Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Specialistica in Informatica

CAO=S UN MODELLO DI PROGETTAZIONE GOAL ORIENTED PER INTERFACCE WEB

Tesi di laurea in Interazione Persona-Computer

Relatore:
Chiar.mo Prof.
Fabio Vitali

Presentata da:
Enrico Zoli

Sessione I
Anno Accademico 2009 - 2010

Ben venga il caos, perché l'ordine non ha funzionato.
Karl Kraus

Indice

Capitolo 1

Introduzione

Lo scopo di questo elaborato di tesi è la progettazione di un modello di design goal-oriented da utilizzare per progetti economicamente contenuti, dove non vi sia la possibilità di coinvolgere direttamente un esperto di usabilità e non si disponga di un budget adeguato per effettuare un'analisi delle categorie di utenze di riferimento. Con questo modello di design si desidera permettere a un team di sviluppo tradizionale di evitare comuni errori nella progettazione di applicazioni usabili senza dover possedere specifiche conoscenze nel campo.

Spesso, a causa dei tempi ridotti di rilascio e di un budget limitato, vengono sviluppate applicazioni che, non tenendo conto del punto di vista dell'utente non lo agevolano nel raggiungimento di obiettivi, mancando così di efficacia, efficienza e soddisfazione. L'usabilità è quindi un aspetto da non trascurare, considerata come la misura che indica la distanza tra il modello mentale del progettista e quello costruito dall'utente durante l'utilizzo del sistema regolandone l'interazione.

Per migliorare l'usabilità del software sono nate negli anni tecniche di progettazione user-centered che si basano sul coinvolgimento attivo dell'utente nella progettazione atte a identificare nella maniera più accurata possibile i requisiti, i loro compiti nei confronti del sistema e il contesto d'uso nel quale si posizionano. 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 attualmente 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, enfatizzando l'uso di descrizioni dettagliate di task realistici come base per la progettazione e la verifica dell'interazione. Il Goal Oriented Design mira invece a soddisfare completamente gli obiettivi dell'utente (Goal) considerando sopratutto quelli personali e non solo quelli lavorativi. Esempi di obiettivi personali sono l'essere rapido, il non fare troppi errori o il non annoiarsi. In questo modello di sviluppo l'utente viene analizzato in maniera più dettagliata introducendo i concetti di personaggio e scenario. Il primo consiste nella modellazione astratta dell'utente per la creazione di pattern comportamentali e motivazionali. I personaggi vengono usati per sviluppare gli scenari che descrivono come gli utenti interagiscono col sistema portando a termine i loro obiettivi. L'utilizzo degli scenari permette quindi di capire quali sono le informazioni necessarie affinché i personaggi soddisfino i loro scopi, permettendo così di sviluppare i prototipi e le interfacce dell'applicazione.

È importante notare che l'utilizzo di questi modelli di design non garantisce che il software risulti effettivamente usabile. Il modello espresso dal sistema può infatti essere distante dal modello atteso dall'utente per vari motivi quali, ad esempio, l'ambiguità linguistica (lo stesso termine può assumere differenti significati per differenti utenti), aspetti procedurali (il numero di operazioni che l'utente deve svolgere è differente da quello atteso), ecc.

Per tentare di risolvere questi problemi è stato sviluppato in questo lavoro un nuovo modello di design chiamato CAO=S. Per ridurre la scarsa usabilità CAO=S, agisce su caratteristiche come l'utilità attesa, la completezza dei contenuti, la comprensibilità del vocabolario (eliminando i termini di difficile interpretazione). Un cambiamento fondamentale di questo modello di design è l'idea di trasformare la fase di analisi degli utenti, task e obiettivi da requisito della progettazione a parametro di progetto.

CAO=S (Concetti+Attori+Operazioni=Strutture dati) semplifica l'analisi delle categorie d'utenza basandosi soltanto sulle caratteristiche fondamentali e utilizza l'analisi dei requisiti come parametro di input del modello. Le prime tre componenti fondamentali (CAO) vengono acquisite attraverso la compilazione di questionari da parte del team di sviluppo. I concetti raffigurano il modo in cui l'utente percepisce e comprende l'informazione. Analizzandoli è possibile definire l'architettura delle informazioni del sistema da realizzare. Gli attori sono le categorie di utenza che interagiscono col sistema tramite le interfacce manipolando le strutture dati percepite attraverso i concetti. A differenza dei personaggi utilizzati nei modelli Goal-oriented, gli attori sono caratterizzati in base al ruolo che assumono all'interno del sistema. Per questo motivo CAO=S deve gestire interfacce potenzialmente diverse e personalizzabili in base al ruolo dell'utente identificando correttamente i task e le operazioni da effettuare. Le operazioni sono le azioni che l'utente effettua sull'interfaccia manipolando i concetti. Il sistema è trasparente in quanto ogni operazione avviene sui concetti e non sulle strutture dati interne.

Tramite l'analisi delle tre componenti fondamentali (CAO) si derivano le strutture (S) che sono l'organizzazione concreta delle idee del progettista. Le strutture sono di tre tipi: le Viste, la Navigazione, il Data storage. Le Viste sono la rappresentazione dei concetti attraverso un'interfaccia; in questa struttura è descritta la modalità di visualizzazione e vengono definiti elementi come form, liste, tabelle, frammenti di altre viste, etc. La Navigazione rappresenta il modo con cui l'utente si sposta fra le viste ed è specificata all'interno delle strutture View. Il Data storage permette il salvataggio dei dati all'interno del sistema in maniera persistente.

CAO=S, oltre a definire un miglior modello formale per la progettazione User Centered, definisce anche un'architettura capace di generare automaticamente applicazioni web usabili, generando interfacce utente che adottano linee guida e pattern di progettazione orientati all'usabilità. In questo modo si abbassano drasticamente i tempi di sviluppo di applicazioni e non sono necessarie particolari conoscenze tecniche (linguaggi di programmazione, usabilità, database, etc).

L'architettura software è concettualmente suddivisibile in due applicazioni distinte. La prima è CAO=S Analyzer, che analizza le tre componenti descritte nel modello formale (CAO) date come input dal team di sviluppo attraverso la compilazione di questionari. Dall'analisi delle componenti vengono generate le tabelle tridimensionali che rappresentano l'insieme di tutte le operazioni svolte dagli attori sui concetti, che vengono mappate

in un file xml di output. La seconda applicazione è CAO=S Generator, che si occupa di sviluppare concretamente l'applicazione attraverso l'analisi del documento xml creato dall'applicazione dall'Analyzer. Viene generato il codice dell'applicazione, l'interfaccia utente e il modulo di persistenza dei dati automatizzando lo sviluppo dell'applicazione.

A dimostrazione della bontà del modello, in questa tesi è stata sviluppata l'applicazione ERIS[1] (CAO=S Generator) lasciando ad un futuro momento l'implementazione di CAO=S Analyzer. Durante la fase di analisi dello sviluppo di ERIS sono state identificate alcune caratteristiche base utili per la realizzazione di applicazioni di qualità, come ad esempio l'uso dei paradigmi REST (Representational State Transfer), MVC (Model View Controller) e ORM (Object-Relational Mapping). Si è scelto poi di utilizzare un framework come struttura di supporto alla creazione di applicazioni. Il framework scelto è Symfony ed è sponsorizzato da Sensio Labs, una nota Web Agency francese.

ERIS sviluppa le applicazioni analizzando il documento xml prodotto da CAO=S Analyzer seguendo quattro fasi: l'inizializzazione, la creazione del modulo di persistenza, la creazione della logica dell'applicazione e la realizzazione dell'interfaccia attraverso il paradigma MVC. Durante la prima fase viene fatto un bootstrap del sistema preparandolo per le fasi successive.

L'inizializzazione è configurata tramite un documento xml che permette di scaricare e installare correttamente il framework e di inizializzare il database e tutti i servizi utili per predisporre correttamente l'ambiente di lavoro.

Nella seconda fase avviene la creazione del modulo di persistenza (model) e vengono analizzati i Concetti descritti nel documento xml per poter generare un documento YAML (YAML Ain't Markup Language) che descrive i dati in maniera generica per tutti i linguaggi di programmazione. Successivamente, attraverso l'utilizzo di un layer interno a Symfony chiamato Doctrine, viene realizzato il modulo di persistenza dei dati dell'applicazione.

Nella terza fase, quella di creazione della logica dell'applicazione, vengono analizzate le Viste in modo da poter creare i moduli (controller) che si occupano di reperire le informazioni dal modulo di persistenza.

[1] "dal greco antico Ἔρις, «conflitto, lite, contesa») è una figura della mitologia greca, la dea del Caos." fonte: http://it.wikipedia.org/wiki/Eris_(mitologia)

Queste informazioni vengono richieste dall'utente tramite la manipolazione dei concetti. Durante questa fase vengono presi in considerazione tutti gli elementi che compongono le Viste: lista, form, contenuto, navigazione, sottoview che descrivono frammenti della Vista.

L'ultima fase permette a ERIS di realizzare l'interfaccia assemblando i componenti interni alle Viste creando così la parte di presentazione di ogni elemento (view).

Per verificare il corretto funzionamento di ERIS è stato deciso di realizzare un'applicazione di test chiamata ALMA che rappresenta alcune funzioni del modulo di registrazione degli esami di un'università. Non avendo a disposizione un'implementazione dell'applicazione CAO=S Analyzer si è dovuto realizzare un documento xml ad hoc che rappresenta le strutture dati realizzate attraverso l'analisi dei concetti, degli attori e delle operazioni.

Capitolo 2

Il problema della scarsa usabilità delle applicazioni web

2.1 Introduzione

Con la diffusione delle tecnologie informatiche negli anni 80, dei personal computer negli anni 90, l'avvento di internet, la proliferazione di siti e applicazioni web, i progettisti che dapprima sviluppavano prodotti software i cui principali fruitori erano utenti esperti, hanno dovuto spostare l'attenzione su un target di utilizzo diverso: questo cambiamento si è verificato nel momento in cui tale utente non era più esperto di informatica.

Da qui nasce il concetto di usabilità; la normativa ISO 9241 del 1993 definisce infatti l'usabilità come: "l'efficacia, l'efficienza e la soddisfazione con le quali determinati utenti raggiungono determinati obiettivi in determinati contesti."[1], specificando il grado di facilità e soddisfazione con cui l'interazione uomo-strumento si compie.

Evidenziando il fatto che l'usabilità ha senso solo in un contesto dove vi sia una relazione d'uso utente/prodotto, con lo studio dell'usabilità si cerca di far corrispondere il più possibile il modello mentale di chi ha progettato il software (design model) con quello costruito mentalmente dall'utente nel momento in cui lo utilizza (user model), in modo da porre al centro dell'attenzione progettuale il fruitore ultimo.

[1] ISO 9241 - 1993, Ergonomic requirements for office work with visual display terminals.

Esperti di usabilità come Jakob Nielsen [NIL93] , Donald Norman [NOR04, NOR90] e Steve Krug [KRU01], hanno dimostrato come un sistema usabile:

- aumenti: l'efficienza, la produttività e le vendite
- riduca: gli errori, i costi di sviluppo e assistenza, il bisogno di training nell'utilizzo

Gli esperti di usabilità interagiscono durante la fase di progettazione e sviluppo del sistema, dalla definizione degli obiettivi e dei requisiti alla costruzione dell'architettura del sistema, al corretto utilizzo dei contenuti fino allo sviluppo dell'interfaccia.

Per sviluppare un sistema usabile bisogna tener conto, infatti, di fattori come:

- i requisiti del prodotto secondo gli utenti
- le caratteristiche dell'utente
- il contesto di utilizzo del prodotto

Per semplificare l'interazione fra l'utente e il sistema si devono mostrare nel modo più chiaro possibile le relazioni che intercorrono fra le manipolazioni dell'utente e i risultati che queste possono provocare.

A supporto dell'usabilità sono nati dei modelli di software design interattivi che fanno uso di prototipi e sono basati sull'attivo coinvolgimento degli utenti stessi allo scopo di ottenerne una chiara identificazione dei requisiti base, dei compiti e dei contesti d'uso in modo da relazionarli correttamente al sistema e alle funzioni che andranno ad utilizzare.

Questi modelli di design, chiamati Human-Centered o User-Centered, fanno perno sull'importanza degli obiettivi degli utenti che utilizzano il sistema, basandosi sulle loro caratteristiche intellettuali, culturali, motivazionali e fisiche, viste come punto di partenza per ottenere sistemi usabili.

La norma ISO che definisce la guida ai processi di progettazione delle applicazioni software interattive è la 13407 "Human Centered Design Processes for interactive systems, 1999" sviluppata dal TC 159 SC4 WG6, il comitato ISO che si occupa di "Ergonomia"; questa definisce lo Human-Centered Design come: 'Un approccio allo sviluppo di sistemi interattivi focalizzato specificamente sul rendere il sistema usabile'; un'attività multidisciplinare, che richiede competenze e tecniche specifiche di ergonomia.

L'applicazione di tali metodi e tecniche al disegno di sistemi interattivi ne aumenta l'efficacia e l'efficienza, migliora le condizioni di lavoro, contrasta possibili effetti nocivi sulla salute, sulla sicurezza e sulle prestazioni.

Applicare l'ergonomia al disegno di sistemi richiede di considerare come fattori primari le capacità, le competenze, le conoscenze, le limitazioni e le esigenze degli utenti.

2.2 I modelli di design User Centered

La progettazione di un sistema orientato sugli utenti, che si basa sull'analisi degli obiettivi delle persone che lo usano è detto "User Centered Design" [AND99].

Le caratteristiche culturali, fisiche, motivazioni, intellettuali, di dominio, ed altre, vengono utilizzate come parametri fondamentali per la progettazione: modificarle o manipolarle a proprio piacimento per ottenere sistemi meno costosi o più efficienti, non rispecchia l' ideologia del modello di design.

Possiamo definire due principali modelli di User Centered Design:

- Task Oriented Design [LR94]: organizza l'intera progettazione sull'analisi dei task che l'utente dovrà svolgere nel sistema. In questo modello di design la fase di analisi dei requisisti deve occuparsi di task reali, completi, che rappresentino realmente i compiti dell'utente. Gli sviluppatori, realizzano il sistema concentrandosi in primo luogo sulla comprensione delle attività dell'utente che vengono poi trasformate in features.

- Goal Oriented Design [CRC07]: orientato alla soddisfazione degli obiettivi dell'utente, poco interessato a conoscere i meccanismi del sistema ma unicamente direzionato al raggiungimento del proprio goal. A differenza del modello task oriented che ha il focus sulle attività che l'utente deve svolgere, questo modello focalizza la progettazione sulla modellazione degli obiettivi dell'utente.

Le fasi dei modelli User Centered Design:

1. Si compie la fase di analisi, effettuando alcune tra le seguenti attività:

   - Incontri con gli stakeholder (portatori di interessi) per capire vincoli e aspettative
   - Analisi dei prodotti esistenti
   - Osservazioni sul campo
   - Interviste o workshop con potenziali utenti
   - Questionari
   - Creazione di profili di utente, elenchi di task e scenari

   Se esiste già un sistema da analizzare viene svolta precedentemente l'expert sociability review e l'expert usability review:

   - *L'expert sociability review* consiste nello studio degli aspetti e delle attitudini sociali connessi con l'interazione online; durante questa fase gli esperti si occupano di rilevare i fattori che promuovono la creazione, il mantenimento e l'evoluzione delle comunità,valutano gli strumenti messi a disposizione agli utenti dal sistema e le valutano questioni relative alla privacy, rilevano i fattori che promuovono le relazioni interpersonali che caratterizzano l'identità dell'utente, della comunicazione sociale e dell'emergenza dei gruppi di utenti. Lo scopo di questa analisi è favorire i comportamenti che portano all'aggregazione e alla condivisione della conoscenza.
   - *L'expert usability review* consiste nell'analisi delle interfacce; attraverso l'individuazione di task-flow è possibile determinare ciò che causa difficoltà ad utenti nel raggiungimento dei loro compiti. Questa analisi è eseguita anche con l'aiuto degli utenti stessi, attraverso test e report, realizzando un elenco prioritario di modifiche da correggere per migliorare l'usabilità del sistema. Oltre ai problemi relativi ai task-utente in questa analisi vengono utilizzati anche quelli relativi all'interfaccia, facendo uso di linee guida, standard e pattern.

2. Si compie la fase di progettazione:

    - Brainstorming, riunioni e discussioni libere
    - Creazione di modelli e schemi di navigazione, bozzetti e schermate(anche in matita)
    - Conduzione di analisi e test sui bozzetti
    - Creazione di prototipi a bassa o alta fedeltà

    In questa fase viene progettata l'interfaccia che, solitamente è concepita prendendo spunto da progetti di successo già esistenti. Di fatto è meglio realizzare un'interfaccia che utilizza pattern di design e linee guida condivise da designer, che realizzarne una innovativa, che potrebbe deludere le non aspettative dell'utente.

3. La fase di valutazione si compie prima e durante l'implementazione vera e propria attraverso:

    - Test con utenti
    - Questionari
    - Analisi euristiche e ispettive

    E' in questa fase in cui ogni funzionalità è confrontata con i task realizzati: per ognuna di queste identificata come requisito del sistema, deve esistere un task del sistema che ne provveda l'utilizzo. Attraverso l'uso di questionari e test con gli utenti si avrà modo di valutare situazioni e problemi non considerati in fase progettuale; in caso di task mal realizzati o funzionalità mancanti all'interno del sistema, bisognerà invece iterare il processo.

4. Fase di implementazione, è la fase di evoluzione del sistema esistente oppure di realizzazione di uno nuovo.

5. Monitoraggio, segnalazione di problemi, questionari e studi sul campo, aiutano gli sviluppatori a raccogliere informazioni da utenti non sottoposti a controllo per identificare problemi o migliorie che vanno eventualmente risolti o sviluppati.

2.3 Il modello di design Goal Oriented

La progettazione tradizionale nei sistemi User Centered è basata sull'individuazione dei task che l'utente deve compiere per potare a termine delle attività.

Nei modelli goal oriented si mira, invece, a soddisfare completamente gli obiettivi dell'utente, rendendolo più soddisfatto e produttivo; quest'ultimo attraverso l'utilizzo dei task raggiunge goal personali, come ad esempio: non fare troppi errori, essere veloci, non sembrare stupido, non annoiarsi, ecc. È da considerarsi un falso goal utente impaginare velocemente un libro, verificare la correttezza di un'indagine di marketing, stampare 10 copie di un documento.

È possibile suddivide i goal dell'utente in tre tipi:

- Experience Goal: lo stato d'animo da raggiungere per un utente (per esempio "sentirsi lodato, intelligenti, alla moda, ecc.").
- End Goal: la motivazione dell'utente ad usare lo strumento prescelto per eseguire il task.
- Life Goal: le motivazioni che descrivono le aspirazioni e le ambizioni dell'utente.

