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: WIDGET USABILI

Tesi di Laurea in Interazione Persona-Computer

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
BRUNO BUCCARELLA

III Sessione
Anno Accademico 2010/11

INDICE

Capitolo 1
Introduzione

Il seguente progetto di tesi suggerisce una possibile soluzione alla mancanza di modelli concreti di progettazione di interfaccia utente che rispettino principi di usabilità. Si è cercato di integrare le teorie di due discipline, da una parte "Interazione persona computer" (HCI) e dall'altra "Model-driven engineering" (MDE). La finalità è quella di fornire attraverso un approccio MDE la generazione automatica di interfacce utente ricche ed interattive. 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. 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 (**PIM** - *Platform Indipendent Model).* 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*).

Nella parte iniziale del progetto si è cercato il linguaggio di modellazione idoneo alle esigenze sopra descritte; la scelta è ricaduta sul linguaggio di modellazione WebML. *WebML (Web Modelling Language)* è una notazione visuale per la specifica di composizione e navigazione di applicazioni ipertestuali per il web.
WebML è basato su standard di grande diffusione come il modello ER (entità-relazione) e UML. Questo linguaggio, appartenendo alla famiglia MDE, permette di

specificare applicazioni Web complesse in modo indipendente dalla piattaforma. 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. WebML fornisce la possibilità di definire una qualunque applicazione Web passando attraverso la modellazione di due modelli principali: il *data-model* (che consiste in un diagramma entità-relazione) e il *web-model* (che consente la definizione di pagine e la loro organizzazione interna in termini di componenti). Punto di forza è il tool di progettazione WebRatio che presenta un'interfaccia molto intuitiva e semplice nell'utilizzo. Il generatore di codice di WebRatio produce applicazioni web in J2EE a partire da modelli WebML.
WebML e WebRatio saranno ampiamente discussi nel capitolo 3.

Nella seconda fase si è individuato il framework JavaScript da utilizzare per la realizzazione dei widget grafici. E' stato scelto ExtJs in quanto è risultato uno strumento molto potente. Ext Js offre uno straordinario range di widget di user interface facilmente estendibili e permette di agganciare semplicemente funzionalità AJAX. Un codice pulito, la ricchezza di interfacce proposte, la completa compatibilità con tutti i browser e con gran parte degli altri framework Javascript fanno di Ext uno strumento ideale. Sarà fatta una panoramica generale del framework nel capitolo 3.

Per creare delle applicazioni che siano usabili, negli ultimi anni sono nate delle tecniche di progettazione user-centered che si basano sul coinvolgimento attivo dell'utente nella progettazione. Questi modelli di sviluppo si basano su processi iterativi di progettazione e fanno ampio uso di prototipi, richiedendo quindi competenze multidisciplinari nel team di sviluppo.
Le due tecniche più utilizzate sono dette Task Oriented Design e Goal Oriented Design. Come si evince dal nome la prima metodologia è incentrata sui task ed è rivolta ad organizzare la progettazione sull'analisi dei compiti che l'utente destinatario deve svolgere nel sistema. Il Goal Oriented Design mira invece a

soddisfare completamente gli obiettivi dell'utente (Goal) considerando sopratutto quelli personali e non solo quelli lavorativi.
Si è scelto di adottare il modello di design Goal Oriented CAO=S. 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 l'acronimo di *Concetti, Attori, Operazioni generano Strutture*. E' 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. Il modello è 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.

Per garantire un alto livello di astrazione e permettere ai widget di essere utilizzati per un ampio numero di domini applicativi, è stata utilizzata l'architettura MVC (Models Views Controllers).
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 che interagiscono tra di loro grazie al controller.
Una volta individuati gli strumenti si è fatta un'analisi degli approcci già presenti in letteratura per cercare di focalizzare le problematiche generali e le possibili soluzioni.

Dalla ricerca è risultato che allo stato attuale nessun approccio fornisce una soluzione completa al problema. Ognuno presenta dei punti critici o comunque è tuttora incompleto. Si è studiato nello specifico WebML e sono stati evidenziati i suoi difetti. E' proprio in questo punto che ci si è inseriti per trovare una soluzione. Alcuni di questi approcci sono stati studiati nel dettaglio e riassunti nel capitolo 2.
Dopo aver definito le tecnologie da utilizzare e la strada da intraprendere si è cercato il modo di intervenire per giungere ad una soluzione idonea al problema. Si è pensato di creare una meta-applicazione che permettesse di coniugare la modellazione WebML con la creazione di Rich Internet Applications (RIA) altamente usabili. Il risultato è stato "BuCaBO".
BuCaBO si pone in mezzo tra il risultato di una modellazione WebML server-side ed una architettura MVC client-side. Il server si occupa della persistenza dei dati, della gestione di accesso al sistema e della creazione di file XML che permettono la comunicazione con il client. Il client si occupa della visualizzazione e della modifica dei dati.
BuCaBO utilizza due strumenti per effettuare la comunicazione e lo scambio di dati fra il client e il server, uno che si può definire di "servizio" e l'altro di "configurazione".
Lo strumento di servizio è quello che si occupa di capire che tipo di informazioni devono essere visualizzate e quindi definire i modelli. Lo strumento di configurazione invece permette di definire chi deve vedere cosa e come.

Per testare il funzionamento di BuCaBO si è simulato un sistema di gestione universitario con due tipologie di utenti, professori e studenti. Gli utenti in questione possono effettuare operazioni diverse e quindi saranno abilitati a viste diverse con widget diversi.
E' stato prima realizzato un modello WebML in grado di rappresentare le caratteristiche di base del sistema ed istruito in modo tale da generare come output dei file XML. Su questi file vengono effettuate delle operazioni di parsing attraverso gli strumenti creati per effettuare la comunicazione tra client e server. Dopo questa

operazione viene creata ed avviata la parte client, vengono richiamati ed aggregati tutti i widget grafici. La composizione delle varie viste avviene attraverso lo strumento di configurazione che definisce quali dati devono essere visualizzati ed in quale modo.

Per mettere in evidenza sia le potenzialità che la totale indipendenza di BucaBo e dei widget creati si è generata un'altra applicazione di dominio completamente diverso da quello precedente. Il risultato è stato la realizzazione di un nuovo sistema che simula la gestione di un magazzino; al posto di professori e studenti ci sono fornitori e prodotti. E' possibile notare come le caratteristiche di usabiltà ed indipendenza dal contesto siano state mantenute.

Capitolo 2
Contesto Scientifico

Questo elaborato cerca di fornire una possibile soluzione alla mancanza di modelli concreti di progettazione di interfaccia utente che rispettino principi di usabilità. Si è cercato di integrare le teorie di due discipline, da una parte "Interazione persona computer" (HCI) e dall'altra "Model-driven engineering" (MDE).
Gli sviluppatori delle tradizionali applicazioni Web hanno concentrato tutta la loro attività intorno ad un'architettura client-server dove tutta l'elaborazione avviene lato server e al client vengono relegate le sole visualizzazioni di contenuti statici.
Questo approccio presenta molte limitazioni, soprattutto per la ricchezza di interfacce applicative che potrebbero essere costruite con un approccio diverso.
Le vecchie applicazioni Web potrebbero essere sostituite dalle cosiddette Rich Internet Applications (RIA) [MGP08] che forniscono interfacce utente più ricche e interattive.
Una tecnologia che consente la creazione di queste nuove interfacce è AJAX (Asynchronous JavaScript and XML) [Pau05] che permette di ottenere un elevato livello di interattività dell'utente.
Dopo la sua nascita, sono comparsi numerosi framework e librerie così la tecnologia ha avuto una rapida evoluzione. Un modo per affrontare questo problema è quello di effettuare delle astrazioni attraverso un approccio MDE quindi attraverso un linguaggio di modellazione ed ottenere un'applicazione Ajax generata automaticamente. Questo permette di generare applicazioni per domini differenti in modo semplice ed automatico.
Tra le diverse soluzioni che sono state implementate, ognuna con i propri pregi e difetti, citiamo:

1. UWE una notazione UML che mira alla semplice generazione di classiche

2. WebML [CFB00], è evidente una forte attenzione alla presentazione e l'assenza di una notazione di supporto per le interfacce AJAX.
3. Rux-model un approccio MDA per Rich Applicazioni Internet (RIA) su base WebML. E' limitato in quanto allo stato attuale è in grado di generare solo interfacce utente senza tenere in considerazione problemi di back-end o toolkit di generazione.
4. [MMV06] Quest' approccio è basato sulla descrizione XML di User Interface e su XSLT. Manca di una notazione visiva e non può essere modellato utilizzando dei tool esistenti. Questo è vero anche per tutti gli altri linguaggi basati su XML.
5. [GMD08] E' una soluzione basta sul framework AndroMda e sulla creazione di un *cartridge* Ajax. E' un approccio molto interessante ma il cartridge è ancora work in progress .

