Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Informatica

Model-driven Usability: alcuni approcci e un prototipo

Tesi di Laurea in Interazione Persona-Computer

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
BEATRICE BACELLI

II Sessione
Anno Accademico 2010-2011

Indice

Capitolo 1

Introduzione

Questo elaborato di tesi si occupa degli aspetti di usabilità nell'interazione degli utenti con le applicazioni, utilizzando le tecniche di generazione model-driven di applicazioni. Vengono esaminati alcuni approcci esistenti e realizzato un prototipo di generatore, che si basa su un modello di design, CAO=S, che integra le caratteristiche descrittive dell'utente che utilizzerà l'applicazione. Con un'analisi accurata, queste caratteristiche vengono usate per curare le modalità di interazione con l'applicazione generata, in modo che abbiano specifiche proprietà di usabilità e adattabilità al contesto di utilizzo.

Le metodologie di sviluppo di applicazioni model-driven si pongono come un supporto agli sviluppatori. Esse consentono di progettare il software in maniera veloce, formalizzando dei modelli e generando automaticamente il codice a partire dal modello. In questi approcci, però, sono carenti i modelli di progettazione dell'interazione dell'utente con l'applicazione da generare. Infatti, gli utenti di un'applicazione hanno caratteristiche differenti, che determinano modi differenti di interagire con essa. Viene quindi richiesta una progettazione dell'interazione che tenga conto di queste caratteristiche. L'interfaccia della medesima applicazione può richiedere cambiamenti sostanziali a seconda di queste caratteristiche. Risulta quindi necessaria un'adattabilità dell'interfaccia che ne faciliti l'utilizzo a ciascun utente coinvolto. Nelle architetture model-driven esistenti l'adattabilità dell'interfaccia spesso viene vista come adattabilità a device diversi, ma non a utenti diversi.

Lo sviluppo di applicazioni a partire da modelli è frutto di due metodi di sviluppo software che si sono evoluti l'uno successivamente all'altro: la Model-driven Engineering (MDE) e la Model-driven Architecture (MDA). MDE è un metodo di sviluppo sofware che si basa sui modelli, che sono astrazioni di un sistema più vicine al

dominio concreto che alle formalizzazioni degli algoritmi. Un modello rappresenta un sistema attraverso un insieme di affermazioni, a sua volta il sistema è un insieme di elementi che interagiscono tra loro. Nella MDE la rappresentazione del sistema da parte del modello deve soddisfare il principio di sostituibilità, che afferma che un modello è detto rappresentante un sistema per un dato insieme di problemi se per ogni problema il modello fornisce la stessa risposta che avrebbe fornito il sistema. Un meccanismo di ulterior astrazione rispetto al sistema è il meta-modello, che definisce solo delle strutture. Un modello definisce uno specifico oggetto (come ad esempio un oggetto nella programmazione object-oriented), il meta-modello rappresenta una definizione astratta e generica del modello stesso (equivalente a una classe nel paradigma object-oriented), quindi un modello può essere conforme a un meta-modello.

L'MDA è una teoria di sviluppo del software basata su un insieme di standard forniti dall'Object Management Group, parte dagli stessi spunti dell'MDE ma l'idea è di rendere un linguaggio di modellazione (come ad esempio UML) un linguaggio di programmazione, piuttosto che un semplice supporto al design.

MDA identifica l'esistenza di due tipi di modelli: indipendente dalla piattaforma (Platform Independent Model, PIM) e specifico per la piattaforma (Platform Specific Model, PSM).

Un Platform Independent Model centra l'attenzione sulla struttura del sistema da un punto di vista indipendente dalla piattaforma su cui sarà implementato. Il punto di vista platform independent mostra le operazioni del sistema nascondendo i dettagli necessari per le specifiche implementazioni. Pertanto un modello PIM è idoneo ad essere usato su diverse piattaforme similari. Un esempio è la descrizione architetturale di un sistema che descrive come le funzionalità sono decomposte in componenti architetturali e relazioni.

Un Platform Specific Model è una descrizione del sistema dal punto di vista di una specifica piattaforma e combina le specifiche del modello PIM con i dettagli di come il sistema usa una specifica piattaforma. La trasformazione da PIM a PSM può permette di ottenere da un modello indipendente dalla piattaforma un'applicazione sviluppata in uno specifico linguaggio.

I vantaggi dell'MDA sono quelli di migliorare la produttività del processo di sviluppo del software e ridurne i costi e i tempi. La Model-driven Architecture, grazie alla notevole diminuzione dello sforzo implementativo, è stata fin da subito considerata come lo strumento in grado di permettere di diminuire i tempi di produzione

del software in modo notevole rispetto ai metodi tradizionali. Il successo è dovuto al fatto che la creazione di un modello è meno problematica dell'implementazione del codice.

Esistono varie tecniche di progettazione delle interfacce incentrate sugli utenti, chiamate User-Centered. La progettazione User-Centered si divide principalmente in due teorie di design, una orientata sui task che l'utente deve svolgere (Task Oriented) e la seconda basata sugli obiettivi dell'utente (Goal-Oriented). Il modello di design Goal-Oriented risulta più efficace per lo scopo di progettare l'interazione orientata agli utenti in quanto, rispetto al modello Task-Oriented mira a soddisfare gli obiettivi personali dell'utente, piuttosto che aiutarlo negli specifici compiti che deve svolgere all'interno dell'applicazione. Gli obiettivi personali analizzati dal modello Goal-Oriented sono, per esempio, l'essere rapido, il non fare troppi errori o il non annoiarsi. In questo modello di design per analizzare l'utente esso viene descritto creando dei personaggi in maniera dettagliata, e vengono creati degli scenari che raccontano come il personaggio interagisce col sistema. I personaggi sono la modellazione astratta dell'utente e vengono usati per sviluppare gli scenari, che descrivono come gli utenti portino a termine i propri obiettivi nell'interazione col sistema. 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.

Il design Goal-Oriented offre uno strumento di progettazione molto efficace, che però richiede una progettazione molto complessa e costosa, poco adatta ad essere utilizzata in team medio-piccoli, dove non sono presenti le competenze e gli strumenti adeguati per una progettazione così accurata. Inoltre, la progettazione con questo modello non fornisce delle soluzioni concrete per lo sviluppo del software. Per questo motivo lo scopo di questo lavoro è quello di realizzare un generatore di applicazioni model-driven a partire da un modello incentrato sugli obiettivi dell'utente, che riduce i costi di sviluppo di strumenti usabili usando tecniche Goal-Oriented.

Vista la complessità del modello Goal-Oriented tradizionale, ne verrà utilizzato uno semplificato, CAO=S, che riduce l'analisi degli utenti alle sole caratteristiche che hanno un impatto importante sull'interazione e senza un'analisi accurata degli obiettivi personali degli utenti.

Le soluzioni che CAO=S utilizza per aumentare l'usabilità delle applicazioni sono agire sulle caratteristiche come l'utilità attesa, la completezza dei contenuti, la comprensibilità del vocabolario (eliminando i termini di difficile interpretazione).

Un cambiamento importante apportato in questo modello di design è il far diventare la fase di analisi degli utenti, task e obiettivi un parametro di progetto e non solo un requisito della progettazione.

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

I *Concetti* rappresentano il modo in cui l'utente percepisce e comprende l'informazione. Si ottengono dall'analisi del dominio che può comprendere anche ambiguità nella definizione delle informazioni, che vengono normalizzati in modo da fornire concetti espressi nel linguaggio dell'utente. Da essi viene definita l'architettura delle informazioni del sistema da realizzare.

Gli *Attori* sono le categorie di utenti 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 descritti tramite le caratteristiche essenziali che hanno un impatto diretto sull'interazione, per esempio: le competenze di dominio, la motivazione ad usare il sistema o le abilità linguistiche. Per questo motivo CAO=S realizza, diversi widget per comporre l'interfaccia e la possibilità di modificare dinamicamente l'interazione dell'utente, sulla base delle sue caratteristiche.

Le *Operazioni* sono le manipolazioni che l'utente effettua sui concetti agendo attraverso l'interfaccia. Sono di quattro tipi: creazione, vista, aggiornamento ed eliminazione, e vengono implementate per l'aggiornamento delle strutture dati interne all'applicazione.

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 viste.

Il *Data Storage* permette il salvataggio dei dati all'interno del sistema in maniera persistente.

CAO=S, oltre a definire un modello per la progettazione User-Centered, può

essere utilizzato per implementare tool per generare automaticamente applicazioni web usabili. L'architettura software di questi tool di CAO=S è concettualmente suddivisibile in due applicazioni distinte.

La prima, *CAO=S Analyzer*, 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 (una per ciascuna componente), che rappresentano l'insieme di tutte le operazioni svolte dagli attori sui concetti, le strutture ottenute 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'Analyzer. Essa genera il codice dell'applicazione, l'interfaccia utente e il modulo di persistenza dei dati automatizzando lo sviluppo dell'applicazione.