Il modello di design Goal Oriented [CRC07] di Alan Cooper migliora i modelli User Centerd descrivendo più adeguatamente il concetto di personaggio e scenario. Il modello è strutturato nelle seguenti fasi:

1. Ricerca: attraverso tecniche di intervista, survey, mira a conoscere il comportamento degli utenti. Fattori importanti di questa fase sono: il dominio sociale e terminologico, il contesto tecnologico, di business e l'esistenza di prodotti simili già in uso.
2. Modellazione: in questa fase si realizzano i personaggi modellando astrattamente gli utenti; osservandoli nella fase di ricerca vengono creati dei pattern comportamentali e motivazionali con cui vengono creati i personaggi ciascuno con un obiettivo proprio, differente dagli altri. I personaggi sono utili per determinare come si deve comportare il prodotto in base agli obiettivi dell'utente e sono utilizzati per migliorare la comunicazione all'interno del team di sviluppo rappresentando un'immagine comune che semplifica la discussione sulle scelte progettuali.

Alan Cooper asserisce che per ogni progetto vi debba essere un insieme che va da 3 a 12 personaggi, ognuno dei quali con tre principali goal:

- Experience Goal: il personaggio ha sempre di solito uno o due experience goal.
- End Goal: solitamente il personaggio ha tre o quattro end goal.
- Life Goal: ogni personaggio ha zero o un solo life goal.

Bisogna essere precisi nella creazione dei personaggi, caratterizzandoli con informazioni che li descrivono dettagliatamente come persone reali: nome, cognome, lavoro, dati anagrafici, foto, ecc.

Per ampliare la visione del progetto, poi, è utile anche creare personaggi che non rappresentino l'utente del prodotto bensì possibili clienti. Ad ogni futura interfaccia del sistema dovrà corrispondere un preciso personaggio chiamato principale perché soddisfatto al 100% da un'interfaccia progettata unicamente sulle sue necessità; altri personaggi, avranno un ruolo secondario utile per aggiungere dettagli alla stessa interfaccia, ma superflui al personaggio principale.

3. Scenari: utilizzando i personaggi, si descrive in forma di storia come questi interagiscono col sistema per portare a termine i loro obiettivi.Esistono tre tipologie di scenari:

   - Scenario di contesto: descrive astrattamente come un oggetto soddisfa l'obiettivo del personaggio.
   - Scenario Key-Path: iterativamente vengono aggiunte features a questo scenario che descrivono dettagliatamente le interazioni più significative fra il personaggio e il sistema stesso.
   - Scenari di validazione: vengono creati e utilizzati per verificare che tutte le soluzione proposte per il personaggio e i suoi obiettivi sono valide.

4. Definizione dei requisiti: creando vari scenari di contesto, si riesce a capire di quali informazioni i personaggi hanno bisogno per raggiungere i loro obiettivi; questi sono, di solito, espressi in linguaggio naturale in modo da risultare comprensibili agli utenti del sistema sprovvisti di conoscenze tecniche.

Queste indicazioni seguono alcune linee guida:

- Evitare l'utilizzo di termini tecnici, se non in caso di un requisito di sistema.
- Evidenziare le parti fondamentali di un requisito.
- Usare un formato standard per tutti i requisiti.

5. Design del sistema: in questa fase viene definita la struttura dell'interfaccia del nostro sistema. Iterando tipicamente più volte ed analizzando i requisiti e gli scenari precedentemente identificati, si arriva a definire la struttura rappresentata, con diverse tecniche, in disegni di possibili organizzazioni dell'interfaccia, facendo riferimento agli scenari di validazione e a quelli di Key-Path.

6. Raffinamento e sviluppo: una volta rappresentata stabilmente l'interfaccia del sistema, questa viene trasformata in un prototipo che iterativamente viene migliorato con aggiunta di dettagli, fino alla vera e propria realizzazione da parte di sviluppatori che devono attenersi il più possibile al prototipo finale di questa fase

2.4 Le applicazioni per la generazione automatica delle interfacce

Lo scopo della realizzazione di sistemi che generano automaticamente interfacce utente è quello di velocizzare e migliorare qualitativamente il lavoro di sviluppatori ed esperti di usabilità, semplificando la creazione e la manutenzione delle interfacce, evitando la riscrittura del codice riutilizzabile e sviluppando rapidamente interfacce per diversi tipi di piattaforme hardware e software.

Gli attuali sistemi di generazione automatica delle interfacce si basano sull'utilizzo del paradigma Model Driven Software Development (MDSD); questo è definito come un'alternativa alla programmazione tradizionale che non prevede codice sorgente da compilare o da eseguire, ma la modellazione del sistema che si desidera implementare. Per far ciò il designer lo modella attraverso dei diagrammi visuali e, successivamente, a partire da questi genera automaticamente il codice in un linguaggio di programmazione tradizionale.

In questi sistemi, i requisiti utente vengono modellati come funzionalità del sistema attraverso dei modelli visivi o matematici [SWM06, MEY06] e, successivamente, i modelli vengono trasformati in interfacce utente.

Possiamo classificare i sistemi che generano interfacce automaticamente in base al tipo di approccio che utilizzano per modellare il sistema:

- L'approccio basato sui linguaggi di modellazione
- L'approccio basato sui modelli discorsivi

I sistemi che adottano l'approccio basato sui linguaggi di modellazione [CF10, TVK09], per esempio ProjectIT+ProjectIT-Studio [SVS06, SIL06], tentano di descrivere formalmente i task, i dati e gli utenti attraverso l'utilizzo di notazioni semi-grafiche e semi-formali per poi generarvi automaticamente le interfacce.

La modellazione avviene attraverso l'utilizzo di linguaggi come UML [BRJ97](Unified Modeling Language), che consentono di rappresentare i sistemi attraverso diverse tipologie di vista.

UML consente di descrivere un sistema sotto tre aspetti:

- Il modello funzionale: rappresenta il punto di vista dell'utente, ovvero ne descrive il comportamento così com'è percepito prescindendo dal funzionamento interno.
- Il modello ad oggetti: rappresenta la struttura e la sotto struttura del sistema utilizzando i concetti object-oriented di classe e di oggetto e le relazioni fra essi.
- Il modello dinamico: rappresenta il comportamento degli oggetti del sistema e le loro interazioni nel tempo.

È costituito dalle seguenti viste:

- Vista dei casi d'uso: utilizzata per analizzare i requisiti degli utenti e capire cosa il sistema debba fare.
- Vista di progettazione: descrive come le funzionalità del sistema devono essere realizzate.
- Vista di implementazione: descrive i packages e le classi.

- Vista dei processi: individua i processi e le entità che li eseguono.
- Vista di sviluppo: mostra l'architettura fisica del sistema.

Una volta modellato graficamente il sistema che si vuole andare a realizzare, il sistema realizzerà attraverso fasi successive la prototipazione e l'implementazione dell'interfaccia.

Un sistema che fa uso di UML e dell'approccio basato sui linguaggi di modellazione OO-Method è chiamato OlivaNova [MOL04] che ha lo scopo di trasformare le specifiche di un sistema in un modello finale. Per far ciò lo sviluppatore modella il sistema concettuale che viene poi tradotto attraverso dei pattern in specifiche OASIS [GCP00, PPI98] che rappresentano il modello finale del sistema da generare. Il modello concettuale che che lo sviluppatore modella graficamente è composto dai seguenti sotto-modelli:

- Modello oggetto: utilizzato per rappresentare il sistema, viene rappresentato attraverso un diagramma delle classi UML.
- Modello dinamico: utilizzato per specificare il ciclo di vita degli oggetti e la loro interazione; l'interazione viene rappresentata attraverso diagrammi non UML.
- Modello funzionale: utilizzato per specificare l'azione che deve essere eseguita quando vi è un cambio di stato di un oggetto.
- Modello di presentazione: utilizzato per specificare come gli utenti interagiscono col sistema.

OliveNova, utilizzando questi modelli, generare l'applicazione.

Parallelamente allo sviluppo di sistemi che basano la generazione automatica delle interfacce sui linguaggi di modellazione, si sono evoluti sistemi che fanno uso di modelli discorsivi.

Questi sistemi, "specificano le interfacce utente per mezzo degli atti comunicativi" [FKP09, FKH06, BFK08, BKF08] attraverso l'analisi della comunicazione umana.

I primi studi sugli atti comunicativi furono fatti da Searle che sosteneva che " il parlare una lingua avviene attraverso l'utilizzo di atti di comunicazione; gli atti ci permettono di fare una dichiarazione, dare dei comandi, fare delle domande, fare delle promesse e così via" e che gli atti linguistici sono le unità base della comunicazione linguistica.

Analogamente alla comunicazione che avviene fra esseri umani, l'interazione uomo-macchina è vista in questi sistemi come un articolato insieme di atti comunicativi.

Le fasi che vengono utilizzate per generare le interfacce in sistemi che utilizzano questa tipologia di approccio sono principalmente tre:

- Fase di analisi e modellazione dell'utente e dei task: viene utilizzata la teoria degli atti comunicativi (Speech Act Theory [SEA69]), che attraverso l'analisi delle conversazioni (Conversation Analysis [LGF90]) identifica gli elementi della comunicazione e li lega attraverso delle relazioni RST (Rhetorical Structure Theory [MT88]). Il modello discorsivo viene quindi rappresentato dagli elementi della comunicazione attraverso il loro raggruppamento, le loro relazioni, il loro ordinamento e la gerarchia.

- Fase prototipale: viene utilizzata per trasformare il modello discorsivo in interfaccia. In alcuni sistemi come TERESA [PSS08, PSM08, PS03, PS02, MPS04, MPS03], questa fase è suddivisa in sotto-fasi utili al raffinamento e all'adattamento del prototipo per la realizzazione di sistemi multimodali.

- Fase di implementazione: implementa concretamente l'interfaccia utente attraverso l'uso di euristiche.

Di seguito riportiamo un esempio di modellazione discorsiva che fa uso degli atti comunicativi per generare le interfacce:

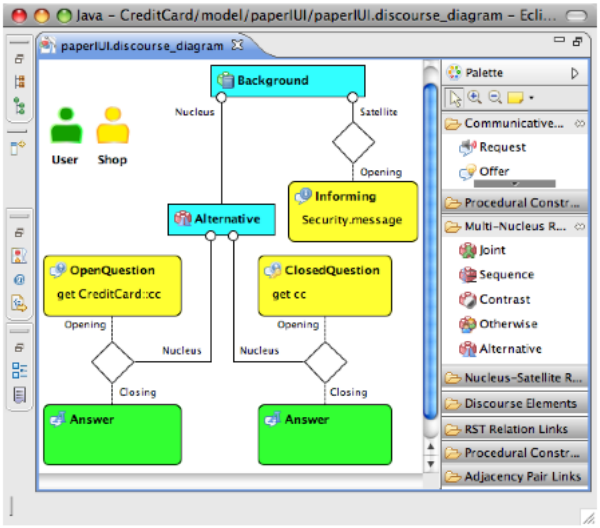

Figura 2.1: Modellazione di un'interfaccia attraverso l'approccio discorsivo

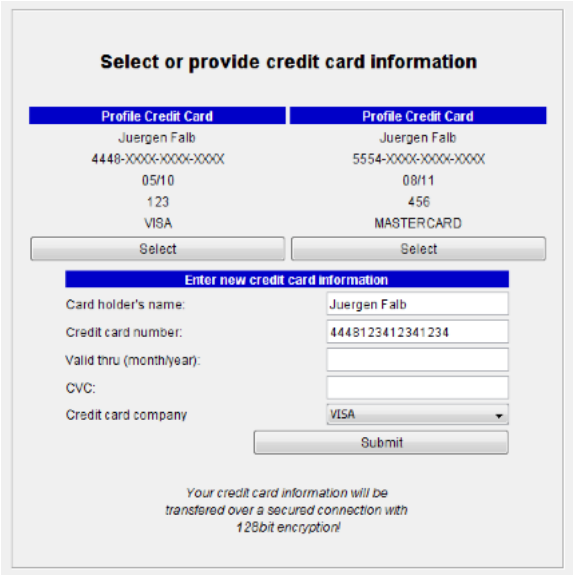

Figura 2.2: Interfaccia realizzata dalla modellazione dell'apporccio discorsivo

Nel contesto della generazione automatica di interfacce, la maggior parte dei sistemi fa uso di meta-linguaggi fra cui UIML [AP99], XUL [BSD01], USIXML [SV03], TERESAXML [MPS04], utilizzati per definire gli elementi dell'interfaccia.

Per favorire la standardizzazione delle interfacce, l'OASIS, ha sviluppato le specifiche di UIML (User Interface Markup Language [AP99]), un meta-linguaggio xml per la definizione delle interfacce utente con cui si possono definire gli elementi dell'interfaccia stessa: pulsanti, menu, liste, tabelle per descriverne poi le relazioni che vi intercorrono.

Lo sviluppo di questo standard è dovuto alla necessità di descrivere interfacce utente indipendentemente dalle piattaforme in cui vengono sviluppate. Possiamo definirlo il pioniere dei linguaggi di markup per descrivere le interfacce e successivamente alla sua standardizzazione, sono stati realizzati altri linguaggi di markup utilizzati in ambienti differenti ma con lo stesso scopo:

- MXML [COE04]: è un linguaggio di markup introdotto da Macromedia nel 2004 e fa parte dell'Open Source Adobe Flex SDK. I file MXML vengono compilati in swf da flash attraverso l'SDK di Flex e renderizzati dai browser, dal plugin Flash di Adobe ecc.

- XUL [BSD01]: è il linguaggio dell'interfaccia con cui è sviluppato Mozilla Foundation. I documenti XUL vengono renderizzati attraverso il motore Gecko che li trasforma in XHTML e SVG e si interfaccia con molte delle esistenti tecnologie, tra cui Css, Javascript, Dtd e Rdf.

- XAL [XAL08]: eXtensible Application Language è il linguaggio di markup della suite di Nexaweb chiamata Enterprise Web 2.0 Suite. Gli sviluppatori possono utilizzare questo linguaggio per definire applicazioni che verrano eseguite come client Java o Ajax.

- XAML [XAM08]: è un sistema di markup che sta alla base delle interfacce utente di Microsoft .Net Framework 3.0; oltre a descrivere l'interfaccia utente come fanno gli altri linguaggi di markup, include anche la logica del programma e gli stili di visualizzazione. Può essere visto come una combinazione di XUL, SVG, CSS, Javascript, in un unico documento XML.

- OpenLaszlo [OPE10]: è un ambiente di runtime che dispone di un linguaggio dichiarativo di definizione dell'interfaccia che dà la possibilità agli sviluppatori di definire i widget, i layout e gli script utilizzati per sviluppare l'interfaccia.

Ne riportiamo di seguito una comparazione:

	Creatore	Ambiente di Runtime
MXML	Adobe	Flash 9 o superiore
XUL	Mozilla	Applicazioni basate su Gecko
XAL	Nexaweb	Java JRE 1.1 / MSJVM, DHTML
XAML	Microsoft	.NET Framework 4.0
Open Laszlo	Lazio	Flash Player 5, DHTML, Java ME annunciato

Tabella 2.1: Alcuni linguaggi di markup per descrivere le interfacce utente

L'avanzamento tecnologico, il numero sempre crescente di dispositivi che vengono resi disponibili agli utenti (telefoni, palmari, smartphone, televisori, computer, tablet, ecc.), l'aumento della complessità delle interfacce e l'incremento di applicazioni che fanno uso di simili features hanno portato alla creazione di sistemi automatici o semi-automatici [ERT09] che generano interfacce per applicazioni multimodali.

Uno fra i migliori ambienti multimodali per la creazione automatica di interfacce attualmente è TERESA [PSS08, PSM08, PS03, PS02, MPS04, MPS03].

TERESA come altri modelli di generazione di interfacce web è basata su modelli discorsivi la cui base di riferimento è l'analisi della comunicazione umana e dei modelli di dialogo che permettono la modellazione del sistema. Attraverso l'utilizzo di queste informazioni viene poi sviluppata automaticamente l'interfaccia.

TERESA offre la possibilità di supportare la generazione di interfacce per una vasta gamme di linguaggi : XHTML MP, VoiceXML, X + V, SVG , Xlet, ecc., ed è un ambiente di sviluppo molto flessibile utilizzato per sviluppare interfacce a vari livelli di automazione adattandosi così all'expertise dei designer.

Si basa principalmente su quattro macro fasi e altrettante strutture:

- Oggetti e Task: in questa fase avviene l'analisi dell'interazione degli utenti con il sistema attraverso l'approccio al modello discorsivo.
- L'interfaccia astratta: descrive astrattamente l'interfaccia. Descrive cosa fa, ma non come lo fa.
- L'interfaccia concreta: descrive concretamente l'interfaccia senza però svilupparla in un linguaggio.
- L'implementazione finale: attraverso un linguaggio dell'interfaccia.

Nella prima fase si individuano le attività dell'utente e le features che il sistema dovrà fornire attraverso la modellazione degli atti comunicativi, specificando gerarchie di azioni e relazioni temporali fra i differenti task individuati. Questa prima fase serve propriamente a identificare cosa il sistema deve fare.

Nella seconda fase avviene la modellazione del sistema attraverso il modello discorsivo che permette al sistema di concepire lo sviluppo dell'interfaccia astratta utilizzata dal sistema per descrivere cosa fa il sistema, introducendo per esempio elementi che rappresentano i futuri oggetti manipolati concretamente dagli utenti durante i task.

La terza fase raffina e migliora l'interfaccia astratta; in base al tipo di piattaforma dove verrà implementata l'interfaccia, gli elementi e i componenti dell'interfaccia astratta possono assumere diverse caratteristiche: differenti se sviluppate per l'interfaccia del pc o dello smartphone.

L'ultima fase conduce alla vera e propria realizzazione dell'interfaccia in base al dispositivo scelto.

Di seguito riportiamo un'immagine del tool di Teresa:

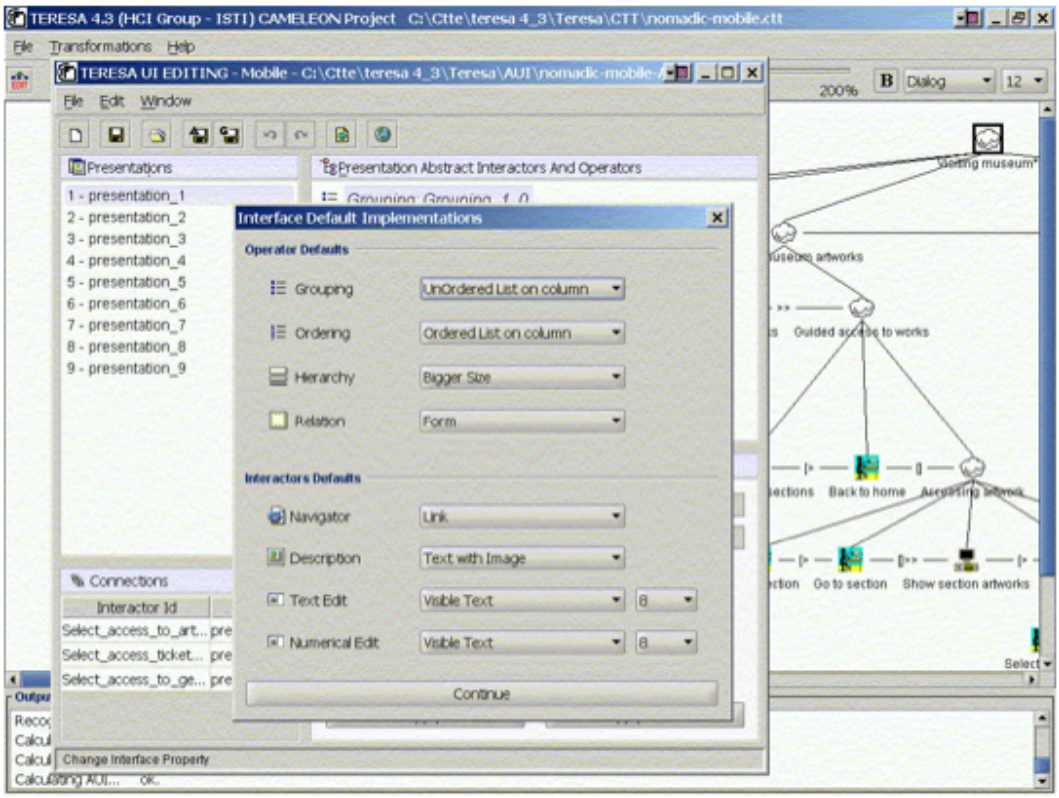

Figura 2.3: Teresa 4.3

I sistemi di generazione automatica di interfacce visti fino ad ora non hanno come parametro fondamentale di progettazione l'utente, bensì i task. TERESA, per esempio, acquisisce le informazioni dell'interazione fra uomo e sistema facendo uso di modelli discorsivi; altri sistemi, come HUMANOID [SLN92, SLN93], tentano di ricavare interfacce da codice, altri ancora dal database (GENIUS [JWZ93]).

