Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Magistrale in Scienze di Internet

GENERAZIONE AUTOMATICA DI APPLICAZIONI WEB: LO STUDIO DELL'USABILITA'

Tesi di Laurea in Interazione Persona-Computer

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
MICHELE CACACE

III Sessione
Anno Accademico 2010/11

CAPITOLO 1 INTRODUZIONE **5**

CAPITOLO 2 CONTESTO SCIENTIFICO **11**

LA SCELTA DEL FRAMEWORK DI MODELLAZIONE: CONFRONTO FRA WEBRATIO E SYMFONY **13**

CAPITOLO 3 CONTESTO TECNOLOGICO **17**

MODEL-DRIVEN ENGINEERING **17**

WEBML **20**
CONCETTI BASE DI WEBML 21
WEBRATIO **23**
TRASFORMAZIONI DEL MODELLO E GENERAZIONE DEL CODICE 25
EXTJS **26**
MODEL VIEW CONTROLLER 28
COMPONENT PRINCIPALI 29
IL MODELLO CAO=S **31**
L'ANALISI DEI REQUISITI IN CAO=S 34
GLI ATTORI IN CAO=S 37
I CONCETTI IN CAO=S 40
LE OPERAZIONI IN CAO=S 42
LE STRUTTURE IN CAO=S 45

CAPITOLO 4 CREAZIONE DI APPLICAZIONI WEB TRAMITE METODOLOGIA MDE **47**

LA MODELLAZIONE CON WEBRATIO **49**
MODELLAZIONE DEL *DATA MODEL* 50
MODELLAZIONE DEL *WEB MODEL* 51
LA SCRITTURA DI UN FILE XML 55

CAPITOLO 5 DA BUCABO ALLA GENERAZIONE DEI PROTOTIPI **59**

BUCABO: LINEE GUIDA **59**
BUCABO: STRUMENTI **61**
IL PRIMO PROTOTIPO GENERATO: SISTEMA UNIVERSITARIO **62**
DESCRIZIONE DEL SISTEMA 65
TOUR STUDENTE 69
TOUR PROFESSORE 71
IL SECONDO PROTOTIPO GENERATO: GESTIONE MAGAZZINO **73**

CAPITOLO 6 CONCLUSIONI **77**

BIBLIOGRAFIA **81**

Capitolo 1
Introduzione

L'elaborato di tesi che segue si occupa di diversi aspetti concernenti la generazione di applicazioni web usabili. Le tecniche su cui si baserà la produzione delle applicazioni sono di tipo *Model-Driven,* implementate mediante un linguaggio di notazione visuale per la modellazione chiamato WebML. Questo tipo di implementazione ha permesso di realizzare applicazioni di qualità in modo rapido e dotate di alte caratteristiche di usabilità e di adattabilità a un determinato contesto.

Le metodologie di sviluppo di applicazioni model-driven si pongono come supporto agli sviluppatori. Esse permettono di progettare il software in maniera veloce, formalizzando modelli e generando automaticamente il codice a partire dal modello. Questi approcci però (come vedremo all'interno del capitolo 2), sono carenti di modelli di progettazione dell'interazione fra l'utente e l'applicazione da generare. Infatti, gli utenti di un'applicazione hanno caratteristiche differenti, che determinano modi differenti di interagire con essa. Viene quindi richiesta una progettazione dell'interazione che tenga conto di queste caratteristiche.

Per poter svolgere un progetto che rispettasse le richieste di rapidità di implementazione e alta usabilità è stato sfruttato il linguaggio di modellazione visuale WebML. WebML è basato su standard di grande diffusione come il modello entità-relazione e UML. Questo linguaggio, appartenendo alla famiglia MDE, permette di specificare applicazioni Web complesse in modo indipendente dalla piattaforma. Esso permette di specificare il modello dei dati di un'applicazione Web e uno o più modelli ipertestuali che possono essere basati sulle specifiche dei processi di business. Con WebML è stato possibile

definire una qualunque applicazione Web passando attraverso la modellazione di due modelli principali: il *data-model* (che consiste in un diagramma entità-relazione per la specifica dei dati sottostanti l'applicazione Web) e il *web-model* (che consente la definizione di pagine e la loro organizzazione interna in termini di componenti).

Per la realizzazione dei modelli WebML è stato utilizzato il tool WebRatio, un plugin di *Eclipse* che rappresenta una nuova generazione di strumenti per lo sviluppo di applicazioni Web secondo le metodologie di MDE. Attraverso l'utilizzo di questo strumento si è potuto implementare un'applicazione di esempio che mostra come sia possibile integrare una parte server Java con un client sviluppato in modo indipendente dal modello. A livello server abbiamo JavaEE, mentre a livello client l'applicazione è in grado di garantire un'alta usabilità grazie all'utilizzo di alcuni widget implementati con il framework *JavaScript ExtJs*.

ExtJs permette con poche righe di codice di creare una varietà di widget avanzati, con la sicurezza che l'applicazione verrà correttamente visualizzata su tutti i browser in circolazione. Per garantire un alto livello di astrazione e per permettere dunque al client di interfacciarsi con il più ampio numero di domini applicativi è stata utilizzata l'architettura MVC (Model View Controller) che ha permesso di organizzare al meglio il progetto e di realizzare dei widget completamente indipendenti dal contesto di utilizzo, in modo tale da poter essere riutilizzati.

Per quanto riguarda le varie tecniche di progettazione delle interfacce incentrate sugli utenti, ne esistono diverse che rientrano nella macro-categoria chiamata User-Centered. La progettazione User-Centered si divide principalmente in due teorie di design, una orientata ai task che l'utente deve svolgere (Task-Oriented) e una basata sugli obiettivi dell'utente (Goal-Oriented). Il modello di design Goal-Oriented risulta più efficace per lo scopo di progettare

l'interazione orientata agli utenti in quanto, rispetto al modello Task-Oriented, mira a soddisfare gli obiettivi personali dell'utente, piuttosto che aiutarlo negli specifici compiti che deve svolgere all'interno dell'applicazione. In questo modello di design l'utente viene descritto in fase di analisi attraverso la creazione di personaggi in maniera dettagliata. In seguito vengono creati degli scenari che raccontano come il personaggio interagisce col sistema. L'utilizzo degli scenari permette quindi di capire quali sono le informazioni necessarie affinché i personaggi possano soddisfare i loro scopi, permettendo così di sviluppare i prototipi e le interfacce dell'applicazione.

Il design Goal-Oriented offre uno strumento di progettazione molto efficace che richiede però una progettazione molto complessa e costosa, poco adatta ad essere utilizzata in team medio-piccoli, dove non sono presenti le competenze e gli strumenti adeguati per una progettazione così accurata. Inoltre, la progettazione con questo modello non fornisce delle soluzioni concrete per lo sviluppo del software. Per questo motivo lo scopo del seguente lavoro è quello di realizzare una *meta-applicazione* in grado di definire alcune linee guida per la generazione di applicazioni Web a partire da un modello. Le applicazioni generate riusciranno ad interfacciarsi con un client usabile, soddisfacendo pienamente gli utenti del sistema.

Vista la complessità del modello Goal-Oriented tradizionale, si è scelto di utilizzarne uno semplificato e rivolto proprio a team di dimensioni ridotte con poche risorse a disposizione: CAO=S. Questo modello, sviluppato all'interno del dipartimento di Scienze dell'informazione dell'Università di Bologna, riduce l'analisi degli utenti alle sole caratteristiche che hanno un impatto importante sull'interazione e senza un'analisi accurata degli obiettivi personali degli utenti. Le soluzioni che CAO=S utilizza per aumentare l'usabilità delle applicazioni consistono nell'agire su alcune caratteristiche come l'utilità attesa, la completezza dei contenuti, la comprensibilità del vocabolario (eliminando i termini di difficile interpretazione).

CAO=S è l'acronimo di Concetti+Attori+Operazioni=Strutture dati. Questo modello di design prevede l'acquisizione delle prime tre componenti (Concetti, Attori e Operazioni) attraverso la compilazione di questionari da parte del team di sviluppo, dall'analisi di queste componenti si realizzano le Strutture dati (S).

Una volta stabilito il contesto scientifico e quello tecnologico sono passato alla definizione della meta-applicazione che è stata "battezzata" BuCaBO.

Da un parte BuCaBO offre linee guida che indirizzano la modellazione del server, dall'altra fornisce alcuni file che permettono al server di interfacciarsi con un client ExtJS usabile. Il lato server è stato modellato e generato a partire dal tool di modellazione WebML WebRatio. Il risultato della generazione è un server di tipo *Java standard* creato tramite l'utilizzo di alcune *unit* (le componenti fondamentali di WebRatio) che permettono di compiere operazioni sui dati definiti all'interno del data-model. Il server, oltre a gestire la parte di accesso al sistema e la persistenza dei dati, si occupa di scrivere dei file XML in cui si trovano i dati e le informazioni impiegate dal client. Per gestire il contenuto dei file e la loro collocazione (URL) sono state ampiamente sfruttate due unit fornite da WebRatio ai suoi sviluppatori: la *XML Out Unit* e la *Script Unit*.

Oltre ai file XML appena descritti, BuCaBO offre altri due strumenti fondamentali per permettere la comunicazione e lo scambio di dati fra il client e il server: 1) i *template* che WebRatio permette di definire per ogni pagina in cui sono state definite le caratteristiche della pagina specifica (come ad esempio il titolo e le componenti grafiche - o widget - offerte dal client). 2) I file *JavaScript* che vengono richiamati dai template e che si occupano della configurazione e dei servizi offerti (come ad esempio le espressioni xPath da effettuare sui vari file XML).

Per dimostrare il funzionamento di BuCaBO è stato implementato un prototipo funzionante che si occupa della gestione di un sistema universitario.
L'applicazione permette l'accesso (tramite login) al sistema da parte due principali tipologie di utenti: i professori e gli studenti. Gli studenti possono compiere le tipiche operazioni offerte da un sistema di gestione degli esami universitari (ad esempio consultare il piano di studi, registrarsi ad un appello, ricercare un professore, etc). Per quanto riguarda i professori, questi sono in grado di visionare gli studenti iscritti ai propri corsi, verbalizzare un esame, compiere operazioni sugli appelli, etc.

Dopo aver creato il primo prototipo, per favorire una migliore comprensione di BuCaBO, si è deciso di implementarne un secondo, cambiando totalmente ambito di applicazione. È stata così creata una semplice applicazione gestionale in cui, al posto di professori e studenti, sono state implementate le entità *prodotto*, *magazzino* e *fornitore*. Così facendo abbiamo dimostrato come con BuCaBO sia possibile creare applicazioni Web in grado di offrire sia un alto livello di usabilità e che rapidi tempi di messa in esercizio.
È forse con un pizzico di orgoglio che concludo affermando che BuCaBO è in grado di gestire il trade-off che separa usabilità e rapidità di implementazione.

Capitolo 2
Contesto scientifico

Nell'ultimo decennio abbiamo assistito ad una crescita esponenziale del numero di applicazioni Web a contenuto informativo (indicate anche con l'acronimo WSI - *Web System Information*). Inizialmente queste applicazioni venivano semplicemente utilizzate come un mezzo per diffondere informazioni. Successivamente, la quantità di informazioni che queste applicazioni hanno dovuto sostenere insieme alla crescita del numero di utenti del sistema, ha portato all'aumento della complessità che le Web application si sono trovate a gestire [LO03]. Questa complessità ha fatto sì che nascesse una nuova disciplina chiama *Web Engineering* (WE). La WE promuove e definisce alcuni principi scientifici, ingegneristici e manageriali per lo sviluppo e la manutenzione dei sistemi *web-based* [GM01]. La Web Engineering tuttavia non sembra aver risolto molti dei problemi che continuano ad affliggere diverse Web application. In alcuni studi [KMP04] sono stati evidenziati i 5 principali tipi di problemi che affliggono le Web application:

- Fallimento nell'individuare le necessità di business;
- Ritardi rispetto allo *scheduling* del progetto;
- Costi eccessivi rispetto al budget;
- Mancanza di funzionalità richieste;
- Scarsa qualità dei *deliverable*.

In [MT09] ci si domanda perché molti committenti di WSI siano ancora restii nei confronti di metodologie di sviluppo basate su MDE (come ad esempio WebML [ABB08]) che in diversi modi, come osservato all'interno del capitolo riguardante il contesto tecnologico, affrontano questi problemi e, in qualche modo, aiutano lo sviluppatore. Alcuni autori [JMS07] sostengono che questa diffidenza nasca dal fatto che le applicazioni Web a contenuto informativo richiedano trattamenti speciali (e quindi soluzioni sviluppate su misura) nei confronti di alcuni attributi di qualità come l'usabilità e l'accessibilità. Gli autori individuano, dunque, una mancanza di fiducia da parte degli *stakeholder* nei confronti degli standard di usabilità che un'applicazione Web sviluppata con metodologia MDE è in grado di offrire.

Possiamo con sicurezza affermare che l'usabilità è quindi un attributo critico di qualità per il successo di un'applicazione Web. Una grande quantità di studi [Don01] hanno evidenziato i benefici che si possono ottenere attraverso un sistema usabile come ad esempio l'aumento di produttività e di soddisfazione degli utenti oppure la riduzione dei costi di *training* e di documentazione.

L'usabilità viene ormai considerata un requisito funzionale dell'applicazione [FB04] e, dunque, possiamo definirla come un parametro fondamentale per la qualità. L'usabilità ha ottenuto un'importanza sempre più elevata nei metodi di Web Engineering, addirittura maggiore che nelle convenzionali applicazioni *Desktop* [HLM06]. Diversi lavori sono stati proposti come metodi per misurare l'usabilità, un esempio sono quelli sviluppati da Olsina [OR02].

All'interno del progetto di tesi l'usabilità non è stato l'unico obiettivo da perseguire. Un altro aspetto di fondamentale importanza ha assunto un ruolo chiave: la possibilità di sviluppare applicazioni di qualità in modo rapido e senza (grossi) errori. Come, affermato in letteratura [LPG10], risulta essere uno dei problemi principali all'interno della Web Engineering. I problemi legati al *trade-off* che viene a crearsi fra velocità di implementazione e qualità

dell'applicazione prodotta nascono dal fatto che i team di sviluppo possono trovarsi di fronte a specifiche che cambiano rapidamente nel tempo. In letteratura [LWE01] questo fenomeno viene anche indicata come la sindrome *"permanent beta"*. Per dominare il trade-off e per liberarsi da questa sindrome ho pensato che l'approccio MDE potesse essere lo strumento che meglio permettessi di soddisfare i requisiti di usabilità e di rapidità di implementazione.

Grazie all'approccio *Model-Driven* è stato possibile aumentare il grado di astrazione cercando, così, di raggiungere quegli alti livelli di qualità descritti in [HT07] [Low03] e che dovrebbero contraddistinguere una buona applicazione Web sviluppata a partire da un approccio MDE.

La scelta del framework di modellazione: confronto fra WebRatio e Symfony

Come già detto in questa sede, lo sviluppo di appicazioni, in questo periodo della storia dell'informatica, si concentra sulle applicazioni Web. La crescente necessità di nuove applicazioni basate sul Web provoca la necessità di nuovi metodi di sviluppo rapidi e di conoscenze professionali che siano in grado di realizzare questo tipo di software. I committenti si aspettano una sempre maggiore rapidità nella realizzazione delle loro richieste tenendo però costanti i requisiti di sicurezza, scalabilità, usabilità ed efficienza [KZ10].

