerato che per:
Attributo si intende una feature
Oggetto si intende una BCP (Built Client Page)
Classe si intende una Classe di Equivalenza o EBCP (Equivalent Built Client Page)

# La candidatura delle features

•Per ogni classe di equivalenza viene proposta una espressione booleana di appartenenza rispettando le seguenti regole:

1. Primo candidato gli ***SPECIFIC***:

   **•se per una classe di equivalenza è presente almeno una feature che sia SPECIFIC essa da sola è sufficiente per la descrizione.**

   **•se per una classe di equivalenza sono presenti più features cha siano SPECIFIC, occorre, per la composizione dell'espressione booleana caratterizzante, fare una *and* logica tra le features.**

2. Secondo candidato le ***RELEVANT AND*** $\overline{IRRELEVANT}$

   **•qualora non sono presenti features SPECIFIC si ricorre alla *and* logica delle features RELEVANT in *and* con la *and* logica degli IRRELEVANT negati**

L'utilizzo degli IRRELEVANT negati è inutile qualora si effettui un'analisi anche sulle **features negate**, perché all'interno di esse sono contenute implicitamente anche le informazioni che si potrebbero ottenere negando gli attributi IRRELEVANT.

3. Terzo candidato le ***CONDITIONALLY SPECIFIC (CSFC)***:

   **•qualora non siano presenti neanche features RELEVANT si procede con la *or* logica tra le features CONDITIONALLY SPECIFIC, nel tentativo di riprodurre almeno parzialmente il potere discriminante delle features SPECIFIC**

# Panoramica sul framework Concept Analyzer

Il framework ***ConceptAnalyzer*** è stato progettato e sviluppato come supporto alla ricerca di un insieme minimo di features discriminanti tra diverse classi di equivalenza relative ad una stessa server page. Esso consente di:

1. Classificare pagine Web mediante il tool PageClassifier
2. Convertire pagine HTML verso lo standard XHTML
3. Manipolare features espresse mediante linguaggio XPath
4. Applicare di features alle pagine XHTML
5. Analizzare i risultati mediante la Concept Analisys, usando il tool Conexp

## Impostazione della sperimentazione

Focalizziamo la nostra attenzione sulla **sperimentazione**:

• Il primo passo è la scelta di una web application (*bookstore*) ed una server page.

•Si provvede, quindi, ad una classificazione in classi di equivalenza delle BCPs generate dalla server page; tale classificazione è ad opera “dell’esperto” che la effettua in base a criteri di similitudine e/o di significato, a seconda delle esigenze discriminanti necessarie.

•Si applicano le features presenti sotto forma di queries XPath alle BCPs costruite a partire dalla server page.

•Si analizzano i risultati ottenuti (presentati in un Lattice)mediante la Concep Analysis. Nel caso in cui la discriminazione non sia soddisfacente si procede nell’applicazione di un nuovo set di features, iterando il passo precedente.

•Si deducono le espressioni booleane minime ed esatte per ogni classe di equivalenza.

# La Sperimentazione (Login.php)1/3

Le BCPs classificate e coinvolte nella sperimentazione

| Build Client Pages | Clusters |
|---|---|
| Login.php.8.html | Login index |
| Login.php.9.html | Login corretta |
| Login.php.10.html | Login errata |
| Login.php.17.html | Login errata |
| Login.php.18.html | Login corretta |
| Login.php.19.html | Login corretta |