Dopo aver elencato i possibili approcci al problema andrò ad effettuare una breve presentazione delle varie soluzioni ad esclusione di WebML che è stato ampliamente discusso nel capitolo precedente.

UWE

Koch and Kraus propongono UWE [KK02], una notazione UML per modellare la navigazione e gli aspetti concettuali di un'applicazione Web. Il loro però è un approccio che va nella direzione delle classiche applicazioni web.
L'UML-based Web Engineering (UWE) descrive una metodologia sistematica per lo sviluppo di applicazioni Web che utilizza esclusivamente le tecniche UML. UWE descrive modelli di navigazione e di presentazione definendo speciali elementi di modellazione UML.
Le attività principali della metodologia UWE sono:

1. l'analisi dei requisiti,
2. la modellazione concettuale,
3. la modellazione navigazionale,
4. la modellazione della presentazione,
5. la modellazione dei tasks,
6. la modellazione della distribuzione dei componenti e delle applicazioni web
7. la visualizzazione degli scenari Web.

E' prevista la generazione semi-automatica delle applicazioni web a partire da tali modelli. Per questo scopo è stata implementata un'estensione del tool ArgoUML.
Il vantaggio di UWE è quello di poter creare con facilità linguaggi di modellazione per specifici domini applicativi con elementi di modellazioni definiti in modo più preciso ed espressivo. Il rischio è che l'eccessivo uso di stereotipi renda un linguaggio troppo difficile da utilizzare e da comprendere.

Rux-Model – WebML

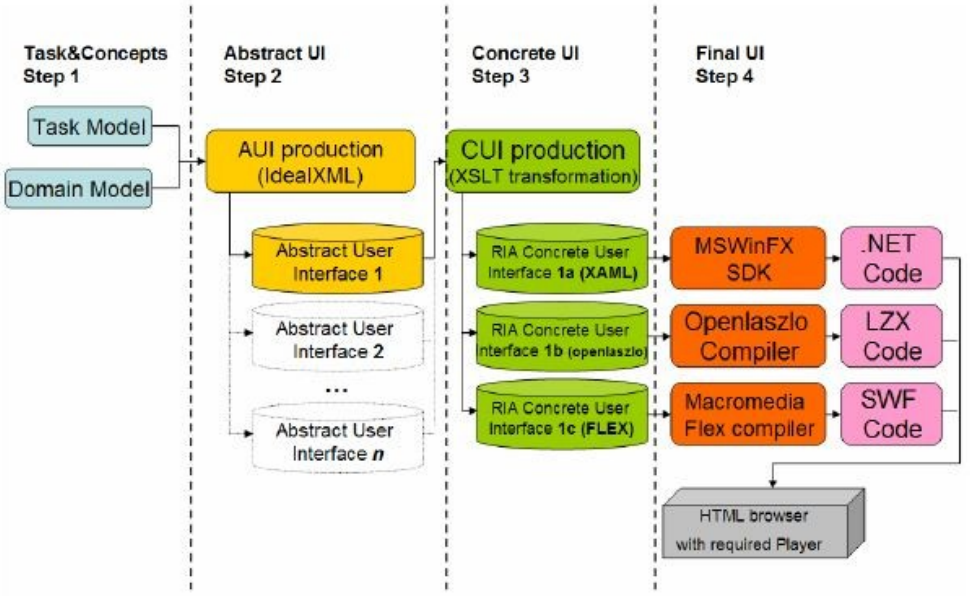
[CLC07] L'obiettivo dell'approccio è duplice: identificare i concetti principali che devono essere presi in considerazione per la progettazione di Rich Internet Applications e mostrare come questi concetti possano essere catturati attraverso l'integrazione di due proposte di Web Engineering WebML e Rux-Model.WebML viene esteso con notazioni che fanno riferimento ad una separazione client-side server-side.
Nella parte iniziale del processo di sviluppo si effettua una modellazione WebML e nelle ultime fasi una modellazione con notazione Rux-model.

XML – XSLT

[MMV06] Questo approccio effettua un processo iterativo di trasformazioni XSLT per tradurre l'interfaccia modellata in un' interfaccia utente finale. Per modellare tutti livelli viene utilizzato UsiXML (USer Interface eXtensible Markup Language), questo linguaggio incorpora quattro livelli di astrazione. La struttura del processo di sviluppo è così definita:

1. Task and concepts,
2. Abstract User Interface (AUI),
3. Concrete User Interface (CUI),
4. Final User Interface (FUI)

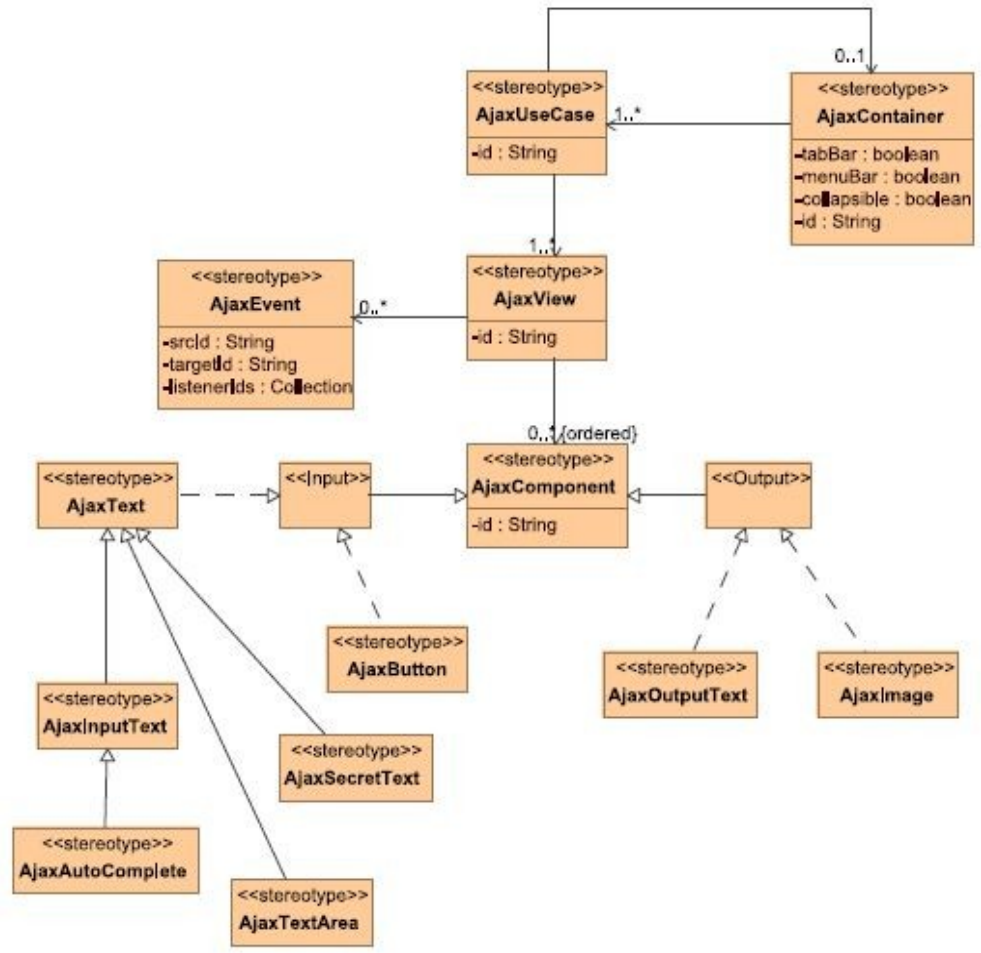
E' richiesto un User Interface Description Language (UIDL) per descrivere una UI a qualsiasi livello di astrazione. Il processo di sviluppo ed il passaggio da un livello ad un altro può essere riassunto nel seguente schema:



AndroMDA - cartidge Ajax

[GMD08] Il lavoro è basato su AndroMda un framework MDA opensource, che viene fornito con una serie di plugin, chiamati cartridge, che utilizzano una combinazione di framework quali JSF, Struts, JSR / EJB, Spring e Hibernate.
L'idea di base è quella di creare un cartridge AJAX. La ragione principale di questa decisione è la facilità di integrazione tra AndroMDA e le tecnologie web. In AndroMDA si possono utilizzare gli stessi modelli per generare una web application che utilizza JSF-Spring-Hibernate, nonché una che utilizza Struts, Hibernate. Inoltre, AndroMDA definisce già dei meta-elementi che dovrebbero valere per diverse di piattaforme. Questo rende l'implementazione di un cartridge AJAX più facile dal momento che ci si può concentrare direttamente sulla trasformazione e la creazione di uno specifico AJAX meta-model.
Le specifiche PSM sono contenute nei cartridge, AndroMDA invece contiene un elenco di elementi indipendenti dalla piattaforma che sono utilizzati per modellare PIM. Questi elementi dovrebbero essere generalmente sufficienti per modellare la maggior parte delle situazioni, è possibile tuttavia aggiungere nuovi elementi PIM come ad esempio stereotipi UML.
La trasformazione tra PIM, PSM e il codice viene effettuata da ciò che AndroMDA chiama metafacades. Per una tipica applicazione web, AndroMDA utilizza diagrammi UML dei casi d'uso per definire i possibili scenari.
Il funzionamento interno di ciascun caso d'uso è specificato attraverso l'utilizzo di diagrammi di attività UML.
I diagrammi descritti riguardano la parte front-end dell'applicazione e devono essere collegati, con dei diagrammi per modellare la parte back-end. Il back-end delle applicazioni web è modellato utilizzando dei diagrammi delle classi. I diagrammi delle classi rappresentano i controller e l' entità persistenti dei framework come ad esempio Hibernate e Spring.