Per valutare la qualità del generatore è stato realizzato un modello di applicazione di test che descrive una rubrica di contatti. Dall'esecuzione del test sul prototipo realizzato, Anthropos, si può apprezzare come esso, grazie all'utilizzo dell'architettura model-driven, permette la generazione dell'applicazione, a partire dal modello descritto, in modo automatizzato, semplificando il lavoro dei programmatori e riducendo notevolmente i tempi di sviluppo.

Anthropos è stato sviluppato con una particolare attenzione ai pattern di sviluppo utili per rendere le applicazioni realizzate nel processo di generazione molto dinamiche e scalabili (Javascript e l'architettura Model-View-Controller in particolare) e questo permette la realizzazione di un'interfaccia che si adatti dinamicamente alle caratteristiche dell'utente che la utilizza. Dai test effettuati si può apprezzare come esso realizzi cambiamenti sostanziali nell'interfaccia basandosi solo su poche righe di descrizione delle caratteristiche dell'utente.

Capitolo 2

Il design User-centered e Model-driven

Come già accennato nell'introduzione, il problema in esame in questo elaborato è la mancanza di modelli concreti di progettazione dell'interazione tra utente e applicazioni. Quello che si vuole fornire è un prototipo di generatore automatico di applicazioni che rispettino i principi dell'usabilità e forniscano un'interfaccia adattabile alle caratteristiche dell'utente. Per farlo, ci si avvale del contributo di teorie da due discipline molto differenti nelle scienze dell'informazione. Da un lato per lo studio dell'interazione dell'utente si utilizzano le teorie di progettazione User-Centered e nello specifico ci si focalizza sul Goal-Oriented Design. Dall'altro per la realizzazione di un generatore automatico di interfacce si utilizzano le teorie della Model-Driven Architecture e si esaminano gli approcci esistenti di generatori model-driven di applicazioni con attenzione all'interazione. Queste due teorie verranno integrate assieme realizzando un generatore di applicazioni model-driven basato su un modello di progettazione Goal-oriented.

2.1 Le teorie di progettazione User-centered

Per User-centered design si intende la progettazione orientata agli utenti, basandosi sull'analisi delle loro caratteristiche Andrews [1999]. A differenza della progettazione orientata al sistema, essa tiene in considerazione le caratteristiche dell'utente, che possono essere intellettuali, motivazionali, fisiche e di vari tipi, che vengono usate come parametri di progettazione e non come variabili modificabili per ottenere sistemi meno costosi o più efficienti e mantenibili.

Esistono due principali teorie di design incentrate sugli utenti:

- **Task-Oriented** Lewis and Rieman [1993] è rivolta ad organizzare l'intera progettazione del sistema sull'analisi dei compiti che l'utente deve svolgere nel sistema. A differenza dell'analisi tradizionale dei requisiti, che si occupa di elementi parziali ed astratti dei task, il task-centered design si occupa di task reali, completi e rappresentativi dell'utilizzo del sistema da parte dell'utente.

- **Goal-Oriented** Cooper et al. [2007] mira a soddisfare gli obiettivi dell'utente considerando soprattutto quelli personali, come il non fare troppi errori o non annoiarsi. L'analisi degli utenti in questo modello avviene tramite l'utilizzo di personaggi che rappresentano gli utenti, creando pattern comportamentali ed emozionali. L'interazione dei personaggi con il sistema porta allo sviluppo degli scenari, che descrivono come gli utenti interagiscono con il sistema per portare a termine i proprio obiettivi.

Queste teorie di User-Centered Design vengono applicate sia per la realizzazione dei nuovi prodotti sia per effettuare dei miglioramenti nei sistemi esistenti. L'applicazione di queste teorie, sia nel caso del Task-centered sia nel Goal-oriented, avviene attraverso delle fasi che verranno seguite anche nel prototipo che si intende realizzare. Queste fasi sono: l'analisi di task e utenti, la progettazione, la valutazione, l'implementazione e il monitoraggio.

Se si sta valutando un sistema esistente, in fase di analisi vengono effettuate queste verifiche:

- **L'expert usability review** consiste nell'analisi delle interfacce da parte di esperti di usabilità. Attraverso l'individuazione di task-flow mira a determinare ciò che causa difficoltà agli utenti nello svolgimento dei 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 vengono analizzati anche quelli relativi all'interfaccia, facendo uso di linee guida, standard e pattern.

- **L'expert sociability review** è uno studio degli aspetti di interazione sociale degli utenti, rivolta a sistemi online. Mira a rilevare i fattori che promuovono la creazione, il mantenimento e l'evoluzione delle comunità di utenti. Si valutano gli strumenti messi a disposizione dal sistema come: la gestione della privacy, i fattori che promuovono le relazioni interpersonali e la fornitura di strumenti per

la comunicazione sociale. Lo scopo di questa analisi è favorire i comportamenti che portano all'aggregazione e alla condivisione della conoscenza.

Nello specifico le fasi si compongono nel modo seguente:

- **Analisi di Task e Utenti:** In questa fase di analisi si incontrano gli stakeholder, cioè tutte le persone coinvolte nel sistema, per capire vincoli e aspettative. Si osservano i prodotti esistenti, si intervistano i potenziali utenti, si creano profili utente ed elenchi di task e scenari.

- **Progettazione:** 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.

- **Valutazione:** Si compie attraverso test con utenti, questionari e l'analisi di euristiche e ispettive. In questa fase 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.

- **Implementazione:** consiste nella fase di modifica del sistema esistente oppure di realizzazione di uno nuovo.

- **Monitoraggio:** si monitora il nuovo sistema attraverso l'analisi delle segnalazione di problemi, di questionari e studi sul campo, per aiutare gli sviluppatori a raccogliere informazioni da utenti non sottoposti precedentemente a controllo. Lo scopo è quello di identificare problemi da risolvere o migliorie da sviluppare eventualmente.

2.2 La progettazione Goal-Oriented

Dall'analisi dei principi generali della progettazione User-centered il modello di design Goal-Oriented Cooper et al. [2007] risulta più efficace per la realizzione del prototipo in oggetto, in quanto esso ha come obiettivo la progettazione dell'interazione costruita ad-hoc per gli utenti. Infatti, i modelli Goal-Oriented rispetto a

quelli Task-centerd mirano a soddisfare gli obiettivi dell'utente, piuttosto che gli specifici compiti che deve svolgere all'interno dell'applicazione.

Come già detto la progettazione Goal-Oriented si focalizza sugli obiettivi personali dell'utente, il quale non viene visto come un operatore ma come una persona reale. L'enfasi non va quindi posta sui compiti da svolgere ma sul fornire un'esperienza gradevole all'utente. Questa gratificazione si realizza tenendo conto dei reali obiettivi dell'utente, che possono essere classificati come:

- Obiettivi di esperienza: come voglio sentirmi nell'usare il prodotto (livello viscerale);
- Obiettivi di fine: come voglio adoperare il prodotto (livello comportamentale);
- Obiettivi di vita: come voglio pensare a me stesso (livello riflessivo).

Il modello Goal-Oriented si incentra sulla accurata descrizione dell'utente. Per fare questo definisce Personaggi e Scenari e li utilizza nella progettazione, che si svolge nelle seguenti fasi:

- **Ricerca:** Scopo di questa fase è produrre un documento che descriva il lavoro da effettuare, analizzando in dominio dell'applicazione e gli utenti. Vengono definiti il dominio terminologico e tecnologico, si cerca di comprendere la visione e i vincoli sul prodotto e infine si analizzano i prodotti simili esistenti.
- **Modellazione:** Si individua il contesto d'uso attraverso la definizione di Personaggi e Scenari. I Personaggi sono utenti fittizi che incarnano le caratteristiche che vogliamo sostenere ed aiutare nel progetto. La cosa importante nel progettare un personaggio è il grado di precisione nella descrizione, non si tratta di scrivere un elenco di caratteristiche ma una descrizione narrativa completa di una specifica persona con un'età, un lavoro, degli interessi e degli obiettivi.

  Ogni progetto ha un cast da 3 a 12 personaggi. Alcuni personaggi sono potenziali utenti, altri sono potenziali non-utenti. Ogni cast contiene un personaggio principale, il protagonista, che va assolutamente soddisfatto al 100%, è colui il quale non può essere soddisfatto dalle interfacce preparate per gli altri, ma viceversa sì. Diventa quindi il punto nodale del progetto. I personaggi secondari svolgeranno storie di contorno e permetteranno di dettagliare aspetti

dell'interfaccia non completamente gestiti dal protagonista. Incentrare la progettazione di un personaggio descitto in questo modo permette di focalizzare la progettazione sul soddisfacimento degli obiettivi degli utenti connessi con la propria soddisfazione di vita, e non con lo svolgimento dei compiti.