Solo attualmente alcuni sistemi di generazione automatica di interfacce iniziano ad esser sviluppati e a far uso dei principi dell'usabilità.

All'università di Harvard è stato sviluppato un sistema chiamato SUPPLE [GWW10] che mira a generare automaticamente interfacce per utenti diversamente abili. Attraverso questo sistema, le interfacce generate si adattano ai contesti dei singoli utenti, perché modellate in base agli individui, ognuno con le proprie capacità, preferenze e esigenze.

SUPPLE fa uso dei modelli discorsivi per la generazione del modello del sistema ed è composto da 3 sottosistemi:

- SUPPLE: usa teorie decisionali per generare interfacce adattandole alle persone e ai device
- ARNAULD: utilizzato per ottimizzare le preferenze dell'utente nell'interfaccia
- ABILITY MODELER: utilizzato per effettuare una valutazione delle capacità motorie dell'utente ed attraverso esse svilupparne un modello

Gli esperimenti fatti attraverso il prototipo di SUPPLE mostrano che questo sistema migliora significativamente la velocità, l'accuratezza e la soddisfazione degli utenti con differenti abilità.

Di seguito riportiamo la stessa interfaccia sviluppata automaticamente per due differenti utenti attraverso l'utilizzo di SUPPLE:

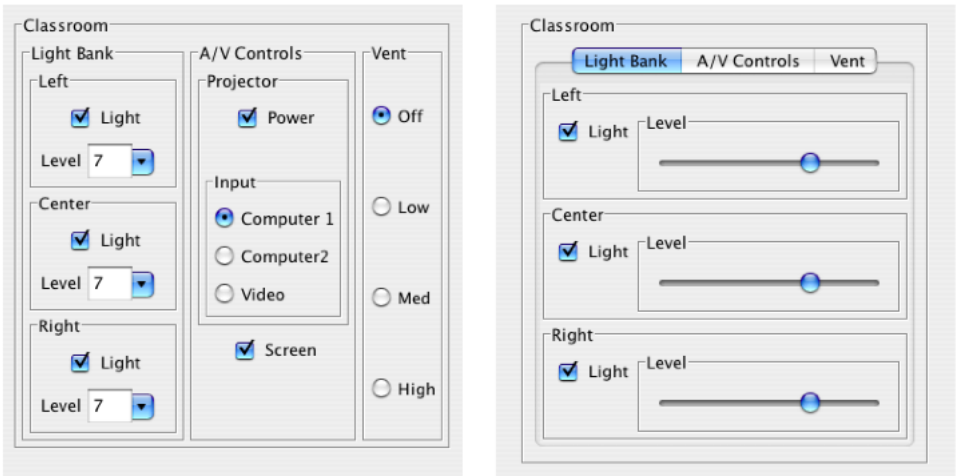

Figura 2.4: Differenti modellazioni della stessa interfaccia generate da SUPPLE

Capitolo 3

CAO=S un nuovo modello di sviluppo goal oriented

Se progettiamo e realizziamo prodotti attraverso cui gli utenti possono soddisfare i propri scopi, quelle persone saranno soddisfatte di aver acquisito i nostri prodotti, li raccomanderanno agli amici e questo si traduce in un successo di business.

Alan Cooper - About face 3

3.1 Idea base di CAOS

Lo scopo di CAO=S è permettere ad un team di sviluppo di buona qualità ma senza esperienza specifica nel campo dell'usabilità di evitare almeno gli errori più comuni nella progettazione di applicazioni usabili. Questo modello è ideato per quei progetti limitati economicamente dove non vi sia la possibilità di coinvolgere direttamente un esperto di usabilità o effettuare un'analisi delle categorie di utenze di riferimento.

Si basa su un'applicazione semplificata del modello goal-oriented in cui la parte analitica sulle categorie di utenti previste è semplificata nell'analisi delle poche caratteristiche fondamentali di queste categorie di utenti. Basare il modello di progettazione sulla semplificazione del modello goal-oriented è un cambio sostanziale che con l'ausilio della generazione automatica di interfacce mira a soddisfare completamente gli obiettivi dell'utente senza più organizzargli solamente i task da svolgere.

Con questo modello si mira ad eliminare le principali cause della scarsa usabilità, ridurre la distanza fra il modello espresso dal sistema e quello percepito dall'utente eliminando le ambiguità linguistiche. Le caratteristiche su cui si agisce sono:

- L'utilità attesa (pensare a chi mi legge)
- La completezza dei contenuti (pensare a dare informazioni)
- La comprensibilità delle informazioni(eliminando le difficoltà inutili lasciandovi solo quelle utili)

CAO=S semplifica e rende più usabile per l'utente tutte le caratteristiche procedurali del sistema, dall'architettura della pagina alle operazioni atomiche, fino alla chiarezza dei comandi nella navigazione.

L'idea base di CAO=S è riassunta nei seguenti punti:

- adottare tutte le linee guida e i pattern di progettazione che le comunità degli utenti reputano utili, indipendentemente da utenti e goal
- trasformare la fase di analisi degli utenti, task e obiettivi, da requisito della progettazione a parametro di progetto. Tanto maggiore saranno accurati i parametri sugli utenti e i loro obiettivi , tanto maggiore sarà il grado di usabilità del sistema.
- Identificazione di tutte le caratteristiche degli utenti che hanno effettivamente impatto noto sul dominio di progettazione
- Generazione automatica di pattern di interfaccia dopo la corretta compilazione di questionari che riguardano i tre aspetti cruciali del modello: concetti, attori e operazioni.

3.2 Il modello CAO=S

CAO=S è l'acronimo di Concetti + Attori + Operazioni = Strutture dati, ovvero le tre componenti fondamentali del modello di progettazione e la risultante.

I concetti sono le informazioni che vengono trattate dall'applicazione e rispecchiano il modo in cui l'utente le comprende e le percepisce, rappresentate nella struttura dati. Da queste entità dipendono l'architettura dell'informazione e una parte della presentazione.

Gli attori in questo modello sono le categorie di utenti che agiscono sull'interfaccia dell'applicazione e che manipolano le strutture dati percepite come concetti per arrivare al raggiungimento dei loro goal e che si differenziano per il ruolo che hanno all'interno del sistema che, per questo, dovrà garantire interfacce potenzialmente diverse che caratterizzino i ruoli degli attori identificandone i task e le operazioni da effettuare.

Gli attori si dividono in diretti e indiretti: i primi sono gli utenti che utilizzeranno il sistema, mentre i secondi, al contrario, sono tutti coloro che per motivi diversi partecipano alla specificazione delle caratteristiche del sistema senza però utilizzarlo.

Le operazioni sono le azioni che l'utente effettua sull'interfaccia manipolando i concetti; possono essere di quattro tipi: creazione, lettura, modifica ed eliminazione secondo i modelli CRUD e REST. Ogni operazione agisce su una o più istanza di un concetto e non direttamente sui dati, così che l'utente non si accorge delle possibili attività che avvengono sui dati sottostanti e il sistema possa effettuare cambiamenti diretti, indiretti, in maniera definitiva o temporanea senza che l'utente se ne accorga.

L'analisi delle tre componenti principali (concetti, attori e operazioni) in questo modello di design rende possibile la descrizione delle strutture. Le strutture sono l'organizzazione concreta dell'idea del progettista sui concetti in cui gli attori svolgono azioni attraverso le operazioni. Possiamo definire tre tipologie di strutture interessanti:

- Le View: ovvero la rappresentazione dei concetti attraverso l'utilizzo di form, liste, tabelle, icone per la manipolazione, frammenti di altre view, ecc.
- La navigazione: logicamente parte della view ma concettualmente rappresentante una struttura separata, utilizzata per spostarci fra le varie view
- Il data storage: che permette il vero e proprio salvataggio dei dati in maniera persistente attraverso l'analisi dei concetti.

Per diminuire la distanza fra il modello espresso dal sistema e quello percepito dall'utente, CAO=S mira ad eliminare le ambiguità provenienti dall'analisi dei requisiti. Lo scopo principale di quest'ultima è definire che cosa il sistema debba fare (mentre le decisioni "sul come" sono rimandate alla fase progettuale) per cui è necessario un dialogo fra gli analisti e gli stakeholder (clienti, fornitori, finanziatori, collaboratori) che mira alla raccolta dei requisiti; successivamente alla loro analisi si determina se le informazioni raccolte sono ambigue, incomplete, poco chiare o contraddittorie. Ultimata questa fase vengono prodotte le specifiche dei requisiti che serviranno ai programmatori a progettare il sistema.

Per eliminare le ambiguità in questa fase si utilizzano solitamente dei linguaggi di modellazione astratti che descrivono le informazioni come:

- UML (Unified Modeling Language [BRJ97]): linguaggio di modellazione e specifica basato sul paradigma object-oriented che esprime le informazioni sotto forma di classi con legami quali associazioni, relazioni, aggregazioni o composizioni.

- Entity-Relationship: modello per la rappresentazione concettuale dei dati ad un alto livello di astrazione, utilizzato nella prima fase della progettazione di una base di dati in cui è necessario tradurre le informazioni risultanti dall'analisi di un determinato dominio in uno schema concettuale. Le informazioni sono descritte dalle entità, ogni entità ha degli attributi e può avere delle relazioni con altre entità.

I progettisti manipolano le informazioni disordinate dei concetti, riorganizzandole e normalizzandole descrivendo entità e classi prive di ambiguità; è proprio il passaggio in cui i progettisti traducono le informazioni grezze a classi o entità dell'analisi a causarne poi l'ambiguità delle stesse. Questa è una delle maggiori cause di complessità d'interazione nelle applicazioni moderne, ed ecco perché CAO=S intende modificare l'analisi dei requisiti nei seguenti modi:

- Registrare tutti i cambiamenti che avvengono tra le informazioni grezze ottenute nella fase di raccolta dei requisiti e le strutture concettuali che vengono realizzate durante la fase di analisi.

- Progettare delle strutture comprensibili all'utente che trasmettano il concetto e non la modellazione che avviene sui dati, tentando così di diminuire la distanza fra il modello utente e quello generato dal sistema.

- Progettare delle operazioni eseguibili dall'utente sui concetti, che a basso livello sottostante vengono mappate nelle funzioni ottenute dall'analisi dei requisiti funzionali.

3.2.1 C di CAO=S, i concetti

I concetti sono le informazioni che vengono trattate dall'applicazione, rispecchiano il modo in cui l'utente comprende e percepisce l'informazione che viene rappresentata dalla struttura dati. Vengono dedotti dall'analisi dei requisiti grezzi prima che le informazioni vengano manipolate, elaborate e normalizzate e da essi dipendono, l'architettura dell'informazione e una parte dell'interfaccia. Le operazioni mostrate nell'interfaccia attraverso CAO=S operano sui concetti(non sono funzioni su strutture dati).

Nella maggior parte dei casi i concetti e le strutture dati non coincidono e non possono essere mappati, portano anzi possono portare a diversi problemi:

- Durante la normalizzazione, dove attori, stakeholder e progettisti utilizzano diverse parole per esprimere lo stesso concetto.
- Differenze lessicali per cui tipi diversi di attori utilizzano termini differenti per esprimere o riferirsi alle stesse cose.
- Differenze concettuali, per cui gli attori utilizzano la stessa parola per riferirsi a concetti e/o domini completamente diversi.
- Polisemie, per cui un attore utilizza la stessa parola per esprimere più significati.
- Raramente possono esserci anche omonimie.

Mostriamo il seguente esempio utile a capire nel migliore dei modi il significato di ciò che è stato precedentemente detto; consideriamo un'applicazione per l'università dove gli attori sono gli amministrativi della sede centrale, delle sedi periferiche, i docenti e gli studenti.

In fase di analisi dei requisiti ci si accorge che:

- La parola "Corso" indica per gli amministrativi un insieme di insegnamenti organizzati in anni di *corso* appunto, mentre per i docenti e gli studenti indica ogni singolo insegnamento.

- La parola "Esame" indica per l'amministrativo e il docente un'insieme di prove che determinano la valutazione finale, mentre per lo studente rappresenta ogni singolo test da superare al termine di ciascun insegnamento, e indica anche, in generale, la materia di cui seguire le lezioni e svolgere le prove.

- La parola "Facoltà" indica per l'amministrativo e il docente un insieme di professori di discipline diverse ma riconducibili allo stesso ambito, mentre per gli studenti rappresenta, il corso di studi, la disciplina studiata, e il luogo fisico in cui si seguono le attività universitarie.

Vi sono quindi dei problemi di normalizzazione: è necessario adottare scelte lessicali per cui il linguaggio dei personaggi diretti abbia sempre priorità su quelle di team e attori indiretti (preferibile non usare, quindi, codici numerici a meno che non siano usate anche dagli utenti stessi).

Si riscontrano differenze lessicali: è necessario trovare espressioni comprensibili a tutti, senza particolare preferenza, esponendo il rapporto tra il termine accettato e tutti gli altri, optando per quelli proposti ed usati da attori diretti con poche competenze di dominio; occorrerà altrimenti realizzare interfacce diverse per ogni tipo di utenza.

Differenze concettuali: mai usare la parola in questione, né in un significato né nell'altro ma preferire specificazioni e precisazioni per disambiguare il concetto; la parola *corso* ad esempio, non è da usarsi, preferendo "Corso di studi" o "Insegnamento" in base alla dominio di riferimento.

Polisemie: nei casi di parole con più significati, è meglio evitarne l'uso.

In caso di fallimento, è necessaria reiterare la fase di raccolta dei requisiti con gli attori, andando a chiarire in modo specifico tutti gli aspetti linguistici delle informazioni grezze.

3.2.2 A di CAO=S, attori e analisi

Con il termine attori in CAO=S vengono individuati tutti quei soggetti interessati nel progetto al (clienti, fornitori, collaboratori ma anche utenti che andranno a utilizzare il sistema).

Si suddividono in due tipologie:

- Attori diretti: risiedono in questa categoria quegli utenti che utilizzeranno il sistema specificati in base al loro ruolo.
- Attori indiretti: coloro che non utilizzeranno il sistema, ma sono interpellati durante le fasi di progettazione e sviluppo.

Nei modelli Goal-Oriented lo sviluppo dei Personaggi e dei successivi scenari è dovuto all'analisi degli attori, perciò la loro categorizzazione e descrizione deve essere approfondita in modo da poter specificare verosimilmente i personaggi che poi li rappresenteranno; molte delle caratteristiche vengono dedotte da finti modelli e senza un'analisi etnografica che mira ad osservare le persone per esaminarne i dettagli di esperienza, domini, attività e relazioni, i personaggi diventano inutili.

Con CAO=S si vuole poter basare i personaggi su caratteristiche credibili; per far ciò gli attori diretti vengono suddivisi per ruolo e non in base alle caratteristiche personali; questa scelta è fondamentale: l'avanzamento tecnologico e la crescente diversità di utilizzo dei sistemi, rendono difficile e arbitraria l'individuazione di pattern di utenti perché troppo differenziati per tipologia, competenze, interessi ecc.

Le analisi sull'utente attraverso indagini etnografiche, oltre ad essere troppo costose, ci mostrano che c'è sempre una piccola percentuale di utenza per cui è utile sviluppare una determinata features nel sistema.

Possiamo, infatti, rappresentare i risultati di questi studi come una curva di una distribuzione che non si sbilancia mai verso gli estremi della scala di valutazione:

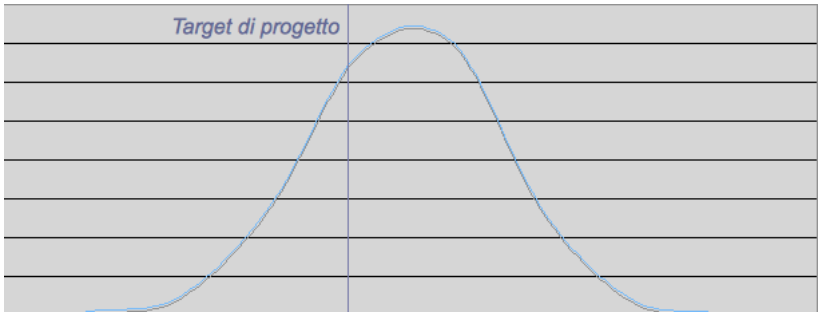

Figura 3.1: Curva del target di progetto

Diventa, quindi, di fondamentale importanza, identificare il comportamento del sistema non più basandosi sulle caratteristiche dell'utente bensì sul ruolo che avrà in quel sistema; in questo modello l'analisi degli utenti diventa il parametro fondamentale della strategia del progetto: trascurare questa fase, o anche solo sorvolare su certi aspetti, può portare alla realizzazione di un sistema diverso da quello che si desidera.

L'idea è di descrivere gli attori in base alle loro caratteristiche; di seguito ne riportiamo solamente alcune a titolo illustrativo:

- Competenza di dominio: capacità di comprendere il contesto in cui opera l'applicazione e di che cosa essa tratta.
- Competenze informatica specifica: capacità di comprendere il contesto informatico in cui l'applicazione opera, l' interazione tipica del web ecc.
- Competenze linguistiche: capacità di comprendere la lingua utilizzata dal sistema, dall'interfaccia, i toni, e le diverse tipologie di vocabolari linguistici.
- Abilità fisiche e visive: capacità di precisione ed abilità nell'interazione con l'interfaccia.
- Motivazione ed Interesse: capacità nell'utilizzo del sistema con relativo scopo (lavorativo o soddisfazione personale).
- Distrazioni d'ambiente: capacità di concentrazione dell'attore senza il coinvolgimento degli eventi esterni.