Venendo al dunque del problema gli autori hanno creato un meta-modello Ajax da utilizzare nello strato PSM di AndroMda ovvero all'interno di un *cartidge*, questo meta-modello dovrebbe essere in grado di catturare la struttura principale della maggior parte delle interfacce utente.



La struttura può essere definita come un insieme di componenti di input, output e layout.

I componenti sono creati sulla base della struttura della libreria Swing API e di altri frameworks Ajax come ad esempio ICEFACES. Un esempio può aiutare a capire meglio, un AjaxUseCase è il risultato della trasformazione di un elemento

FrontEndUseCase del PIM. Nonostante a livello teorico sembra tutto ben definito, il cartridge Ajax è ancora work in progress quindi la generazione automatica dei widget Ajax non è ancora del tutto possibile.

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%

Immagine: Tabella di confronto tra l'approccio allo sviluppo tradizionale e MDE

http://www.omg.org/mda/mda_files/M1Global.htm

- 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.

- 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).

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.
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.
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:

- l'*OK link*: è il link che viene seguito quando l'operazione va a buon fine;
- il *KO link*: che viene esguito 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 appliazioni Web che seguono le metodologie di MDE.

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

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 templates di pagina per le piattaforme JSP e .NET. WebRatio è progettato e realizzato da Web Models, 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.
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;
- 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;

Immagine:PatternMVC
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 dati frequentemente 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.

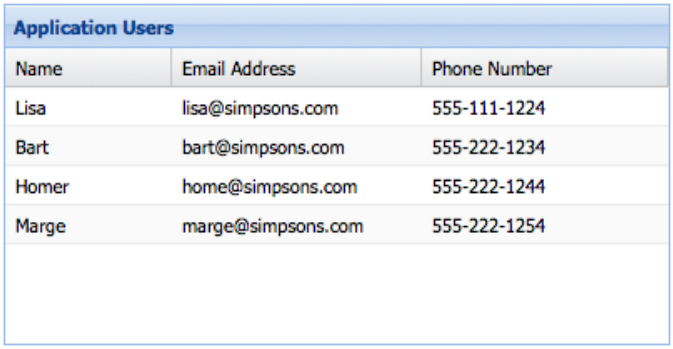


Immagine: Grid

http://docs.sencha.com/ext-js/4-0/#!/guide/grid

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.*

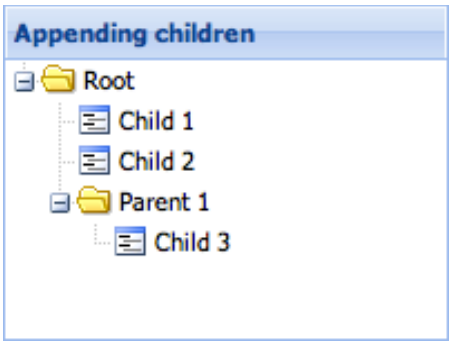


Immagine: Tree Panel

http://docs.sencha.com/ext-js/4-0/#!/guide/tree

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.

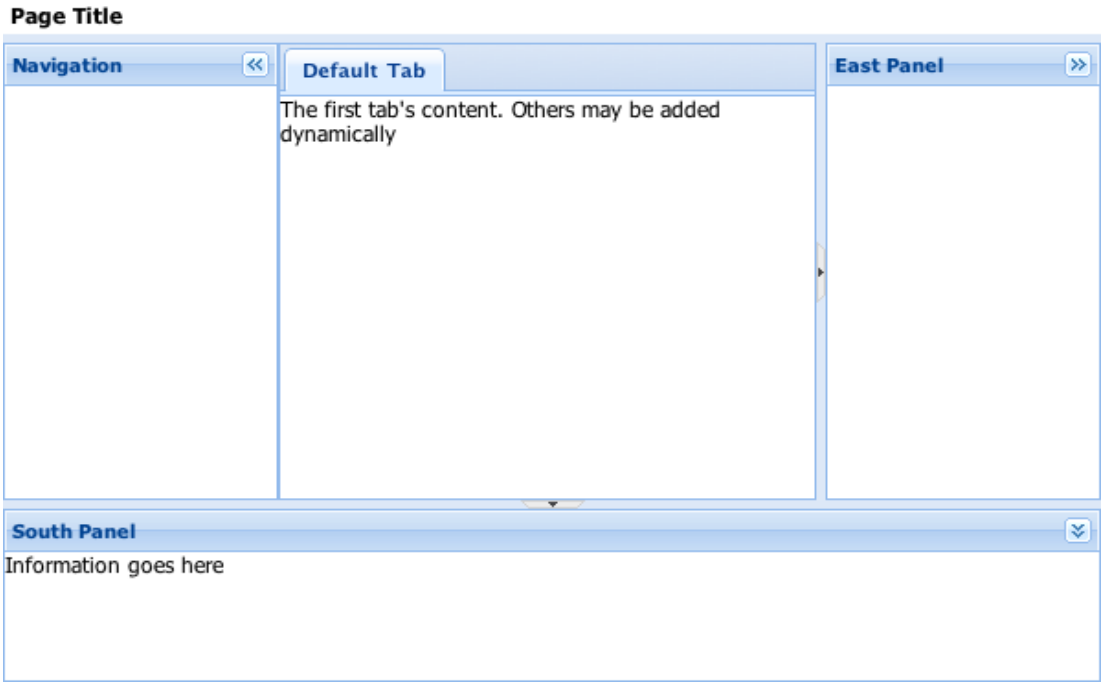


Immagine: Viewport

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 adottato il modello di design Goal Oriented CAO=S [Zol10]. CAO=S è stato realizzato 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: 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*.

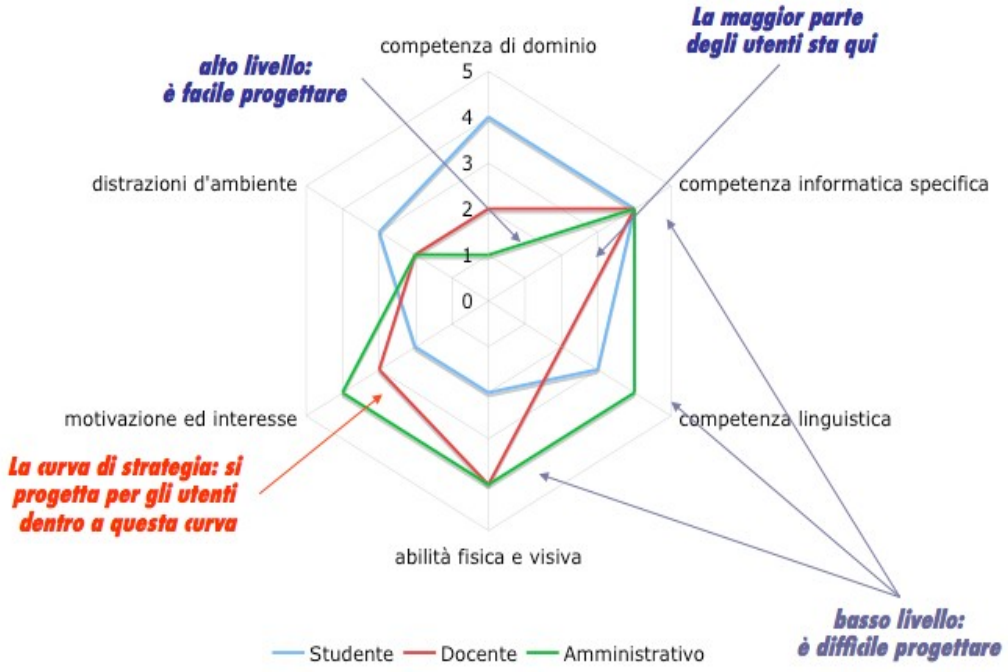


Immagine: Grafico di Strategia [Zol10]

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:

- *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 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 Widget con ExtJS

Dopo aver analizzato tutte le componenti che hanno portato alla realizzazione del progetto di tesi si può passare ad un'analisi attenta dei widget realizzati con il framework ExtJS. L'idea di base è quella di realizzare degli oggetti indipendenti dal dominio dell'applicazione e con delle alte caratteristiche di usabilità.
Per poter realizzare ciò si è pensato di effettuare una netta separazione tra *client-side e server-side.* Le due parti entrano in comunicazione attraverso dei file XML.
Per prima cosa, si è pensato a quale fosse l'architettura migliore per realizzare la parte client-side, l'architettura che ha risposto a queste esigenze è stata quella *Models Views Cotrollers.*
L'architettura MVC permette di separare le operazioni di gestione e visualizzazione dei dati e affida al Controller il compito di fare interagire queste due componenti.
Il Controller dell'applicazione, nel momento in cui viene chiamato in causa per la visualizzazione delle informazioni riguardanti un Concetto, prima reperisce tutte le informazioni interrogando lo Store e successivamente le trasmette alla View.