Durante lo svolgimento del progetto di tesi mi sono trovato davanti alla scelta del framework di modellazione da sfruttare per la realizzazione dei modelli che sono alla base dell'applicazione Web prodotta. Andremo ora ad effettuare un

confronto fra due prodotti che sono stati individuati come potenziali candidati e cioè WebRatio e Symfony.

Anche se apparentemente diverse, queste due applicazioni sono simili se viste dall'interno e seguono alcune regole base di design per le applicazioni Web: il pattern architetturale *Model, View, Controller*. In MVC il Model garantisce l'accesso ai dati, la View si cura della prestazione dei dati e il Controller contiene la *business logic* che permette la comunicazione fra i primi due.

Symfony[1] rappresenta un approccio alla programmazione per lo sviluppo di applicazioni Web. La maggior parte del lavoro che si svolge con questo tool è di codifica in PHP. WebRatio[2], invece, rappresenta un approccio al design per lo sviluppo di applicazioni Web.

La scelta è ricaduta sul tool WebRatio che introduce il linguaggio di modellazione visuale chiamato WebML. I fattori che hanno portato alla scelta di WebRatio rispetto a Symfony sono fondamentalmente legati ad alcuni fattori come la possibilità che offre WebRatio di effettuare in maniera rapida e veloce la sincronizzazione con il database. La sincronizzazione con il database permette di:

- Sincronizzare il *data-model* (cioè il diagramma ER in cui vi sono tutte le informazioni che l'applicazione Web deve gestire) con una entità per volta (se necessario) permettendo quindi di modificare il modello dell'applicazione senza perdere le tabelle contenute nel database;

- Mostrare gli elementi del data model che non sono ancora stati sincronizzati;

[1] http://www.symfony-project.org/. Consultato in data 21 febbraio 2012
[2] http://www.webratio.com/portal/homePage.do. Consultato in data 21 febbraio 2012

- Selezionare delle entità e delle relazione ed importarle nel database.

In tutte queste operazioni WebRatio fornisce un *wizard* che aiuta e guida lo sviluppatore permettendo di risparmiare grandi quantità di tempo di lavoro. Un altro punto di forza presentato da WebRatio è la possibilità, tramite un tipo particolare di *units*, di selezionare dei dati e di scriverli all'interno di un file XML. Con questi file si è implementata la comunicazione fra client e server rendendo questa operazione del tutto indipendente dal dominio di applicazione.

Capitolo 3
Contesto tecnologico

Model-Driven Engineering

Il progetto di tesi che è stato affrontato ha visto come obiettivo la creazione di applicazioni web sviluppate secondo la metodologia Model-Driven Engineering. Questo tipo di approccio vede come fondamentale il ruolo del modello (cioè la rappresentazione astratta delle attività e delle conoscenze che caratterizzano il dominio di applicazione). In MDE la modellazione assume di conseguenza una posizione di maggior riguardo rispetto allo sviluppo, all'implementazione e alla codifica di algoritmi funzionali al software.

Negli ultimi anni abbiamo assistito (e stiamo tutt'ora assistendo) ad un aumento del numero di progetti software e applicazioni web che vengono implementante seguendo la metodologia MDE.

Questo è legato a diversi fattori:

- MDE permette di incrementare la produttività grazie alla riduzione dei tempi di codifica. Secondo una stima realizzata dall'*Object Management Group* è possibile avere dei risparmi che possono raggiungere il 273%.

	HAND WRITTEN LOC	COST/ LOC	TIME	TOTAL COST	PERCENT SAVINGS
Traditional U.S. Based Development	177,963	$ 34.83	32 Months	$ 6,198,812	N/A
With MDE	65,457	$ 34.83	12 Months	$ 2,280,000	272%

Fig 1. Tabella di confronto fra l'approccio allo sviluppo tradizionale e MDE[3]

- MDE permette di massimizzare la compatibilità fra sistemi. Con questo approccio si definiscono, inizialmente, le funzionalità del sistema utilizzando un modello totalmente indipendente dalla piattaforma (il cosiddetto **PIM** - *Platform Indipendent Model)*. Solo dopo la fase di modellazione del PIM verrà utilizzato un appropriato linguaggio specifico di dominio (**DSL** - *Domain Specific Lenguage*) per arrivare alla traduzione del PIM in uno o più modelli specifici di piattaforma (**PSM** - *Platfrorm Specific Model*) che i computer sono in grado di eseguire. PSM può utilizzare diversi DLS o un generale GPL (*General Purpose Language* - ad esempio Java, C++, PHP, etc). È importante notare che esistono diversi tool automatici che sono in grado di effettuare questa traduzione permettendo grandi risparmi. Un PIM può dare origine a diversi PSM per permettere all'applicazione di:

    - essere utilizzata su piattaforme con diverso sistema operativo;

    - poter funzionare anche nel caso di sistemi distribuiti, cioè nel caso in cui diversi componenti sono eseguiti su piattaforme o livelli diversi.

[3] Fonte: http://www.omg.org/mda/mda_files/M1Global.htm (consultato in data 29 febbraio 2012)

- Semplificazione del processo di design. Uno dei principale scopi in MDE è la separazione del design dall'architettura. Separare queste due permette allo sviluppatore di scegliere il meglio in entrambi i domini.

- Nella fase di design ci si occupa dei requisiti funzionali (cioè dei casi d'uso);

- Nella fase di studio dell'architettura ci si occupa delle infrastrutture attraverso le quali i requisiti non funzionali (come la scalabilità, l'affidabilità e le performance) sono realizzati.

- Aumento del ciclo di vita del software. Con MDE, la pretesa di OMG è quella di avere sistemi con un ciclo di vita più lungo (si parla addirittura di 20 anni[4]).

La Model-Driven Engineering si riferisce quindi ad una serie di approcci allo sviluppo, incentrati sulla modellazione del software. In alcuni casi i modelli sono costruiti con un certo livello di dettaglio e poi codificati a mano in seguito. Altre volte, però, i modelli sono costruiti in modo tale da permettere la codifica a partire direttamente dal modello stesso.

Un paradigma di modellazione per MDE è considerato efficace, se i suoi modelli hanno senso dal punto di vista di un utente che abbia familiarità con il dominio e se possono servire come base per i sistemi di attuazione. I modelli sono sviluppati attraverso una comunicazione ampia fra product manager, progettisti, sviluppatori e utenti del dominio applicazione.

[4] Fonte: http://www.omg.org/mda/mda_files/M1Global.htm (consultato in data 29 febbraio 2012)

Molta della popolarità raggiunta da MDE è legata a UML (*Unified Modeling Language*) e alla quantità di tool che permettono la generazione di codice a partire da un modello UML. Le tecnologie MDE che pongono una maggiore attenzione all'architettura e all'automazione della fase di codifica permettono elevati livelli di astrazione nello sviluppo del software. Questa astrazione permette quindi di avere modelli più semplici con una maggiore attenzione al dominio del problema.

WebML

Data la natura del progetto di tesi, dopo un periodo in cui si è ricercato il linguaggio di modellazione più idoneo alle nostre esigenze, la nostra scelta è ricaduta sul *Web Modelling Language*: una notazione visuale per la specifica di composizione e navigazione di applicazioni ipertestuali per il web[AB08].

WebML è basato su standard di grande diffusione come il modello entità-relazione e UML. Questo linguaggio, appartenendo alla famiglia MDE, permette inoltre di specificare applicazioni Web complesse in modo indipendente dalla piattaforma. Esso permette di specificare il modello di dati di un'applicazione Web e uno o più modelli ipertestuali che possono essere basati sulle specifiche dei processi di business.

L'approccio WebML per lo sviluppo di applicazioni Web è costituito da diverse fasi. Ispirato da modello a spirale di Boehm, il processo WebML è applicato in modo iterativo e incrementale, le fasi vengono ripetute e raffinate fino al momento in cui i risultati soddisfano i requisiti richiesti.

Il linguaggio WebML è un *Domain Specific Language* (DSL) per la progettazione di applicazioni web.

Concetti base di WebML

In questa sezione andiamo a descrivere i concetti base di WebML con particolare attenzione al *data model* e all'*hypertext model* [CFB03].

- **WebML data model.** Per la specifica dei dati sottostanti l'applicazione Web, WebML sfrutta il modello di dati **ER** (*Entity-Relationship* equivalente al diagramma delle classi primitive UML). Il modello di dati può anche includere la specifica dei dati calcolati. Gli attributi calcolati, le entità e le relazioni sono dette *derived* e le loro regole di computazione sono specificate come espressioni logiche scritte utilizzando linguaggi dichiarativi come OQL (*Object Query Language* - standard di linguaggio di interrogazione per basi di dati a oggetti sul modello di SQL).

- **WebML Hypertext Model.** Il modello ipertesto permette la definizione dell'interfaccia front-end dell'applicazione Web. Esso consente la definizione di pagine e la loro organizzazione interna in termini di componenti (chiamate *content-unit*) per visualizzare il contenuto. Esso permette la definizione di collegamenti tra pagine e *content-unit* che supportano informazioni di localizzazione e navigazione. I componenti possono specificare le operazioni, come ad esempio la gestione dei contenuti e le procedure di *login*/*logout* dell'utente (chiamate *operation-unit)*. Una *site-view* è un particolare ipertesto, progettato per soddisfare uno specifico insieme di requisiti. È costituita da delle aree, che sono le sezioni principali dell'ipertesto e comprende in modo ricorsivo altre sub-aree o pagine. Le pagine sono i contenitori reali di informazioni

trasmesse all'utente.

Pagine all'interno di un'area o di una *site-view* possono essere di tre tipi:

- home page ("H") è l'indirizzo di default della *site-view;*
- pagina di default ("D") è quella presentata di default quando si accede all'area che la contiene;
- una *landmark-page* ("L") è raggiungibile da tutte le altre pagine o aree all'interno della *site-view* che la contiene

Le pagine sono composte da *content-unit*, che sono le componenti elementari che pubblicano pezzi di informazioni all'interno delle pagine. In particolare, le *data-unit* rappresentano alcuni degli attributi di una certa istanza di un'entità; *multidata-unit* rappresentano alcuni degli attributi di una serie di istanze di entità, le *index-unit* presentano una lista di chiavi descrittive di un insieme di istanze di entità e consentono la selezione di una di loro; le *entry-unit* infine sono dei *form* che permettono di raccogliere valori di input inseriti dall'utente.

Unità e pagine sono collegate da link, formando così un ipertesto.
Comportamenti diversi possono essere specificati a partire da tipologie diverse di link:

- *Link automatico*. Contrassegnato come "A", è quello di default che effettua un collegamento di tipo "navigazione" e viene seguito in assenza di interazione di un utente quando questo accede alla pagina;
- Un *collegamento di trasporto* (rappresentato con una freccia tratteggiata), viene utilizzato solo per passare informazioni di contesto da un'unità ad un'altra.

WebML offre primitive aggiuntive per esprimere le operazioni di aggiornamento, come ad esempio la creazione, l'eliminazione, la modifica di un'istanza di un'entità (rappresentate attraverso la *create, delete* e *modify unit*), l'aggiunta o l'eliminazione di una relazione tra due istanze (rappresentato rispettivamente dalla *connect* o *disconnect unit*). Ogni operazione può avere due tipi di link in uscita:

- *OK link*: è il link che viene seguito quando l'operazione va a buon fine;
- *KO link*: che viene eseguito quando l'operazione non riesce.

La metodologia WebML è pienamente supportata dal tool *WebRatio*, un plugin di *Eclipse* che rappresenta una nuova generazione di strumenti per lo sviluppo di applicazioni Web che seguono le metodologie di MDE.

WebML è accompagnato dal software CASE WebRatio, che genera automaticamente applicazioni Web dinamiche da diagrammi WebML[AB08].

WebRatio

Rispetto al processo di sviluppo WebML, WebRatio copre le fasi di progettazione dati e progettazione ipertesto, e supporta l'implementazione attraverso la generazione automatica di database relazionali e di template di pagina per le piattaforme JSP e .NET[5].

[5] Fonte: http://www.webml.org/webml/page1.do?stu2.values=IT. Consultato in data 12 febbraio 2012.

WebRatio è progettato e realizzato da *Web Models Srl*, una società italiana costituita nel 2001 come spin-off del Politecnico di Milano. Web Models offre al mercato internazionale WebRatio e tutti i servizi ad esso correlati: supporto, consulenza, formazione e sviluppo di Web application[6].

WebRatio è stato implementato come un insieme di plug-in di *Eclipse*. Eclipse è un *framework* per IDE (*Integrated Development Enviroment* - cioè un software che aiuta i programmatori nello sviluppo del codice), nel senso che, oltre ad essere esso stesso un IDE, fornisce l'infrastrutture per la definizione di un nuovo IDE. Eclipse è un *framework open source* multi-piattaforma, eseguibile su Linux, Windows e Mac OS X.

Come detto in precedenza WebRatio supporta pienamente il metamodello WebML, comprese le più recenti estensioni per *workflow driven* per applicazioni e servizi Web.

La fase di progettazione in WebRatio è effettuata attraverso una GUI per la progettazione di applicazioni che comprende una serie di editor, un set di trasformatori e validatori del modello, e alcuni *workspace* per la gestione componenti.

I modelli vengono salvati come documenti XML. La GUI di WebRatio per la fase di design definisce una speciale prospettiva Eclipse di progettazione per migliorare e soddisfare le esigenze di modellazione visuale. Questa si compone di diversi pannelli, tra cui:

- gli editor per i diagrammi di modello che vengono utilizzati per la progettazione dei dati di modello WebML e per i modelli di ipertesto;

[6] Fonte: http://www.webratio.com/portal/contentPage/it/Chi%20siamo. Consultato in data 13 febbraio 2012

- gli editor di testo avanzati per la progettazione dei descrittori XML, componenti Java, e così via;

- gli editor per la specifica di nuovi componenti e per la definizione di proprietà appartenenti alle istanze dei componenti;

- i *Wizard* per il sostegno delle attività di progettazione più comuni (come ad esempio nuovi progetti);

- gli editor di documentazione per la documentazione del progetto generato.

Trasformazioni del modello e generazione del codice

I modelli WebML vengono trasformati con un approccio MDE per aumentare la produttività del processo di sviluppo.

Il generatore di codice di WebRatio produce applicazioni web in J2EE a partire da modelli WebML. Essi sono stati sviluppati utilizzando ANT, XSLT, e tecnologie Groovy.

Groovy è un linguaggio agile con una sintassi simile a Java e pienamente integrato all'interno della piattaforma Java.

WebRatio è in grado di generare automaticamente la documentazione dei componenti WebML e dei progetti sviluppati. Per ciascun componente o progetto, il progettista può specificare una serie di proprietà descrittive o documentali che vengono poi sfruttate da trasformazioni groovy per generare la

documentazione in diversi formati (PDF, HTML, RTF, e così via) e con diversi stili grafici.