- **Definizione dei requisiti:** Vengono definiti gli scenari dei contesti d'uso, tramite la creazione di storie che esprimono delle esperienze dei personaggi. Lo scopo è individuare le funzionalità e le capacità richieste al prodotto.
- **Design del sistema:** Viene definita la struttura dell'interfaccia, il mddo in cui gli elementi che la compongono esprimano le informazioni e implementino le funzionalità. Per farlo vengono utilizzati gli scenari definiti precedentemente. Il risultato di questa fase sono dei wireframe che rappresentano l'interfaccia.
- **Raffinamento:** L'interfaccia rappresentata nella fase precedente viene trasformata in un prototipo al quale, con diverse iterazioni, vengono man mano aggiunti sempre più dettagli di comportamenti, forma e contenuto. Il prototipo realizzato sarà lo strumento di supporto agli sviluppatori che dovranno attenersi ad esso per lo sviluppo.
- **Supporto:** Per garantire supporto anche in presenza di cambiamenti non previsti in sede di progetto, durante lo sviluppo del sistema vero e proprio si apportano modifiche al prototipo e alla timeline, aggiornando il documento di specifica della forma e del comportamento.

Come si può vedere il design Goal-Oriented richiede una progettazione molto articolata e complessa, sicuramente efficace, ma poco adatta ad essere utilizzata in ambiti dove non vi siano grosse risorse da mettere in campo per la realizzazione di un software. Per questo motivo verrà utilizzato un modello Goal-Oriented semplificato, CAO=S, che riduce l'analisi degli utenti alle sole caratteristiche che hanno un impatto effettivo sull'interazione e non fa un'analisi accurata degli obiettivi personali degli utenti.

2.3 Model-Driven Architecture

Per realizzare lo scopo di fornire un supporto concreto agli sviluppatori è stato scelto di sviluppare un generatore di applicazioni seguendo i principi della Model-Driven Architecture. MDA Schmidt [2006], Atkinson [2003] si presta a questo scopo

perché si occupa della realizzazione del software in modo automatico. Infatti verrà utilizzata per generare l'applicazione a partire da un modello definito in modo da gestire gli aspetti di usabilità.

Per descrivere la Model-driven Architecture è necessario partire dalla Model-driven engeneering. MDE è un metodo di sviluppo sofware che ha l'intento di gestire la complessità delle piattaforme software e i relativi problemi. Si basa sui modelli, che sono astrazioni di un sistema più vicine al dominio concreto che alle idee e agli interessi degli algoritmi. Un modello rappresenta un sistema attraverso un insieme di affermazioni, a sua volta il sistema è un insieme di elementi che interagiscono tra loro. In MDE la rappresentazione del sistema da parte del modello deve soddisfare il principio di sostituibilità, che afferma che un modello è detto rappresentante un sistema per un dato insieme di problemi se per ogni problema il modello fornisce la stessa risposta che avrebbe fornito il sistema. Un meccanismo di astrazione ulteriore è il meta-modello che definisce solo delle strutture. Un modello definisce uno specifico oggetto (come ad esempio un oggetto nella programmazione object-oriented), il meta-modello rappresenta una definizione astratta e generica del modello stesso (equivalente a una classe nel paradigma object-oriented), quindi un modello può essere conforme a un meta-modello.

L'MDA è una teoria di sviluppo del software basata su un insieme di standard forniti dall'Object Management Group (OMG), parte dagli stessi spunti della MDE ma l'idea è di rendere un linguaggio di modellazione (come ad esempio UML) un linguaggio di programmazione, piuttosto che un semplice supporto al design. Non è un approccio innovativo, ha infatti antenati di rispetto come: gli schemi dei database, i tool CASE, le interfacce WYSWYG.

I vantaggi della MDA sono quelli di migliorare la produttività del processo di sviluppo del software e ridurne i costi e i tempi. La Model-driven Architecture, grazie alla notevole diminuzione dello sforzo implementativo, è stata fin da subito considerata come lo strumento in grado di permettere di ridurre i tempi di produzione del software in modo notevole rispetto ai metodi tradizionali. Il successo è dovuto al fatto che la creazione di un modello è meno problematica dell'implementazione del codice.

L'approccio MDA è composto da quattro tipi di modelli: Computation Independent Model, Platform Independent Model, Platform Specific Model e Platform Model. Il codice finale è generato partendo dal Platform Specific Model.

Un *Computation Independent Model* si focalizza sul sistema e sul suo ambiente;

i dettagli strutturali del sistema sono nascosti o non ancora determinati. E' un modello indipendente da come il sistema verrà poi implementato, in cui si presenta cosa ci si aspetta dal sistema e come si comporta nell'ambiente. Un esempio di CIM sono gli Use Case Model.

Un *Platform Independent Model* centra l'attenzione sulla struttura del sistema da un punto di vista indipendente dalla piattaforma su cui sarà implementato. Il punto di vista platform independent mostra le operazioni del sistema nascondendo i dettagli necessari per le specifiche implementazioni. Pertanto un modello PIM è idoneo ad essere usato su diverse piattaforme similari. Un esempio è la descrizione architetturale di un sistema che descrive come le funzionalità sono decomposte in componenti architetturali e relazioni tra di esse.

Un *Platform Specific Model* è una descrizione del sistema dal punto di vista di una specifica piattaforma e combina le specifiche del modello PIM con i dettagli di come il sistema usa una specifica piattaforma.

Un *Platform Model* fornisce una serie di concetti rappresentanti le parti che compongono la piattaforma e i servizi offerti dalla stessa. Inoltre offre una serie di concetti che rappresentano i diversi elementi da usare nel PSM per specificare l'uso della piattaforma da parte del sistema.

Il processo di trasformazione permette di passare da un modello ad un altro. La trasformazione più interessante è quella da PIM a PSM che permette di ottenere da un modello indipendente dalla piattaforma, un modello specifico sulla piattaforma considerata. Al fine di eseguire la trasformazione è necessario avere anche il Platform Model che spesso è solo in forma di manuali hardware o software o nelle conoscenze di chi effettua la conversione. Esistono diversi approcci alla trasformazione da PIM a PSM tra cui:

marking: un mark è un concetto nel PSM applicato ad un elemento del PIM che indica come questo elemento deve essere trasformato; questo processo di trasformazione passa attraverso la costruzione di un modello PIM marcato dal quale applicando le regole di mapping si ottiene il PSM;

metamodel transformation: si esprime il modello PIM attraverso un metamodello, si specifica la trasformazione per la piattaforma scelta in termini di mapping tra metamodelli e con questa si produce il metamodello del PSM;

model transformation: il processo è simile al precedente ma viene preso in considerazione un modello PIM composto da elementi che sono sotto tipi di quelli

Platform Independent, si utilizza una specifica trasformazione tra tipi e si genera un modello PSM;

pattern application: si identificano dei pattern nel modello PIM che, attraverso regole di mapping più specifiche delle precedenti, vengono trasformati in altri pattern nel modello PSM; anche in questo caso può essere generato un modello PIM marcato;

La generazione del modello PSM può richiedere anche l'applicazione iterativa dei pattern e delle regole di trasformazione, in cui ad ogni stadio del processo, il PSM generato diventa un PIM per lo stadio successivo. L'ultima trasformazione che produce il codice a partire dal modello PSM avviene invece in modo quasi automatico. Il principale vantaggio dell'approccio MDA nella progettazione software rispetto al classico modello "waterfall è nel concentrarsi maggiormente sulla definizione formale di cosa il sistema deve fare piuttosto che sui dettagli implementativi. Inoltre permette di costruire un modello del sistema riutilizzabile anche su piattaforme al momento della progettazione sconosciute.

Per via di questi vantaggi questa tecnologia è l'ideale per gli scopi del progetto che si vuole realizzare, difatti si vedrà come il prototipo implementerà un generatore di applicazioni basandosi sul modello definito secondo i principi del design User-centered.

2.4 La generazione automatica di interfacce

Lo sviluppo software Model-Driven consiste in un approccio basato sui modelli. La progettazione del software avviene a livello astratto utilizzando un modello, a partire dal quale può venire generato in maniera automatica il codice sorgente. Questo approccio permette allo sviluppatore di astrarsi dallo specifico linguaggio di programmazione, e porre l'enfasi sugli aspetti più importanti della progettazione come l'interazione, spesso scorrettamente tralasciati.

Negli ultimi anni si è sviluppato l'utilizzo degli approcci model-driven per fornire supporto alla modellazione di sistemi secondo criteri di usabilità.

Uno dei primi approcci è Ceri et al. [2002], si tratta di uno studio in cui viene realizzato un modello e un ambiente di sviluppo che genera delle applicazioni web focalizzandosi sull'aspetto comunicativo del web. Il lavoro presta attenzione agli aspetti di navigazione e presentazione dei contenuti web, ma non viene comunque

considerato l'aspetto dell'usabilità. Il design parte da una specifica concettuale dei contenuti, gli aspetti e la navigazione desiderati.