Per ogni caratteristica viene data una valutazione numerica con valore che varia da 1 (buono) a 5(mediocre). Eccone un esempio:

Motivazione ad apprendere:

1. Si accorge che attraverso il sistema può ottenere più informazioni, anche più velocemente. Attraverso di esso riesce a semplificarsi la vita.
2. Attraverso l'utilizzo del sistema l'utente si velocizza, ottiene le informazioni che desidera.
3. Capisce che l'utilizzo del sistema potrebbe semplificargli il raggiungimento dei goal ma non ha interesse o tempo nell'imparare il suo utilizzo
4. Non è interessato alla conoscenza del sistema, lo utilizza solo qualche volta per arrivare ai suoi obiettivi base.
5. Non comprende la potenzialità del sistema, non gli interessa approfondirle e non percepisce l'utilità

Successivamente, dopo aver raccolto tutte le informazioni utili, vengono mappate in un grafico detto *diagramma di strategia*:

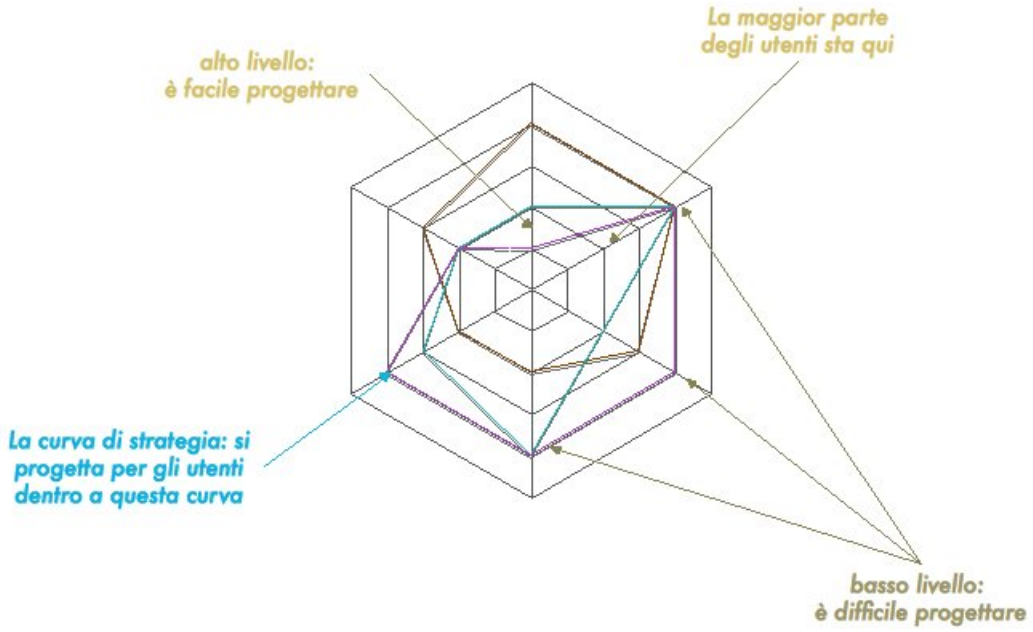

Figura 3.2: Diagramma di strategia di CAO=S

Questo diagramma riassume tutte le caratteristiche per cui bisogna sviluppare il sistema. Ogni vertice del diagramma rappresenta una caratteristica che permette, così, il tracciamento delle peculiarità degli attori rappresentati dai tracciati del colore verde, rosso e azzurro. CAO=S svilupperà quindi un interfaccia comune a tutti e tre gli attori analizzando l'area comune e personalizzerà l'area di un attore nel momento del suo utilizzo analizzando ogni aspetto non comune al momento del suo utilizzo.

3.2.3 O di CAO=S, operazioni

Le operazioni in CAO=S sono associate alle azioni sui concetti e non sulle strutture dati, tutte le azioni che effettua l'utente sull'interfaccia manipolano i concetti stessi. Le operazioni prese dai modelli CRUD E REST sono di quattro tipi: creazione, vista, modifica, cancellazione. Analizziamole con ordine:

Operazione Creazione

Attraverso questa operazione si crea una o più istanze di un concetto, ove l'operazione di creazione ha diverse caratteristiche intrinseche:

- Tipo di creazione: automatica, manuale o implicita
- Molteplicità: una sola o molteplici istanze
- Default dell'istanza : lo stato iniziale non deve essere mai vuoto, in modo da abbassare il carico cognitivo e di lavoro dell'utente.
- Persistenza: possibilità di memorizzazione dell'istanza del concetto
- Notifiche utente e d'interfaccia: avvengono durante l'operazione e nell'istante il cui il sistema trova il problema (così detto real-time).
- Memoria dell'utente: vengono segnalati i valori precedentemente immessi dall'utente per velocizzare la creazione rapida e suggerire possibili valori all'utente.

Operazione Vista

Attraverso l'operazione vista riusciamo a visualizzare le istanze del concetto recuperandole dalle strutture dati. Vediamone possibili:

- Vista individuale completa: le caratteristiche di un concetto sono tutte visibili.
- Vista individuale ridotta: solo alcune caratteristiche principali del concetto di riferimento vengono mostrate; sarà possibili comunque passare ad un'operazione di vista di tipo completa.
- Vista lista multipla: mostrata una lista di istanze del concetto, da qui sarà possibile passare a una vista individuale ridotta o completa sulla cui lista sarà possibile effettuare operazioni quali: ordinamento, selezione, filtri ecc.
- Vista lookup, questa tipologia di vista è utile per selezionare un'istanza di un concetto da eseguire in seguito.
- Vista ricapitolato multipla, vengono raggruppati i fatti sulle istanze del concetto e vengono mostrati insieme es: contatori, data di creazione ecc.

Dalle viste vengono mostrati e gestiti anche i criteri di filtri, selezione, raggruppamento, ordinamento utili a gestire correttamente i concetti.

Operazione Modifica

Questa operazione serve per modificare le proprietà di un'istanza di un concetto senza doverlo ricreare. Due sono le differenti tipologie di operazioni che la rendono possibile: quelle globali e quelle specifiche. La prima viene mostrata con la stessa maschera di quella di creazione ma con un contenuto diverso, mentre la seconda modifica solo alcune proprietà utilizzando maschere specifiche.

Operazione Cancellazione

Possiamo eliminare un'entità dal sistema oppure archiviarla. Eliminandola, questa viene rimossa dall'interfaccia e dalle strutture dati con una vera e propria cancellazione totale dell'entità.Attraverso l'archiviazione, invece, cancelliamo l'entità solo dalla "vista" dell'utente senza l'eliminazione definitiva dalla struttura dati.

3.2.4 S di CAO=S, le strutture

Per sviluppare lo scheletro dell'applicazione, CAO=S fa uso di una tabella tridimensionale che ha per assi Concetti, Attori e Operazioni; le celle descrivono le proprietà che li relaziona, come l'attore A debba eseguire l'operazione O sul concetto C. Attraverso questa tabella si riescono a creare le strutture dati di CAO=S:

- Viste: modelli di interfaccia, frammenti con cui si visualizzano le proprietà delle entità; in questa struttura è descritta la modalità di visualizzazione e vengono esposti i comandi attivabili durante la navigazione.
- Navigazione: il meccanismo utilizzato per passare da una struttura di tipo vista ad un'altra con le relative operazioni.
- Strutture dati: qui vengono memorizzate persistentemente le entità attraverso l'utilizzo di database.

Lo scheletro dell'applicazione è formato da viste collegate da comandi di navigazione. Ad ogni cella della tabella tridimensionale descritta precedentemente deve corrispondere una vista e/o un comando.

Sul diagramma vengono espresse le view che raccolgono celle diverse che sono coerenti tra loro per scopo e vincoli e che dunque permettono le operazioni specificate sui concetti specificati da parte degli attori specificati.

Di seguito riportiamo un'esempio di tabella che mostra le operazioni che possono essere effettuate dall'attore Docente sui concetti Insegnamento e Prova d'esame.

Docente	Insegnamento	Prova d'esame
Create	no	Singolo e multiplo default: su template istanze precedenti
Vista	Singolo:dati pubblici Multiplo: dati minimi	Default: propri insegnamenti
Update	Programma del corso Pubblicazione del programma	Manuale: update globale
Remove	no	prove passate: archiviazione

Tabella 3.1: Tabella Concetti-Attori-Operazioni

L'applicazione è composta da viste collegate da comandi della struttura navigazione. Partendo dalla vista iniziale, si naviga verso le altre viste composte da comandi attivabili e presentazione dei dati. Ogni passo effettuato durante la navigazione è la rappresentazione delle informazioni che si trova in una specifica cella della tabella tridimensionale e che corrisponde ad una vista e/o a un comando.

Di seguito viene riportato lo schema di processo di CAO=S, che descrive le fasi utilizzate dal modello di design, dal reperimento delle informazioni di progettazione fino all'implementazione dell'applicazione. Nel caso un'applicazione risulti troppo complessa per essere sviluppata attraverso il modello di design CAO=S, si coinvolge un esperto di usabilità e la progettazione avviene tradizionalmente.

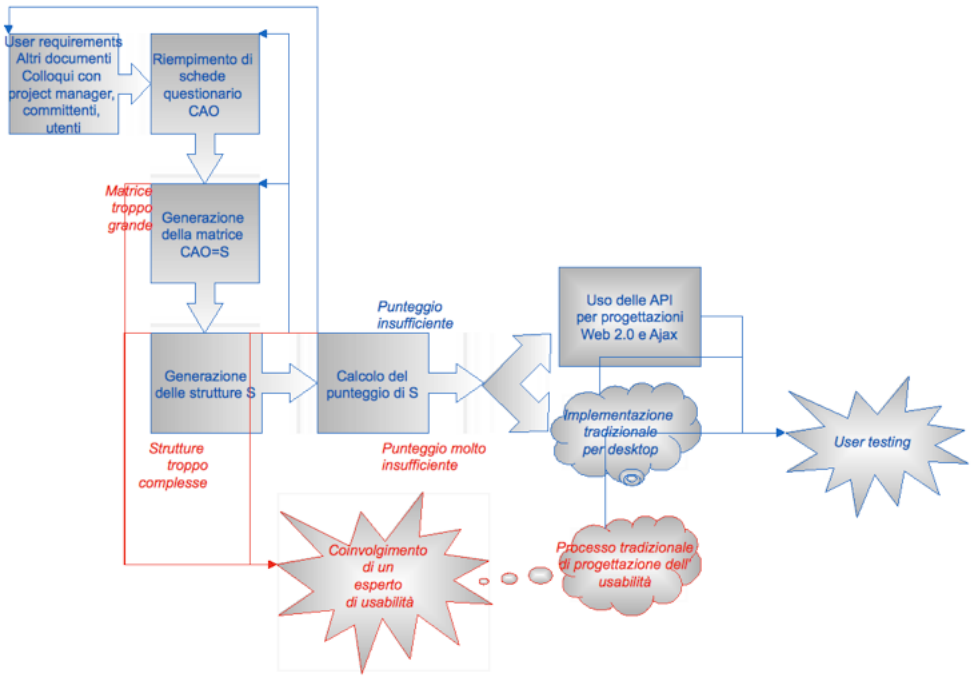

Figura 3.3: Schema di processo

3.3 L'architettura per CAO=S

Il modello di progettazione CAO=S è implementabile sviluppando due applicazioni differenti:

- CAO=S Analyzer: consente attraverso la compilazione di report e form da parte del team di sviluppo, di analizzare i dati in input e creare una tabella tridimensionale che ha per assi Concetti, Attori e Operazioni utilizzando il diagramma di strategia di CAO=S.

- CAO=S Generator: si occupa di sviluppare concretamente l'applicazione delineata dal team di sviluppo attraverso l'analisi del documento xml in input, generando automaticamente il codice server e client side, l'interfaccia utente e il modulo di persistenza dei dati.

  L'immagine seguente rappresenta la suddivisione dei passi fondamentali di un'architettura CAO=S.

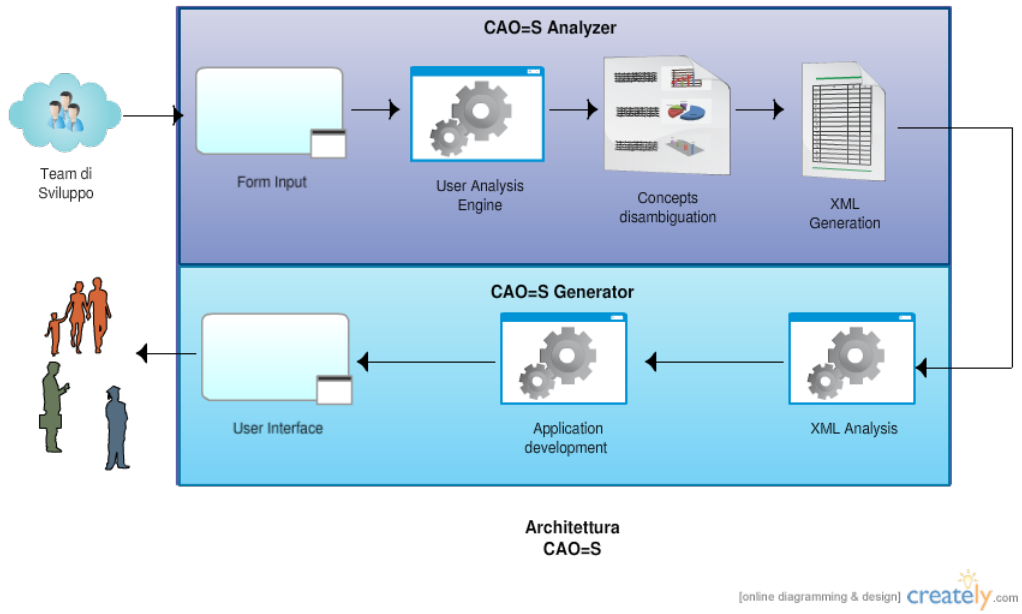

Figura 3.4: Architettura di CAO=S

Capitolo 4

ERIS (CAO=S generator)

Nel corso di questo lavoro, oltre ad aver preso parte allo sviluppo del modello CAO=S e alla progettazione dell'architettura, mi sono occupato della realizzazione di ERIS[1].

ERIS, attraverso l'analisi del documento xml creato dall'applicazione di analisi, genera il codice dell'applicazione (client side e server side), l'interfaccia utente (UI) e il modulo di persistenza dei dati automatizzando lo sviluppo dell'applicazione.

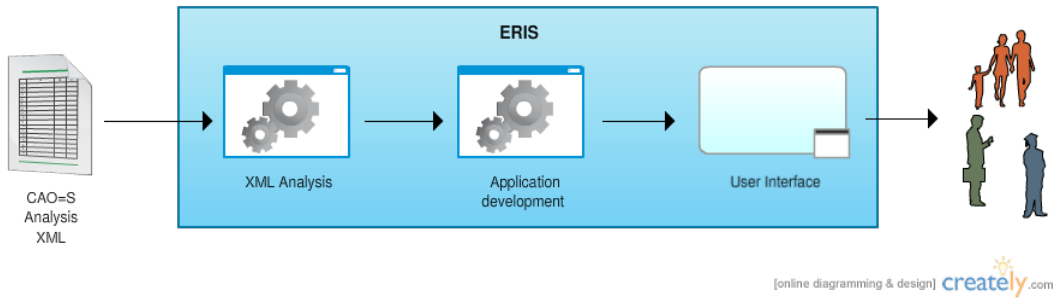

Figura 4.1: ERIS

[1]"dal greco antico Ἔρις, «conflitto, lite, contesa») è una figura della mitologia greca, la dea del Caos. È strettamente legata ad Ares, cui spesso si accompagna, e secondo alcuni faceva da guardia al palazzo del dio della guerra." fonte: http://it.wikipedia.org/wiki/Eris_(mitologia)

4.1 Scelte progettuali

Durante la fase di analisi su come sviluppare ERIS, ho identificato alcune caratteristiche fondamentali che le applicazione realizzate attraverso di essa dovrebbero avere:

- Uso del paradigma REST (Representational State Transfer [FIE00]) è un tipo di architettura software per i sistemi di ipertesto distribuiti, si riferisce ad un insieme di principi di architetture di rete che delineano come le risorse sono definite e indirizzate. È un paradigma utilizzato per la realizzazione di applicazioni Web che permette la manipolazione delle risorse per mezzo dei metodi GET, POST, PUT e DELETE del protocollo HTTP. Prevede un'architettura di tipo:

  - Client-Server: i client sono separati attraverso l'interfaccia universale.
  - Stateless: lo stato del client nella comunicazione con il server. Lo stato non viene salvato nel server e per ogni richiesta che viene effettuata dal client, esso è in grado di rispondere senza di mantenere informazioni.
  - Cacheable, i client gestiscono la cache per memorizzare vecchie richieste utili.
  - A livelli, un client non è a conoscenza del tipo di sistema con cui è connesso, non sa se la comunicazione avviene direttamente con un server oppure attraverso l'uso di intermediari.

  Un concetto fondamentale di REST è l'utilizzo delle risorse che possono essere accedute attraverso l'utilizzo un identificatore chiamato URI; le risorse vengono scambiate fra i client e i server, che comunicano attraverso l'uso di un'interfaccia standard (HTTP); la cosa interessante è che un numero qualsiasi di connettori possono fare da mediatori nella richiesta e ognuno dei connettori conoscendo l'identificatore della risorsa (URI) e l'azione richiesta, media per reperire l'informazione cercata.

- Faccia uso di ORM [ANU08]: ORM (Object Relational Mapping), fornisce mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico database utilizzato. L'uso di questo sistema favorisce il superamento dell'incompatibilità di fondo tra il progetto orientato agli oggetti ed il modello relazionale,

aumenta notevolmente la portabilità evitando la riscrittura delle routine che implementano lo strato di persistenza nel cambio fra Database Manager System, maschera semplicemente tutte quelle complesse operazioni che vengono effettuate sui dati dette CRUD[CRU10](Create, Read, Update, Delete) riducendo drasticamente la quantità di codice da redigere e i relativi errori, gestisce la concorrenza ai dati durante l'utilizzo del sistema evitando conflitti di accesso ai dati in caso accesso contemporaneo, fornisce automaticamente un supporto alla caching attraverso meccanismi sui dati. Ma la scelta principale per cui le applicazioni sviluppate con ERIS devono far uso di ORM è la gestione della comunicazione mediante l'uso del pattern di design Unit of Work, che ritarda tutte le azioni di aggiornamento dei dati al momento della chiusura della comunicazione. Per far ciò vengono inoltre inviate, in un unico statement, query multiple al Database Manager System limitando cosi al minimo i tempi di risposta dell'applicazione.

- Uso del i18n/l10n [AYK05]: l'inserimento di questo processo fra le caratteristiche base che le applicazioni dovranno avere mira a rendere ERIS fruibile da tipologie di team di sviluppo prodotti per un mercati specifici o multinazionali. I18n e i10n sono abbreviazioni di internalization e localization, utilizzati per adattare il software a lingue diverse e a differenze regionali. Internalization è il processo di progettazione di una applicazione software in modo che possa essere adattato alle varie lingue e regioni, senza modifiche di codice e struttura. Localization è invece il processo di adattamento software per una specifica regione o lingua con l'aggiunta di alcune impostazioni locali di specifiche componenti e traduzione del testo.