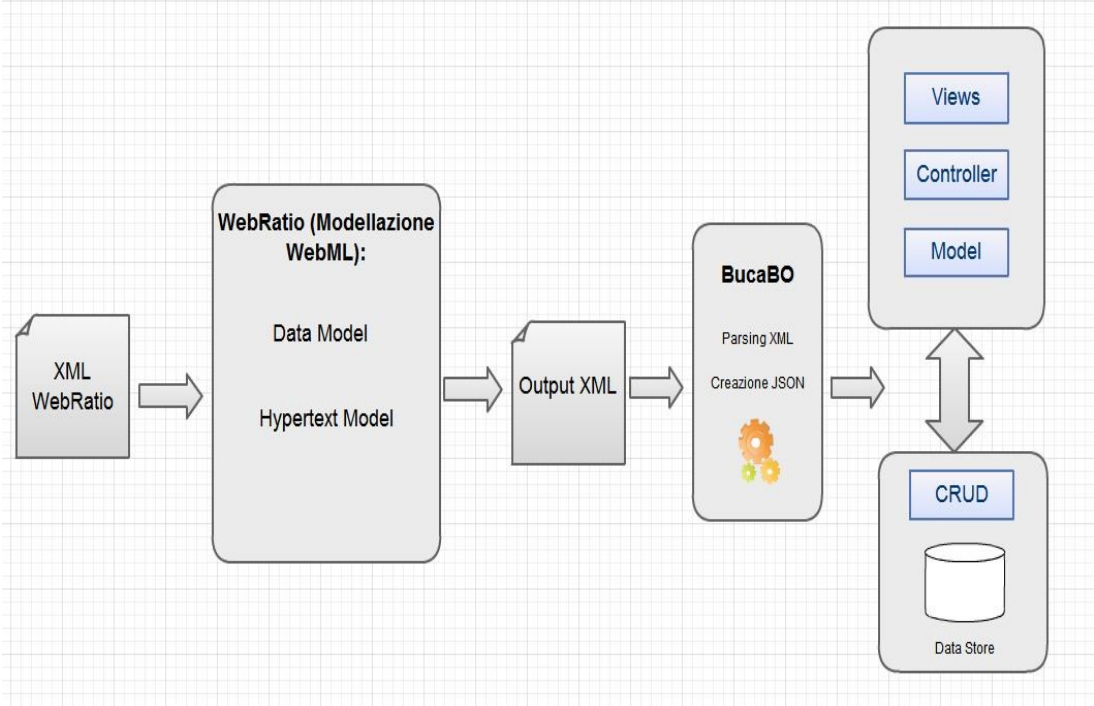


Immagine: Struttura progetto di Tesi

L'immagine mostra la struttura del progetto di tesi, le operazioni di parsing del file Xml e della creazione dei Json avvengono all'interno della *directory* "Js". In questa cartella sono presenti tutti i file di configurazione che permettono sia il passaggio che la trasformazione dei dati provenienti dall'output di WebRatio ed il file "app.js" che avvia l'applicazione client e quindi la creazione dell'interfaccia.
Invece nella *directory* "app" è realizzata la struttura MVC.

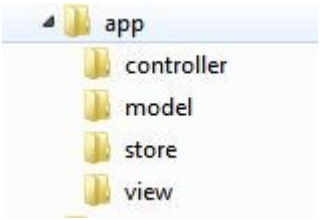


Immagine: Strutture MVC

MVC, realizza un sistema di sviluppo dell'interfaccia a moduli intercambiabili in modo da poter utilizzare a proprio piacimento le varie View del sistema, ottenendo dei frammenti componibili in modo semplice.
I Modelli e gli Store di ExtJs sono i componenti che permettono di rappresentare le informazioni. 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 model viene realizzato per permettere allo Store di funzionare, lo Store è fondamentalmente un insieme di Model instances.
Viene quindi generato uno Store per ciascun Concetto e viene associato al rispettivo Model. Lo Store si compone di un *proxy* che ha due funzioni:

1. lettura attraverso un *reader* di tipo "xml"
2. scrittura attraverso un *writer di tipo "xml"*.

Grazie a queste due funzioni si implementa un modello CRUD : create, read, update e destroy.
Dall'attenta analisi di CAO=S e delle esigenze del progetto si è giunti alla conclusione che le View possono essere di diverso tipo: liste, sommario, form, navigazione. Si è partito da ciò per stabilire quali fossero i widget da creare.
Quindi i widget realizzati sono stati:

1. Tree panel (per la navigazione)
2. Grid (per la visualizzazione delle griglie)
3. Combobox (per la ricerca all'interno di una lista)
4. Cruscotto (insieme di panel per i sommari).

Grid (Liste)

Le viste che vengono generate sono diverse in base alla possibilità o meno da parte di un utente di effettuare un'operazione. Ad esempio nello specifico caso dell'applicazione generata (sistema universitario), si è ipotizzato di avere due tipologie di utenti: studenti e professori. I primi abilitati a funzioni di *create, delete, update, read* e quindi con maggiori possibilità di modifica.
Conseguentemente le griglie che andranno a rappresentare i dati avranno delle *actioncolumn* che permetteranno le operazioni sopra citate.

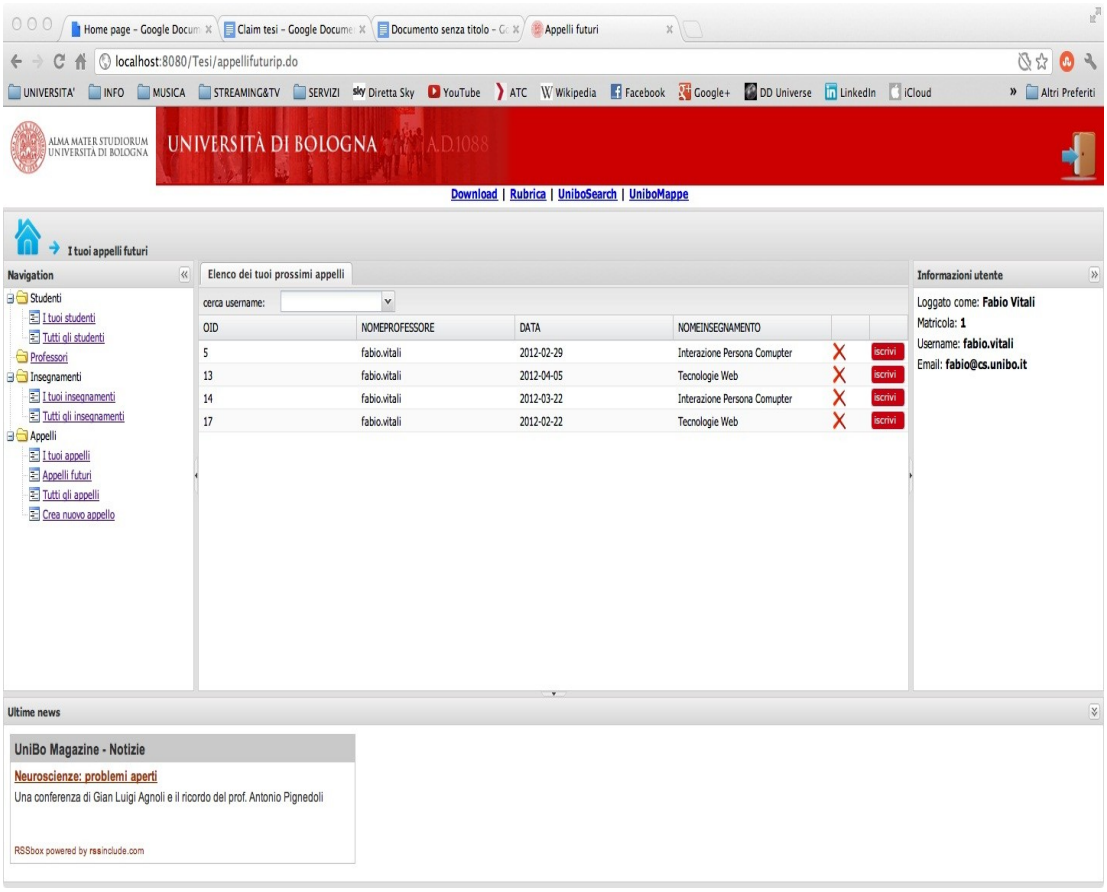


Immagine: Esempio Grid

Nello specifico caso dell'immagine sono presenti due *actioncolumn,* una con la "X" che effettua un'operazione di *Delete* e una con "Iscrivi" che ne effettua una di *Update.*

Nella creazione del widget *Grid* si è fatta molta attenzione a crearlo in modo che esso sia completamente indipendente dal dominio di applicazione e quindi che possa essere riutilizzato in più contesti. Ciò è stato reso possibile grazie al passaggio tra BucaBo e l'applicazione client di variabili contenenti delle strutture JSON. Queste variabili sono generate in modo dinamico a partire dall'output Xml di WebRatio.