L'usabilità è un aspetto di particolare interesse da diversi anni, e sin da subito sono stati adottati diversi sistemi di valutazione e verifica degli usability requirements. Fino al 2006 la principale mancanza in questo ambito è stata l'impossibilità di verificare gli aspetti di usabilità sin dalle prime fasi di progettazione, ma solo in fase di sviluppo. Per questo motivo il Department of Information Systems and Computation di Valencia si è occupato di studiare un modello di sviluppo basato sul Model Driven Architecture (MDA) Abrahao and Insfran [2006]. Questo framework è stato instanziato nel modello di esecuzione del tool OlivaNova. L'aspetto principale di questo modello è la valutazione ciclica dei requisiti di usabilità effettuata sui modelli concettuali, Platform Indipent Model (PIM), a partire dai quali si svilupperà l'applicazione. Quest'approccio garantisce eleveti livelli di usabilità del prodotto, e riduce in maniera importante gli interventi sul codice in fase id sviluppo. In figura 2.1 viene messo a confronto, il modello concettuale, sulla sinistra e un'istanza concreta, in cui vengono dettagliati i singoli step che vanno dalla raccolta dei requisiti fino all'implementazione. Si può notare anche il non coinvolgimento degli utenti nelle fasi di valutazione, allo scopo di velocizzare questi micro-cicli del processo. Come si può inoltre notare, il modello prevede una fase di compilazione astratta per passare dal PMI all'implementazione, e che si struttura in due passaggi:

1. Viene definito un mapping tra i criteri di usabilità scelti in base ad altri studi in materia, e i concetti astratti di progettazione, come i pattern.

2. Si applicano delle regole di trasformazione per ottenere un'implementazione, o almeno uno scheletro, a partire dai concetti più astratti definiti al punto 1).

In figura 2.2 questi due passagi sono rappresentati con maggior dettaglio e chiarezza. Analizzando nuovamente la prima figura, 2.1, è da notare un'altra particolarità del processo, ovvero la possibilità di intervenire non solo sul modello PIM, ma anche sulle regole di trasformazione, che hanno un impatto importante sull'interfaccia utente ottenuta al termine. Infine è stato proposto OlivaNova ModelMolina [2004], un modello concreto di quanto descritto fin'ora, e propone diverse fasi:

1. Definizione dei goals, cioè del livello dei task e delle caratteristiche di usabilità che il sistema dovrà rispettare. Quest'analisi dev'essere svolta con la visione di ognuno degli stakeholder;

2. Divisione dei goals in caratteristiche più specifiche e facilmente valutabili;

3. Relazione tra le caratteristiche definite al punto (2) e vincoli specifici (dimensioni, range di valori, ecc..)

4. Associazione di metriche di valutazione, sia quantitative che qualitative, come per esempio associare un valore di chiarezza ai messaggi d'errore: un generico Errore: impossibile completare l'operazione avrà una valutazione praticamente nulla, al contrario di un messaggio tipo Impossibile inserire i dati, il campo codice fiscale è obbligatorio

Tutto questo permette al sistema OlivaNova di fornire una serire di segnalazione, e se possibile suggerimenti, sui potenziali problemi di usabilità.

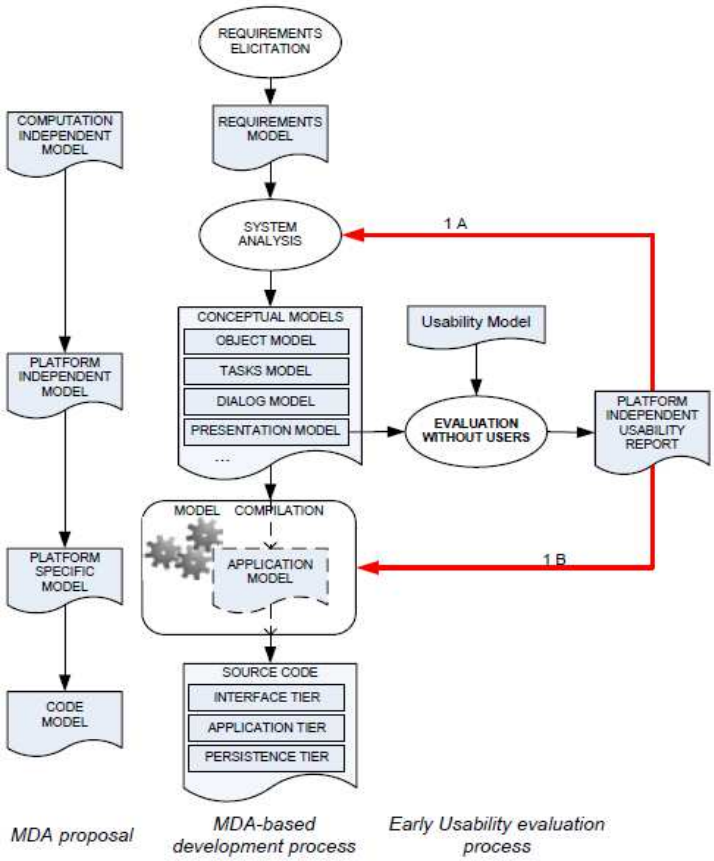

Figura 2.1: *Confronto tra il modello concettuale e un'istanza concreta dettagliata dai singoli step che vanno dalla raccolta dei requisiti fino all'implementazione*

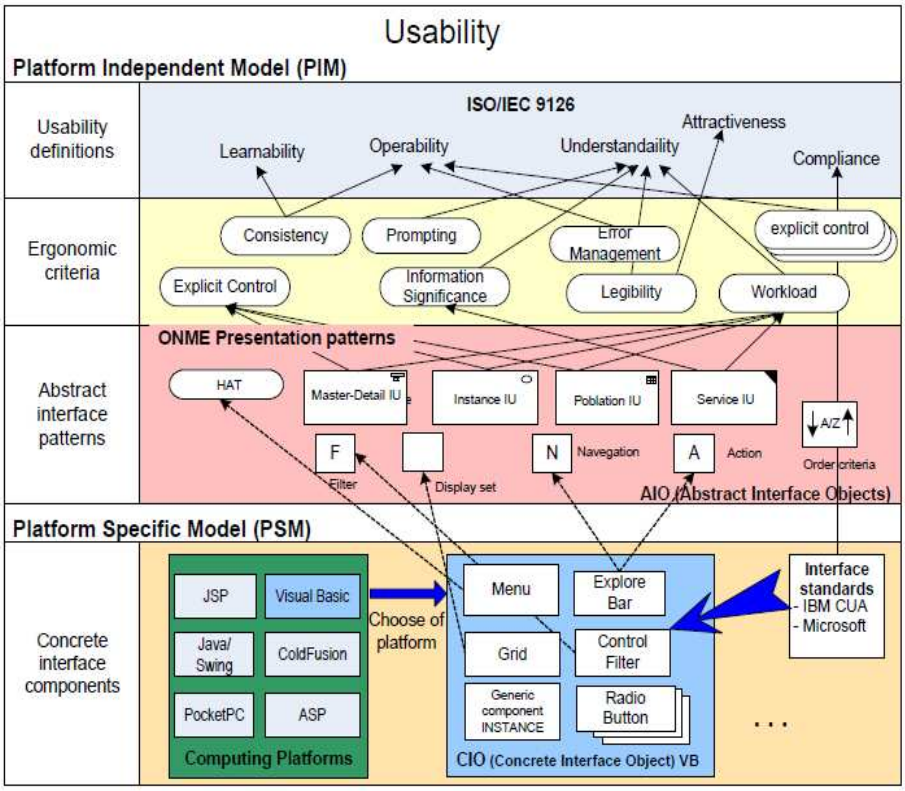

Figura 2.2: *I passaggi nella fase di compilazione astratta per passare dal PMI all'implementazione nel modello realizzato in* **?**

TERESA Paternò et al. [2008b,a] è uno tra i migliori sistemi di generazione automatica di interfacce, studiata principalmente per la modellazione delle applicazioni sulla base del device da cui vengono utilizzate. Il cuore di quest'ambiente multimodale sono i modelli di dialogo, che hanno come riferimento l'analisi della comunicazione umana. La creazione di un'interfaccia passa attraverso 4 fasi:

1. Modello ad alto livello per un'applicazione multi piattaforma;

2. Creazione del modello dell'interfaccia, in grado di descriverne le funzionalità senza entrare nello specifico di come queste devono essere realizzate;

3. Definizione dell'interfaccia tramite concetti astratti;

4. Implementazione dell'interfaccia in uno dei linguaggi supportati da Teresa: XHTML, MP, VoiceXML, X+V, SVG, XLet,...

Entrando nel dettaglio delle fasi sopra riportate, nella prima vengono definite le attività dell'utente e le features che il sistema dovrà fornire per consentire la realizzazione dei task utente. Attraverso il modello ConcurTaskTrees (CTT) viene definita una struttura composta da atti comunicativi messi in relazione gerarchica e temporale tra loro, in grado di descrivere le interazioni tra utente e sistema. A questo livello emerge la necessità di avere un modello più completo, che non solo permetta di definire task ed elementi, ma che a questi associasse i dispositivi su cui è possibile usarli per questo viene definito il CTTE. In figura 2.3 viene evidenziata questa nuova caratteristica, sia per i task che per gli oggetti, mentre in figura 2.4 c'è la maschera che permette di ridefinire il modello in base al device.