Uno dei punti di forza di WebRatio è la sua estendibilità che permette di definire nuove unità WebML. Tuttavia, per esigenze semplici, lo sviluppo di nuove unità può non essere necessario, perché è disponibile una speciale unità generica di WebML denominata *Script Unit*. Questa unità può essere configurata tramite uno script Groovy e può essere immediatamente utilizzata nei modelli. Ogni volta che viene inserita una *script-unit* in un progetto Web, lo script file sorgente deve essere specificato. Scrivere uno script Groovy è un compito relativamente semplice e veloce poiché richiede solo una conoscenza di base di Java. Lo script che deve essere associato all'unità può comprendere alcuni aspetti peculiari: l'inizio dello script può contenere commenti che dichiarano gli ingressi e le uscite della *script-unit*, con una sintassi semplice:

```
// input = INPUT1 | INPUT2 | ...
// output = OUTPUT1 | OUTPUT2 | ...
```

Gli ingressi possono essere utilizzati direttamente come variabili nel codice script. Il tipo di variabile di ingresso dipende dal tipo di unità che fornisce l'*input* alla *scipt-unit*, e quindi la *script unit* deve essere adattata per diversi tipi di variabile di *input* (stringhe, array di stringhe, array di oggetti, oggetti e così via). Gli *output* devono essere restituiti attraverso l'istruzione *return*.

ExtJS

La necessità di utilizzare un linguaggio con una sintassi semplice e che funzioni per ogni browser ha fatto ricadere la scelta sul framework ExtJs. ExtJs semplifica la creazione di layout, anche molto complessi, che risultano

accattivanti per l'utente e cross-browser. Con poche righe di codice è possibile creare una varietà di widget avanzati, con la sicurezza che l'applicazione verrà correttamente visualizzata su tutti i browser in circolazione. Le caratteristiche di ExtJS che si adattano perfettamente allo scopo del progetto sono:

1. la scrittura del codice con un paradigma object-oriented;

2. un supporto cross-browser;

3. strutturazione del codice secondo il pattern Model-View-Controller; permette di
   organizzare al meglio il progetto e di realizzare dei widget completamente indipendenti dal contesto in modo tale da poter essere riutilizzati. L'utilizzo di questo pattern permette una gestione del codice che semplifica la generazione automatica e permette una buona scalabilità.

4. potenti librerie di widget, ovvero delle classi che realizzano componenti standard
   dell'interfaccia che sono facilmente estendibili;

Fig2. Pattern MVC[7]

[7] Fonte: http://www.sencha.com/products/extjs/whats-new-in-ext-js-4/

Un altro dei motivi che ha portato alla scelta di ExtJs è stata la sua piena compatibilità con i dati JSON. Ciò ha permesso di poter espandere ed adattare i vari widget al contesto e quindi al modello.

JSON (JavaScript Object Notation) è un formato per la serializzazione e l'interscambio di datifrequentemente usato in applicazioni AJAX. È facilmente comprensibile da agenti umani ed è facilmente generabile ed interpretabile da una macchina a stati. Pur derivando dalla sintassi usata da JavaScript per rappresentare gli oggetti, JSON è di fatto indipendente dal linguaggio di programmazione. I file JSON nel progetto hanno il ruolo fondamentale del passaggio dei parametri che permettono l'estensione dei widget, ovvero hanno permesso la creazione di viste diverse in base alle caratteristiche dell'utente.

ExtJs si adatta perfettamente alle esigenze del progetto poichè fornisce due strumenti principali per rappresentare le informazioni gestite dall'applicazione, i Modelli e gli Store. I primi contengono la definizione dei campi che rappresentano le entità e tramite gli Store gestiscono i rispettivi dati persistenti. Questa struttura permette di mappare ciascun Concetto di CAO=S in un Model ExtJs che contiene la descrizione dei fields che lo compongono.

Il vero punto di forza di ExtJS sono le viste. Le viste vengono indicate come *component* nell'architettura del framework, la loro particolarità è l'alto livello di usabilità che presentano. Particolarità che si sposa alla perfezione con un progetto di human-computer interaction, HCI.

Model View Controller

Il principio su cui si basa questo pattern è la separazione di un'applicazione in componenti che:

1. rappresentano i contenuti o dati dell'applicazione (Model);

2. producono ed elaborano i contenuti (Controller);

3. presentano i contenuti (View).

Il principale vantaggio è dato dalla separazione delle viste dai modelli facendoli interagire tramite il controller. Questo permette di creare e modificare viste senza dover cambiare il modello e rappresentare un modello per mezzo di viste multiple indipendenti.

Component principali

In questa sezione vedremo elencati i componenti principali, tra i quali: Grid panel, Tree panel e Viewport.

La **Grid Panel** è un componente estremamente versatile che fornisce un modo semplice per visualizzare, ordinare, raggruppare e modificare i dati. Con semplici e veloci operazioni permette di rappresentare gli stessi dati secondo prospettive diverse, permette all'utente di fare qualsiasi operazione che può essere fatta su una lista.

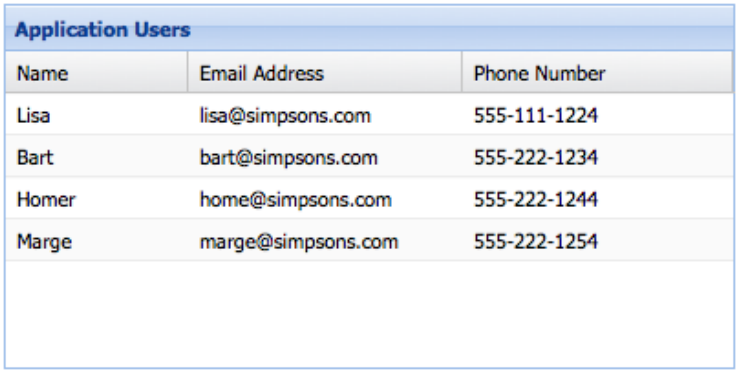

Fig 3. Esempio di Grid Panel[8]

[8] Fonte: http://docs.sencha.com/ext-js/4-0/#!/guide/grid. Consultato in data 29 febbraio 2012

Il **Tree Panel** è un ottimo strumento per la visualizzazione gerarchica dei dati all'interno dell'applicazione ed è ideale per la navigazione. Può essere generato in modo statico o dinamico e si compone di *root e child*.

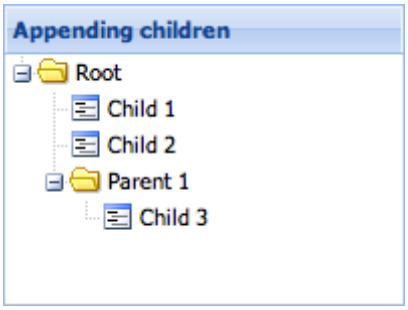

Fig 4. Esempio di Tree Panel[9]

Il **Viewport** è un contenitore che rappresenta l'area visualizzabile dell'applicazione. In modo automatico assume le dimensioni della finestra del browser e gestisce il suo ridimensionamento. E' possibile creare un solo Viewport in una pagina e come qualsiasi altro contenitore si occupa solo del posizionamento dei propri componenti figlio all'interno del layout.

[9] Fonte: http://docs.sencha.com/ext-js/4-0/#!/guide/tree. Consultato in data 29 febbraio 2012

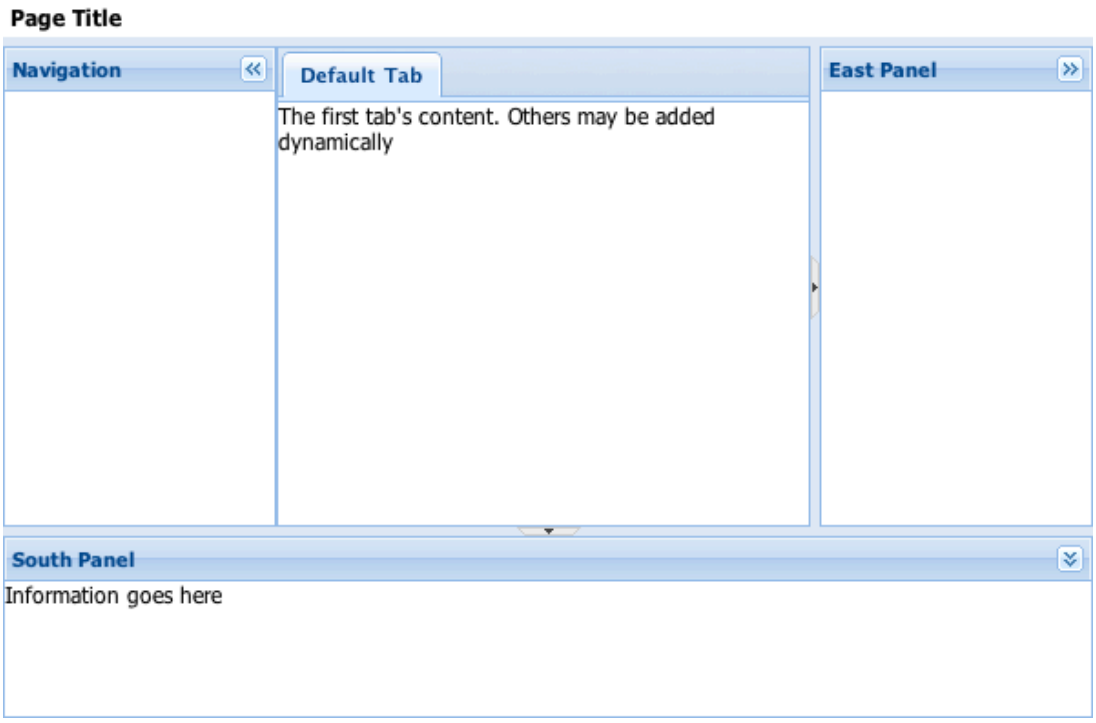

Fig 5. Esempio di Viewport

Un **Combobox** è una combinazione tra un tradizionale HTML text <input> field ed un campo <select>, l'utente può scrivere liberamente nel campo oppure scegliere dei valori da una dropdown selection list. La lista è popolata attraverso uno Store.

Il modello CAO=S

Per il progetto di tesi è stato adotto il modello di design Goal Oriented CAO=S [Zol10]. CAO=S è stato sviluppato all'interno del dipartimento di *Scienze dell'Informazione* dell'Università di Bologna. La finalità di questo modello è quella di permettere ad un team di sviluppo, senza esperienza nel campo dell'usabilità e senza disponibilità economiche, di avere delle linee guida da seguire per evitare quelli che sono gli errori di usabilità più comuni.

CAO=S è dunque un modello di processo per la formazione di sistemi applicativi usabili, si occupa di aspetti che appartengono al design dell'usabilità. L'idea è quella di prendere un processo descritto in un documento e passare dall'analisi astratta alla specifica dei criteri fondamentali di un'interfaccia.

CAO=S è l'acronimo di *Concetti, Attori, Operazioni generano Strutture*. Il modello è quindi basato sullo studio delle informazioni che devono essere manipolate all'interno dell'applicazione (cioè i *Concetti*) che sono messe a disposizione degli utenti (cioè gli *Attori*) i quali agiscono sul sistema mediante dei comandi (cioè le *Operazioni*). Se il modello è applicato in modo corretto e dunque se l'analisi è stata effettuata in maniera puntuale, le *Strutture* saranno generate per permettere un'elevata usabilità dell'applicazione.

Le *Strutture* proposte dal modello sono 3:

- *Viste* (maschere di visualizzazione e manipolazione delle proprietà dei concetti);

- *Strutture Dati* (modelli che persistono all'interno del sistema - ad esempio all'interno di un Database - e che vengono manipolati all'interno dell'applicazione. Le *Strutture Dati* sono utilizzate al fine di descrivere le proprietà dei concetti);

- *Navigazione* (meccanismi che permettono all'utente di passare da una *Vista* ad un'altra).

Per rendere il processo di design il più semplice possibile, CAO=S si basa su alcune idee chiave che sono state utilizzate nella fase di progettazione del progetto di tesi.
Queste possono essere riassunte in 4 punti fondamentali:

1. Adozione sistematica e automatica di tutte le linee guida ed i pattern di progettazione (indipendentemente da utenti e task).

2. Trasformazione della fase di analisi di utenti, task e obiettivi dai requisiti alla
   progettazione parametrica del progetto: tanto più accurate e corrette sono le informazioni su utenti, task e obiettivi, tanto maggiore sarà la garanzia di usabilità.

3. Identificazione solo delle caratteristiche degli utenti che hanno effettivamente un impatto noto sulla progettazione (ad esempio: competenze di dominio, alfabetizzazione informatica, livello di interesse, etc).

4. Generazione automatica di pattern di interfaccia (e magari anche di scheletri di
   interfaccia) dopo la compilazione di schede e questionari sui tre aspetti fondamentali del modello: analisi dei concetti, analisi degli attori, analisi delle operazioni da effettuare.

Come accennato precedentemente una delle finalità di CAO=S è di permettere al team di sviluppo di non incappare in errori che causano inusabilità. Questi errori possono nascere nel momento in cui i progettisti si focalizzano esclusivamente sui requisiti funzionali dell'applicazione ignorando alcuni aspetti che, invece, sono di estrema importanza per l'usabilità di un'applicazione.

Secondo CAO=S per evitare questi errori bisogna gestire:

- La distanza fra il modello espresso dal sistema e il modello atteso dall'utente. Questa distanza può essere di natura *linguistica* (se i termini non sono comprensibili, le spiegazioni sono insufficienti o l'output non è esplicativo) oppure *operazionale* (cioè se la navigazione è poco chiara o se risulta incoerente con le aspettative dell'utente).

- Le complicazioni procedurali. Questo significa che il numero di operazioni atomiche che può compiere l'utente sia proporzionato alle sue attese.

- La completezza delle viste. Un'applicazione risulterà non usabile se porzioni importanti delle informazioni utili all'utente non sono visualizzate, o non sono visualizzate insieme, o richiedono navigazione eccessiva per essere visualizzate.

L'analisi dei requisiti in CAO=S

Essendo CAO=S un modello di progettazione di tipo Goal-Oriented prevede come prima fase l'analisi dei requisiti. L'analisi dei requisiti, in CAO=S, può durare dai 20 minuti fino ad un massimo di 4 giorni. Lo scopo è quello di identificare le opinioni degli *stakeholder* e di metterle a confronto con quelle dei progettisti. È importante notare che se esistono diverse opinioni fra gli *stakeholder* queste devono essere confrontate attentamente per trovare una soluzione comune.

L'obiettivo principale dell'analisi dei requisiti è dunque la risoluzione dell'ambiguità per arrivare a descrivere le funzioni in termini immediatamente implementabili come le operazioni su input che generano output. Come strumento per eliminare le ambiguità si utilizzano tre strumenti fondamentali

che consistono in modelli astratti di descrizione delle informazioni:

1. il modello *Entity-Relationship*, in cui le informazioni sono descritte come entità che hanno attributi e relazioni. Questo modello fornisce un alto livello di astrazione per la rappresentazione concettuale dei dati ed è utilizzato nella prima fase di progettazione per capire come strutturare le basi di dati permettendo quindi coerenza fra il dominio di progetto e la schematizzazione concettuale.

2. Il *Class Diagram* di *UML*, in cui le informazioni sono espresse sotto forma di classi con legami quali associazioni, relazioni, aggregazioni e composizioni.

L'utilizzo di questi due modelli di astrazione per la rappresentazione dei dati permette ai progettisti di compiere strade diverse per arrivare allo stesso punto. La doppia via che viene percorsa porta:

- alla manipolazione delle informazioni disordinate dei concetti;

- alla riorganizzazione dei concetti;

- alla normalizzazione che avviene nel momento in cui si arriva alla descrizione delle entità e delle classi (che risulteranno totalmente prive di ambiguità).

Questo processo risulta essere uno dei maggiori fenomeni che portano complessità nell'interazione con le applicazioni moderne. È per questo motivo che CAO=S fornisce delle ulteriori linee guida per agevolare lo svolgimento dell'analisi dei requisiti:

- Registrare i cambiamenti che avvengono tra i concetti ottenuti in fase di raccolta dei requisiti (che potrebbero risultare grezzi e potenzialmente ambigui) e le strutture concettuali ottenute in fase di analisi (che invece sono più raffinate e non ambigue).