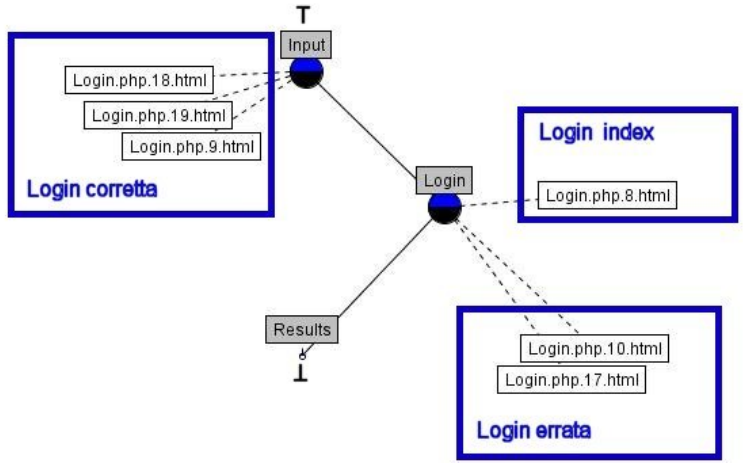
1. Applichiamo un primo set di features

| | ServerPage | Descrizione | XPathQuery |
|---|---|---|---|
| | Login.php | Input | boolean(//input) |
| | Login.php | Login | boolean(//input[@name='Login']) |
| | Login.php | Results | boolean(//a[@id='Results']) |

Il contesto ottenuto mediante ConceptAnalyzer

| A | B | C | D |
|---|---|---|---|
| | Login | Input | Results |
| Login.php.10.html | X | X | |
| Login.php.17.html | X | X | |
| Login.php.18.html | | X | |
| Login.php.19.html | | X | |
| Login.php.8.html | X | X | |
| Login.php.9.html | | X | |

Il lattice prodotto con Conexp




Le espressioni booleane ottenute manualmente

Login Corretta --> INPUT AND $\overline{\text{LOGIN}}$ AND $\overline{\text{RESULTS}}$

Login Index --> INPUT AND LOGIN AND $\overline{\text{RESULTS}}$

Login Errata --> INPUT AND LOGIN AND $\overline{\text{RESULTS}}$

È evidente, come si può anche denotare dal lattice, che con l'insieme di features utilizzato non può esserci una corretta discriminazione tra le classi di equivalenza in quanto esistono due espressioni booleane identiche per clusters diversi

# La Sperimentazione (Login.php)2/3

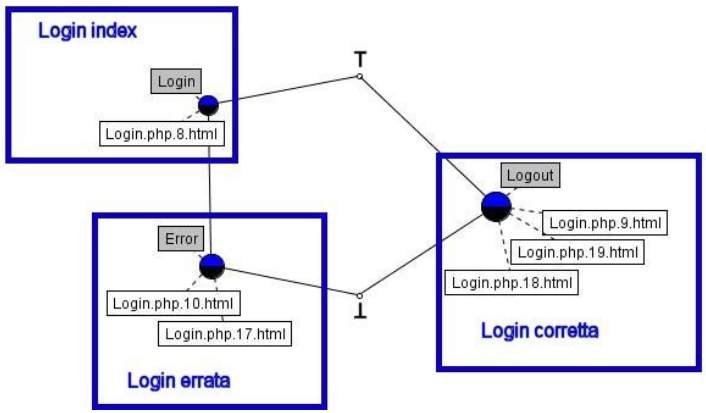
2. Proponiamo un nuovo insieme di features

| | ServerPage | Descrizione | XPathQuery |
|---|---|---|---|
| | Login.php | Error | #error,incorrect,errore,impossibile# |
| | Login.php | Login | boolean(//input[@name='Login']) |
| | Login.php | Logout | boolean(//input[@value='Logout']) |

Il contesto ottenuto mediante ConceptAnalyzer

| A | B | C | D |
|---|---|---|---|
| | Error | Login | Logout |
| Login.php.10.html | X | X | |
| Login.php.17.html | X | X | |
| Login.php.18.html | | | X |
| Login.php.19.html | | | X |
| Login.php.8.html | | X | |
| Login.php.9.html | | | X |

Il lattice prodotto con Conexp




Dal lattice deduciamo che:

per la classe di equivalenza (EBCP) *Login corretta*:
- l'attributo *Login* è **Irrelevant**
- l'attributo *Error* è **Irrelevant**
- l'attributo *Logout* è **Specific**

per la classe di equivalenza *Login index*:
- l'attributo *Login* è **Relevant**
- l'attributo *Error* è **Irrelevant**
- l'attributo *Logout* è **Irrelevant**

per la classe di equivalenza *Login errata*:
- l'attributo *Login* è **Relevant**
- l'attributo *Error* è **Specific**
- l'attributo *Logout* è **Irrelevant**