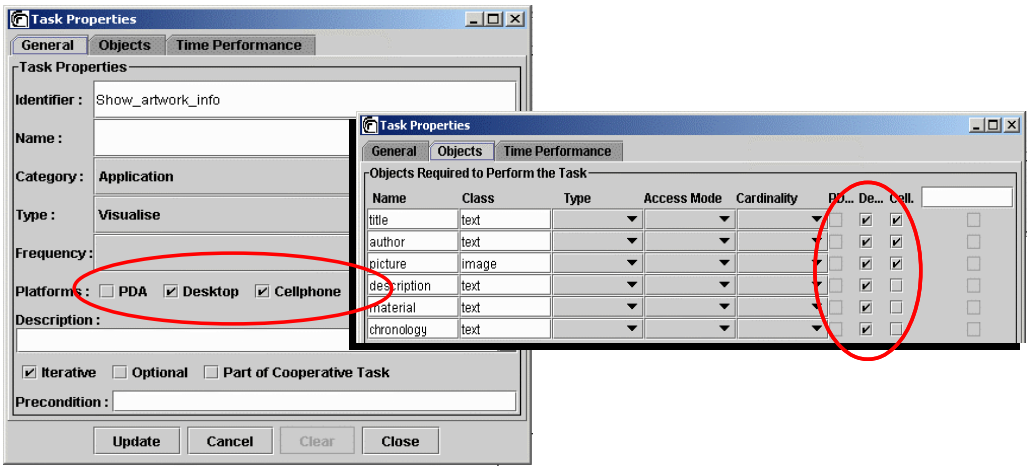

Figura 2.3: *Interfaccia di Teresa modificata in modo da permettere di associasse i dispositivi su cui è possibile usare i task definiti.*

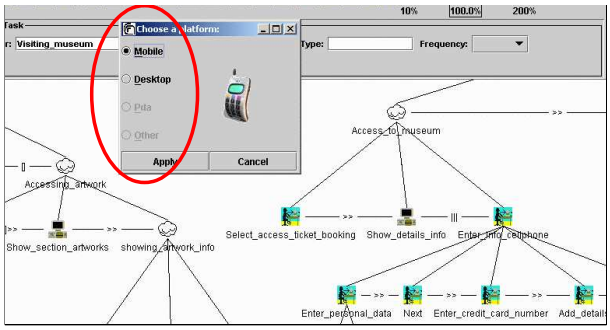

Figura 2.4: *Maschera che permette di ridefinire il modello in base al device in Teresa*

Nella fase successiva vengono creati, sempre ad alto livello, degli elementi tramite

i quali vengono descritti in un modello discorsivo come l'utente manipolerà i componenti dell'interfaccia. Questi elementi sebbene di alto livello, vengono definiti per ogni possibile device, e verranno istanziati con oggetti concreti nella fase di sviluppo.

La terze fase si occupa della definizione astratta dei concetti dell'interfaccia. E' in questo momento che vengono messi in relazione tutti gli elementi, tramite raggruppamenti, ordinamentie, gerarchizzazione e sequenza temporale. Queste caratteristiche forniranno delle regole importanti per l'implementazione, per esempio degli elementi raggruppati potranno essere inseriti all'interno di una stessa tab o fieldset. Il frutto di questa elaborazione è un file XML strutturato appositamente per gestire un modello task-oriented.

La quarta e ultima fase è strettamente legata alla piattaforma software e hardware dell'applicativo. E' in questo momento che vengono introdotte le specifiche di dettaglio di ogni elemento, che possono dipendere dal device (dimensione, posizione,...) o dalla definizione astratta del concetto e dalla relativa semantica. Per esempio un'interfaccia pensata per un netbook avrà dimensione degli elementi differente da una realizzata per smartphone touch screen. Un altro esempio legato alla definizione astratta è la scelta della relazione tra gli elementi che può essere di continuità logica delle operazioni (wizard) o di prossimità, ovvero di vicinanza fisica dettata da una relazione di frequenza oppure di similutidine di contenuto.

Un'idea innovativa di Teresa è l'introduzione di un proxy in grado di lavorare sulla conversione dell'interfaccia in maniera dinamica, permettendo ad un utente di svolgere le proprie attività passando da un device all'altro con continuità. L'interfaccia del client di migrazione è visibile in figura 2.5. Per esempio un utente potrebbe effettuare una ricerca di oggetti da comprare online tramite il proprio smartphone mentre è in autobus, quindi concludere l'acquisto dal pc desktop di casa una volta rientrato, sfruttando così semrpe il miglior device consentito dalla situazione.

Un altro approccio è quello mostrato in Sottet et al. [2008] il quale esamina le interfacce web plastiche, che si modificano adattandosi al contesto d'uso. L'approccio utilizzato è quello del Model Driven Engineering (MDE) che si concentra sulla creazione di modelli di dominio. Il lavoro svolto nella pubblicazione non fornisce alcun tool di sviluppo ma si limita a definire un meta-modello. L'interfaccia utente di un'applicazione viene descritta attraverso un grafo di modelli collegati tra loro. Sono presenti dai modelli concettuali più astratti a quelli più concreti dell'interfaccia finale. Nel grafo le relazioni tra i modelli sono dei "mapping" come definiti in MDE. I "mapping" sono utilizzati sia per collegare modelli astratti in fase di progettazione,

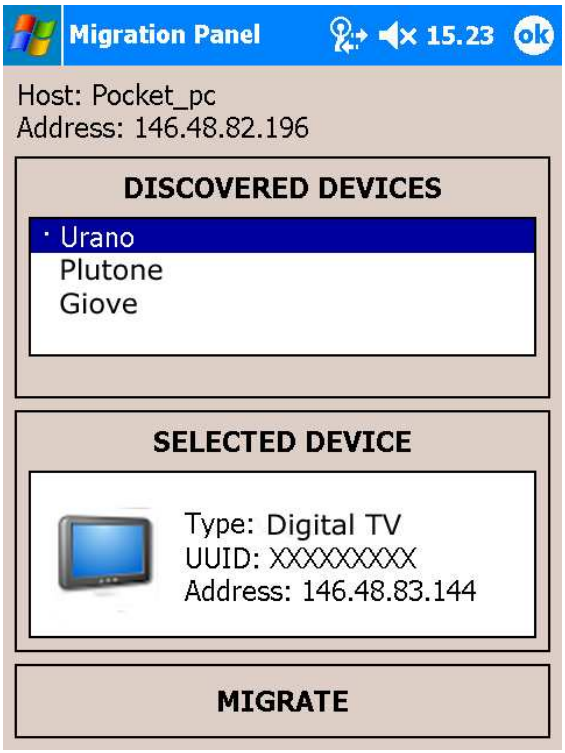

Figura 2.5: *Interfaccia del cambio di device in Teresa*

sia modelli concreti a run-time. In fase di progettazione, la mappatura esprime le proprietà che aiutano il progettista a selezionare le funzioni di trasformazione più appropriate. A run-time selezionano una diversa tipologia di layout sulla base di proprietà di usabilità. Le proprietà di usabilità utilizzate in questo progetto sono otto criteri estratti dalle recommendation di Bach and Scapin [2003].

In figura 2.6 si possono osservare due esempi di meta-modelli prodotti. Sono presenti 3 tipi di modelli, sulla sinistra un modello dei task e un modello concettuale. Tramite le diverse proprietà di usabilità i modelli astratti vengono collegati a diversi modelli di layout di interfaccia, scegliendo il più opportuno in base alla proprietà di usabilità utilizzata. Nell'esempio seguendo un principio di raggruppamento dei concetti si predilige un layout dell'interfaccia suddivisa in un'area per la selezione della stanza e un'altra area per la selezione della temperatura. Nel secondo caso, applicando un principio di diminuzione del carico di lavoro che porta a diminuire il numero di azioni necessarie per svolgere un task, l'interfaccia permette direttamente la manipolazione della temperatura per ogni stanza disponibile.