Le griglie (definite nel modello CAO=S come viste multiple) permettono diverse operazioni:

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

La griglia è il widget più potente e con le più alte caratteristiche di usabilità.

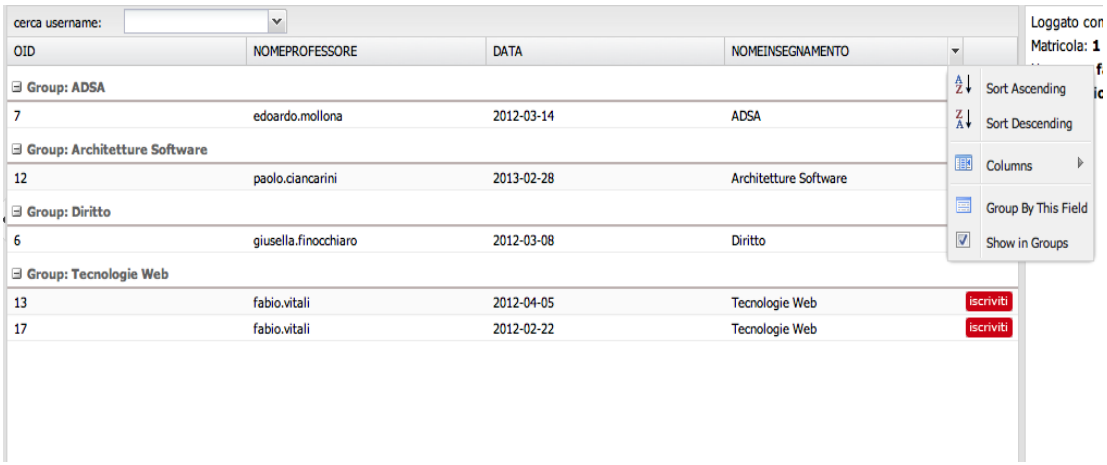


Immagine: Esempio Grid (filter, sort, group by)

Nell'immagine si nota come sia possibile realizzare le operazioni sopra elencate attraverso un menù *dropdown* presente su ogni colonna.

Cruscotto (Sommario)

Nella Homepage di ogni utente, è presente un "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)). Tecnicamente è un insieme di panel all'interno di un *Container.*

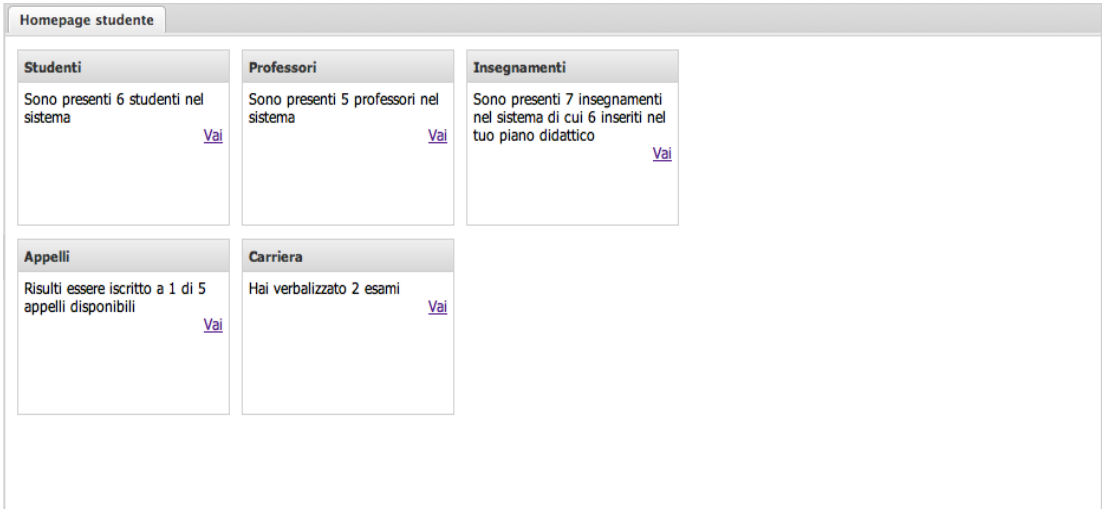


Immagine: Esempio Cruscotto

Questo è il caso di un cruscotto di uno studente, ogni pannello contiene una serie di informazioni ed un link alla vista di riferimento.
La creazione dei panel all'interno del *Conteiner* viene anche questa volta effettuata attraverso una variabile contenente una struttura JSON creata in BucaBo.

Tree panel (navigazione)

La navigazione viene gestita attraverso il componente *tree panel* che presenta una struttura ad albero formata da *root e child*, la composizione dell'albero è dinamica e viene creata nei file di configurazione di BucaBo. Nelle voci di menù dell'albero ci saranno solo le pagine che l'utente è abilitato a visualizzare.

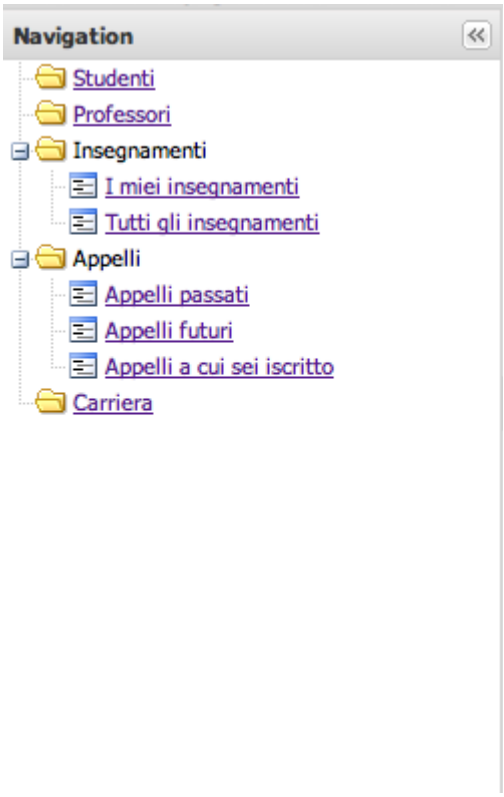


Immagine: Esempio Tree Panel

L'immagine di esempio rappresenta il menù di navigazione di uno studente ed al suo interno vi sono solo le viste a lui consentite.
La gestione dei permessi sulle pagine viene gestita da WebRatio con un struttura ER *Group-Module-User,* l'utente potrà visualizzare solo le pagine presenti nel suo modulo.

Combobox (ricerca in una lista)

Il Combobox è il widget che permette la ricerca su un attributo di un'istanza, può essere collegato a diversi componenti. Nello specifico dell'applicazione di esempio generata (sistema universitario) è stato utilizzato per filtrare gli elementi di una griglia e nella compilazione di un form.