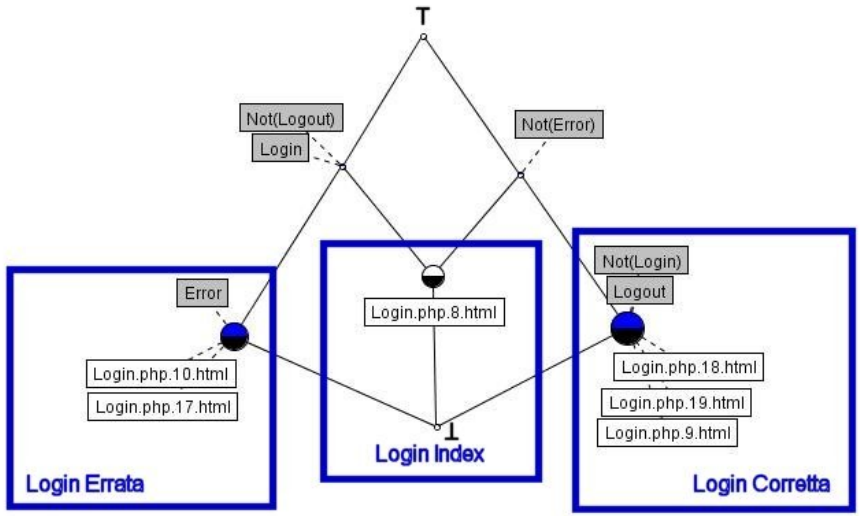
Da cui derivano le espressioni booleane carateristiche

- Login Corretta --> LOGOUT
- Login Index --> LOGIN and $\overline{\text{LOGOUT}}$ and $\overline{\text{ERROR}}$
- Login Errata -->ERROR

# La Sperimentazione (Login.php)3/3

3. Nel tentativo di ridurre ulteriormente le espressioni proposte o trovarne delle nuove equivalenti, proviamo ad applicare oltre alle tre features sopra indicate anche i loro negati

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|
| | Error | Login | Logout | Not(Error) | Not(Login) | Not(Logout) |
| Login.php.10.html | X | X | | | | X |
| Login.php.17.html | X | X | | | | X |
| Login.php.18.html | | | X | X | X | |
| Login.php.19.html | | | X | X | X | |
| Login.php.8.html | | X | | X | | X |
| Login.php.9.html | | | X | X | X | |




per la classe di equivalenza *Login errata*:
l'attributo *Login* è **Relevant**
l'attributo *Not(Login)* è **Irrelevant**
l'attributo *Error* è **Specific**
l'attributo Not(Error) è **Irrelevant**
l'attributo *Logout* è **Irrelevant**
l'attributo Not(Logout) è **Relevant**

per la classe di equivalenza *Login index*:
l'attributo *Login* è
l'attributo *Not(Lo*
l'attributo *Error* è
l'attributo Not(Err
l'attributo *Logout*
l'attributo Not(Lo

per la classe di equivalenza (EBCP) *Login corretta*:
l'attributo *Login* è **Irrelevant**
l'attributo *Not(Login)* è **Specific**
l'attributo *Error* è **Irrelevant**
l'attributo Not(Error) è **Relevant**
l'attributo *Logout* è **Specific**
l'attributo Not(Logout) è **Irrelevant**

Usando le implicazioni di Duquenne-Guigues (tramite il tool Conexp) per ridurne le espressioni booleane otteniamo:

Login Corretta --> LOGOUT and NOT(LOGIN)

Login Index --> LOGIN and NOT(ERROR)
*oppure*
NOT(LOGOUT) and NOT(ERROR)

Login Errata -->ERROR

**Sviluppi futuri:** Automatizzare la creazione delle espressioni booleane caratteristiche per ogni classe di equivalenza.