- Sia sviluppata attraverso il paradigma MVC (Model View Controller [LR01]): questo paradigma rende la progettazione e la gestione delle applicazioni Web più agevole. Il pattern è strutturato sulla separazione dei compiti del software in 3 livelli:

  - Il Controller: è un modulo che si occupa di gestire le richieste che vengono fatte tipicamente dall'utente, invoca il model per reperire le risorse giuste e la view che le rappresenta correttamente.
  - Il Model: rappresenta i dati e le regole da applicare per la loro gestione. Come dati si intende a tutte le informazioni che verrano salvate dal sistema in modo

persistente e che devono sottostare a regole di cui il Model è responsabile. Questo livello fornisce al Controller i dati richiesti che si occuperà poi di richiamare la giusta View per presentarli all'utente.

– La View: è un livello che fornisce differenti modalità di presentazione dei dati, si occupa di visualizzare i dati contenuti nel model e d'interagire con l'utente.

Volendo basarmi su queste caratteristiche, lo sviluppo delle applicazioni con l'utilizzo ERIS risultava abbastanza complicato, e il codice sarebbe risultato sporco e non performante; per evitare questi problemi ho deciso di far uso di un framework con cui ERIS riesce, interfacciandosi, a sviluppare applicazioni più velocemente.

Con l'aiuto di un framework il disegno architettonico dell'applicazione è migliore e semplificato ed i tempi di sviluppo del progetto diminuiscono notevolmente. Il framework stesso è caratterizzato da componenti standard sulla tecnologia di riferimento e mette a disposizione degli sviluppatori funzionalità che fanno uso di design-pattern e best-practice; per sua stessa natura, dovendo essere utilizzato genericamente anche per altre tipologie di progetto, è necessario che sia estendibile ed adattabile alle esigenze di qualsiasi sviluppatore.

Dopo aver scelto di utilizzare un framework, sono state fatti studi che hanno portato a identificare quale fosse il migliore nello sviluppo del progetto. Di seguito riporto i framework di maggior interesse per progetti PHP [PHP08].

Framework	Versione stabile attuale	Data di rilascio	Tipo di Licenza
CakePhp	1.3.0	24-04-2010	MIT
CodeIgniter	1.7.2	11-09-11	BSD-Style
Horde	3.3.6	15-12-2009	LGPL
Kohana	3.0.6	08-06-2010	BSD-Style
Qcodo	0.4.10	23-12-2009	MIT
Seagull	0.6.7	23-02-2010	BSD
Symfony	1.4.4	06-04-2010	MIT
Zend Framework	1.10.4	28-04-2010	BSD

Tabella 4.1: Framework php

Tre sono i framework php più diffusi con cui lavora la maggior parte della community php: CakePhp [CRU10], Symfony [PZ07] e Zend Framework [ALB08]. Nella tabella 4.2 riportiamo alcune delle loro caratteristiche.

Framework	Cake PHP	Symfony	Zend Framework
Linguaggio	php	php5	php5
Ajax	Prototype/script.aculo.us, jQuery/jQuery UI, MooTools/MooTools, etc.	Prototype, script.aculo.us, Unobtrusive Ajax with UJS	Toolkit-independent
MVC framework	Yes	Yes	Yes
MVC Push/Pull	Push	Push	Push and Pull
i18n & l10n	Yes	Yes	Yes
ORM	Active record pattern (1.x), Data Mapper Pattern (2.x)	Propel, Doctrine (YAML)	Table and Row data gateway
Testing framework	Unit Tests, Object Mocking, Fixtures, Code Coverage, Memory Analysis	Yes	Unit Tests
DB migration framework	Yes	Yes	Yes
Security Framework	ACL-based	Plugin	ACL-based
Template Framework	Themes, Layouts, Views	Yes	Yes
Caching Framework	Memcache, Xcache, APC, File	Yes	Yes
Form Validation Framework	Validation and Security	Yes	Yes

Tabella 4.2: Caratteristiche tecniche di: Cake Php, Symfony, Zend Framework

Sono state analizzate anche altre caratteristiche non tecniche del framework. Di seguito ne vengono riportate alcune:

- Funzionalità aggiuntive: anche se non espressamente richieste possono essere ritenute importanti (la generazione automatica del codice, il supporto ai web services ecc..)
- Verifica dello stile di progettazione: il framework propone una metodologia di progettazione che si avvicina al modo di lavorare dei programmatori.
- Documentazione: ufficiale e non deve essere completa ed esauriente e fruibile in base alle tipologie di documentanzione, api, tutorial, how-to; permettendo di iniziare ad essere produttivi molto rapidamente, abbassando la curva di apprendimento.
- La licenza del framework e i requisiti minimi richiesti: devono essere compatibili con i progetti universitari.
- La vivacità della community: è un fattore determinante di un prodotto software; una community attiva permette di risolvere problemi, bug, errori in tempi molto ristretti.

In base a tutte queste caratteristiche, il framework migliore di cui ERIS fa uso nello sviluppo di applicazioni è Symfony.

Symfony [PZ07] è un web application framework per progetti PHP ed attraverso di esso possiamo facilmente:

- Creare applicazioni automatizzando molte funzionalità
- Seguire una struttura logica di sviluppo di tipo step-by-step che ne rende l'utilizzo più facile.
- Dà la possibilità al programmatore di eseguire operazioni anche complesse attraverso il richiamo a semplici istruzioni facendo uso di script e funzioni già previste all'interno del framework.

Per gli utenti abituati ad utilizzare php e i pattern di progettazione per applicazioni web, la curva di apprendimento nell'utilizzo del framework è ridotta a meno di un giorno; si tratta di un framework personalizzabile, adatto alla creazione di applicazioni robuste, dove si ha il pieno controllo della sua configurazione: la struttura delle directory, le librerie esterne ecc.. Include tool che permettono di testare, individuare bug e documentare velocemente l'applicazione che si va a realizzare, è completamente gratuito e pubblicato sotto licenza MIT[2] e gestisce nativamente l'escaping degli output aumentando il livello generale di sicurezza relativo all'applicazione. Un'ultimo pregio scegliendo Symfony è ottenere i benefici di una community molto attiva che comunica attraverso blog, forum, chat, mailing-list, newsletter ecc. Symfony è sponsorizzato da Sensio Labs, una Web Agency francese nota per le sue innovazioni in materia di sviluppo web.

Richiede per essere installato un'architettura LAMP (Linux, Apache, MySQL e Php 5 [LB02]) offrendo la possibilità di essere utilizzato anche con altri database come PostgreSQL, Oracle, SQlite, Microsoft SQL Server. Fa uso del pattern architetturale Model-View-Controll (MVC).

Symfony separa il codice relativo alla logica dell'applicazione da quello relativo all'interfaccia in modo, che lo sviluppatore possa dedicarsi completamente allo sviluppo della parte logica mentre il designer allo sviluppo della parte del templating senza dover interferire e mettendo a disposizione degli sviluppatori:

- Uno strumento per il testing in fase di sviluppo
- Un pannello di opzioni di debugging
- Funzionalità per la generazione di codice;
- Un sistema di log per analizzare del funzionamento dell'applicazione

[2]È una licenza permissiva, cioè permette il riutilizzo nel software proprietario sotto la condizione che la licenza sia distribuita con tale software. Fonte: http://it.wikipedia.org/wiki/Licenza_MIT

4.2 Le fasi di ERIS

Attraverso l'analisi del file xml e utilizzando il framework Symfony, ERIS riesce a sviluppare un'applicazione web seguendo determinate fasi che ne descrivono la logica di sviluppo:

1. Inizializzazione dell'applicazione
2. Creazione del modulo di persistenza
3. Creazione della logica dell'applicazione
4. Realizzazione dell'interfaccia

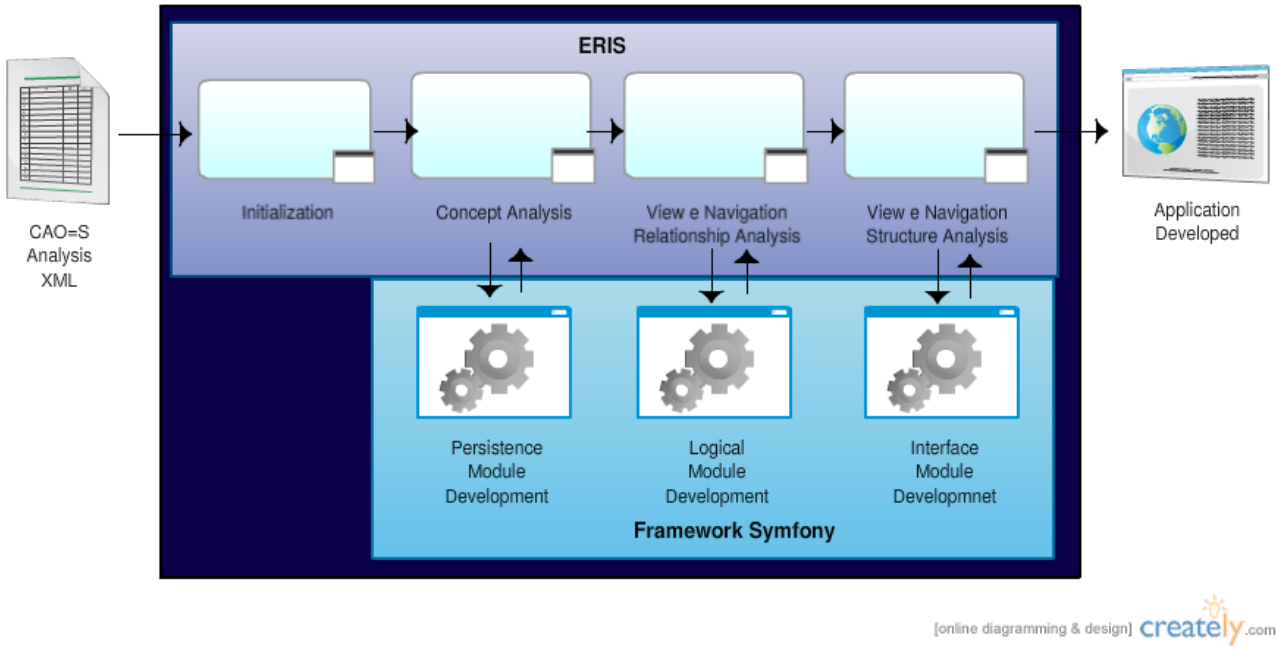

Figura 4.2: ERIS

Come si può notare dall'immagine dell'architettura, vi è un'interazione molto forte fra ERIS e il framework Symfony: il primo riceve in input un file xml che descrive 3 diverse strutture dati: i concetti, le view e la navigazione; ogniuna di queste anche se ben distinta è altamente correlata con le altre e viene sviluppata e analizzata seguendo il pattern architetturale MVC.

Nella prima fase viene inizializzato il sistema, preparato l'ambiente di lavoro in cui ERIS sviluppa l'applicazione; nella seconda vengono analizzati i concetti descritti nel file xml e creato il modulo di persistenza dei dati. I concetti rappresentano le informazioni percepite dall'utente e ne rappresentano le strutture dati; attraverso la loro analisi si riesce ad individuare quella parte dei concetti che serve a creare il modulo di persistenza dati. La terza fase, la più complessa ha lo scopo di sviluppare la logica di controllo dell'applicazione attraverso l'analisi delle strutture View e Navigation del file in input. L'ultima fase, sviluppa il modello dell'interfaccia, scomponendo le strutture View e Navigation del documento xml, riesce a creare il modulo View che si occupa di presentare le informazioni dei concetti all'utente attraverso l'interfaccia.

4.2.1 Inizializzazione dell'applicazione

La prima fase, detta d'inizializzazione, prepara il sistema per le fasi successive, predisponendone l'ambiente di lavoro. Tutte le informazioni che sono necessarie in questa fase sono descritte in un file xml di configurazione.:

- l'url di Symfony: utile per scaricare il framework
- projectRoot: il PATH dove l'applicazione viene sviluppata
- projectName: il nome del progetto da sviluppare
- database: tutte le informazioni necessarie alla connessione col database
- altre: informazioni che aiutano ERIS a configurare l'ambiente di sviluppo.

Mostriamo un esempio del file config.xml:

```
<symfony>www.symfony-project.org/get/symfony-1.4.3.tgz</symfony>
<projectRoot>/Users/enricozoli/Sites/sfproject/</projectRoot>
<projectName>alma</projectName>
<database>
        <type>mysql</type>
        <hostname>localhost</hostname>
        <port></port>
        <dbname>alma</dbname>
        <user>root</user>
        <password></password>
</database>
```

ERIS, dopo aver preparato l'ambiente di lavoro ed installato il framework, si prepara a creare il progetto utilizzando l'interfaccia a riga di comando. Il sistema fa un uso abbastanza massiccio inizialmente dell'interfaccia a linea di comando che l'aiuta ad automatizzare gran parte del lavoro. Di seguito viene riportato proprio il comando utilizzato per generare il progetto *ALMA* descritto nel documento xml di configurazione:

```
php symfony generate:project alma
```

Il task *generate:project alma*, crea la struttura di default delle cartelle e i file necessari per il progetto:

- apps: contiene tutte le applicazioni del progetto
- cache: i file di cache del progetto
- config: i file di configurazione del progetto
- lib: le librerie e le classi di progetto
- log: i file di log del framework
- plugins: i plugin installati
- test: unari e funzionali
- web: la cartella principale

Successivamente all'organizzazione dell'ambiente di sviluppo, ERIS inizializza il sistema di persistenza dei dati tramite l'interfaccia di Symfony; per memorizzare tutte le informazioni si è deciso di utilizzare un database relazionale (MySQL, PostgreSQL, SQLite, Oracle, MSSQL) già supportati dalla versione base di Symfony, che fa uso di PDO. PDO [PDO10] acronimo di "Php Data Objects" è un layer di astrazione in grado di pilotare database di diverso tipo tramite un'interfaccia unica, indispensabile per rendere un applicativo portabile su multi-piattaforma e fa uso di diverse tipologie di database. Symfony è un framework orientato agli oggetti e dà la possibilità agli sviluppatori di utilizzare due strumenti ORM che utilizzano l'estensione php PDO: Propel e Docrine

Vediamo brevemente le differenze:

- Propel [PRO10]: permette làaccesso al database usando un insieme di oggetti e mette a disposizione delle API per scrivere e leggere i dati; questo ORM [PRO10] permette allo sviluppatore di lavorare nello stesso modo con cui si lavora con le altre classi e gli oggetti in PHP.

- Doctrine [WBB10]: si appoggia nella parte superiore del database abstraction layer; una delle sue principali particolarità è la possibilità di scrivere query in una specie di sottolinguaggio SQL chiamato DQL. Questo permette allo sviluppatore di sfruttare un'alternativa all'SQL per mantenere la massima flessibilità evitando la duplicazione del codice.

ERIS utilizza Symfony con *ORM Doctrine* [WBB10].

4.2.2 Modulo di persistenza

Ultimata la fase di starup, il sistema è pronto a iniziare le varie fasi utili alla realizzazione del progetto; ERIS, in questa seconda fase, analizza inizialmente i concetti descritti all'interno del file xml preso in input per poterne ricavare informazioni utili allo sviluppo del modello di persistenza dei dati. I Concetti descritti rappresentano le informazioni percepite dall'utente e ne rappresentano le strutture dati; attraverso la loro analisi si riesce ad individuare quella parte dei Concetti utile a creare il modulo di persistenza dati.

Di seguito viene mostrato un esempio di Concetto chiamato *verbale*:

```
<concept name="verbale">
<automaticField name="codiceverbale" type="string" id="true"/>
<joinField concept="studente" name="matricola" type="string"/>
<lookupField concept="studente" name="cognome" type="string"/>
<lookupField concept="studente" name="nome" type="string"/>
<joinField concept="insegnamento" name="codicecorso" type="string"/>
<lookupField concept="insegnamento" name="nomecorso" type="string"/>
<lookupField concept="insegnamento" name="facolta" type="string"/>
<lookupField concept="insegnamento" name="codicefacolta" type="string"/>
<automaticField name="AA" type="string" listable="true"/>
<editableField name="dataverbale" type="date" queryable="true"/>
<editableField name="votoverbale" type="string" listable="true"/>
</concept>
```

Per ricavarne le informazioni utili alla creazione della struttura dati persistente del progetto, ERIS analizza il tagname degli elementi che compongono i Concetti. Durante l'analisi decide se mappare i Concetti in Entità del database e gli elementi del Concetto come record di quell'entità. Esistono differenti elementi che possono descrivere le caratteristiche di un Concetto ed ognuno di essi verrà analizzato dal sistema differentemente:

AutomaticField: attraverso l'utilizzo di questo tagname è indicato un elemento il cui legame col concetto è alla base del Concetto stesso identificando tutti quei campi di dati permanenti che non possono essere creati o modificati dall'utente, ma unicamente dal sistema. Un esempio è il codice di un verbale, la matricola per uno studente, ecc.

JoinField: serve a specificare la relazione che vi è fra il concetto corrente ed un'altro.

LookupField: identifica un campo che ha un legame debole col concetto. L'elemento descritto da questo tagname, non viene utilizzato mappato come record nella memorizzazione persistente del Concetto; per esso vi è sempre un corrispettivo elemento di tipo *AutomaticField* o *EditableField* in un altro Concetto, ciò significa che per ogni legame debole al concetto che si sta analizzando deve esisterne uno forte in un Concetto diverso. Tutti gli elementi descritti con questo tagname rappresentano in realtà quelle informazioni che l'utente percepisce collegate al concetto, ma

di cui il sistema non ha bisogno di mantenere nella struttura dati. Le informazioni riguardanti questo elemento verranno reperite attraverso l'elemento *joinFiled* che specifica la relazione del concetto con cui è legato. Se esaminiamo, per esempio, il concetto mostrato precedentemente, notiamo che l'elemento che rappresenta il *nome dello studente* all'interno del Concetto *verbale* fa parte del Concetto *studente*, perciò il sistema andrà a reperire l'informazione *nome studente* attraverso il Concetto studente, ove è memorizzata.

EditableField: specifica un elemento con un legame molto forte col concetto, attraverso l'utilizzo di questo tagname riusciamo a identificare successivamente quali sono i campi del sistema dove possiamo effettuare delle operazioni di modifica.

ListField: identifica come la *LookupField* un campo che ha un legame debole col concetto, ma a differenza di essa rende fruibile una lista di informazioni reperibili da altri concetti.

Ogni elemento, inoltre, può avere degli attributi che lo descrivono:

- Name: il nome che lo identifica
- Concept: il concetto a cui è relazionato
- Type: il tipo dato che lo definisce
- Queryable: indica se possono esser effettuate operazioni di tipo query
- Listable: indica se possono essere effettuate operazioni di ordinamento
- Id: indica se è l'identificativo dell'istanza del concetto.

Per estrapolare le informazioni utili alla creazione della struttura dati, ERIS effettua un parsing di tutti i concetti descritti all'interno del documento xml. Dopo un'attenta analisi crea le strutture dati per la memorizzazione permanente delle informazioni che sono la rappresentazione semplificata della struttura dei Concetti. Di fatto ERIS crea una tabella per ogni Concetto da cui prende anche il nome e ne definisce le colonne attraverso il mapping con tutti gli elementi del Concetto con tagname: *AutomaticField, EditableField, JoinField*. Le colonne delle tabelle del database dell'applicazione, rappresentano quegli elementi che hanno un legame forte col Concetto.