- Progettare le strutture in modo tale da trasmettere i concetti degli attori indipendentemente dal modello utilizzato.

- Progettare le operazioni eseguibili dall'utente sotto forma di manipolazione dei concetti degli attori e mapparle nelle funzioni ottenute dall'analisi dei requisiti funzionali.

Nella modellazione CAO=S che porta alla formalizzazione di concetti, attori, operazioni e strutture ci si può appoggiare a delle regole pratiche che aiutano a mantenere una distanza ridotta fra il mapping del modello e l'applicazione da generare. Queste regole di natura pratica sono elencate nella tabella sottostante.

Modello dell'attore	**Modello del team di progetto**
Concetti (frasi, sostantivi, aggettivi)	Strutture dati (classi, attributi, relazioni) (entità, attributi, relazioni)
Operazioni (verbi, avverbi)	Funzioni (input, elaborazioni, output)

Concludendo: è erroneo pensare che le scelte implementative riguardanti l'usabilità dipendano dalla caratterizzazione dell'utente. Queste dipendono dalla caratterizzazione dell'aspettativa che il team ha nei confronti dell'utente.

Gli Attori in CAO=S

Uno degli aspetti più importanti, preso in considerazione all'interno del progetto di tesi, è l'adattabilità che il *client* deve avere nei confronti delle caratteristiche dell'utente. In CAO=S il termine *attore* viene utilizzato per definire gli utenti che il sistema ospiterà.

L'attore, in generale, è uno *stakeholder* le cui idee aiutano i progettisti nel design del sistema. Possiamo distinguere fra:

- *Attori diretti*: sono quelli che utilizzeranno personalmente il sistema. È importante dividere gli attori diretti a seconda dei ruoli ricoperti all'interno del sistema, e dare loro un nome per evitare ambiguità. Gli attori diretti sono anche detti *utenti*.

- *Attori indiretti*: sono quelli che, anche se non utilizzeranno direttamente l'applicazione, per particolari motivi sono influenti sulle caratteristiche che il sistema dovrà avere. Esempi di attori indiretti sono il committente, il responsabile del sistema informatico e il legislatore.

In CAO=S le caratteristiche prese in considerazione sono solo quelle degli utenti (quindi degli attori diretti) poiché l'interfaccia grafica è rivolta a loro. Gli utenti sono divisi in categorie in base al tipo di ruolo che debbono (o non debbono) avere all'interno dell'applicazione (e quindi al tipo di interfacce da creare).

L'analisi goal-oriented prevede di identificare e caratterizzare in maniera dettagliata gli utenti, disegnando personaggi che ben li rappresentano. Per fare ciò, solitamente, ci si serve dell'*analisi etnografica* senza la quale i personaggi sono poco utili. Le analisi etnografiche portano sempre, come output, ad una

curva a campana. Tale curva tende asintoticamente a zero sia nella parte sinistra (valori nulli per la caratteristica X posseduta dagli utenti) che nella parte destra (valori alti per la caratteristica X). Il risultato di questa analisi porta dunque all'individuazione di una percentuale di utenza per la quale sarebbe utile implementare una determinata *feature* del sistema.

Questa analisi è indubbiamente di grande aiuto nella fase di design del progetto ma ha dei costi che possono essere onerosi. Per rendere la progettazione più semplice ma allo stesso tempo conservare un discreto livello di qualità, quello che bisogna fare è individuare almeno cinque o sei caratteristiche di base che abbiano un chiaro impatto sull'implementazione. Sulla base di queste proprietà verranno poi plasmati i personaggi.

Il risparmio a cui si può arrivare con questo tipo di analisi è notevole [Bac11]: si pensi che con l'analisi etnografica si arriva mediamente all'individuazione di 20 caratteristiche, alcune delle quali, però, sono prive di un impatto preciso sulla progettazione.

Le caratteristiche che CAO=S propone come fondamentali, che hanno un forte impatto sulle scelte implementative e che guidano la progettazione del sistema sono sei:

1. *Competenza di dominio*: capacità dell'attore di comprendere esattamente il dominio dell'applicazione, della specificità del contesto in cui opera e della terminologia specifica utilizzata.

2. *Competenza informatica specifica*: capacità dell'attore di comprendere il contesto informatico in cui opera specificamente l'applicazione, le consuetudini, il vocabolario e l'interazione tipica del mezzo (nel nostro caso il Web).

3. *Competenza linguistica*: capacità dell'attore di comprendere la lingua utilizzata dall'interfaccia, le sottigliezze linguistiche, i vari registri del linguaggio (es. burocratico, giuridico, medico), i toni (es. ordini, suggerimenti, richieste, avvisi, etc).

4. *Abilità fisica e visiva*: capacità dell'attore di percepire ed agire sull'interfaccia in maniera agile e naturale. Precisione, agilità, percezione dei colori e dei contrasti, etc.

5. *Motivazione ed interesse*: capacità dell'attore di trovare giustificazioni all'utilizzo del sistema che vadano oltre il mero obbligo lavorativo. Capacità di concentrazione e soddisfazione personale nello svolgimento con successo dei compiti supportati dall'applicazione. Rischi interni alla creazione del *flow* (per flow si intende la concentrazione contenta e totale che si ottiene nel fare qualcosa).

6. *Distrazioni d'ambiente*: capacità dell'attore di immergersi nell'applicazione e di portare a termine i compiti senza distrazioni esterne. Rischi esterni alla creazione del flow.

Per ciascuna delle caratteristiche appena descritte si dà una valutazione che va da uno a cinque (con uno che indica un valore molto buono e cinque che indica un valore molto basso). I valori ottenuti da questa valutazione vengono riportati in un grafico radar detto di *strategia*.

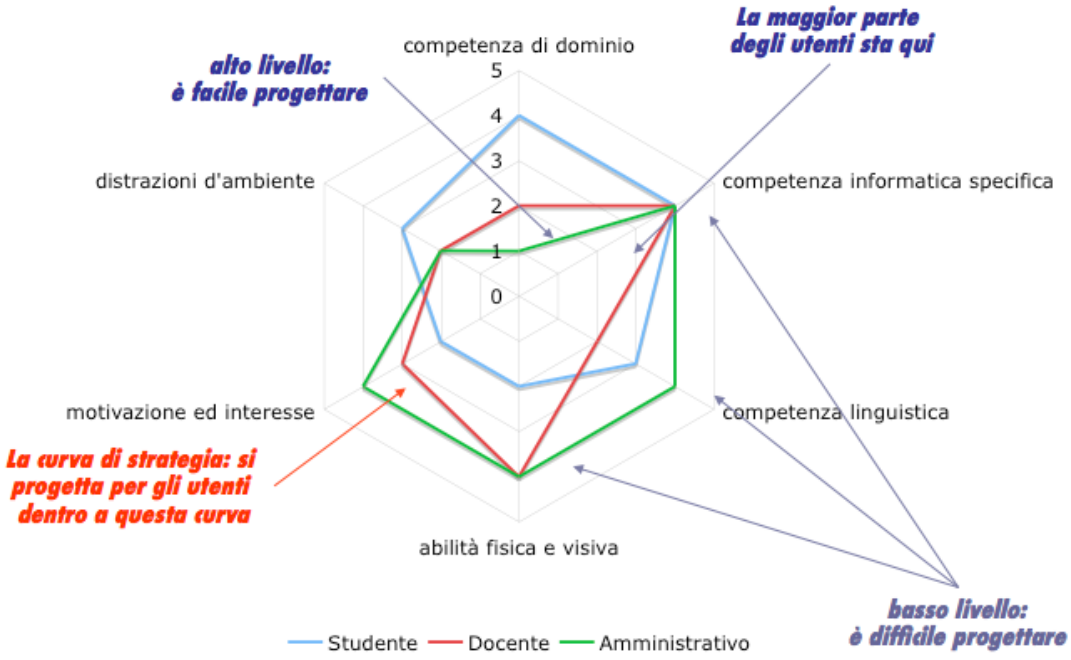

Fig 6. Esempio di Grafico di Strategia[10]

I Concetti in CAO=S

I *Concetti*, secondo il modello CAO=S, indicano le informazioni trattate dall'applicazione. I Concetti sono dunque il modo in cui gli utenti percepiscono l'organizzazione delle informazioni gestite dal sistema. Nel caso in cui non sia possibile un'analisi più dispendiosa, i Concetti possono essere ricavati tramite l'analisi dei requisiti grezzi ottenuti durante la fase di pre-elaborazione.

Quello che l'interfaccia dovrà fare, per rendere usabile l'applicazione, è indicare chiaramente le operazioni che possono essere effettuate sui concetti, e non le funzioni richiamabili sulle strutture dati. In casi particolarmente fortunati concetti e strutture dati coincidono. Se così non fosse occorrerebbe tenere a mente la possibilità di trovarsi di fronte a diversi tipi di problemi come ad esempio:

[10] Fonte: [Zol10]

- *Problemi di normalizzazione*: quando attori e team usano parole diverse per esprimere la stessa cosa, oppure quando attori indiretti e diretti usano parole diverse per esprimere la stessa cosa. Le scelte lessicali degli attori diretti hanno sempre priorità su quelle del team e degli attori indiretti. Bisogna modificare i termini e adattarsi a quelli degli utenti.

- *Differenze lessicali*: quando tipi diversi di attori utilizzano parole diverse per esprimere le stesse cose. Per non incappare in errori di questo tipo è opportuno:

    - trovare un termine accettabile da tutti, anche se non è il preferito da nessuno;

    - esplicitare il rapporto tra il termine scelto e tutti gli altri termini, magari nelle informazioni aggiuntive (tipo i *tooltip*);

    - non preferire un termine proposto ed usato dagli attori indiretti. In generale va utilizzato il termine usato dagli utenti con minore competenza di dominio;

    - realizzare interfacce diverse per ogni tipo di utenza se non si riesce a trovare un termine accettabile.

- *Differenze concettuali*: quando la stessa parola viene usata da attori diversi per descrivere cose diverse. In questi casi è opportuno non utilizzare la parola in questione, né per un significato né per l'altro.

- *Polisemie*: quando la stessa parola viene usata dagli stessi attori per descrivere cose diverse. Come per le differenze concettuali, in questi casi è consigliabile non utilizzare la parola in questione.

Le Operazioni in CAO=S

Per comprendere al meglio cosa siano le *Operazioni* in CAO=S occorre da subito specificare che non sono le operazioni che si possono effettuare sulle strutture dati ma sui *Concetti*. Ogni comando, ogni etichetta, ogni *Widget* deve riportare termini connessi con i concetti, e non con i termini del sistema a cui sono associati.

CAO=S, per le sue operazioni, prende spunto dai modelli CRUD (*Create, Read, Update, Delete*) e REST (*Representational State Transfer*). Individuiamo quattro tipo di operazione:

1. *Creazione*: generazione di una o più istanze di concetto. Le sue caratteristiche sono:

    a. *Tipo di creazione*. Può essere: **manuale** (in cui si attiva una specifica azione il cui scopo principale e quello di creare una nuova istanza di un concetto), **automatica** (in cui l'esecuzione di un certo comando crea in modo automatico la nuova istanza), **implicita** (quando per le esigenze della struttura dell'istanza se ne crea una nuova).

    b. *Valori di default*: sono fondamentali per diminuire il lavoro dell'utente e rassicurarlo sul senso e la destinazione dell'operazione; non devono mai mancare.

    c. *Molteplicità*: possibilità di creare solo un'istanza o molte allo stesso tempo.

d. *Persistenza*: l'istanza del concetto dovrà esistere dopo la fine dell'operazione, se è persistente, deve esistere un operazione di vista associata.

e. *Memoria dell'utente*: oltre al valore di default proposto dal sistema, i valori
precedentemente immessi dall'utente sono utili per la creazione rapida ed efficace di istanze del concetto. La memoria utente esprime il concetto proposto da Alan Cooper [CRC07] secondo il quale "se vale la pena richiedere una cosa all'utente, vale la pena ricordarsela per la prossima volta".

f. *Notifica di insuccesso*: questo tipo di notifica deve fornire *feedback* all'utente nel caso in cui il sistema riscontri un problema. Per una migliore usabilità è opportuno che le notifiche siano tempestive ed immediate.

2. *Vista*: consiste nella visualizzazione di una o più istanze del concetto. La vista o *read* è l'operazione più frequente, esistono diverse tipologie e sono:

a. *Vista individuale completa*: tutte le proprietà associate al concetto sono visibili. Se l'utente è abilitato, da queste è possibile il passaggio ad un update globale. Una regola d'oro è che le modalità di inserimento e di visualizzazione siano simili e riconducibili il più possibile.

b. *Vista individuale ridotta*: vengono visualizzate solo alcune proprietà che sono eventualmente modificabili con la possibilità di passaggio ad una vista individuale completa.

c. *Vista multipla (lista)*: viene fornita una vista ridotta di ciascuna istanza della lista e deve essere possibile passare ad una vista individuale. Si devono poter eseguire operazioni come ordinamenti, raggruppamenti, filtri e ricerche sul database. Se l'utente è abilitato, deve essere possibile eseguire una creazione di istanze.

d. *Vista multiple (lookup)*: permette la visualizzazione connesse ad un concetto associato. Scopo del lookup è selezionare una o più istanze del concetto da usare in seguito. In alcuni casi può essere necessaria una vista individuale estremamente ridotta (in cui compare ad esempio anche solo il nome). Se esistono molte istanze, deve essere possibile ordinare, raggruppare e filtrare.

e. *Vista multipla (ricapitolo)*: in cui fatti raggruppati sulle istanze del concetto vengono visualizzati insieme. Possono essere totali, contatori, ultime creazioni, etc (un esempio di *ricapitolo* è il *Cruscotto* che è stato implementato nell'applicazione prototipo).

3. *Update*. È la modifica di una o più proprietà, di una o più istanze, senza la creazione di nuove. Può essere:

a. *Globale*: se tutte le proprietà delle istanze sono modificabili;

b. *Specifico*: se dipende dal tipo di valore aggiornato (ad esempio i valori booleani possono essere aggiornati con un semplice pulsante dentro ad un'altra vista).

4. *Remove*. È la rimozione di una o più entità e può essere di due tipi:

    a. *Eliminazione*: l'istanza (o le istanze) non esistono più e non sono più recuperabili (equivale quindi ad una cancellazione totale).

    b. *Archiviazione*: l'istanza o le istanze semplicemente non sono più disponibili nelle operazioni di vista generali.

    Anche all'interno di questo tipo di operazione è opportuno notificare il successo o l'insuccesso in modo tale da fornire *feedback* all'utente.

Le strutture in CAO=S

Una volta effettuata l'analisi delle prime tre componenti di CAO=S (*Concetti, Attori e Operazioni*), vengono generate le strutture che descrivono la memorizzazione dei dati e le componenti di interazione tra l'utente e il sistema.

Le strutture nel modello CAO=S sono tre:

1. *Strutture dati*: qui vengono memorizzate in modo persistente le entità attraverso l'utilizzo di database.

2. *Viste*: modelli di schermata attraverso i quali vengono visualizzate le proprietà delle entità. Ogni vista è composta dalla visualizzazione vera e propria e da comandi attivabili durante la visualizzazione. Alcuni di questi comandi sono di navigazione.