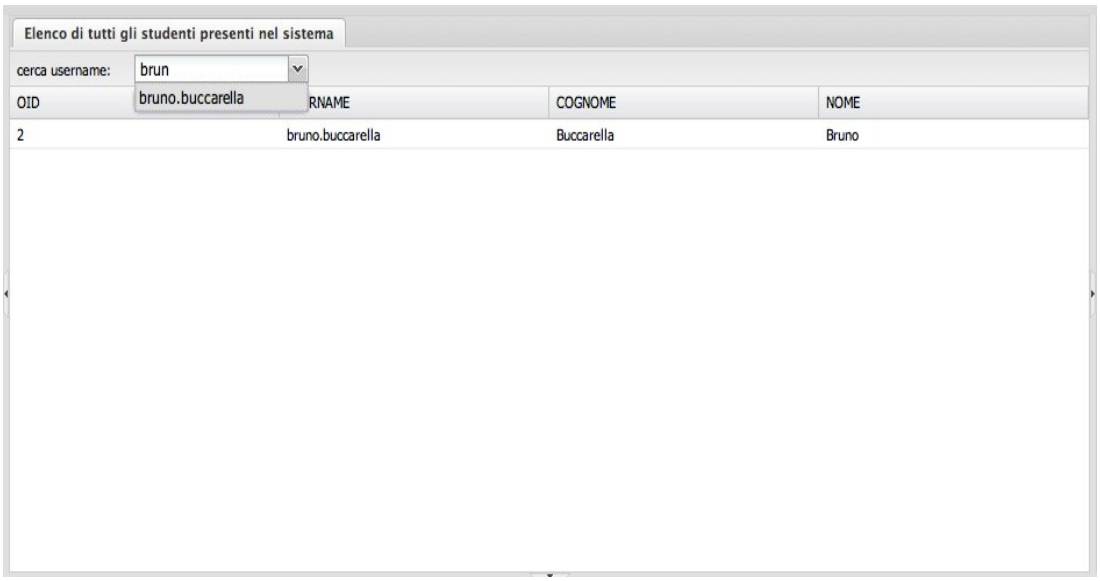


Immagine: Esempio Combobox

Nell'esempio si può notare come avvenga il filtro sulla griglia già all'inserimento della quarta lettera. Il combobox è molto utile nella gestione di grandi quantità di dati, per poter funzionare deve essere associato ad uno Store e deve essere definito il parametro su cui effettuare la ricerca. C'è da notare che il filtro non viene effettuato sulla griglia ma sullo Store che rinvia alla griglia i dati filtrati.

Aggregazione Widget

L'aggregazione dei widget all'interno di una vista avviene grazie al *Viewport.* Si è scelto di utilizzare una struttura con cinque regioni in quanto permette una buona gestione degli spazi ed un corretto posizionamento dei widget all'interno della pagina. Il Viewport viene creato all'interno del file "app.js" ovvero all'avvio dell'applicazione client-side.

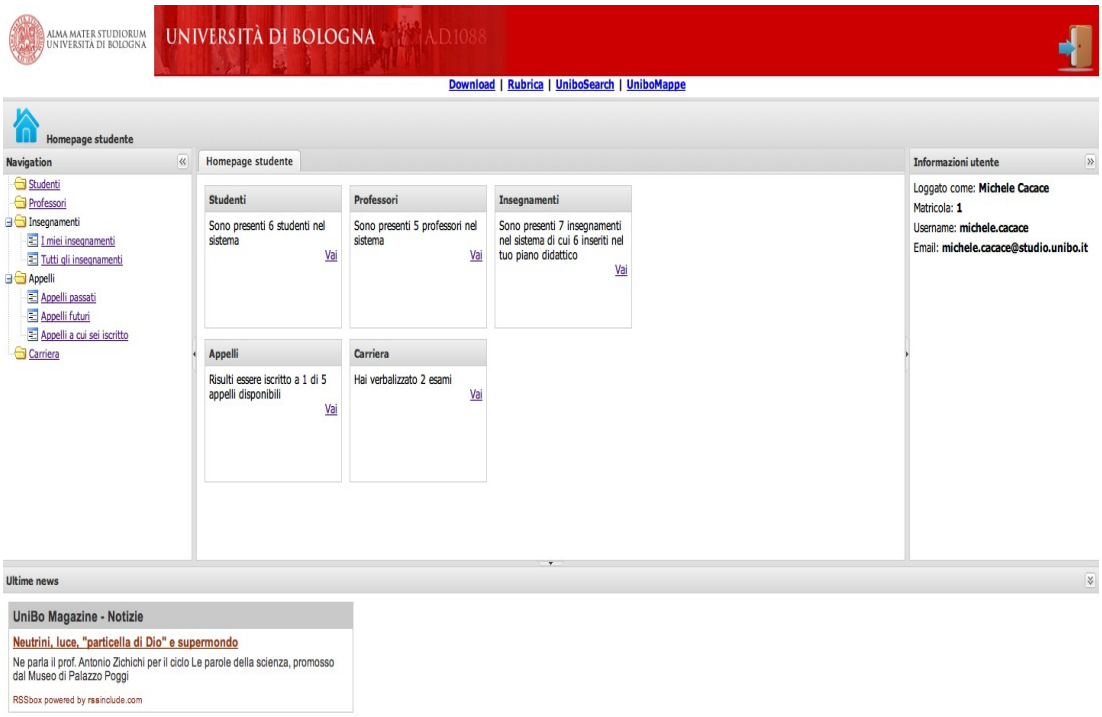


Immagine: Esempio Viewport

Questo è l'esempio della Homepage di uno studente. A sinistra nella regione ovest è presente il widget "Tree" che permette la navigazione all'interno dell'applicazione, nella regione centrale (in questo caso) abbiamo il cruscotto che corrisponde al widget "panel2". Nella regione nord è presente il collegamento alla Homepage e l'indicazione della pagina in cui l'utente si trova, nella regione est invece un pannello fornisce delle informazioni utili all'utente. Infine nella regione sud è stato inserito un lettore rss esterno.

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.*

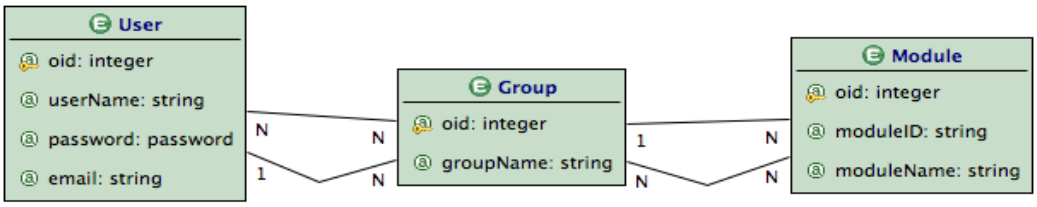


Immagine: Porzione Data-model gestione utenti

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 possibile 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.

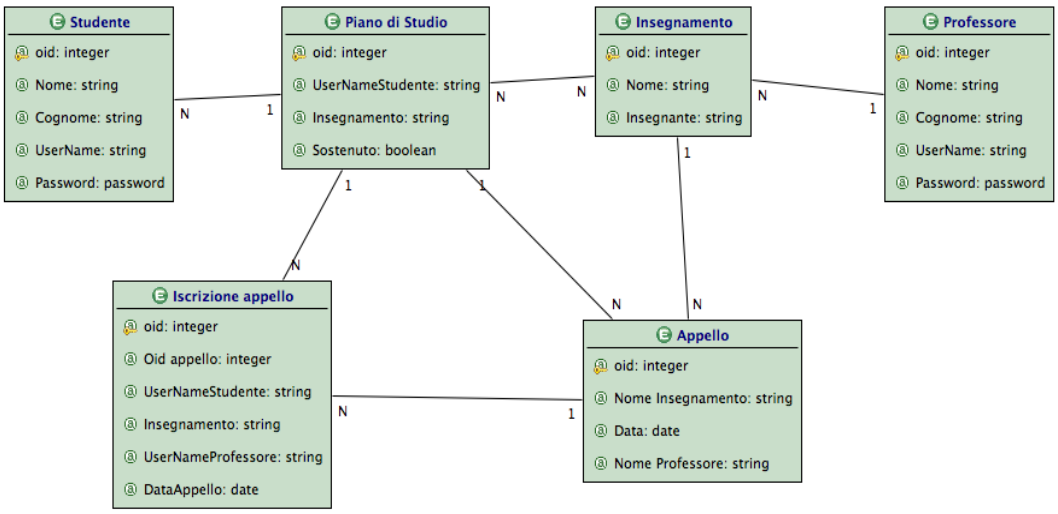


Immagine: Data-model sistema universitario

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

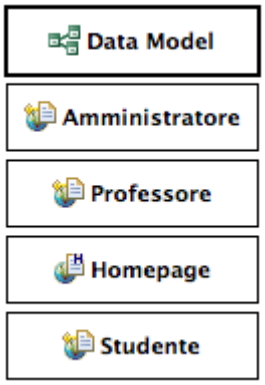


Immagine: Site-view

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.

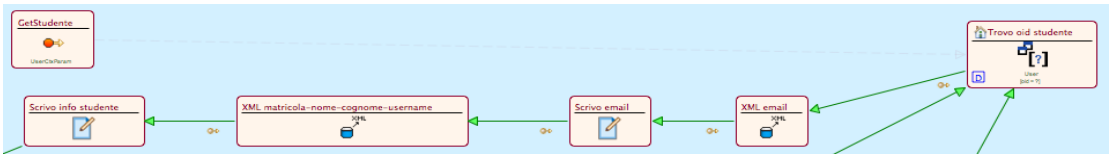


Immagine: Esempio XML Out Unit e Script Unit

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 contenente 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.