Presso l'università di Harvard è stato sviluppato SUPPLE Gajos et al. [2010], un sistema in grado di generare automaticamente interfacce utente per persone diversa-

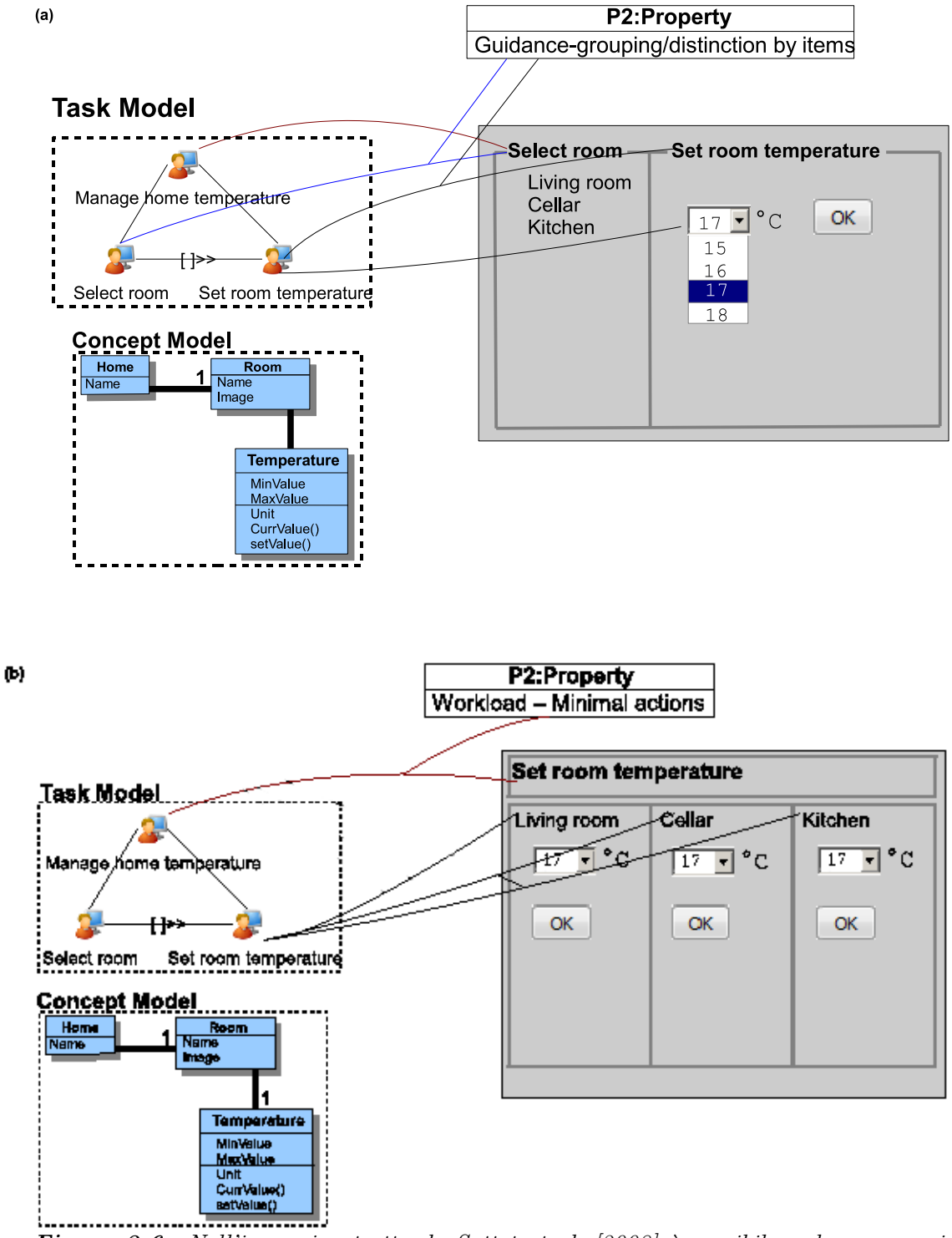

Figura 2.6: *Nell'immagine tratta da Sottet et al. [2008] è possibile vedere come viene cambiata l'interfaccia a seconda dell'applicazione di diverse proprietà; nel primo caso (a) di raggruppamento e nel secondo caso (b) di diminuzione del numero di azioni*

mente abili, prendendo in considerazione il contesto di utilizzo, le funzionalità effettivamente utilizzate, le caratteristiche specifiche di ogni utente e le proprie conoscenze e preferenze.

La maggiore differenza tra Supple e altri studi nello stesso ambito è l'obiettivo con il quale si realizzano interfacce differenti, poiché è la prima volta che ci si concentra sulle differenze fisiche ed ambientali degli utenti, piuttosto che sugli aspetti tecnologici di piattaforma e device. Per questo motivo vengono sfruttate delle tecniche di ottimizzazione per definire con estrema precisione le caratteristiche di ogni singolo elemento dell'interfaccia, a partire dalla navigazione (tab, popup, tree) fino al dimensionamento dei bottoni e del font.

Supple si basa su tre concetti per definire l'interfaccia:

1. Interface specification: definisce quali funzioni venogno messe a disposizione dell'utente, specifcando il tipo primitivo degli elementi che descrivono l'interfaccia, le aggregazioni concettuali degli stessi e gli eventuali vincoli. I tipi primitivi sono i classici interi, booleani, ecc... più quelli specifici delle interfacce, come immagini e mappe, le aggregazioni dipendono dal contesto, per esempio specifcare che nome, cognome e indirizzo fanno parte dei dati di fatturazione, e infine i vincoli possono specificare dei range di valori per gli interi, oppure impediscono o consentono l'aggregazione di alcuni elementi. La caratteristica fondamentale dell'Interface Specfication è che non definisce concretamente come dovrà essere fatto un elemento dell'interfaccia, ma ne specifica solo le caratteristiche.

2. Device Model: è il modello in cui vengono raccolti tutti i widget primari (radio button, checkboxes,...), i widget container (tab,frame,...) ed eventuali vincoli, come per esempio che una finestra non superi la dimensione dello schermo per evitare lo scroll. Questo servirà in fase di generazione dell'interfaccia per scegliere quali widget usare per ogni elemento dell'interfaccia, basando la scelta sulle caratteristiche utente.

3. User Traces: modellizza le attività svolte dagli utenti, creando dei trail, ovvero delle sequenze di cambiamenti di stato di elementi dell'interfaccia, senza entrare nella specifica istanziazione dell'elemento stesso. Questa generalizzazione permette per esempio di ottenere interfacce per device differenti semplicemente utilizzando il widget corretto, mantenendo invariata la relazione con l'attività svolta dall'utente, fondamentale per ottimizzare l'usabilità dell'interfaccia.

In figura 2.7 è riportata l'interfaccia di un programma di gestione dei dispositivi di un'aula, generata da Supple sulla base del modello graficamente schematizzato. Le frecce tratteggiate indicano aggregazioni istanziate tramite widget container, mentre le frecce continue mostrano la relazione tra elementi dell'Interface Specification e il Widget scelto per quest'interfaccia.

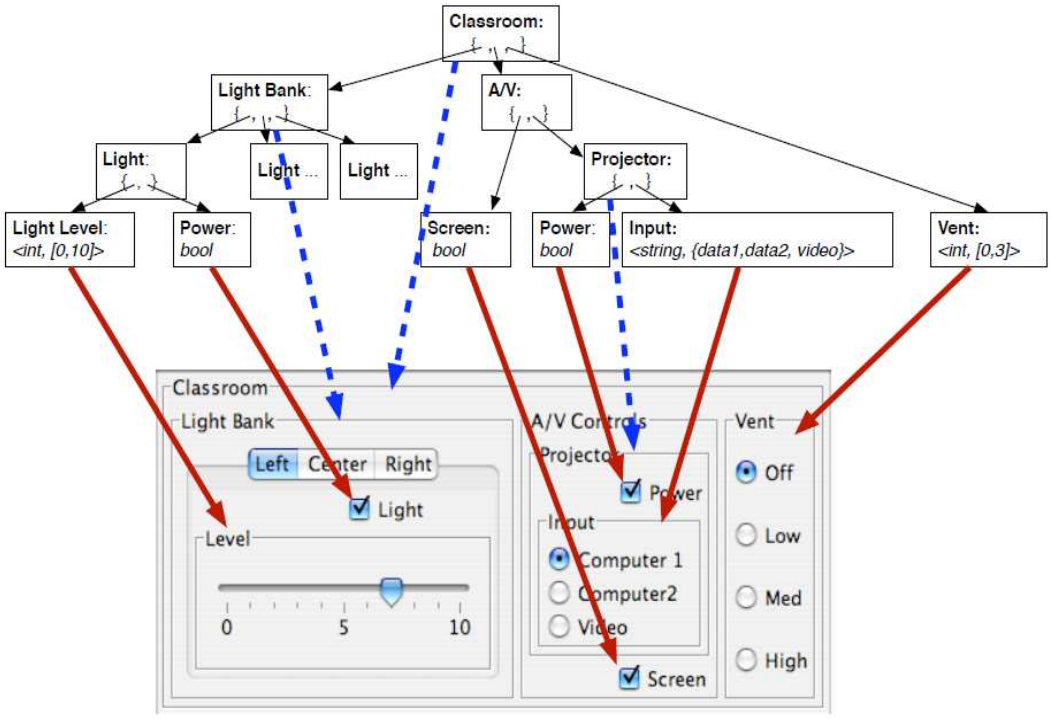

Figura 2.7: *Interfaccia di un programma di gestione dei dispositivi di un'aula generata da Supple*

SUPPLE è infine strutturato in 3 sotto sistemi, ognuno dedicato ad una specifica funzione:

- SUPPLE: genera le interfacce basandosi sulle caratteristiche utente e i modelli discorsivi
- ARNAULD: si concentra sulle preferenze dell'utente per ottimizzare la generazione dell'interfaccia
- ABILITY MODELLER: analizzza in maniera dettagliata le capacità motorie dell'utente in base alla quali sviluppa un modello da utilizzare per la generazione dell'interfaccia.