Gli elementi con tagname *LookupField* non vengono utilizzati per la realizzazione delle strutture dati poiché sono banalmente degli elementi utilizzati come riferimenti alle informazioni di altri Concetti: saranno, invece, parte fondamentale nella creazione delle View ed utilizzati per far giungere all'utente le informazioni legate al concetto.

Attraverso il parsing del file e l'analisi del file xml, ERIS, sviluppa il modulo di persistenza dei dati descrivendo le tabelle dell'applicazione in formato YAML [**?**]; successivamente utilizza l'ORM di Symfony per tradurre la descrizione del database in codice SQL e ultimare, così, lo sviluppo del modulo.

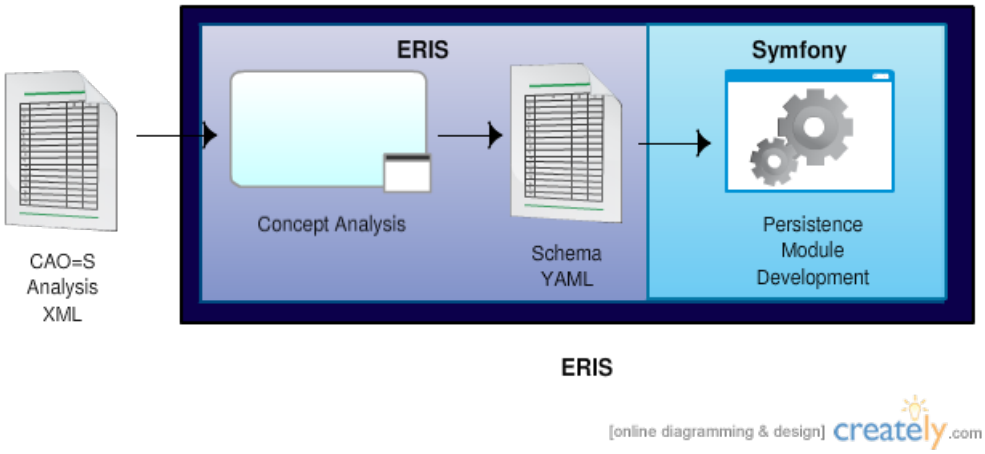

Figura 4.3: Creazione del modulo di persistenza

Il formato YAML viene descritto come uno standard di serializzazione dei dati human friendly per tutti i linguaggi di programmazione, con questo formato possiamo descrivere semplicemente stringhe, interi, dati , array, ed altri. La struttura con cui si descrivono i dati è basata sul concetto d'indentazione, infatti le coppie chiave/valore vengono separate dai ":" mentre entità consecutive vengono indicate con il trattino. Symfony utilizza questo formato per i suoi file di configurazione.

ERIS genera un file YAML che contiene le informazioni di tutte le tabelle e relative colonne.

Ogni colonna all'interno del file è descritta nel seguente modo:

- type: il tipo di colonna (boolean, integer, float, string, array, blob, timestamp, date, enum, ecc..), questa colonna mostra l'attributo type dell'elemento specifico interno al concetto analizzato.

- notnull: attributo booleano di una colonna che mostra se si desidera che la colonna sia obbligatoria. Durante il parsing del documento xml, gli elementi con l'attributo id di valore true(che rappresentano la chiave primaria delle tabelle), non possono essere descritti nel file YAML attraverso una colonna la cui proprietà è null. Lo stesso succede per tutti gli elementi con tagname *JoinField*, che rappresentano le chiavi secondarie delle tabelle e che permettono cosi l'esistenza di relazioni fra concetti.

- unique: attributo di tipo booleano della colonna utilizzato nel caso in cui si voglia creare una chiave univoca per la colonna stessa.

Di seguito si riporta il codice del file YAML realizzato da ERIS attraverso l'analisi del Concetto *verbale* precedentemente mostrato:

```
1  verbale:
2   columns:
3    id: { type: integer, autoincrement:true ,primary: true}
4    AA: { type: string(255), notnull: true }
5    dataverbale: { type: date(255), notnull: true }
6    votoverbale: { type: string(255), notnull: true }
7    matricola:
8     type: integer
9    codicecorso:
10    type: integer
11  relations:
12   studente:
13    local: matricola
14    foreign: id
15   insegnamento:
16    local: codicecorso
17    foreign: id
```

Utilizzando alcuni task integrati all'ORM Docrine, riusciamo a generare automaticamente il codice SQL necessario alla creazione delle tabelle del nostro modulo di persistenza attraverso l'utilizzo del file YAML realizzato da ERIS. Doctrine riesce a trasformare il codice interno al documento YAML in codice SQL e successivamente crea le strutture dati all'interno del database.

Symfony, nel momento in cui vengono realizzate le strutture dati, genera automaticamente delle librerie php che danno la possibilità ad ERIS di utilizzare le entità del database manipolandole come se fossero degli oggetti; i valori delle colonne di un record possono essere quindi manipolati tramite un oggetto del modello, utilizzando i metodi getter e setter.

4.2.3 Logica dell'applicazione

In seguito allo sviluppo del modello di persistenza dei dati, ERIS analizza la struttura dati View interne al documento xml; la struttura dati view è composta da diversi elementi View che rappresentano per l'utente una specifica interfaccia che gli consente di effettuare operazioni sui dati in un determinato task, ma per ERIS descrive come i dati devono essere rappresentati, specificando i moduli che fanno parte dell'interfaccia e i frammenti che la compongono.

Attraverso l'analisi di questa struttura ERIS riesce a sviluppare i moduli di view e controller della futura applicazione. Una View può essere composta da diversi elementi/strutture: liste, sommari, form, navigazione, sottoviste, ecc.. Analizziamole nel dettaglio.

List

Questo elemento definisce una struttura dinamica composta da elementi dello stesso tipo, fa riferimento ad un concetto e mostra come colonne tutti gli elementi (con l'attributo *listable*) contenuti in esso. Una List viene presentata all'utente attraverso l'interfaccia con una tabella formata da elementi che si riferiscono ad uno specificato concetto. Viene mostrata di seguito un esempio di List.

```
<list concept="verbale">
        <heading>Lista dei verbali</heading>
        <operations>
                <select/>
                <multipleSelect/>
                <show view="registroForm"/>
                <edit view="registroForm"/>
                <sort/>
                <group/>
                <filter/>
        </operations>
</list>
```

Il controller dell'applicazione, nel momento in cui viene richiesta la visualizzazione delle informazioni riguardanti un Concetto, prima reperisce tutte le informazioni dal modulo di persistenza dei dati e successivamente le invia alla View.

Non essendo un Concetto una struttura dati permanente, e dato che solo una parte degli elementi che lo compongono vengono realmente salvati all'interno della tabella che lo rappresenta, nel momento in cui il controller dell'applicazione, attraverso i suoi meccanismi di view richiama il component *Lista*, si dovrà occupare anche di reperire tutte le informazioni mancanti per renderle fruibili. Di fatto le informazioni mancanti nella struttura dati che rappresenta il concetto saranno solamente quelle che il Concetto descrive come utili per l'utente nella comprensione e nella percezione del concetto stesso.

Gli elementi mancanti all'interno delle strutture dati, infatti, sono proprio quelli indicati dal tagname lookup definiti nei concetti, che vengono reperiti tramite le relazioni che hanno con gli altri concetti. Il controller, nel momento di creazione del componente *List*, va a ricercare le informazioni degli elementi *lookup* presso altri Concetti seguendo i legami che vi sono fra i Concetti. Utilizzando come punto di partenza l'elemento *JoinField* che ha il valore dell'attributo concept uguale a quello dell'elemento *lookup* preso in considerazione nel concetto considerato.

La List è formata da sotto elementi che la descrivono:

Heading: rappresenta l'intestazione della tabella

Operations: è un elemento che compone List che serve a definire le operazioni che possono essere effettuate su quella specifica lista. ERIS effettuando il parsing di questo elemento, modifica le caratteristiche del controller della lista rendendo possibile il concreto utilizzo delle operazioni sull'elemento all'interno dell'applicazione. Alcune delle operazioni definibili tramite questo elemento sono la selezione di una riga, la selezione multipla di più righe, l'ordinamento in base alla colonna, la possibilità di filtrare i risultati in base a certe caratteristiche scelte dall'utente ed altri ancora.

Summary

L'elemento Summary viene utilizzato per esprimere informazioni riguardanti la View che lo contiene; ogni Summary fa riferimento a un concetto preciso in cui può applicare anche operazioni e sistemi di filtraggio sui dati, ha un Heading che ne rappresenta l'intestazione e un Content che definisce qual'è il contenuto della Summary e quali sono le operazioni che ERIS deve implementare all'interno del controller per soddisfare successivamente la futura richiesta della View. Le operazioni si basano sui dati che il controller reperisce dai Concetti, applicandovi se ve ne sono dei filtri. Queste operazioni sono definite in XPath, un linguaggio che fa parte della famiglia XML e che permette di individuare i nodi all'interno di un documento di nodi. Il sistema, analizzando questo elemento, dovrà, quindi, mostrare le informazioni incluse all'interno dell'head e del content applicandovi dei filtri descritti come attributi dell'elemento stesso. Eccone un esempio:

```
<summary concept="verbale"  filter="dataVerbale='' " >
        <heading>
                Verbali da completare
        </heading>
        <content>
                Sono presenti {count(.)} verbali da completare.
        </content>
</summary>
```

Form

L'elemento form indica la realizzazione di una struttura HTML di tipo form che dia la possibilità all'utente di interagire con la vista stessa, è caratterizzato da un sotto elemento Header, che ne definisce il contenuto dell'intestazione e da l'elemento Operations che ne definisce le operazioni. Come tutti i form, questo elemento può essere utilizzato per creare una nuova istanza di un concetto, modificarne una già esistente o addirittura cancellarla.

Un'ulteriore operazione è stata aggiunta con *archivie*, con questa operazione il sistema dovrà sviluppare un meccanismo di archiviazione delle informazioni, mantenendo le informazioni interne al sistema senza dunque cancellarle ma dando l'impressione all'utente che tali informazioni non siano più reperibili. ERIS creerà dunque i form con diverse proprietà basandosi sull'analisi dell'elemento descritto nel file xml preso in input. Come nelle List, anche qui il Concetto è il punto fondamentale con cui il form è legato e quindi il sistema andrà a reperire tutte le informazioni richieste dal controller attraverso la rappresentazione del concetto nel modulo di persistenza.

```
<form concept="registro" >
        <heading>Registri lezione</heading>
        <operations>
                <archivie>
                <remove condition="not(firmato)"/>
                <create>
        </operations>
</form>
```

Subview

Questo è uno degli elementi più importanti che compongono le View, attraverso di esso, possiamo include View ad altre View, costruendo "View" sempre più complesse. ERIS dovrà sviluppare nell'applicazione un controller e un sistema di sviluppo dell'interfaccia in modo da poter utilizzare a proprio piacimento le varie View del sistema, trattando tutte le View descritte nel file xml come dei frammenti riutilizzabili sviluppando interfacce per composizione.

```
<subview>
        <view ref="verbaleDaCompletareCompact"/>
        <view ref="verbaleCompletatoCompact"/>
        <view ref="registroCompact"/>
</subview>
```

Navigation

La navigazione è il meccanismo utilizzato per passare da un elemento di tipo View ad un altro; durante la sua analisi, ERIS è a conoscenza che questa struttura non identifica solamente la barra di navigazione dell'interfaccia ma di tutti gli elementi a cui è associata.

Di seguito mostriamo un esempio del concetto appena espresso:

```
<view name="verbaleList">
        <list concept="verbale">
        </heading>
        </operations>
        </list>
        <navigation>
        <link filter="dataVerbale=''" view="verbale">
        Visualizza i verbali che devono essere firmati
        </link>
        </navigation>
</view>
```

Possiamo notare che la struttura di navigazione interna a questa View *non* è specificata per una *barra di navigazione* che permette di visitare View differenti dell'applicazione, ma si riferisce all'elemento di tipo List. La navigazione quindi è una struttura utilizzata per descrivere tutti i meccanismi di navigazione interni alla view che si riferiscono a tutte le tipologie di elementi ed è compito di ERIS analizzare correttamente questo elemento e implementarne la navigazione.

Le strutture dati View e Navigation del documento xml vengono analizzate e mappate nei moduli Controller e View dell'applicazione finale.

Per capire come viene sviluppata la logica dell'applicazione, dobbiamo prima analizzare la struttura di Symfony per poi capire come ERIS vi interagisce.

In Symfony, le applicazioni che condividono lo stesso modello di dati persistenti sono raggruppate in un *Progetto*; se dovessimo sviluppare un portale con Symfony, avremmo

bisogno di due diverse applicazioni che interagiscono sugli stessi dati,ad esempio un'applicazione front-end per gli utenti che fruiscono delle informazioni e un applicazione back-end per gli amministratori.

D'ora in poi il concetto di *applicazione* è propria di Symfony e chiameremo Progetto il sistema che ERIS deve sviluppare.

Ogni applicazione interna al Progetto ha una struttura di questo tipo:

- config: contiene i file di configurazione dell'applicazione.
- lib: contiene le librerie e le classi dell'applicazione.
- modules: contiene il codice della View e del Controller dell'applicazione organizzati secondo i principi del pattern MVC.
- templates: contiene i file dei template utili alla definizione del layout.

Ogni applicazione è suddivisa ulteriormente in moduli contenuti nella directory modules, ciascuno dei quali contiene il codice che rappresenta l'insieme delle operazioni che l'utente o altri moduli dell'applicazione possono fare su un oggetto del modello (un modulo d'esempio per un blog può essere l'insieme del codice che offre la possibilità all'utente di effettuare operazioni sull'oggetto Articolo o Post).

Ogni modulo suddivide il codice della View e del Controller dell'applicazione in due sottocartelle:

- Actions: le azioni (il Controller del modulo).
- Templates: i template (la View del modulo).

Un esempio di come viene suddiviso il modulo della View *verbaleList:*

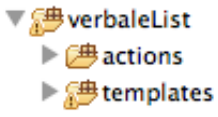

Figura 4.4: Struttura View VerbaleList

Con l'uso dell'interfaccia a linea di comando di Symfony, si possono generare automaticamente i moduli secondo due modelli determinati in base all'argomento del comando:

- Modulo *base*: viene creata la struttura del modulo, anche se la logica di controllo e visualizzazione deve essere sviluppata da ERIS.
- Modulo *entità*: passando come parametro al comando il nome di un'entità interna al database, il framework, oltre a realizzare la struttura del modulo, definisce tutte le azioni che possono essere effettuate sull'entità e sui record; quelle definite automaticamente nel file actions.class.php, situato all'interno della cartella actions, sono:
  - index, mostra i dati della tabella che il modello rappresenta
  - show, mostra i campi e i loro valori per una data riga della tabella
  - new, mostra un form per creare una nuova riga nella tabella
  - create, crea una nuova riga
  - edit, mostra un form per modificare una riga esistente
  - update, aggiorna una riga con i valori inseriti dall'utente
  - delete, cancella una riga dalla tabella.

ERIS interagisce con Symfony creando entrambe le tipologie dei moduli.

Nella costruzione del controller dell'applicazione finale vengono analizzate le strutture dati view e per ogni view viene creato un modulo base in quanto l'elemento view non descrive quali sono i dati interni al sistema ma ne gestisce la presentazione. Durante il parsing dell'elemento View, ERIS realizza automaticamente il modulo di tipo *entità* del concetto di riferimento della view e in questo modo vengono creati tutti i metodi e le classi basi, responsabili della gestione dell'oggetto rappresentato dal concetto nel database.

Per sviluppare la parte controller dei sotto-elementi della View (list, summary, form, navigation, subview, ecc.) nell'applicazione finale, ERIS sfrutta il concetto di frammento o componente: nell'architettura del documento xml una view può essere composta da sottoview, ma in symfony il modulo che rappresenta l'elemento view del file xml non

può essere utilizzato come frammento perché non può essere incluso o richiamato in o da un'altro modulo. Per mappare correttamente la logica del documento xml, ERIS gestisce tutti i sotto elementi delle View interne al documento xml, come se fossero frammenti interni alla view stessa.

Per questo motivo, e per evitare di scrivere codice identico più volte, ERIS utilizza tre tipologie diverse di frammenti disponibili in Symfony:

- Partial: viene utilizzato se la logica è leggera e si desidera includere un template con accesso agli stessi dati utilizzati dal controller e non è altro che un pezzo di codice di template riutilizzabile (principalmente dal modulo View dell'applicazione). In un'applicazione per pubblicazioni, ad esempio, il codice del template che visualizza un articolo è utilizzato nella pagina del dettaglio dell'articolo, nella lista dei migliori o ultimi articoli; questo codice è il candidato ideale per esser sviluppato attraverso i partial. Questa tipologia di frammento non può accedere, però, ai dati del modello di persistenza e non può, quindi, reperire informazioni. È un frammento utilizzato principalmente principalmente per il modulo View dell'applicazione dove la logica è inesistente.

Figura 4.5: Partial

- Component: viene utilizzato se la logica del controller è più complessa, dando la possibilità al frammento di accedere al modello dei dati ed effettuare operazioni. Questo particolare frammento è sviluppato separando la logica di controllo da quella di presentazione. Supponiamo, per esempio, di dover realizzare una barra laterale che mostra le ultime notizie di un dato soggetto a seconda del profilo utente che viene reso visibile in diverse pagine: le interrogazioni necessarie sono, quindi, troppo complesse per un partial che si occupi solo di template, a cui si se ne preferisce uno di questo tipo che separi la parte logica da quella di presentazione.

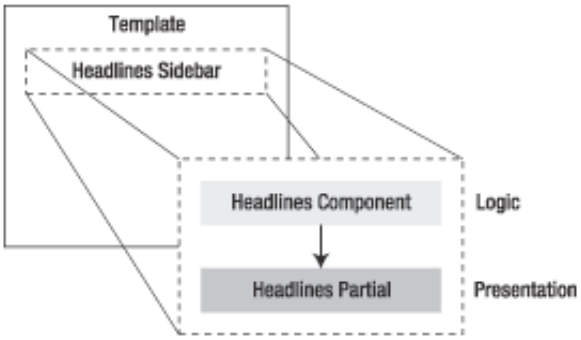

Figura 4.6: Component

- Slot: viene utilizzato nel caso sia necessario rimpiazzare diverse volte una parte del layout con un contenuto standard; viene utilizzato come una variabile fissa, ovvero il suo contenuto è definito globalmente e può essere incluso ovunque.

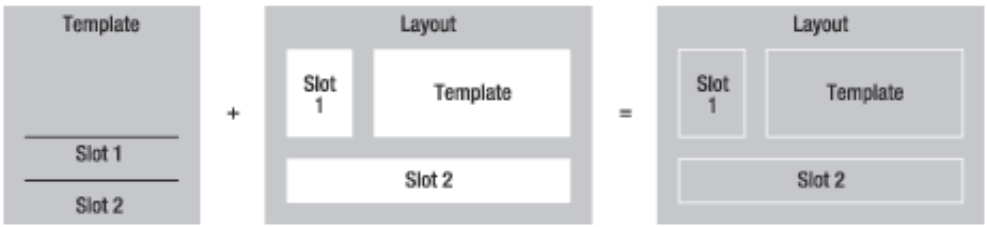

Figura 4.7: Slot

ERIS sviluppa gli elementi che compongono le View come frammenti di tipo Component: questa decisione è dovuta dalla possibilità di scambiare variabili in tutto il componente. La logica di questo frammento è inserita in un file chiamato components.class.php, inserito all'interno della cartella action del modulo, e il file che ne gestisce la presentazione viene inserito nella cartella template.