Immagine: Esempio Login

Gli elementi che accompagnano l'utente durante tutta l'esperienza 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).

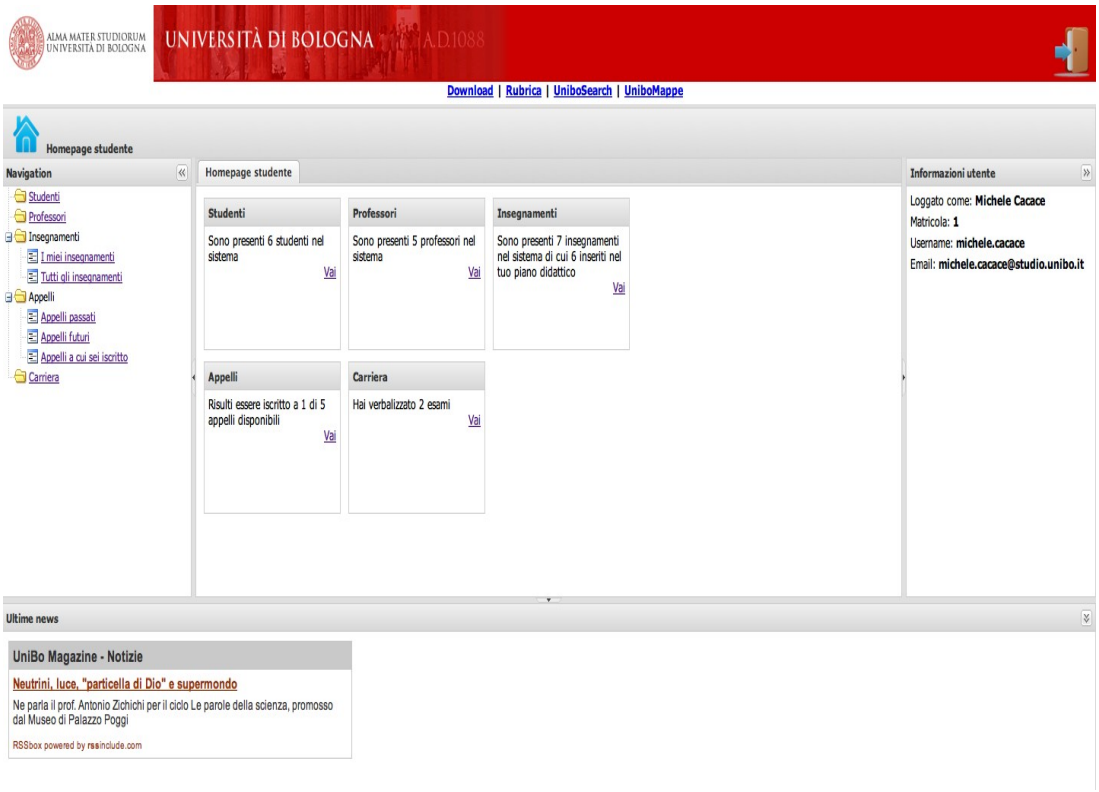


Immagine: Esempio Interfaccia

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.

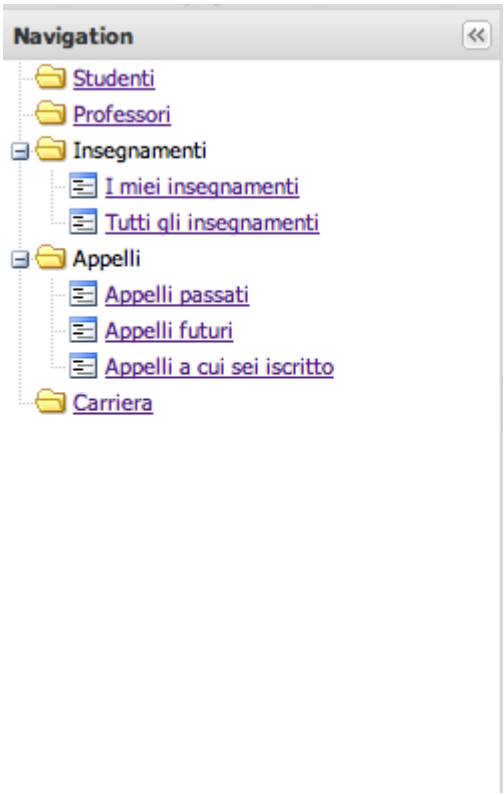


Immagine: Tree studente

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.

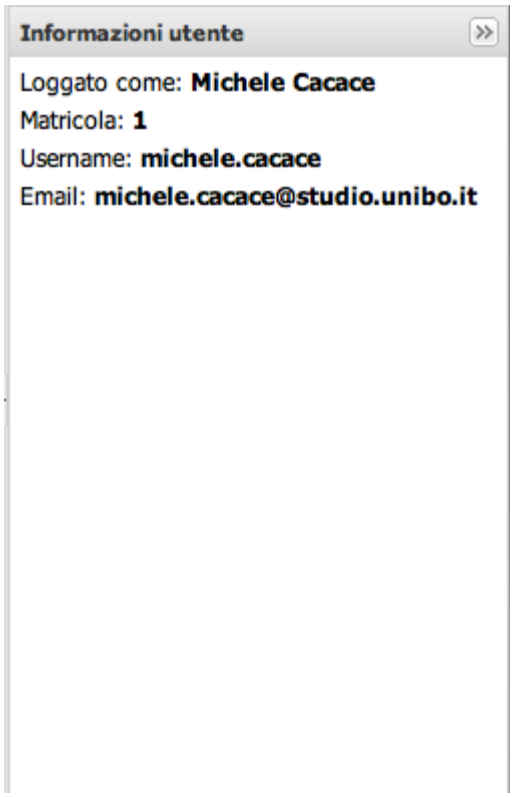


Immagine: Informazioni studente

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.

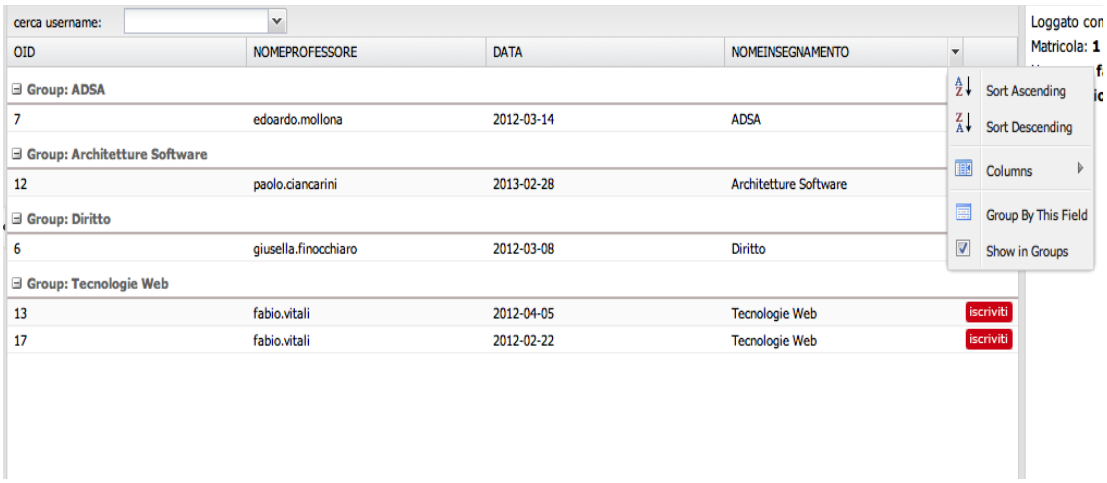


Immagine: Grid Insegnamenti

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

Immagine: Action column iscrizione appello

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 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.

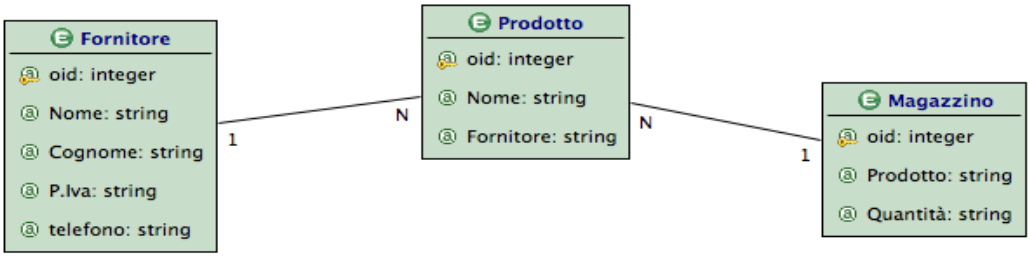


Immagine: Data-model sistema gestione magazzino

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.