3. *Navigazione*: meccanismo di attivazione di viste e comandi per passare dalla visualizzazione iniziale all'esecuzione del comando cercato.

Il prototipo realizzato è composto da viste collegate da comandi della struttura dell'albero di navigazione. Partendo dalla vista iniziale (*Homepage*), si naviga verso le altre viste. Si può osservare come la modellazione in concetti, attori, operazioni e strutture di CAO=S, consenta una facile progettazione dell'architettura dell'applicazione. In particolare le caratteristiche degli utenti in questo modello permettono la creazione di componenti dell'interfaccia intercambiabili (i widget), che vengono progettati in modo da facilitare l'utilizzo del programma ad ogni tipologia di utente e di contesto.

Capitolo 4
Creazione di applicazioni Web tramite metodologia MDE

Come detto in precedenza, per la realizzazione del progetto di tesi si è deciso di utilizzare la metodologia MDE ed, in particolare, è stata adottata la modellazione WebML che è pienamente supportata dal tool WebRatio. WebRatio mette a disposizione un editor di diagrammi WebML tramite i quali si sono potuti esprimere tutti i requisiti dell'applicazione Web. Le Web application prodotte da WebRatio sono conformi allo standard Java/JSP 2.0[11].

La parte *server* della Web application ottenuta per testare il funzionamento del progetto di tesi è di tipo *Java standard* e non ha alcun componente proprietario. Per questo motivo può essere messa in esercizio su qualsiasi application server Java: Apache, Tomcat, JBoss, Caucho Resin, Oracle AS, etc. In fase di implementazione è stato scelto l'application server Tomcat dato che è quello che viene installato di default nel momento in cui si installa WebRatio.

[11] Fonte: http://www.webml.org/webml/page1.do?stu2.values=IT. Consultato in data 12 febbraio 2012

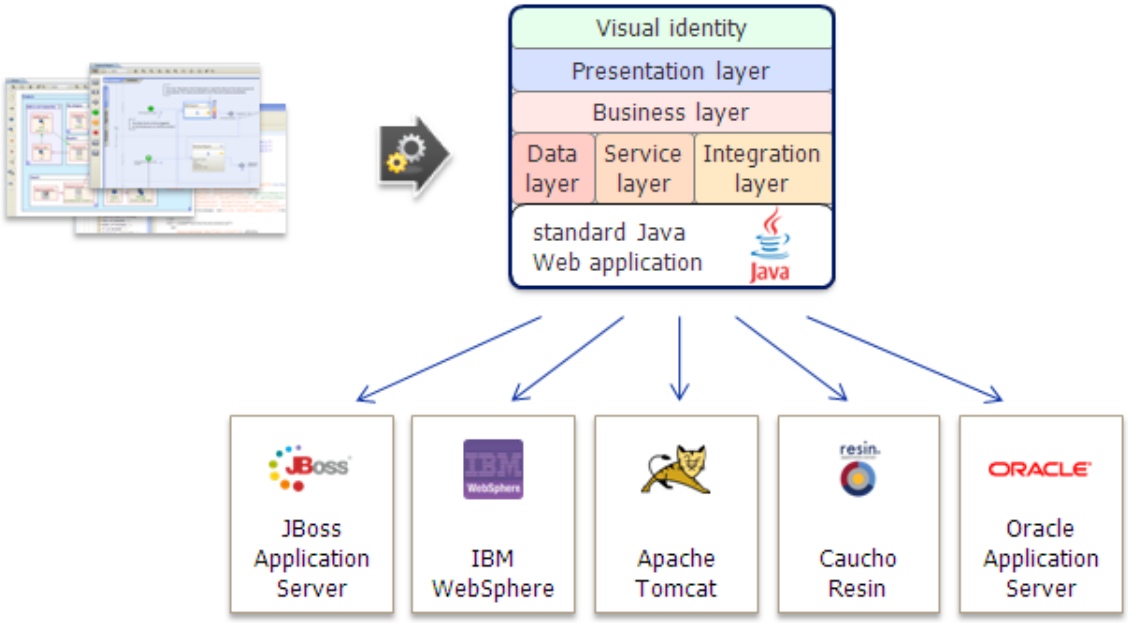

Fig 7. Application Server supportati da WebRatio[12]

WebRatio ha permesso di generare il server in qualsiasi momento del processo di sviluppo. Si è potuto, quindi, ottenere senza alcun costo un prototipo funzionante e realistico dell'applicazione per poter verificare la sua fedeltà rispetto ai requisiti. In questo modo ci si è trovati davanti ad un ciclo virtuoso di analisi, modellazione, generazione e verifica, che ci ha permesso di convergere verso la soluzione finale.

[12] Fonte: http://www.webratio.com/portal/contentPage/it/Genera%20l%27applicazione. Consultato in data 29 febbraio 2012

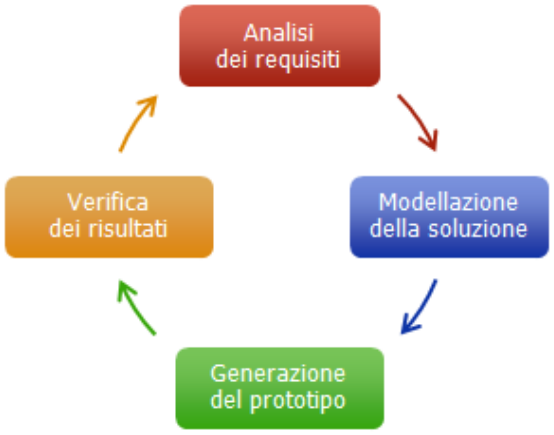

Fig 8. Il processo di sviluppo in WebRatio[13]

La modellazione con WebRatio

WebRatio permette di creare un'applicazione Web tramite la creazione di due modelli: il ***data-model*** e il ***web-model***.

Il *data-model* è un diagramma ER in cui vi sono tutte le informazioni che l'applicazione Web deve gestire. Il *web-model* è, invece, un gruppo di diagrammi WebML che specificano le pagine Web dell'applicazione ed il loro contenuto. Per specificare il contenuto della pagina il progettista deve fare riferimento agli oggetti contenuti nel *data-model*, che rappresentano il punto di partenza per la creazione di un nuovo progetto web. In questo modo è possibile creare tutto il modello di una qualsiasi applicazione Web senza considerare il database che verrà utilizzato. Vi è un alto livello di astrazione che permette di ritardare la scelta del database e di modificare questa scelta durante lo sviluppo.

Una volta che il database è stato scelto, le informazioni riguardanti ogni oggetto del *data-model* (entità, relazioni o attributi) sono collegate con un oggetto di database (tabella, vista o colonna) e memorizzate attraverso una mappatura.

[13] Fonte: http://www.webratio.com/portal/contentPage/it/Genera%20l%27applicazione

Una definizione della mappatura può essere[AB08]: "Una relazione uno-a-uno tra ogni elemento nel *data-model* e l'elemento corrispondente nel database. Con la mappatura ogni operazione che si riferisce al modello di dati può essere tradotta in istruzioni SQL per il database specifico."

Modellazione del *Data Model*

Come è già stato accennato, la modellazione WebML passa attraverso la definizione del *Data Model*. Con questo termine indichiamo il modello dei dati *Entity-Relationship* utilizzato per la specifica dei dati sottostanti l'applicazione Web. All'interno della nostra applicazione di prova sono state modellate alcune entità principali che possono essere viste come le classi primitive UML che caratterizzano il sistema.

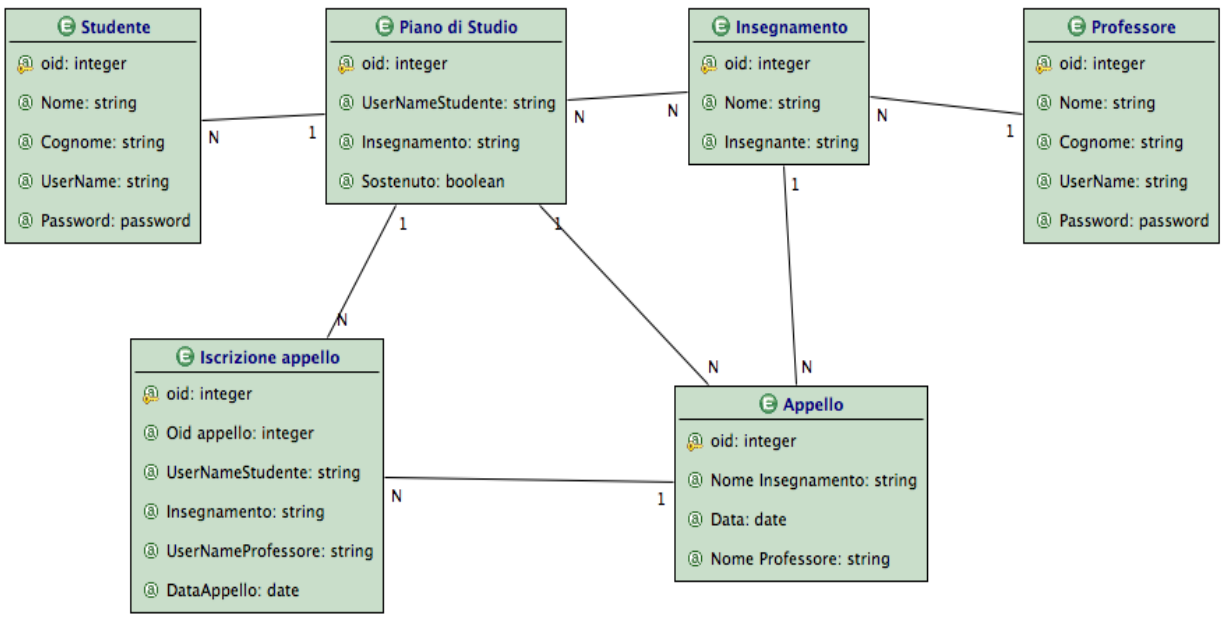

Fig 9. Il Data-Model realizzato per l'applicazione prototipo

Le entità principali sono state modellate seguendo i concetti CAO=S individuati in fase di analisi dei requisiti.

Per la gestione degli accessi al sistema, WebRatio mette a disposizione delle entità che si trovano di default all'interno del data-model.

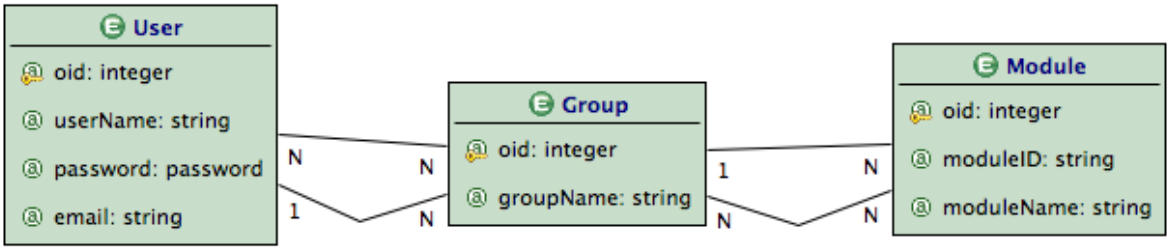

Fig 10. Porzione del Data-Model che gestisce gli utenti.

L'entità **User** permette la creazione di istanze di utenti che potranno accedere al sistema. Il **gruppo** non è altro che la definizione di un insieme in cui possono essere inseriti diversi utenti in base al ruolo che questi avranno all'interno del sistema.

Per l'applicazione prototipo abbiamo utilizzato tre gruppi: gli studenti, i professori e l'amministratore del sistema.

Ogni gruppo è collegato ad un modulo, cioè ad una serie di viste ed operazioni raggiungibili dai vari utenti. Queste entità si sono prestate molto bene ai fini del progetto di tesi poiché permettono di astrarre la gestione degli accessi al sistema. Queste permettono, inoltre, al progetto di tesi di potersi adattare facilmente a diversi contesti d'uso in modo facile e veloce.

Modellazione del *Web Model*

Andiamo ora ad esaminare com'è stato modellato il *Web-Model* all'interno del progetto di tesi.

Il *Web-Model* in WebRatio corrisponde all'*Hypertext Model* di WebML cioè al modello ipertesto che permette la definizione dell'interfaccia front-end dell'applicazione Web. Il Web Model è solitamente strutturato attraverso una o più *Site-View*. Una Site-View è un particolare ipertesto progettato per soddisfare un insieme di requisiti ed è costituita da delle aree, che sono le sezioni principali dell'ipertesto e comprende in modo ricorsivo altre sub-aree o pagine.

Poiché il progetto di tesi funziona con un client realizzato ad hoc, la modellazione delle Site-View non si è curata della realizzazione delle interfacce grafiche.

Le Site-View vengono modellate per poter gestire i dati e in modo tale da mettere questi a disposizione del client. In altre parole si occupano dei servizi che l'applicazione è in grado di offrire e della loro accessibilità in base al gruppo di utenti che le richiede.

Le Site-View che sono state modellate nell'applicazione prototipo sono quattro:

1. Homepage (login);
2. Studente;
3. Professore;
4. Amministratore.

Ad esclusione della homepage, le altre Site-View sono state definite come "protette" permettendo così di restringere l'accesso ad un singolo modulo da parte di un gruppo di utenti. Notiamo che nel momento in cui attribuiamo la

proprietà *protected* ad una Site View, WebRatio la definisce in automatico come *modulo* collegabile ad un gruppo.

Login

Nel progetto di tesi, la gestione degli accessi al sistema è regolata tramite la *Login Unit* messa a disposizione da WebRatio.

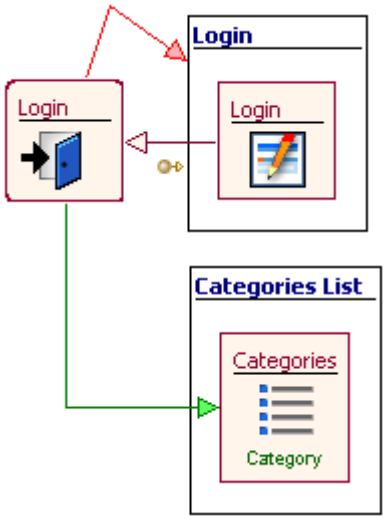

Fig 11. Esempio di utilizzo della *Login Unit*

Collegando un form con i campi *username* e *password* ad una *Login Unit* è possibile reindirizzare l'utente esplicitamente ad una pagina protetta. La *Login Unit* si occupa di controllare se i dati sono corretti. Una volta inseriti i dati di accesso due sono le possibili azioni: se le credenziali di accesso sono corrette la Login Unit si occupa di portare l'utente sul modulo di sua compentenza; se le credenziali sono sbagliate l'utente viene portato su una pagina di errore dove si comunica l'incorrettezza di username e/o password.

Fig 11. Esempio del risultato ottenuto dopo la generazione

Site View

Vista la necessità di comunicare con un client che sia del tutto indipendente dal sistema, il progetto di tesi sfrutta le potenzialità offerte da WebRatio nella creazioni di file XML che siano così leggibili e modificabili dai widget grafici. Per questi motivi, all'interno delle Site-View si è fatto ampio uso di *XML Out Unit* e *Script Unit* che permettono la creazione di file XML in un preciso URL che verrà poi utilizzato dal client per fare la presentazione dei dati.