La decisione nell'utilizzo di questa tipologia di frammento è dovuta dalla possibilità di scambiare variabili in tutto il componente. La logica di questo frammento è inserita in un file chiamato components.class.php che è inserito all'interno della cartella action del modulo e il file che ne gestisce la presentazione viene inserito nella cartella template.

ERIS analizza passo per passo tutti gli elementi che definiscono le view e modifica in base alle specifiche esigenze dell'elemento le classi per la gestione del modello creando, modificando e cancellando i metodi che gestiscono l'accesso ai dati e il loro recupero.

Facciamo un esempio, analizzando, nello specifico, il comportamento nell'analisi del seguente elemento xml:

```
<view name="verbaleDaCompletareCompact">
        <summary concept="verbale" filter="dataverbale=''">
                <heading>Verbali da completare</heading>
                <content>
                Sono presenti {count(.)} verbali da completare
                </content>
        </summary>
        <navigation>
                <link view="verbaleCompletatiCompact">
                        Visualizza i verbali completati
                </link>
        </navigation>
</view>
```

ERIS inizializza il modulo base della view chiamandolo verbaleDaCompletareCompact ed inizia ad analizzare gli elementi che la compongono.

Identifica un elemento di tipo summary che si riferisce al concetto verbale ed ha un filtro il cui contenuto è "dataverbale=' ' " controllando se il modulo del concetto esiste (in caso contrario va a realizzare il modulo entità del concetto utile per il recupero delle informazioni interne alla struttura dati che lo rappresenta). Viene creato, successivamente, il component Summary interno al modulo della View verbaleDaCompletareCompact per la cui realizzazione vengono prodotti differenti file posizionati differentemente in base al compito che assolvono. Tutta la logica di recupero ed invio delle informazioni al modulo della View dell'applicazione viene inserita all'interno del file components.class.php che è situato nella directory action del modulo che stiamo analizzando. Potendo avere diversi component per lo stesso modulo, la logica di ciascuno di essi è definita dentro differenti funzioni che prendono il nome dal compont che utilizzano.

In seguito ERIS passa ad analizzare gli elementi che compongono summary, heading e content creando il component per heading e inserendo nella parte di presentazione del component il contenuto "verbali da completare": questo tipo di component è equivalente

a un partial di symfony perché non fa uso di variabili fra la parte logica e quella di presentazione.

Realizzato il component dell'elemento content, individua al sul interno un comando XPath per cui analizza la funzione ed effettua il conteggio dei verbali: per far ciò controlla se esiste un filtro nell'elemento summary con i relativi metodi utili a reperire le informazioni dal concetto verbale attraverso quel filtro (in caso contrario ERIS si occupa di scrivere i metodi utili al component per effettuare le sue operazioni), successivamente viene scritta la logica interna al component analizzato da ERIS e i metodi che permettono di reperire le informazioni utili al component.

ERIS genera alla fine il template del frammento dove verrà inserito il contenuto "testuale" dell'elemento content, includendovi una variabile utilizzata per ricevere l'informazione elaborata dal controller, a questo punto, ha analizzato e sviluppato correttamente la logica degli elementi heading e content,includendoli nel componente summary.

Possiamo, così, notare che ogni elemento interno alle view è, per ERIS, un component: per poterlo individuare in caso di riutilizzo si ricorre al modulo in cui è inserito, mediante il nome della view e il tipo di tag che lo descrivono (questo procedimento avviene per tutti i sottoelementi delle view, esclusi i form).

ERIS utilizza il framework di Symfony per gestire i form del sistema. Scrivere i form è uno dei compiti più complessi di un sistema, ERIS dovrebbe occuparsi di scrivere i form, implementare le regole per la validazione di ogni campo, processare i valori per salvarli nel database, visualizzare i messaggi d'errore, ripopolare i campi in caso d'errore, ecc.

Attraverso l'utilizzo del framework di Symfony molte di questi processi sono automatizzati, Symfony gestisce i form attraverso tre parti fondamentali e distinte:

- validazione: il framework mette a disposizione diverse classi per la validazione dell'input (email, stringa, intero, booleano, ecc..)
- widget: il framework mette a disposizione delle classi per l'output html dei campi (select, textarea, checkbox)
- form: rappresenta i form costituiti da widget e validatori mettendo a disposizione dei metodi per la loro gestione. Ogni campo di un form in symfony ha il suo validare e il suo widget.

Un form in Symfony non è altro che una classe costituita da campi ed ogni campo ha un nome, un validatore e un widget.

ERIS fa uso del framework dei form di symfony in due occasioni:

- quando inizializza i concetti del documento xml
- quando individua l'elemento form all'interno delle view del file xml.

Quando inizializza i concetti, attraverso il comando:

```
php symfony doctrine:build --forms
```

ERIS, fa uso del framework di symfony per serializzare i form per il database in quanto questo conosce il modello del database e lanciando questo task genera automaticamente le classi del form nella cartella lib/form per tutte le tabelle interne al sistema.

Quando individua l'elemento fom all'interno delle view del file xml, oltre a realizzare i component, ERIS va a controllare che le classi per la gestione del form esistano. Se ciò non avviene, ERIS si occupa di realizzare il form per il concetto descritto come attributo dell'elemento nel file xml, personalizzandolo in base alle operazioni e alle sottoparti descritte nel documento xml.

4.2.4 L'interfaccia

Durante la fase di analisi del documento xml, e contemporaneamente allo sviluppo dei moduli che rappresentano il Controller, avviene la creazione dei moduli che rappresentano la Vista (MVC) dell'applicazione finale.

Come Vista, ERIS definisce lo strato che rappresenta ciò con cui l'utente interagisce. Come accennato, le cartelle action e template vengono create durante la creazione dei moduli nell'analisi delle View del documento xml.

All'interno della cartella template, vengono creati tutti quei file utili al sistema per rendere fruibili all'utente le informazioni rinvenute dal controller; durante l'analisi degli elementi sottostanti la view del documento xml, vengono realizzate le parti presentazionali di ogni component nella cartella templates.

Facciamo un esempio prendendo in considerazione il frammento xml precedentemente visto:

Durante l'analisi del frammento, una volta sviluppata la logica di controllo dell'elemento summary del documento xml, viene sviluppata la logica di Vista del modulo stesso; per far ciò, ERIS realizza la parte View del component identificata dal tag content inserendovi il contenuto:

```
Sono presenti <?php echo $verbalecount; ?> verbali da completare.
```

Al suo interno la variabile $verbalecount, per esempio, assume i valori che il controller del component content gli passa.

Di seguito mostriamo una parte della logica del component utilizzata per ottenere le informazioni richieste dalla Vista. Come si puo notare dentro il controller del component summary viene creato una nuova istanza dell'oggetto verbale utilizzata per reperire le informazioni richieste dalla view.

```
function executeSummaryContentverbaleCompletatoCompact(){
     $verbale= new verbale();
     $this->verbalecount = $verbale->getVerbalesCountWithFilter('dataverbale','!=','');
}
```

Tutte le Viste dei moduli utilizzano questo modello di interazione con il controller, in questo modo vengono realizzati tanti component quanti sono gli elementi interni alla vista analizzata nel documento xml.

In seguito alla creazione dei component, ERIS va a costruire la pagina di visualizzazione di ogni modulo ovvero la presentazione nell'interfaccia di una View, raggruppando gerarchicamente i component uno dentro l'altro. Vediamo, ad esempio, il contenuto del file index.php del modulo verbaleDaCompletareCompact:

```
1 include_component('verbaleDaCompletareCompact','
    summaryHeadingverbaleDaCompletareCompact');
2 include_component('verbaleDaCompletareCompact','
    summaryContentverbaleDaCompletareCompact');
3 include_component('verbaleDaCompletareCompact','
    navigationverbaleDaCompletareCompact');
```

Attraverso la funzione include_component è possibile richiamare un frammento interno ad un modulo da qualsiasi altro modulo o parte di esso, basta infatti richiamare correttamente il frammento che si desidera includere specificando alla funzione come parametro il nome del modulo in cui è salvato il frammento e il nome del frammento.

Visualizzando la cartella templates (modulo Vista, MVC) contenuta nei moduli che rappresentano gli elementi View del documento xml, si nota che le Viste realizzate dalla composizione di frammenti che rappresentano gli elementi figli dell'elemento di riferimento. Ogni figlio è composto ricorsivamente dalla composizione dei frammenti degli elementi che lo compongono e così via...

Per facilitarne la comprensione, mostriamo l'architettura del modulo verbaleDaCompletareCompact realizzata da ERIS.

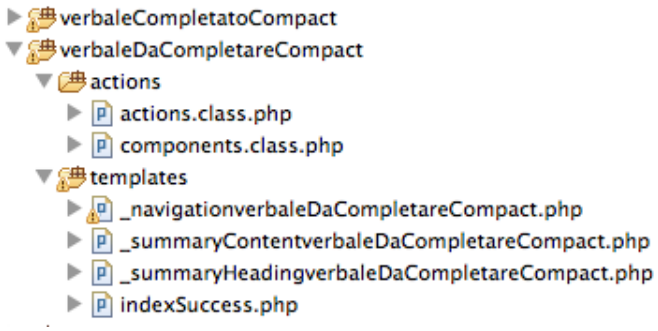

Figura 4.8: Architettura generata da ERIS della View *verbale da completare compact*

ERIS analizza e genera il codice dell'elemento View *verbaleDaCompletareCompact* come in figura.

Nell'immagine viene mostrato il modulo verbaleDaCompletareCompact; al suo interno vengono realizzate da ERIS la cartella actions (il controller del modulo) e la cartella template (che rappresenta il modulo di View).

Per ogni elemento che compone la View nel documento xml viene creato un compoment suddividendone la parte di logica (inserita all'interno del documento component.class.php) da quella di presentazione (viene creato un file php in cui all'interno viene inserito il contenuto dell'elemento analizzato nell'xml); per ogni View dell'xml esiste un file file index.php responsabile della presentazione dei contenuti nell'interfaccia dove al suo inteno vengono aggregati i frammenti che lo descrivono.

Tutti i frammenti creati da ERIS creano dei template. I template in Symfony, sono la rappresentazione dei dati elaborati dal controller in formato html (la Vista). Di solito i template non vengono sviluppati dai programmatori ma bensì dai grafici, per questo ERIS, nella loro creazione cerca di inserirvi meno codice php possibile.

Symfony gestisce i template insieme al concetto di Layout che è quella parte della Vista dell'applicazione finale che contiene il codice html comune a tutte le pagine.

Di fatto in qualsiasi applicazione web si può notare una quantità di codice ripetuta diverse volte in diverse pagine, per cui è efficiente implementare un container globale e inserirvi al suo interno unicamente le parti delle pagine variabili, come mostrato nella figura successiva:

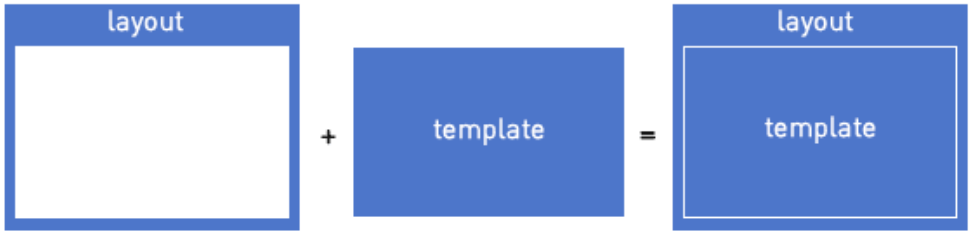

Figura 4.9: Layout + template

Il container in Symfony è un file chiamato Layout.php salvato dentro la cartella template dell'applicazione; l'immagine sopra mostra la composizione di una pagina web attraverso la composizione fra layout e template (il primo identico per tutte le pagine e il secondo definito in base al modulo di View dove si trova).

Successivamente mostriamo un esempio della View verbaleList dell'applicazione ALMA realizzata da ERIS attraverso il framework Symfony:

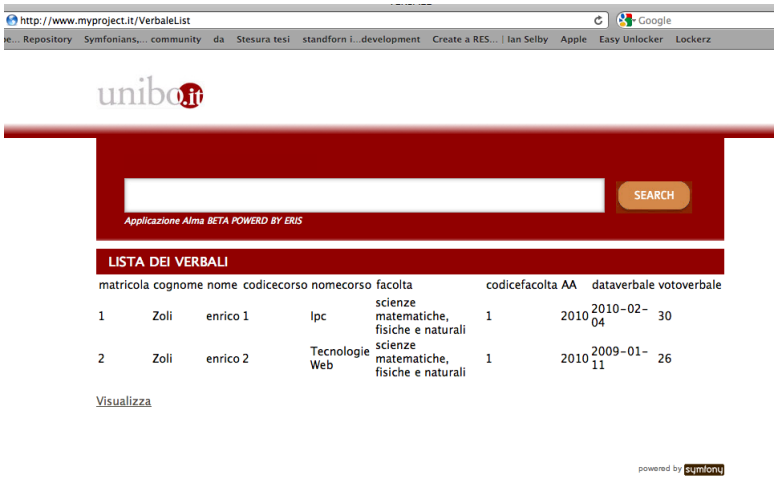

Figura 4.10: View dell'applicazione

Capitolo 5

Valutazioni

5.1 Il Test di ERIS

Per valutare il funzionamento di ERIS si è deciso di realizzare un'applicazione chiamata ALMA che rappresenta alcune funzioni del modulo di registrazione degli esami di un'università.

ALMA è stata descritta in un documento xml identico a quello prodotto in output dall'applicazione *CAO=S Analyzer*. Il documento xml è stato costruito ad hoc perchè l'applicazione di analisi non è stata ancora implementata.

I Concetti di ALMA descritti nelle strutture del documento xml sono i seguenti: verbale, studente, docente, insegnamento, corso di laurea, facoltà, dipartimento, registro e lezione. Le relazioni fra i concetti e gli elementi che li caratterizzano sono stati pensate per poter creare complicazioni al sistema.

Le strutture View di ALMA prese in cosiderazione ci permettono di controllare se ERIS analizza correttamente gli elementi che descrivono la lista, il form, i sommari, i contenuti e se applica correttamente ad essi, i filtri descritti all'interno degli elementi e delle strutture dati.

L'obiettivo del test di ALMA è quello di controllare il corretto funzionamento della:

- Creazione del modulo di persistenza dei dati
- Creazione del modulo di logica dell'applicazione
- Creazione dell'interfaccia

Dando in input il documento xml ad ERIS si vede subito che l'applicazione funziona correttamente.

La creazione del modulo di persistenza è avvenuta correttamente, infatti sono state controllate tutte le tabelle che rappresentano i concetti del file xml, verificando che siano state generate e inizializzate correttamente nel database. Anche le relazioni fra i concetti sono state mappate correttamente.

Successivamente sono stati controllati il modulo di logica e l'interfaccia, generati correttamente attraverso la creazione dei moduli interni al sistema che rappresentano le View del documento xml.

Di seguito mostriamo la View *lista di verbali* generata da ERIS e una View per la creazione di un'istanza del Concetto *studente*:

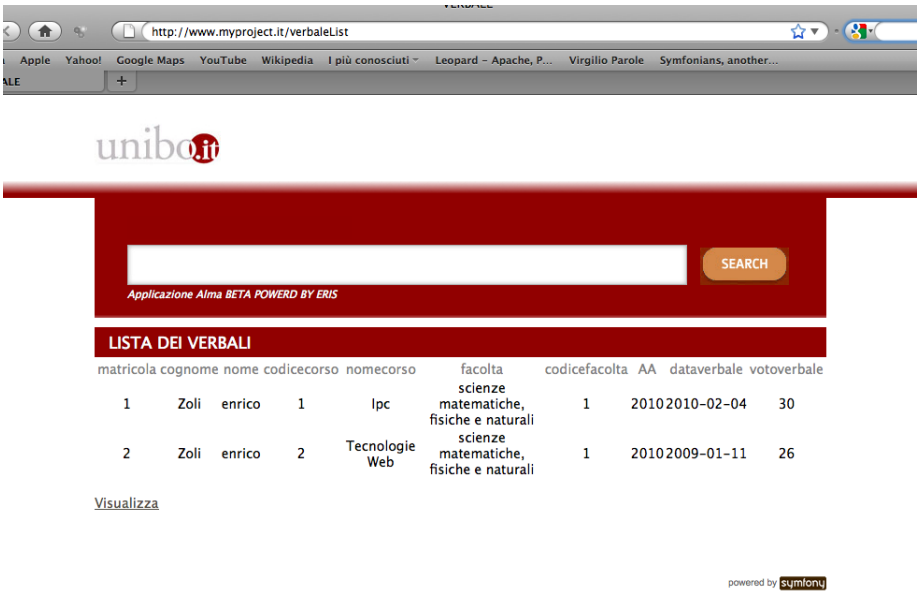

Figura 5.1: View lista di verbali

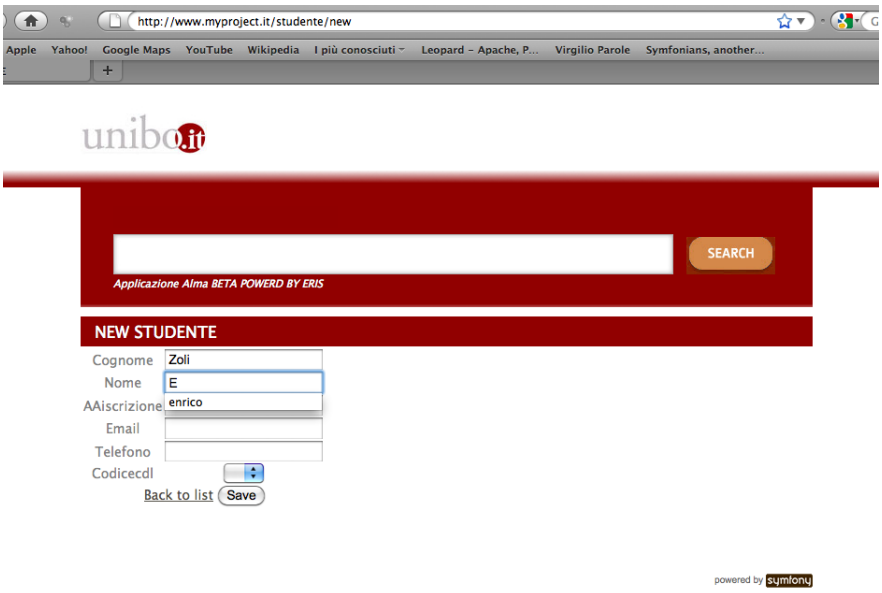

Figura 5.2: View studente, operazione Create

Di seguito effettuiamo una valutazione qualitativa del sistema realizzato che ci permette di individuare i miglioramenti apportati da ERIS rispetto allo sviluppo tipico di un'applicazione web.

5.2 Valutazione qualitativa

Anche se ERIS per ora sviluppa solo le funzionalità di base, si possono già identificare i vantaggi e i miglioramenti riscontrati attraverso il suo utilizzo.