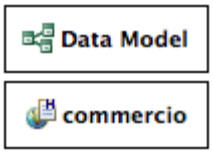


Immagine: Site View

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

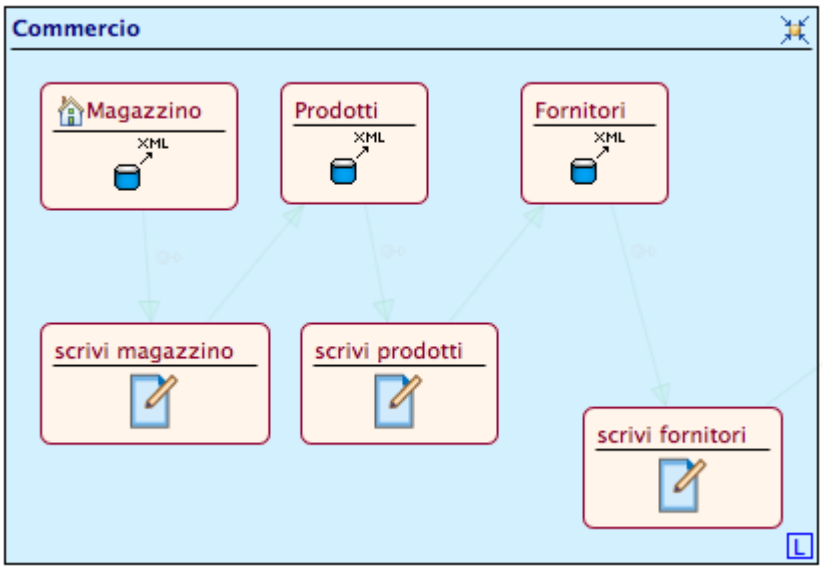


Immagine: XML Out Unit e Script Unit

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

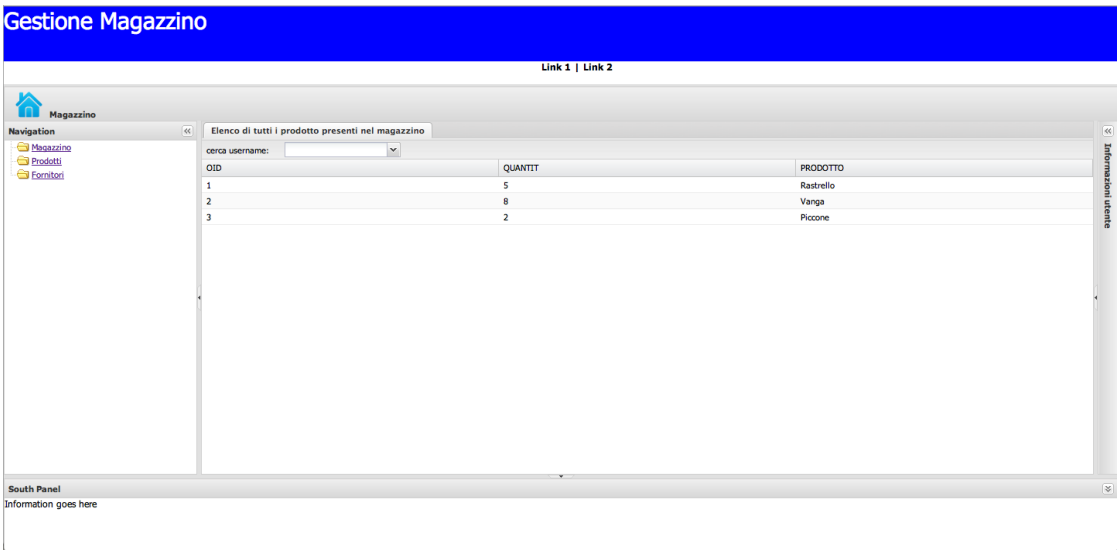


Immagine: Esempio Interfaccia Sistema di Gestione Magazzino

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 1 ora.

Conclusione

In questa dissertazione si è cercato di integrare le teorie di due discipline, da una parte "Interazione persona computer" (HCI) e dall'altra "Model-driven engineering" (MDE) per la realizzazione di applicazioni web che rispettino principi di usabilità.
Si è fatta un'analisi degli approcci già presenti in letteratura per cercare di focalizzare le problematiche generali e le possibili soluzioni. Dalla ricerca è risultato che allo stato attuale nessun approccio fornisce una soluzione completa al problema.
E' stata perciò realizzata la meta-applicazione BuCaBO che permette di creare applicazioni web altamente usabili a partire dal linguaggio di modellazione WebML e dal suo tool di sviluppo WebRatio.
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.
Per permettere all'interfaccia grafica di avere delle caratteristiche di usabilità sono stati creati dei widget con il Framework ExtJS completamente indipendenti dal contesto. Per garantire un alto livello di astrazione e permettere ai widget di essere utilizzati per un ampio numero di domini applicativi, è stata utilizzata l'architettura MVC (Models Views Controllers).
Per creare delle applicazioni che siano usabili, si è pensato ad una tecnica di progettazione user-centered, si è quindi scelto di adottare il modello di design Goal Oriented CAO=S.
Per testare le capacità di BuCaBO sono stati realizzati due prototipi, un sistema di gestione degli esami universitari ed un sistema di gestione del magazzino. Si è potuta notare la semplicità e la facilità con cui possono essere create applicazioni web usabili in poco tempo.

BuCaBO potrebbe essere ulteriormente migliorato rendendo “adattabili“ le interfacce grafiche messe a disposizione degli utenti. Tutto ciò potrebbe essere implementato sia inserendo nuove proprietà al data-model che ai widget grafici che permettono la creazione delle interfacce.
Le caratteristiche che CAO=S prevede per l'utente dovrebbero poter essere specificate nel data-model e passate al lato client attraverso le unit “XML Out Unit” e “Script Unit” che permettono appunto la comunicazione client-server.
Il client potrebbe essere evoluto attraverso la parametrizzazione di alcune caratteristiche grafiche (come ad esempio la grandezza dei font) che cambierebbero in base alle 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 (ICTE 2008), 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. http://amslaurea.cib.unibo.it/2734/1/bacelli_beatrice_tesi.pdf (Last Visited: 29 February 2012).

[CFB00] Ceri, S., Fraternali, P., Bongio, A. (2000). *Web modeling language (WebML): a modeling language for designing web sites*. In Computer Networks: The International Journal of Computer and Telecommunications Networking, 33 (1-6):137–157. Elsevier Amsterdam, Netherlands. DOI: 10.1016/S1389-1286(00)00040-2.

[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.

[CLC07] Preciado, J., Linaje, M., Comai, S., Sanchez-Figueroa, F. (2007). *Designing Rich Internet Applications with Web engineering methodologies*. In Huang, S., Di Penta, M. (Eds.), Proceedings of the 9th IEEE International Symposium on Web Site Evolution (WSE'07): 23–30. Washington, DC, USA: IEEE Computer Society. DOI: 10.1109/WSE.2007.4380240.

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

[GMD08] Gharavi, V., Mesbah, A., van Deursen, A.. (2008). *Modelling and Generating AJAX Applications: A Model-Driven Approach*. In Gaedke, M., Bieliková, M., (Eds.), Proceedings of the 7th International Workshop on Web-Oriented Software Technologies (IWWOST'08): 38-43. Bratislava, Slovakia: Vydavateľstvo STU. ISBN-13: 9788022728997

[KK02] Koch, N., Kraus, A. (2002). *The expressive power of UML-based web engineering.* In Schwabe, D., Pastor, O., Rossi, G., Olsina, L. (Eds.), Proceedings of the 2nd International Workshop on Web-oriented Software Technology (IWWOST'2002): 105–119. Berlin, Germany: Springer. ISBN:3540002332

[MGP08] Meliá, S., Gómez, J., Pérez, S., Díaz, O. (2008). *A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA*. In Schwabe, D., Curbera, F., Dantzig, P. (Eds.). In Proceedings of the 2008 Eighth International Conference on Web Engineering (ICWE '08). Washington, DC, USA: IEEE Computer Society. DOI: 10.1109/ICWE.2008.36.

[MMV06] Martìnez-Ruiz, F., Munoz Arteaga, J., Vanderdonckt, J., Gonzàlez-Calleros, J. (2006). *A first draft of a model-driven method for designing graphical user interfaces of Rich Internet Applications.* In Sanchez, A. (Eds.), Proceedings of the Fourth Latin American Web Congress: 32–38 (LA-WEB

'06). Washington, DC, USA: IEEE Computer Society. DOI: 10.1109/LA-WEB.2006.1.

[Pau05] Paulson, L. (2005). *Building Rich Web Applications with Ajax.* In Computer, 38(10):14–17. Washington, DC, USA: IEEE Computer Society. DOI: 10.1109/MC.2005.330.

[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. http://amslaurea.cib.unibo.it/1243/1/Zoli_Enrico_tesi.pdf (Last Visited 29 February 2012).