Le diverse Site-View su cui si basa il progetto di tesi si compongono di un'area all'interno della quale avvengono dei meccanismi per la creazione di tutti i file XML che servono al client. La "catena" di operazioni che porta alla scrittura degli XML inizia tramite l'individuazione dell'utente che la richiede e, dunque, con una *Get Unit* che restituisce l'identificativo dell'utente. Questo viene passato ad una *Selector Unit* che si occupa di ritrovare tutti gli attributi collegati all'utente. Successivamente abbiamo la creazione dei diversi file XML.

In ogni Site-View del progetto di tesi è importante implementare tutte la pagine attraverso le quali l'utente potrà navigare. È da notare come queste pagine siano vuote poiché sarà il lato client ad occuparsi delle interfacce grafiche

Lo studente, una volta terminata la sua esperienza all'interno di un'applicazione prototipo potrà lasciare il sistema effettuando il *logout*. Questa operazione è stata implementata tramite la *Logout Unit* di WebRatio.

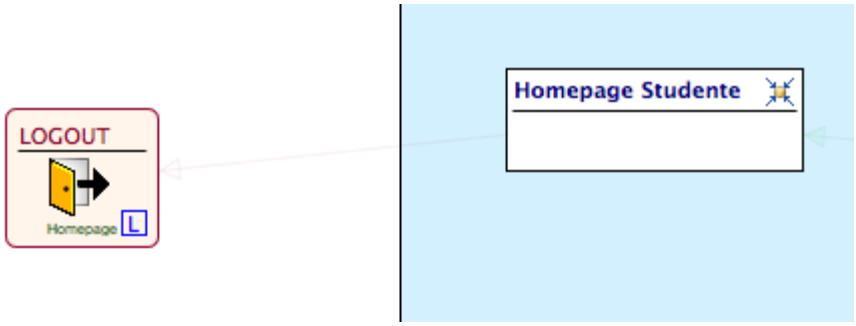

Fig 12. Esempio di utilizzo di una *Logout Unit*

La *Logout Unit* si occuperà di terminare la sessione e di riportare l'utente alla pagine di login. La "L" che si nota in figura all'interno della *Logout Unit* indica che questa è raggiungibile a partire da tutte le pagine contenute nell'area.

La scrittura di un file XML

Andiamo ora ad analizzare brevemente come avviene la scrittura di un file XML all'interno del progetto di tesi. WebRatio rende possibile questa azione tramite due *unit*:

- Una *XML Out Unit* che si occupa di selezionare l'entità che si vuole esportare e di effettuare delle operazioni sulle istanze alle quali si è interessati.

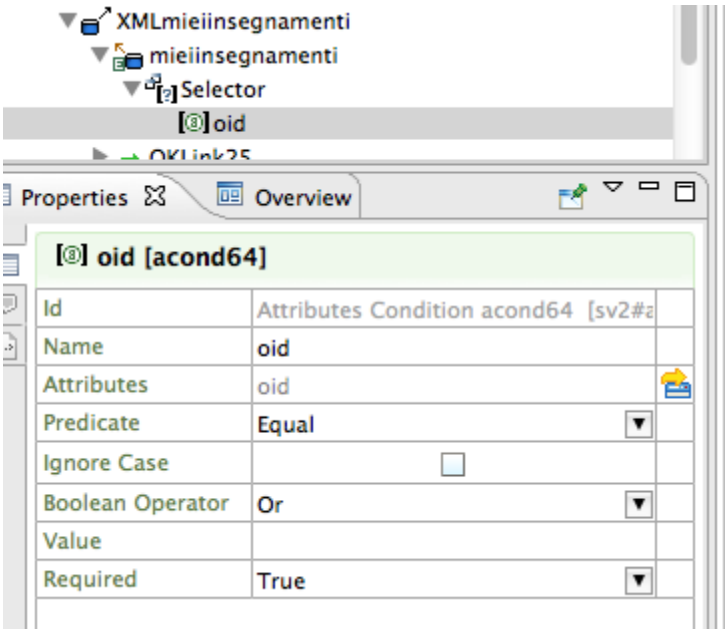

Fig 13. Esempio di utilizzo di una *XML Out Unit*

Nell'esempio in figura possiamo osservare come avviene la selezione degli insegnamenti di un professore: abbiamo un'entità di tipo *insegnamento*" e su questa effettuiamo delle operazioni tramite un selettore (*selector*). Le operazione avvengono tramite un confronto sull'*oid* fra tutti gli insegnamenti e quelli che appartengono al professore (e che sono stati precedentemente individuati tramite una *Selector Unit*). Facendo così, all'interno della *XML Out Unit* verranno conservati solo gli insegnamenti di cui il professore (che ha avviato la sessione) è titolare.

- La *Script Unit*. In WebRatio, una script unit, permette di definire delle funzioni extra rispetto a quelle fornite di default dal tool. In questa *unit* è possibile esplicitare delle funzioni nel linguaggio *Groovy*. Prendiamo in esame l'esempio degli insegnamenti del professore: all'interno della *Script Unit* troviamo le seguenti linee di codice

```
#input Document mieiinsegnamenti
import org.apache.commons.io.FileUtils;
String uploadDir = rtx.getUploadDirectory();
File myXMLDoc = new
File(uploadDir+"//professore/mieiinsegnamenti.xml");
myXMLDoc.createNewFile();
FileUtils.writeStringToFile(myXMLDoc,
mieiinsegnamenti.asXML())
```

Possiamo osservare come, inizialmente, si definisca un documento di input (in questo caso di nome *mieiinsegnamenti*) e, dopo aver richiamato delle librerie e settato la directory per la scrittura, si vada a creare e a scrivere il nuovo file. La *XML Out Unit* si occuperà di passare sulla variabile di input *mieiinsegnamenti* le istanze selezionate tramite il *coupling* che collega le due *unit*.

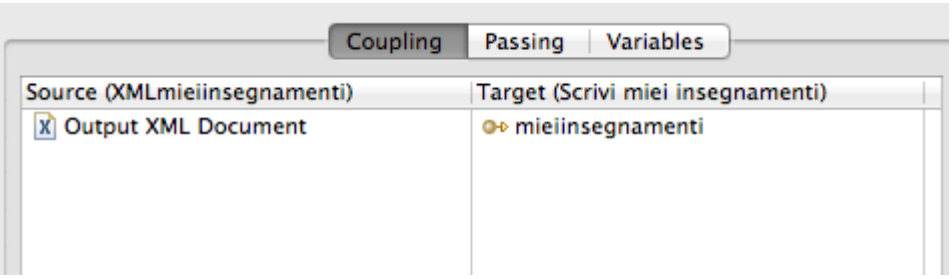

Fig 14. Esempio di *coupling* fra una *XML Out Unit* e una *Script Unit*

Capitolo 5
Da BuCaBO alla generazione dei prototipi

Dopo aver preso in esame le funzionalità offerte dal tool di modellazione che sono state utilizzate all'interno del progetto di tesi, si andrà ora a definire la meta-applicazione realizzata.

BuCaBO offre, da una parte, un insieme di linee guida per la modellazione WebML e dall'altra, una serie di strumenti indispensabili per permettere la comunicazione fra il lato client e quello server.

BuCaBO: linee guida

Le linee guida offerte da BuCaBO permettono allo sviluppatore di generare un'applicazione Web server side in grado di interfacciarsi con una data applicazione client side. Le linee guida interessano la creazione dei due fondamentali modelli WebML: il data-model (il modello dei dati *Entity-Relationship*) e il web-model (il modello ipertesto che permette la definizione dell'interfaccia front-end dell'applicazione Web).

Nella modellazione del data-model bisogna far corrispondere alle entità i Concetti CAO=S individuati in fase di analisi dei requisiti. Questa operazione è importante in quanto permette di eliminare i termini di ambiguità.

Per quanto riguarda gli Attori diretti CAO=S, questi vengono sempre definiti all'interno del data-model sfruttando le relazioni fra le entità *user, group, module*.

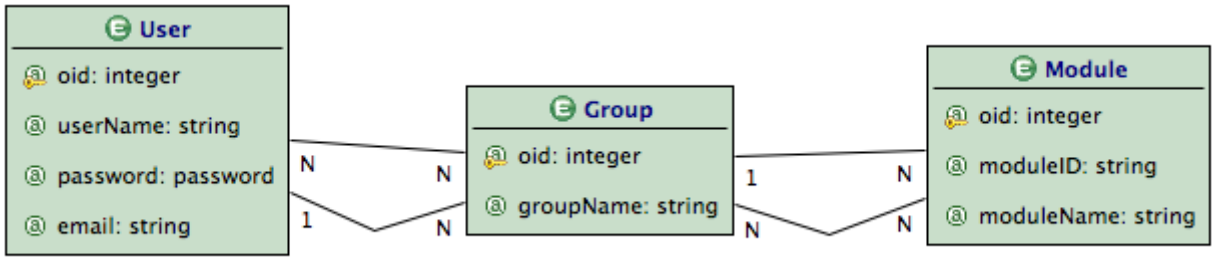

Fig 15. Le entità *User, Group, Module*

Uno dei vantaggi nell'usare il tool di modellazione WebRatio è che queste entità e relazioni si trovano già modellate all'interno del data-model. L'entità User permette la creazione di tutti gli utenti che possono accedere al sistema. Con Group è possibile definire una categoria di utenti diretti, mentre attraverso la relazione che collega Group e Module è possible stabilire quali viste sono raggiungibili da un determinato gruppo di utenti.

La modellazione del web-model consiste nella creazione di site-view con particolari caratteristiche:

1. Ogni site-view corrisponde ad un modulo;

2. Ogni site-view può contenere una o più aree;

3. Nell'area si definiscono le operazioni che l'applicazione sarà in grado di offrire;

4. Nell'area si utilizzano due strumenti per rendere disponibili gli output delle diverse operazioni: le XML Out Unit e le Script Unit;

5. Ogni site-view contiene le pagine raggiungibili dagli utenti (viste).

6. Le pagine sono vuote in quanto sarà l'applicazione client a gestirne la presentazione.

Il compito delle XML Out Unit è quello di selezionare e strutturare i dati che saranno utilizzati dal client. Le Script Unit invece ricevono la struttura dati dalla XML Out Unit e ne scrivono il contenuto all'interno di un file.

BuCaBO: Strumenti

Come visto in precedenza la comunicazione tra client e server avviene tramite l'utilizzo di file XML. Andiamo ora ad analizzare gli strumenti che BuCaBO offre per estrarre ed organizzare i dati che saranno utilizzati dal client.

BuCaBO fornisce strumenti definiti di *servizio* e di *configurazione*. Gli strumenti di servizio sono implementati all'interno del file *xPath.js* dove vi sono le espressioni *xPath*. In particolare troviamo due funzioni:

- *creoArray():* in cui abbiamo la chiamata *Ajax* che recupera un file XML a partire da un URL. Dopo aver individuato l'XML, questa funzione sfrutta alcune capacità del framework *jQuery* per creare un array contenente gli attributi presenti nel file XML. *CreoArray()* è molto sfruttata dall'applicazione *client side* in quanto permette di avere

dinamicità e adattabilità per le varie viste.

- *getInfoUtente()*: questa funzione effettua il *parsing* sul file XML contenente i dati
  dell'utente, li estrapola e li inserisce all'interno di un array.

Gli strumenti di configurazione invece sono implementati attraverso due file JavaScript:

1. *call:* indica al file xpath.js dove andare a reperire i file XML

2. config: gestisce la presentazione dei dati estratti dall' XML e crea le strutture JSON che verranno utilizzate dai widget grafici.

Il primo prototipo generato: Sistema Universitario

In questa sezione andiamo ad analizzare il primo prototipo che è stato generato tramite la meta-applicazione BuCaBO.

Il prototipo simula il funzionamento di un sistema universitario dove professori e studenti possono effettuare diverse operazioni.

Professore

	Create	Read	Update	Delete
Professore	--	29. Vista totale dei professori	--	--
Studente	--	30. Vista totale degli studenti 31. Vista degli studenti iscritti ai propri corsi	32. Iscrizione di studente ad appello 33. Rimozione di studente da appello 34. Verbalizzazione di studente per corso	--
Corso o Insegnamento	--	35. Vista totale dei corsi 36. Vista dei propri corsi	37. Modifica del programma del corso	--
Appello	38. Creazione totale dell'appello	39. Vista dei propri appelli futuri 40. Vista dei propri appelli 41. Vista totale degli appelli	**Su appelli passati** = data appello < now() 42. Verbalizzazione studente 43. Iscrizione di studente ad appello 44. Rimozione di studente da appello **Su appelli futuri** = data appello > now() 45. Update totale appello (tranne voto studenti) 46. Iscrizione di studente ad appello 47. Rimozione di studente ad appello	**Su appelli passati** = data appello < now() 48. Archiviazione appello (*constraint*: solo se tutti gli studenti sono stati verbalizzati) **Su appelli futuri** = data appello > now() 49. Rimozione appello

Le operazioni che il professore può effettuare sono di quattro tipi: *Create*, *Read*, *Update*, *Delete*.

Lo studente invece è abilitato solo ad operazioni del tipo *Read e Update*.

Studente

	Create	Read[2]	Update	Delete
Professore	--	50. Vista totale dei professori	--	--
Studente	--	51. Vista totale degli studenti	--	--
Corso o Insegnamento	--	52. Vista totale dei corsi 53. Vista dei propri corsi	--	--
Appello	--	54. Vista dei propri appelli futuri 55. Vista dei propri appelli passati 56. Vista della carriera (cioè appelli verbalizzati) 57. Vista totale degli appelli	**Su appelli futuri** = data appello > now() 58. Iscrizione ad appello 59. Rimozione da appello	--

In un sistema di questo tipo bisogna necessariamente garantire una netta differenziazione tra gli utenti che, di conseguenza, si troveranno di fronte a

viste diverse. A tale scopo è stata implementata una gestione degli utenti per gruppi e moduli. La modellazione del data-model ha portato all'individuazione di sei entità che rappresentano i Concetti CAO=S raccolti in fase di analisi dei requisiti.

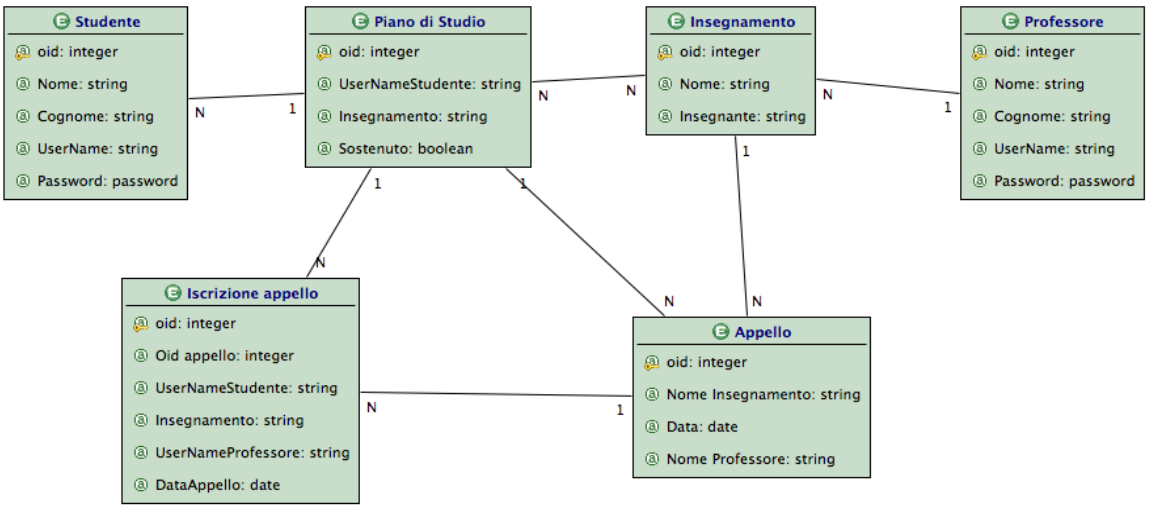

Fig 16. Il Data-Model del sistema di gestione degli esami universitari.