Lo studio svolto in Luna et al. [2010], invece è incentrato sui requisiti di usabilità delle applicazioni web e tiene in particolare considerazione il fatto che questo tipo di programmi richiede una certa dinamicità per rimanere al passo coi tempi. Questo lavoro consiste nell'integrazione dei requisiti di usabilità strettamente correlati alle funzionalità del software, in un modello basato su due differenti approcci model driven: test-driven e MDSD(Model-Driven Software Development). Il primo prevede cicli piuttosto brevi di test, e l'utilizzo di MDSD consente di avere una generazione automatica (o semi-automatica) di codice rapida e con una notevole riduzione degli errori, e tutto ciò si sposa bene con la necessità di frequente cambiamento. I test vengono eseguiti sin dalle prime fasi in cui si può lavorare solo sui mockup, e proseguono anche nella fase di sviluppo durante la quale le suite di test sono costruite per complessità crescente, con un'aggiunta graduale di features. A differenza del classico TDD (test driven developement), quando si terminano un ciclo di test si interviene prima sul modello dal quale viene generato il codice, e solo se necessario viene modificato o corretto il codice vero e proprio. (Vedi 2.8)

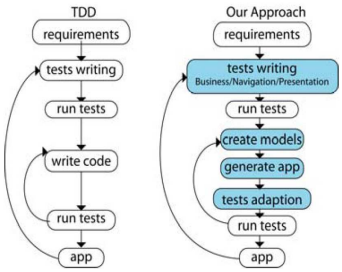

Figura 2.8: *differenza tra il ciclo di sviluppo Test-Driven e quello del lavoro di Luna et al. [2010]*

Praticamente questo modello di sviluppo prevede sei fasi, descritte di seguito:

1. Analisi dei requisiti di usabilità che impattano sulle funzionalità del programma. Per ogni requisito è possibile individuare uno o più meccanismi in grado di soddisfarlo, tra quelli già noti in letteratura grazie a specifiche linee guida. Quest'approccio alla progettazione obbliga l'analisi dei requisiti di usabilità all'inizio del processo, a garanzia del fatto che questi saranno rispettati fino alla conclusione dello sviluppo.

2. Creazione del modello dei requsiti funzionali tramite degli User Interaction Diagrams (UIDs) e dei mockup.

3. Definzione delle suite di test, anche in questo caso strettamente legate ai requisiti di usabilità definiti nella fase (1). Questo garantirà il pieno rispetto delle specifiche iniziali, semplificando l'individuazione di errori ben prima dello sviluppo completo dell'applicazione.

4. Costruzione del Design Model che consentirà la generazione automatica del codice, tramite un tool di MDSM, come per esempio Web Ratio.

5. Testing del risultato ottenuto in fase 4. Al termine dei test vengono evidenziati sia problemi insiti nel codice, piuttosto frequenti quando questo viene prodotto in maniera semi-automatica, sia problemi di concetto, ovvero che richiedono una modifica al modello.

6. Refactoring volto al miglioramento dell'usabilità del prodotto finale. Può accadere infatti che a questo punto del processo si individuino ulteriori requisit di usabilità, pertanto è previsto un refactoring che intervine a partire dalla fase (1), che richiederà chiaramente una revisione anche delle fasi seguenti.

2.5 Conclusioni

Lo studio di Ceri et al. [2002] è uno dei primi approcci che sottolinea come il model-driven software development possa comprendere nella fase di modellazione aspetti della navigazione web e sulla presentazione dei contenuti, ma non considera come questi possano essere usati per ottenere miglioramenti sull'usabilità.

OlivaNova **?** introduce la valutazione dell'usabilità all'interno della progettazione sin dalle prime fasi. Come si può osservare dalla figura 2.1 il processo di sviluppo di questo modello inserisce una fase di valutazione dell'usabilità, senza però coinvolgere gli utenti finali, per semplificarne e velocizzarne lo svolgimento. Questo però rischia di rendere scorrette queste valutazioni, se non supportate quantomeno da euristiche consolidate sull'usabilità, e di conseguenza l'intero sistema può risultare meno affidabile dal punto di vista dell'usabilità.

Il modello sviluppato in Teresa Paternò et al. [2008b,a] esamina nella fase di analisi iniziale i task che l'utente dovrà compiere nell'applicazione finale. Come si è osservato nei paragrafi precedenti i modelli di design Task-Oriented non permettono di raggiungere gli stessi risultati di usabilità dei modelli Goal-Oriented, in quanto non mirano agli obiettivi reali degli utenti.

Un altro aspetto interessante del modello realizzato nel progetto Teresa è la possibilità di definire diverse tipologie di device in modo da sviluppare per ciascuno di essi un'interfaccia adatta alle sue caratteristiche. Non si studia però come rendere l'interfaccia facilmente utilizzabile per l'utente, ma ci si sofferma solo sul fornire tutte le funzionalità anche su terminali limitanti, come ad esempio gli schermi ridotti degli smartphone.

L'approccio mostrato in Sottet et al. [2008] produce un meta-modello che partendo da modelli astratti, attraverso l'applicazione di principi di usabilità, rappresenta dei modelli di interfaccia che si adattano ai principi applicati. Questo lavoro però non fornisce un tool di sviluppo ma solo un meta-modello astratto.

SUPPLE Gajos et al. [2010] è un sitema Model-Driven con lo scopo di generare interfacce che prendano in considerazione le caratteristiche dell'utente, come le differenze fisiche ed ambientali degli utenti. Le interfacce generate sono pensate per rispettare i principi di accessibilità per le persone diversamente abili. Questo lavoro di modellazione sulle caratteristiche degli utenti potrebbe essere esteso per rivolgersi anche ad utenti generici.

Il lavoro pubblicato in Luna et al. [2010] propone un modello basato su due differenti approcci Model-Driven: Test-Driven e MDSD(Model-driven Software Development). Il primo prevede cicli piuttosto brevi di test, il secondo consente di avere una generazione automatica di codice. Questo lavoro consiste dell'unione di queste due teorie, introducendo nel MDSD, i test relativi ai requisiti di usabilità strettamente correlati alle funzionalità del software. Anche questo risulta, quindi, un approccio all'usabilità più focalizzato sui task che sugli obiettivi degli utenti.

Capitolo 3

Il modello CAO=S

Il prototipo da sviluppare in questa tesi ha alla base l'idea di fornire uno strumento di progettazione di applicazioni usabili, che sia di supporto agli sviluppatori. Per questo, è stato utilizzato il modello di design Goal-Oriented CAO=S. Questo modello è stato creato con l'intento di fornire gli strumenti base per progettare applicazioni in condizioni dove, per vari motivi, ad esempio economici, non sia possibile un'analisi sistematica dell'utenza di riferimento, coinvolgere un esperto di usabilità esterno o magari non si possano coinvolgere utenti nella progettazione. Il modello cerca di fornire soluzioni generiche agli usuali problemi di usabilità, cercando di evitare gli errori più comuni.

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 usa)
- La completezza dei contenuti (pensare a dare informazioni)
- La comprensibilità delle informazioni (eliminando le difficoltà inutili lasciandovi solo quelle utili)

Per rendere più semplice la modellazione User-centered, CAO=S propone una progettazione semplificata del modello goal-oriented, in cui la parte di analisi delle categorie di utenti previste è semplificata alla semplice analisi delle poche caratteristiche fondamentali di queste categorie di utenti.

Per semplificare la progettazione, CAO=S utilizza le seguenti idee, che sono i pilastri di base del generatore realizzato:

- Adozione sistematica e automatica di tutte le linee guida ed i pattern di progettazione utili sempre, indipendentemente da utenti e task.
- Trasformazione della fase di analisi di utenti, task e obiettivi da requisito alla progettazione a parametro di progetto: tanto più accurate e corrette sono le informazioni su utenti, task e obiettivi, tanto maggiore sarà la garanzia di usabilità.
- Identificazione delle sole caratteristiche degli utenti che hanno effettivamente un impatto noto sulla progettazione (e.g., competenze di dominio, alfabetizzazione informatica, livello di interesse, ecc.).
- Generazione automatica di pattern di interfaccia (e magari anche di scheletri di interfaccia) dopo la compilazione di schede e questionari sui tre aspetti fondamentali del modello: analisi dei concetti, analisi degli attori, analisi delle operazioni da effettuare.