Solitamente, lo sviluppo di un'applicazione web segue le seguenti macro fasi:

- Fase di analisi: dove avviene l'incontro con gli stakeholder, l'analisi dei prodotti esistenti, interviste con i possibili utenti, l'expert usability review, l'expert sociabilty review, ecc. .
- Fase di progettazione: brainstorming, creazioni di modelli e di schemi di navigazione, realizzazione di bozzetti delle schermate e prototipi di bassa fedeltà servono a progettare l'interfaccia.

- Fase di test: dove ogni funzionalità è confrontata con i task realizzati attraverso test sui prototipi dell'interfaccia.

- Fase di implementazione: è la fase che implementa concretamente il prototipo dell'interfaccia.

- Fase di monitoraggio: utilizzata per raccogliere informazioni utili all'evoluzione dell'applicazione, per identificare problemi del sistema ecc.

CAO=S e di conseguenza ERIS, modifica il processo di sviluppo standard automatizzando la fase di progettazione, di test e di implementazione, riducendo fortemente il carico di lavoro del team di sviluppo. Le fasi di un team di sviluppo che fa uso di CAO=S sono:

- Fase di analisi: in questa fase, molto accurata, il team di sviluppo descrive in CAO=S il sistema attraverso i concetti, gli attori e le operazioni. E' la fase fondamentale e decide lo sviluppo dell'applicazione.

- Fase CAO=S: viene utilizzato CAO=S per realizzare automaticamente l'applicazione

- Fase di monitoraggio: utilizzata per identificare problemi, possibili migliorie ecc. .

L'utilizzo di ERIS permette di generare applicazioni in modo automatizzato facendo risparmiare così tempo e soldi a sviluppatori e clienti.

L'esempio utilizzato precedentemente come test di funzionamento (ALMA) è stato una dimostrazione della rapidità dello sviluppo di un'applicazione utilizzando ERIS. Lo sviluppo dello stesso applicativo avrebbe richiesto a programmatori esperti almeno una giornata di lavoro.

CAO=S rende possibile la creazione di applicazioni web e interfacce non solo ad utenti esperti di informatica o programmatori, ma anche a utenti che non hanno conoscenza di linguaggi di programmazione; abbassa inoltre il livello di conoscenza che i programmatori devono possedere per poter sviluppare applicazioni: non richiede particolari configurazioni, non richiede conoscenze di sistemi come middleware o IDE.

ERIS facendo uso del framework Symfony è molto flessibile, ovvero può essere esteso introducendo dei nuovi moduli, plugin e si ha a disposizione una comunità di sviluppatori per la risoluzione di problemi. D'altro canto la scelta di utilizzare un framework può

rallentare la modifica delle applicazioni: se non si conosce il framework e ci si addentra per la prima volta nella sua struttura, la curva di apprendimento risulta essere alta, questo può sembrare un difetto ma una volta compreso il suo funzionamento e la sua struttura, le eventuali modifiche internet al sistema risulteranno estremamente semplici e rapide.

I sistemi di generazione semi-automatica delle interfacce utilizzano linguaggi di modellazione o modelli discorsivi per poter implementare l'interfaccia richiedendo agli sviluppatori la conoscenza di linguaggi e strumenti come: UML, ECLIPSE, etc. . A differenza di questi, un sistema che implementa completamente CAO=S, permetterebbe di realizzare *automaticamente* applicazioni web senza l'ausilio di conoscenze aggiuntive.

Di seguito elenchiamo una serie di caratteristiche positive di ERIS:

1. paradigma MVC
2. conformità alle specifiche
3. usabilità ed accessibilità
4. utilizzo delle tecnologie moderne (AJAX, REST)
5. riduzione degli errori di sviluppo del codice
6. Aumenta la leggibilità e la chiarezza dichiarativa del codice (uso di costanti, scelta dei nomi per le costanti, variabili, funzioni, ecc.)

Attraverso le considerazioni appena fatte, possiamo considerare il sistema come un'ottimo strumento per i progetti a budget limitato, permettendo a un team di sviluppo di applicazioni, senza conoscenze di usabilità, di evitare comuni errori nella progettazione di applicazioni usabili.

Capitolo 6

Conclusioni

Oggigiorno l'usabilità è una caratteristica troppo importante per essere tralasciata.

Abbiamo visto l'utilizzo dei modelli di progettazione user-centered non colma definitivamente la distanza che c'è tra il modello mentale del progettista e il modello mentale dell'utente.

A questo scopo abbiamo ideato un modello di sviluppo innovativo chiamato CAO=S che permette di realizzare applicazioni usabili per progetti con budget limitato e che non richiede al team di svilippo conoscenze di usabilità. Come abbiamo visto, CAO=S modifica il tradizione modello di sviluppo incentrato sull'utente attraverso la parametrizzazione dell'analisi dei requisiti del modello goal-oriented, riducendo al minimo le caratteristiche degli attori con l'introduzione del concetto di ruolo e facendo uso di linee guida e pattern di design.

Attraverso questo modello si riescono a produrre strutture dati analizzando i tre componenti chiave che sono gli attori, le operazioni e i concetti.

Presentando l'architettura di CAO=S si è tentato di dare una soluzione principale al problema della generazione automatica delle interfacce. È stato quindi implementato ERIS che rappresenta il modulo di generazione automatica dell'applicazione nel modello CAO=S e permette attraverso l'acquisizione di un documento xml di sviluppare il modulo della persistenza, il modulo di logica dell'applicazione e l'interfaccia utente.

La maggior parte dei sistemi che attualmente generano interfacce automaticamente, basano lo sviluppo dell'interfaccia interpretando dei modelli realizzati da sviluppatori attraverso linguaggi di modellazione (UML).

Con CAO=S abbiamo tentato di eliminare questa fase automatizzandola il più possibile evitando così che un team di sviluppo debba avere competenze di modellazione ed abbassando il dispendio di tempo nella fase di analisi e della realizzazione dell'architettura. Infatti con questo sistema, un team di sviluppo non ha bisogno di modellare nessuna struttura ma solamente di analizzare le informazioni utili per la realizzazione dell'applicazione.

Tramite il test ALMA (una piccola applicazione che rappresenta alcune funzioni del modulo di registrazione degli esami di un'università) abbiamo mostrato che ERIS riesce a generare correttamente i moduli di persistenza dei dati, della logica dell'applicazione e l'interfaccia in tempi rapidissimi.

Per migliorare CAO=S andrebbe implementata l'applicazione CAO=S Analyzer che permetterebbe attraverso dei questionari compilati da un team di sviluppo di analizzare i componenti fondamentali di CAO=S (Concetti, Attori, Operazioni) e di produrre il documento xml che rappresenta le strutture da dare in input ad ERIS.

È necessario comunque comprendere che dal punto di vista architetturale ci sono ancora ampi margini di miglioramento.

Bisognerebbe infatti ampliare e formalizzare lo schema del documento xml utilizzato come scambio di dati permettendo in questo modo ad eventuali altre applicazioni di interfacciarsi con il sistema. Inoltre, andrebbero apportate migliorie all'aspetto grafico attraverso due possibili direzioni:

- estendere CAO=S permettendo di inserire anche dei dati relativi alla formattazione grafica
- fornire come input dei templates di CSS.

Come ultimo sviluppo futuro, bisognerebbe migliorare ERIS, perfezionando l'applicativo sviluppato e ampliandolo con un modulo Client Side per la gestione di tecnologie quali AJAX.

Bibliografia

[ALB08] R. Allen, R. Lo, and S Brown, *Zend Framework in Action*, Manning Pubn; 2008.

[AND99] K. Andrews, *Human-Computer Interaction Lecture Notes*. Final Version of 13 July 1999, http://www.iicm.edu/hci/

[ANU08] K. Anuja, *Object Relational Mapping*, 4 Feb 2008 http://hdl.handle.net/123456789/71

[AP99] M. Abrams, C. Phanouriou, "UIML: An XML language for building device-independent user interfaces", *In Proceedings of the XML 99*, 1999

[AYK05] N. Aykin, *Usability and internationalization of information technology*, Routledge, 366 pagine, 2005

[BFK08] Cr. Bogdan, J. Falb, H. Kaindl, S. Kavaldjian, R. Popp, H. Horacek, E. Arnautovic, A. Szep, "Generating an Abstract User Interface from a Discourse Model Inspired by Human Communication", hicss pp.36 *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, HICSS, 2008

[BHL94] F. Bodart, A. Hennebert, J. Leheureux, I. Provot, B. Sacré, J. Vanderdoncktj "A Model-based Approach to Presentation: A Continuum from Task Analysis to 18 Computer-Aided Design of User Interfaces Prototype", in *Proceedings of 1st Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'94,* (Bocca di Magra, 8-10 June 1994), F. Paternó (Ed.), Focus on Computer Graphics Series Springer Verlag Berlin, pp. 77-94, 1995

[BHL95] F. Bodart, A. Hennebert, J. Leheureux, I. Provot, B. Sacré, J. Vanderdonckt, "Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide", in *Proceedings of 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95* (Château de Bonas, 7- 9 June 1995), R. Bastide and Ph. Palanque (Eds.), Eurographics Series, Springer Verlag, pp. 262-278. Vienna, 1995 http://www.info.fundp.ac.be/cgi-bin/pub- spec-paper?RP-95-019

[BHN95] H. Balzert, F. Hofmann, C. Niemann, Vom Programmieren zum Generieren, "Auf dem Weg zur automatischen Anwendungsentwicklung", in *Proceedings of GI- Fachtagung Software-technik,* Braunschweig, pp. 126-136, October 1995

[BKF08] C. Bogdan, H. Kaindl, J. Falb, R. Pop, "Modeling of interaction design by end users through discourse modeling", *Proceedings of the 13th international conference on Intelligent user interfaces*, 2008

[BRJ97] G. Booch, J. Rumbaugh e I. Jacobson, *The Unified Modeling Language user guide: Addison-Wesley*, 1997

[BSD01] V. Bullgard, K. Smith, and M. Dacota*, Essential XUL Programming*, Wiley, July 2001.

[CRU10] CakePHP, Retrieved November 12, 2008, from http://cakephp.org/

[CF10] A. M. R. d. Cruz, J. P. Faria, *Automatic Generation of User Interface Models and Prototypes from Domain and Use Case Models*, INTECH, May 2010

[COE04] C. Coenraets, *An overview of MXML: The Flex markup language*, Mar 2004, http://www.adobe.com/ devnet/flex/articles/paradigm.html

[CRC07] A. Cooper, R. Reimann, D. Cronin, *About Face 3: The Essentials of Interaction Design*, John Wiley & sonsm,3rd edition, May 7 2007, ISBN: 978-0470084113

[CRU10] CRUD, Retrieved: 31 May 2010, http://en.wikipedia.org/wiki/Create-,_read,_update_and_delete

[ERT09] D. Ertl, "Semi-automatic multimodal user interface generation", *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, 2009

[FIE00] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, University of California, 2000

[FKH06] J. Falb, H. Kaindl, H. Horacek, C. Bogdan, R. Popp e E. Arnautovic, "A discourse model for interaction design based on theories of human communication". In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2006. ACM Press, pp. 754–759.

[FKP09] J. Falb, S. Kavaldjian, R. Popp, D. Raneburger, E. Arnautovic, H. Kaindl , "Fully automatic user interface generation from discourse models", *Proceedings of the 13th international conference on Intelligent user interfaces*, 2009

[FPR09] J. Falb, R. Popp, T. Röck, H. Jelinek, E. Arnautovic, H. Kaindl, "Fully automatic generation of web user interfaces for multiple devices from a high-level model based on communicative acts", Volume 5, Number 2 / 2009, *International Journal of Web Engineering and Technolog*, 135 - 161

[GCP00] J. Gomez, C. Cachero e O. Pastor, Extending a Conceptual Modelling Approach to Web Application Design. In *CAiSE '00. 12th International Conference on Advanced Information Systems*, volume 1789, pages 79–93. Springer-Verlag. Lecture Notes in Computer Science, 06 2000.

[GWW10] K. Z. Gajos, D. S. Weld, and J. O. Wobbrock. "Automatically generating personalized user interfaces with Supple", *Artificial Intelligence*, 2010.

[JWZ93] C. Janssen, A. Weisbecker, J. Ziegler, "Generating User Interfaces from Data Models and Dialogue Net Specifications", in *Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93* « Bridges Between Worlds » (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollna- gel, T. White (Eds.), ACM Press, New York, pp. 418-423, 1993

[KRF09] S. Kavaldjian , D. Raneburger , J. Falb, H. Kaindl, D. Ertl, "Semi-automatic user interface generation considering pointing granularity", *in Proceedings of the 2009 IEEE international conference on Systems, Man and Cybernetics*, 2009

[KRU01] S Krug, *Don't make me think*, Hops libri, Milano, 2001

[LB02] J. Lee, W. Brent, *Open Source Web Development with LAMP: Using Linux, Apache, MySQL, Perl, and PHP*, Addison Wesley, December 2002

[LGF90] P. Luff, N. Gilbert, D. Frohlich, *Computers and Conversation, Academic Press*, 1990.

[LR01] A. Leff, J. T. Rayfield, "Web-Application Development Using the Model/View/Controller Design Pattern", *in Fifth IEEE International Enterprise Distributed Object Computing Conference*, pp.0118, 2001

[LR94] C. Lewis, J. Rieman, *Task-centered user interface design, a practical introduction,* 1994

[MEY06] B. Meyer, "Dependable Software". *In Dependable Systems: Software, Computing, Networks*, Springer Berlin / Heidelberg, Lecture Notes in Computer Science vol. 4028, 2006, pp. 1-33.

[MOL04] P.J. Molina: "User interface generation with olivanova model execution system", in *IUI '04 9th international conference on Intelligent user interfaces*, pp. 358–359. ACM, New York, NY, USA (2004)

[MPS03] G. Mori, F. Paternò, C. Santoro, "Tool Support for Designing Nomadic Applications" in *Proceedings of IUI*, Miami, Florida, January 12-15, 2003.

[MPS04] G. Mori, F. Paternò, C. Santoro, "Design and Development of Multidevice User Interfaces through MultipleLogical Descriptions" in *IEEE Transactions on Software Engineering*, pp.507-520, August 2004

[MT88] W. C. Mann e S. A. Thompson, *Rhetorical Structure Theory: Toward a functional theory of text organization.* Text, 8(3): 243–281, 1988.

[NIL93] J. Nielsen, "Usability Engineering", *Academic Press*, Boston, MA. 1993

[NOR90] D. Norman, *La caffettiera del masochista*, Giunti, Milano, 1990

[NOR04] D. Norman, *Emotional design*, Apogeo, Milano, 2004

[OPE10] OpenLaszlo, Laszlo Systems, Inc, 2010, http://www. openlaszlo.org.

[PDO10] PDO, update: Jun 2010 http://php.net/manual/en/book.pdo.php

[PHP08] PHP, Retrieved November 12, 2008, http://www.php.net/

[PPI98] O. Pastor, V. Pelechano, E. Insfran, e J. Gomez, "From Object Oriented Conceptual Modeling to Automated Programming in Java", in *ER '98. International Conference on the Entity Relationship Approach*, pages 183– 196, 1998.

[PRO10] Propel, Object-Relational Mapping (ORM) for PHP5, 2010 http://www.propelorm.org

[PS02] F. Paternò, C. Santoro, "One Model, Many Interfaces*", Proceedings of CADUI 2002*, Valenciennes, France, May 2002

[PS03] F. Paternò, C. Santoro, *A Unified Method for Designing Interactive Systems Adaptable to Mobileand Stationary Platforms* Interacting with Computers 15, pp. 349-366, Elsevier, 2003

[PSS08] F. Paternò, C. Santoro, e A. Scorcia, "User interface migration between mobile devices and digital tv" in *HCSE-TAMODIA '08: Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams*, Berlin, Heidelberg, Springer-Verlag, pages 287–292, 2008

[PSM08] F. Paternò, C. Santoro, J. Mäntyjärvi , G. Mori G., S. Sansone, "Authoring Pervasive MultiModal User Interfaces", *International Journal of Web Engineering and Technology*, Inderscience Publishers, pp.235-261, 2008

[PZ07] F. Potencier and F. Zaninotto, *The Definitive Guide to Symfony*, Apress, 2007.

[SEA69] J. R. Searle, “Speech Acts: An Essay in the Philosophy of Language”, *Cambridge University Press*, Cambridge, England, 1969

[SIL06] Silva, *The ProjectIT Research Program* (whitepaper). Retrieved 05/06/2006 from http://isg.inesc-id.pt/alb/uploads/1/193/pit-white-paper-v1.0.pdf

[SLN92] P. Szekely, P. Luo, R. Neches, “Facilitating the Exploration of Interface De- sign Alternatives: The HUMANOID Model of Interface Design”, in *Proceedings of the Conference on Human Factors in Computing Systems CHI’92 « Striking a balance »*, Monterey, 3-7 May 1992, P. Bauersfeld, J. Bennett, G. Lynch (Eds.), *ACM Press*, New York, 1992, pp. 507-514. http://www.isi.edu/isd/CHI92.ps

[SLN93] P. Szekely, P. Luo, R. Neches, “Beyond Interface Builders: Model-Based In- terface Tools”, in *Proceedings of the Conference on Human Factors in Computing Systems INTERCHI’93 « Bridges Between Worlds »* (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), *ACM Press,* New York, 1993, pp. 383-390. http://www.isi.edu/isd/Interchi-be-yond.ps

[SV03] N. Souchon, J. Vanderdonckt, “A Review of XML-Compliant User Interface Description Languages”, in *Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS’2003* (Madeira, 4-6 June 2003), Jorge, J., Nunes, N.J., Falcao e Cunha, J. (Eds.), *Lecture Notes in Computer Science,* Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391

[SVS06] A. Silva, C. Videira, J.Saraiva, D. Ferreira, R. Silva, “The ProjecIT-Studio, an intefrated environment for development of information systems”, in *Proc. of Second International Conference of Innovative Views of.NET Technologies (IVNET 06),* pages 85-103

[SWM06] B. Schoeller, T. Widmer, and B. Meyer. “Making specifications complete th- rough models”, R. Reussner, J. Stafford, and C. Szyperski editors, “Architecting Systems with Trustworthy Components”, volume 3938 of *Lecture Notes in Computer Science*. Springer-Verlog, 2006.

[TVK09] V. Tran, J. Vanderdonckt, M. Kolp, S. Faulkner, "Generating User Interface from Task, User and Domain Models", *Second International Conference*

on Advances in Human-Oriented and Personalized Mechanisms, Technologies, and Services, centric, pp.19-26, 2009

[WBB10] J. H. Wage, R. S. Borschel, G. Blanco, B. Eberlei, Doctrine, http://www.doctrine-project.org, 2010

[XAL08] Nexaweb: XAL – eXtensible Application Language, http://dev.nexaweb. com/home/us.dev/index.html@cid=1784.html 2008

[XAM08] XAML, "XAML: Extensible application markup language", in *Microsoft Developer Network* (MSDN), 2008. URL http://msdn.microsoft. com/en-us/library/ms747122.aspx.

[YAM09] Yaml 1.2, 01 Oct 2009, http://yaml.org

Elenco delle figure