Per quanto riguarda il web-model, sono state modellate 4 site-view che rappresentano i moduli raggiungibili all'interno dell'applicazione.

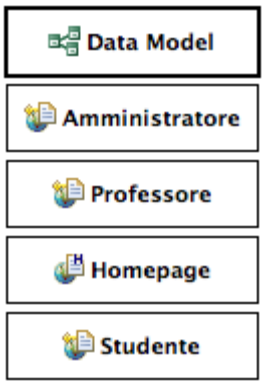

Fig 17. Le Site-View modellate per il sistema di gestione degli esami universitari.

Come detto in fase di definizione delle linee guida, si è fatto un ampio uso di XML Out Unit e Script Unit per permettere il passaggio di dati al client.

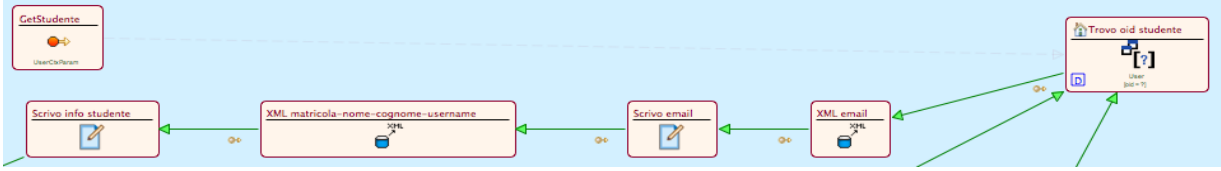

Fig 18. Porzione del modello che compone una Site-View.

Nell'esempio in figura possiamo osservare come l'utente (in questo caso di tipo Studente) venga identificato dal sistema e come venga prodotto il file XML contentente alcune informazioni personali.

Descrizione del sistema

All'avvio ci si trova di fronte ad una schermata di Login, avvenuto il riconoscimento dell'utente questo viene indirizzato al modulo di appartenenza ed abilitato alle proprie operazioni.

Fig 19. La finestra di *Login*.

Gli elementi che accompagnano l'utente durante tutta l'eseperienza all'interno dell'applicazione sono:

1. *Albero di navigazione* (parte sinistra della pagina);

2. *Pannello informazioni utente* (parte destra della pagina)*;*

3. *Top Menù* (in alto nella pagina).

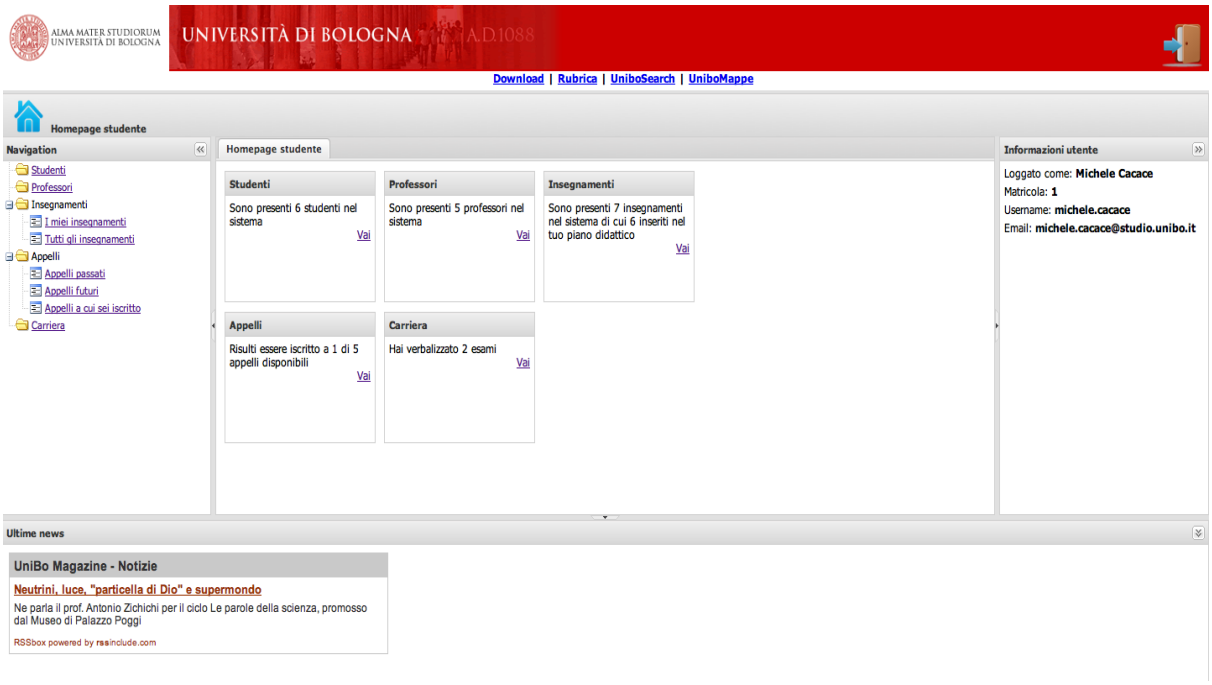

Fig 20. *Homepage* dello studente

L'interfaccia può essere considerata come un puzzle, perchè composta da widget completamente indipendenti l'uno dall'altro posizionati all'interno pagina. Nella *Homepage* ogni utente, sia esso professore che studente, si trova di fronte al "*cruscotto*" ovvero una serie di *overview* che possiamo definire come un resoconto dei concetti direttamente collegati all'utente (corrispondente alle operazioni di CAO=S *Vista multipla-ricapitolo*).

Il layout della pagina è composto da 5 regioni (Nord, sud, est, ovest e centro) tutte ridimensionabili a piacere dall'utente.

All'interno della regione ovest abbiamo l'albero di navigazione che permette il passaggio da una vista ad un'altra. Naturalmente questo menù sarà diverso in base alla tipologia di utente.

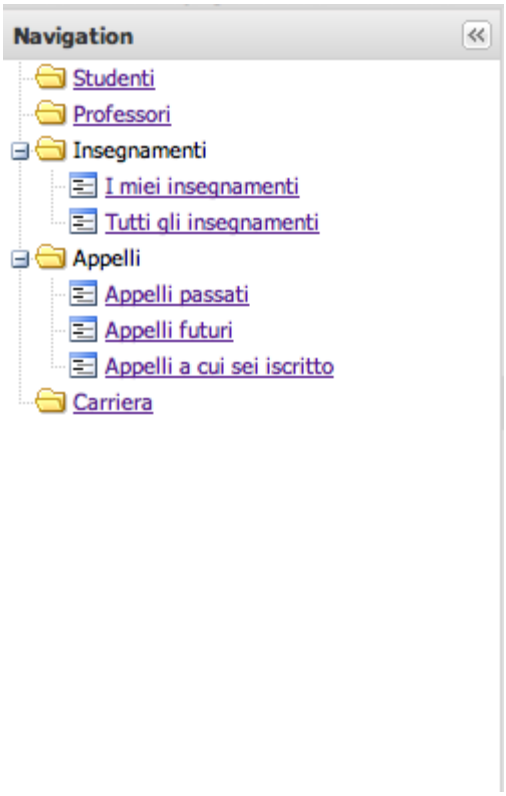

Fig 21. Il Menù

Nella regione est abbiamo un pannello con tutte le informazioni utili riferite all'utente cioè nome, cognome, matricola ed email. Questo permette la costante individuazione della sessione attiva e di avere sempre in primo piano informazioni che potrebbero ritornare utili in qualsiasi momento.

Fig 22. Il pannello contenente le *Informazioni utente*.

La regione centrale può essere considerata la più importante. All'interno di essa si alternano le viste fondamentali come il cruscotto e le griglie. Le griglie permettono diverse operazioni:

1. ordinamenti;
2. raggruppamenti,
3. filtri,
4. ricerche,

Le griglie, ove possibile, consentono all'utente la creazione di un'istanza. La griglia è il widget più potente e con le più alte caratteristiche di usabilità.

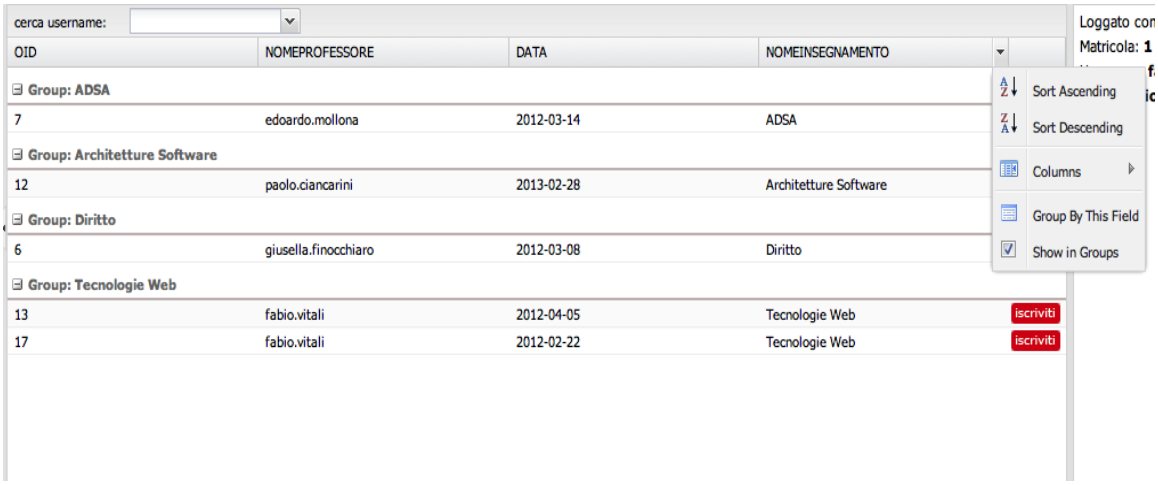

Fig 23. Elenco dei comandi applicabili su una *Grid*.

In alcune griglie è permesso l'inserimento e la rimozione di istanze.

Fig 24. Esempio di *Action Column*.

Tour studente

Lo studente che accede al sistema con le sue credenziali, una volta nella Homepage, avrà subito a disposizione una serie d'informazioni fornite dal cruscotto:

1. Numero degli studenti presenti nel sistema;

2. Numero dei professori presenti nel sistema;

3. Numero di insegnamenti inseriti nel proprio piano di studi;

4. Numero di appelli a cui è iscritto;

5. Carriera (numero di esami sostenuti).

Lo studente può navigare all'interno dell'applicazione grazie al menù di navigazione che trova a sinistra nella pagina. Il menù è composto da cinque nodi di root: *Studenti, Professori, Insegnamenti, Appelli, Carriera*. La *root Insegnamenti* ha due child: *I miei Insegnamenti* e *Tutti gli Insegnamenti*. Allo stesso modo la root *Appelli* ha tre child: *Appelli passati, Appelli futuri* e *Appelli a cui sei iscritto*.

Nella pagina *Studenti* l'utente si troverà di fronte ad una griglia posizionata nella parte centrale del layout. Questa griglia permette di visualizzare un elenco di tutti gli studenti ed effettuare solo delle operazioni di *read*: raggruppamenti, filtri, ricerche ed ordinamenti. La pagina *Professori* è pressoché simile a quella precedente, la differenza sta solo nel contenuto della griglia che ovviamente visualizzerà l'elenco dei professori. Anche in questa vista abbiamo solo operazioni di *read*.

Le pagine *I miei Insegnamenti* e *Tutti gli Insegnamenti* presento due griglie: in una abbiamo la lista dei nomi di tutti gli insegnamenti inseriti nell'intero sistema universitario e nell'altra quella dei soli insegnamenti inseriti nel piano di studi dello studente. Anche in questo caso abbiamo solo operazioni di *read*.

Le uniche due pagine che permettono delle operazione di *Update* nel modulo studente sono: *Appelli futuri* ed *Appelli a cui sei iscritto*. La prima visualizza all'interno di una griglia tutti gli appelli con una data futura alla data attuale. Accanto ad ogni riga (che rappresenta dunque un'istanza) è presente una *action column* con un bottone "iscriviti" che da la possibilità all'utente di iscriversi

all'esame. Se l'utente deciderà di iscriversi, la riga selezionata sarà eliminata dalla griglia ed andrà a popolare la griglia della pagina *Appelli a cui sei iscritto*. In quest'ultima griglia abbiamo un elenco degli appelli a cui l'utente è iscritto. L'utente ha la possibilità di poter eliminare una o più iscrizioni grazie al pulsante "X" contenuto nella action column. Se l'utente decide di effettuare l'eliminazione la riga selezionata sarà eliminata dalla griglia e ritornerà a popolare la griglia di *Appelli futuri*. L'ultima pagina che può essere visualizzata dall'utente è *Carriera:* anche qui abbiamo una semplice griglia il cui contenuto è una lista degli esami sostenuti, in questo caso abbiamo delle semplici operazioni di *read*.

Tour professore

Il professore accede alla sua Homepage dopo aver inserito le credenziali ed essere stato riconosciuto dal sistema. La Homepage presenta un cruscotto centrale che è composto dalle seguenti voci:

1. Numero degli studenti presenti nel sistema;
2. Numero dei professori presenti nel sistema;
3. Numero degli insegnamenti totali di cui "n" suoi
4. Numero degli appelli totali di cui "n" suoi.

Sulla sinistra è presente il menù di navigazione che permette all'utente il passaggio da una vista ad un'altra. Si compone di quattro nodi di root: *Studenti, Professori, Insegnamenti* e *Appelli*. Ad esclusione della root *Professori* tutte le altre hanno dei child, *Studenti* ha: *I tuoi studenti* e *Tutti gli studenti,* la root *Insegnamenti* ha: *I tuoi insegnamenti* e *Tutti gli insegnamenti,* infine la root

Appelli ha quattro child: *I tuoi appelli, Appelli futuri, Tutti gli appelli* e *Crea nuovo appello*.

Nella pagina *I tuoi studenti* abbiamo una griglia con una lista di studenti iscritti ai propri insegnamenti. In questa griglia oltre alle operazioni di *read* (raggruppamenti, filtri, ricerche ed ordinamenti) sono abilitate delle operazioni di tipo Update mediante una *action column* con all'interno un bottone "registrare". L'utente ha quindi la possibilità di registrare l'esame effettuato dallo studente (permettendo quindi l'*Update specifico* del modello CAO=S). Nella pagina di pari livello *Tutti gli studenti* è presente una griglia con l'elenco di tutti gli studenti presenti nel sistema universitario, le sole operazioni abilitate sono quelle di *read*.

La pagina *Professori* è composta da una griglia con una lista di tutti i professori presenti nel sistema universitario. Le sole operazioni abilitate sono quelle di *read*.

Nei due child della root *Insegnamenti* abbiamo due griglie: una con la lista degli insegnamenti dell'utente e l'altra con la lista di tutti gli insegnamenti presenti nel sistema. Le uniche operazioni possibili sono quelle di *read*.

L'ultima root è quella degli *Appelli*. Nei suoi child sono concentrate le diverse operazioni di *create, update e delete* che l'utente può realizzare.

Nella pagina *Appelli Futuri*, il professore può visualizzare all'interno di una griglia la lista dei suoi appelli futuri. In questa pagina è abilitato ad effettuare le operazioni di rimozione di un appello e di iscrizione di uno studente ad un appello. Queste operazioni sono possibili grazie ai bottoni presenti nella *action column* a destra.

L'unica operazione di *create* è presente nella pagina *Crea nuovo Appello*. Qui l'utente può compilare un semplice form e creare una nuova istanza (implementando così la creazione manuale di CAO=S). La nuova istanza andrà a popolare le griglie presenti nelle pagine *I tuoi appelli, Appelli futuri* e *Tutti gli Appelli*.

Nella pagina *I tuoi Appelli* è presente una griglia che visualizza una lista degli appelli creati dall'utente. Nella pagine *Tutti gli Appelli* abbiamo una griglia contenente tutti gli appelli inseriti nel sistema. In entrambe le pagine si possono effettuare solo delle operazioni di *read*.

Il secondo Prototipo generato: Gestione Magazzino

Per mettere in risalto le potenzialità e le capacità di BuCaBO di svincolarsi dal dominio di applicazione, è stato generato un secondo prototipo che simula un sistema di Gestione del magazzino.

Sono state modellate delle nuove entità:

1. Fornitore;
2. Prodotto;
3. Magazzino.

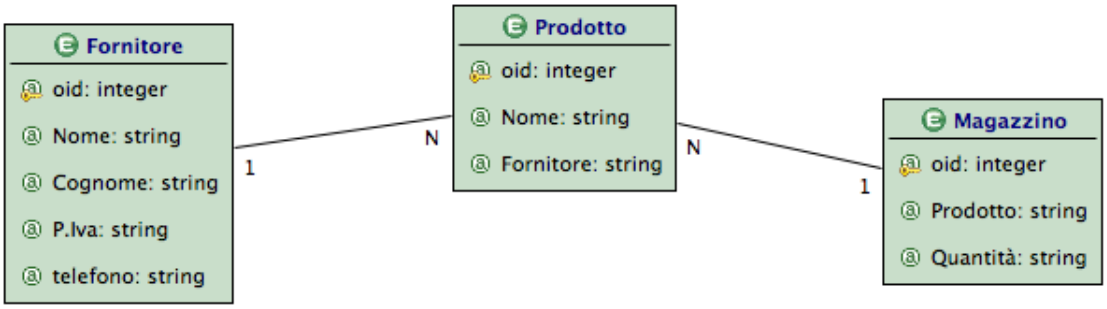

Fig 25. Il Data-Model del secondo prototipo generato.

Per comodità è stata definita una solo tipologia di Attore e quindi generata una sola Site-view. La Site-view contiene tre pagine che sono le viste a cui l'utente può accedere.

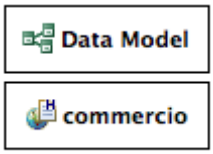

Fig 26. Le Site-View del secondo prototipo generato.

Come nel prototipo precedente sono state utilizzate delle XML Out Unit e delle Script Unit che permettono la creazione dei file XML.

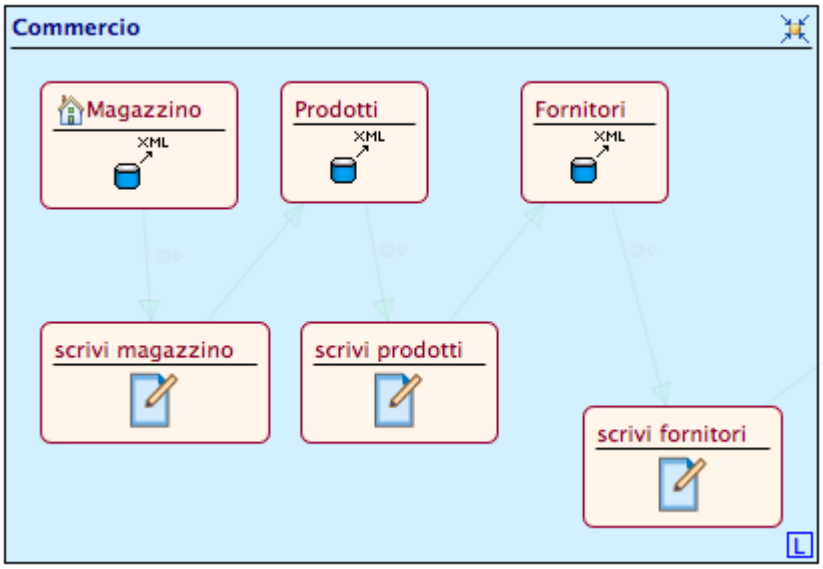

Fig 27. Utilizzo di *XML Out Unit* e *Script Unit* nel secondo prototipo generato.

Il risultato della generazione è stato un piccolo sistema di gestione del magazzino.

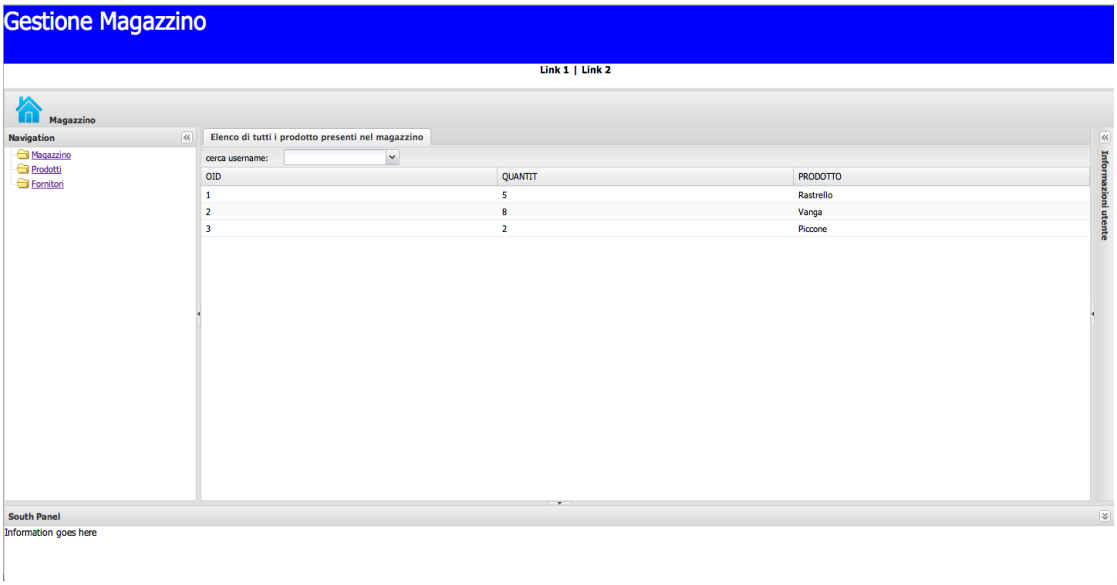

Fig 28. Interfaccia del secondo prototipo generato.

L'immagine mostra come sia stata generata un'applicazione con caratteristiche completamente diverse da quella precedente mantenendo, però, le

caratteristiche di usabilità. Il tempo necessario per realizzarla è stato di circa un'ora.

Capitolo 6
Conclusioni

In questa dissertazione sono stati esaminati i diversi aspetti riguardanti la generazione di applicazioni web usabili. Le tecniche prese in considerazione sono state di tipo *Model-Driven* e implementate mediante un linguaggio di notazione visuale per la modellazione chiamato WebML. Con WebML è stato inoltre possibile definire una qualunque applicazione Web passando attraverso la modellazione di due modelli principali: il *data-model* e il *web-model*. Questo tipo di implementazione ha consentito la realizzazione di applicazioni Web di qualità in tempi rapidi e con alte caratteristiche di usabilità.

Per la realizzazione dei modelli WebML è stato utilizzato il tool WebRatio. Attraverso l'utilizzo di questo strumento si è potuta implementare un'applicazione di esempio che mostra come sia possibile integrare una parte server Java con un client sviluppato in modo indipendente dal modello. Il risultato della generazione del livello server è un'applicazione JavaEE mentre il livello client è stato implementato utilizzando JavaScript ed un suo framework grafico chiamato ExtJs.

ExtJs permette con poche righe di codice di creare una varietà di widget avanzati, con la sicurezza che l'applicazione verrà correttamente visualizzata su tutti i browser in circolazione.

Per quanto riguarda le varie tecniche di progettazione delle interfacce incentrate sugli utenti è stato adottato il design Goal-Oriented che offre uno strumento di progettazione molto efficace ma allo stesso tempo richiede team di sviluppo di medie dimensioni. Vista quindi la complessità del modello Goal-Oriented

tradizionale, si è scelto di utilizzarne uno semplificato e rivolto proprio a team di dimensioni ridotte con poche risorse a disposizione: CAO=S. Questo modello, sviluppato all'interno del dipartimento di Scienze dell'informazione dell'Università di Bologna, riduce l'analisi degli utenti alle sole caratteristiche che hanno un impatto importante sull'interazione e senza un'analisi accurata degli obiettivi personali degli utenti.

Una volta definiti gli strumenti da utilizzare all'interno del progetto di tesi è stata definita la meta-applicazione BuCaBO che offre due principali strumenti per la realizzazione di applicazioni Web usabili:

- Alcune linee guida da seguire in fase di modellazione del server;

- Alcuni file che permettono al server di interfacciarsi con un client usabile.

Per dimostrare il funzionamento di BuCaBO sono stati implementati due prototipi di applicazioni Web con domini applicativi completamente diversi: un sistema di gestione degli esami universitari e un sistema per la gestione di un magazzino.

BuCaBO potrebbe prestarsi ad alcuni sviluppo ulteriori per rendere "adattabili" le interfacce grafiche messe a disposizione degli utenti. Questa feature potrebbe essere implementata passando attraverso l'estensione sia del data-model che dell'applicazione client-side.

Il data-model potrebbe permettere la specifica (per ogni utente) delle caratteristiche che CAO=S prevede per l'utente (competenza di dominio, competenza informatica specifica, competenza linguistica, abilità fisica e visiva, motivazione ed interesse, distrazioni di ambiente). Questi valori potrebbero essere messi a disposizione del client utilizzando le stesse unit che

permettono in BuCaBO la comunicazione con il client (XML Out Unit e Script Unit).

Il client potrebbe essere evoluto attraverso la parametrizzazione di alcune caratteristiche grafiche (come ad esempio la grandezza dei font) che verrebbero a modificarsi a partire dalle caratteristiche (CAO=S) degli utenti che le richiedono.

Bibliografia

[ABB08] Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., Fraternali, P. (2008). *Web applications design and development with WebML and WebRatio 5.0*. In Paige, R., Meyer, B. (Eds.) Proceedings of the 46th International Conference, Tools Europe 2008 (ICTE2008), 11: 392-411. Berlin, Germany: Springer. DOI: 10.1007/978-3-540-69824-1_22.

[Bac11] Bacelli, B. (2011). *Model-driven Usability: alcuni approcci e un prototipo*. In Tesi di laurea in Interazione Persona-Computer. Corso di laurea magistrale in informatica. Università di Bologna, Italy. URL: http://amslaurea.cib.unibo.it/2734/1/bacelli_beatrice_tesi.pdf (Last Visited: 29 February 2012).

[CFB03] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M. (2003). *Progettazione di dati e applicazioni per il Web*. Milano, Italy: McGraw Hill. ISBN-13: 9788838661389.

[CRC07] Cooper, A., Reimann, R., Cronin, D. (2007). *About Face 3: The essentials of interaction design*. Indianapolis, USA: Wiley Publishing Inc. ISBN: 0470084111.

[Don01] Donahue, GM. (2001). *Usability and the bottom line*. In IEEE Software, 18 (1): 31-37. Washington, DC, USA: IEEE Computer Society. DOI: 10.1109/52.903161.

[FB04] Folmer, E., Bosh, J. (2004). *Architecting for usability: A Survey*. In Journal of System and Software, 70 (1-2): 61-78. Amsterdam, Netherlands: Elsevier. DOI: 10.1016/S0164-1212(02)00159-0.

[GM01] Ginige, A., Murugesan, S. (2001). *Guest editors' introduction: web engineering - an introduction*. In IEEE MultiMedia Magazine, 8 (1): 14-18. Washington, DC, USA: IEEE Computer Society. DOI:10.1109/93.923949.

[HLM06] Hilz, M., Leitner, G., Melcher, R. (2006). *Usability of Web Applications*. In Web Engineering. Hoboken, NJ, USA: John Wiley & Sons, Ltd. ISBN-13: 978-0-470-01554-4.

[HT07] Hailpern, B., Tarr, P. (2007). Model-Driven Development: the good, the bad, the ugly. In IBM System Journal, 45 (3): 451-461. Armonk, NY, USA: IBM. DOI: 10.1147/sj.453.0451.

[JMS07] Juristo, N., Moreno, AM., Sanchez-Segura, MI. (2007). *Guidelines for eliciting usability functionalities*. In IEEE Transformation Software Engineering, 33 (11): 744 - 758. Washington, DC, USA: IEEE Computer Society. DOI: 10.1109/TSE.2007.70741.

[KMP04] Kappel, G., Michlmayr, E., Pröll, B., Reich, S., Retschitzegger, W. (2004). *Web engineering – old wine in new bottles?* In Koch, N., Fraternali, P., Wirsing, M. (Eds.) In Proceedings of the 4th International Conference on Web Engineering (ICWE2004): 6-12. Berlin, Germany: Springer.

[KZ10] Kesik, J., Zyla, K. (2010). *Usability comparison of WebRatio and Symfony for educational purposes*. In Proceedings of Electrotechnical Institute. Issue 247: 223-240. Institute of Computer Science, Lublin University of Technology, Poland.

[Low03] Lowe, D. (2003). *Web System requirements: an overview*. In Requirements Engineering Journal, 8 (2): 102-113. Berlin, Germany: Springer. DOI: 10.1007/s00766-002-0153-x.

[LPG10] Luna, E., Panach, J., Grigera, J., Rossi, G., Pastor, O. (2010). *Incorporating usability requirements in a test/model driven Web Engineering approach*. In Journal of Web Engineering, 9 (2): 132-156. Paramus, USA: Rinton Press, Incorporated. URL: http://dl.acm.org/citation.cfm?id=2011309.2011312 (Last Visited: 29 February 2012).

[LWE01] Lawrence, B., Wiegers, K., Ebert, C. Top risk of requirement engineering. (2001). In IEEE Software,18 (6): 62-63. Washington, DC, USA: IEEE Computer Society. DOI: 10.1109/52.965804.

[MT09] Molina, F., Toval, A. (2009). *Integrating usability requirements that can be evalueted in design time into Model Driven Engineering of Web Information System*. In Advances in Engineering Software Journal, 40 (12): 1306–1317. Amsterdam, Netherlands: Elsevier. DOI: 10.1016/j.advengsoft.2009.01.018.

[OR02] Olsina, L., Rossi, G. (2002). *Measuring Web Application Quality with WebQEM*. In IEEE Multimedia Magazine, 9 (4): 20-29. Washington, DC, USA: IEEE Computer Society. DOI: 10.1109/MMUL.2002.1041945

[Zol10] Zoli, E. (2010). *CAO=S un modello di progettazione Goal Oriented per interfacce web*. In Tesi di Laurea in Interazione Persona Computer. Corso di Laurea Specilistica in Informatica. Università di Bologna, Italy. URL: http://amslaurea.cib.unibo.it/1243/1/Zoli_Enrico_tesi.pdf (Last Visited 29 February 2012).