3.1 Il modello CAO=S

Il modello CAO=S semplifica la progettazione utilizzando poche componenti: i Concetti, gli Attori e le Operazioni, che insieme generano Strutture (CAO=S è l'acronimo).

I *concetti* sono le informazioni trattate dall'applicazione, da essi dipendono la struttura dei dati memorizzati, il tipo di presentazione proposta, l'architettura dell'informazione, ecc. La distinzione importante da fare è che non sono le strutture dati manipolate dall'applicazione, ma rappresentano il modo in cui l'utente percepisce e comprende l'informazione che viene rappresentata dalle strutture dati.

Gli *attori* sono le categorie di utenti che agiscono sulle interfacce dell'applicazione per svolgere i loro task manipolando le strutture dati che loro interpretano come concetti. Non vengono rappresentati tramite le proprie caratteristiche personali, ma per il ruolo che svolgono all'interno dell'applicazione, che differenzia quindi l'interazione che il sistema deve proporre.

Le *operazioni* sono azioni che l'attore compie sul sistema, a cui corrispondono manipolazioni sui concetti. La tipologia di operazioni è conforme al modello CRUD (e REST) e sono quindi: creazione, lettura, modifica, eliminazione. Ogni operazione agisce o su uno o su più istanze del concetto, direttamente o indirettamente, in maniera definitiva o temporanea.

Dall'analisi delle prime tre componenti vengono generate le *strutture*, che sono l'organizzazione concreta dell'idea del progettista sui concetti in cui gli attori svolgono azioni attraverso le operazioni Ci sono tre tipi di strutture che interessano lo sviluppo:

- La view, ovvero la presentazione dei concetti, sotto forma di form, elenchi, schede, insieme di icone a manipolazione diretta, ecc.
- La navigazione, ovvero il modo in cui si passa da una view all'altra e la pagina iniziale dà accesso alle altre.
- Le strutture dati, che permettono di memorizzare in maniera persistente le informazioni che caratterizzano i concetti presentati all'utente

Come è stato analizzato nel capitolo 2 la prima fase dei modelli di progettazione Goal-Oriented è l'analisi dei requisiti. Questa fase consiste nell'analisi dei requisiti raccolti dagli stakeholder, dalla quale si producono le specifiche. I requisiti raccolti vengono analizzati per determinare se sono poco chiari, incompleti, ambigui, o contraddittori, e lo scopo è di risolvere questi problemi. La risoluzione delle ambiguità serve a fornire concetti chiari e requisiti funzionali implementabili. Ci si limita a definire 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 Booch et al. [2005]): 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 modifica 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.

La modellazione in concetti, attori, operazioni e strutture di CAO=S, consente una facile realizzazione dei mapping tra modello e architettura dell'applicazione da generare. Per questo motivo verrà formalizzato utilizzando un linguaggio di modellazione, XML, e utilizzato come input per il generatore model-driven.

3.2 Le caratteristiche degli Attori in CAO=S

L'aspetto più importante che si vuole sviluppare nel prototipo è quello dell'adattabilità dell'applicazione alle caratteristiche dell'utente. Nel modello CAO=S viene utilizzato il componente dell'attore per definire gli utenti del sistema, e viene rappresentato attraverso le sole caratteristiche essenziali per la progettazione.

L'attore in generale è uno stakeholder le cui idee contribuiscono alla progettazione del sistema. Si possono distunguere due tipi di attori:

- i *diretti* che sono quelli che useranno personalmente il sistema;
- gli *indiretti* che pur non utilizzando direttamente l'interfaccia influiscono nelle caratteristiche del sistema, possono essere ad esempio i committenti, il CED o la legislazione vigente.

Nella progettazione di CAO=S le caratteristiche prese in analisi sono solo quelle degli attori diretti. L'analisi goal-oriented prevede di rappresentare gli utenti in maniera precisa, disegnando personaggi che li rappresentano in maniera dettagliata. Per ottenere una descrizione così precisa degli utenti è necessaria un'analisi etnografica, altrimenti i personaggi sono inutili. Però l'analisi etnografica è costosa e rende i personaggi poco pratici e predittivi.

Le analisi etnografiche sugli utenti hanno sempre come risposta una curva a campana della distribuzione (come in figura 3.1), che non è mai a zero, né a sinistra (persone con inesistente caratteristica X), né a destra (persone con un eccellente caratteristica X). Quindi il risultato mostrato è che c'è sempre una piccola percentuale di utenza per la quale sarebbe utile sviluppare una certa feature del sistema.

Per semplificare la progettazione, conservando un livello di qualità minima, c'è bisogno di qualcosa di intermedio. Invece di avere una descrizione accurata di numerose caratteristiche, molte delle quali prive di un impatto preciso sulla progettazione, si identificano (almeno) cinque o sei caratteristiche base con impatto chiaro sull'implementazione e si caratterizzano gli utenti su di esse. In CAO=S l'analisi delle caratteristiche diventa un parametro della strategia del progetto.

Le caratteristiche di base che si possono prendere in considerazione sono ad esempio:

- Competenza di dominio: capacità dell'attore di comprendere esattamente di

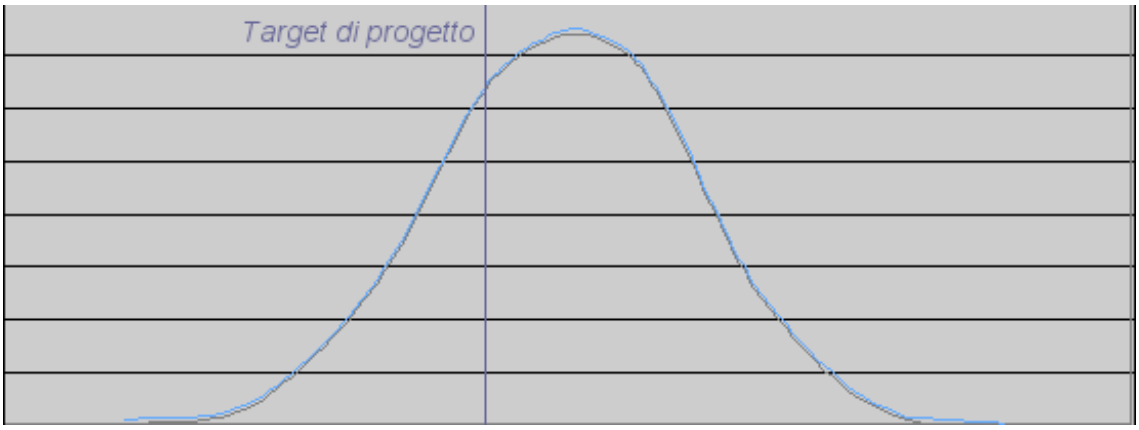

Figura 3.1: *Curva a campana della distribuzione degli utenti a seconda delle analisi entografiche.*

cosa tratta l'applicazione, della specificità del contesto in cui opera, e della terminologia specifica utilizzata.

- Competenza informatica specifica: capacità dell'attore di comprendere il contesto informatico in cui opera specificamente l'applicazione, e le consuetudini, il vocabolario e l'interazione tipica del mezzo (es. web).
- Competenza linguistica: capacità dell'attore di comprendere la lingua utilizzata dall'interfaccia, le sottigliezze linguistiche, i vari registri del linguaggio (es. burocratico, giuridico, medico), i toni (es. ordini, suggerimenti, richieste, avvisi, ecc.)
- Abilità fisica e visiva: capacità dell'attore di percepire ed agire sull'interfaccia in maniera agile e naturale. Precisione, agilità, percezione dei colori e dei contrasti, ecc.
- Motivazione ed interesse: capacità dell'attore di trovare giustificazioni all'utilizzo del sistema che vadano oltre il mero obbligo lavorativo. Capacità di concentrazione e soddisfazione personale nello svolgimento con successo dei compiti qui supportati. Rischi interni alla creazione del flow
- Distrazioni d'ambiente: capacità dell'attore di immergersi nell'applicazione e di portare a termine i compiti senza distrazioni esterne. Risci esterni alla creazione del flow.

Per ogni caratteristica, si dà una valutazione numerica (ad esempio, da 1 a 5, con 1 valore molto buono e 5 valore molto basso).

Ad esempio per la Competenza di dominio:

1. È uno specialista della materia ha una padronanza completa di terminologia e procedure, riesce a trovare il modo per affrontare un problema tecnico.

2. È particolarmente interessato all'ambito o ne ha già avuto esperienza. Riesce ad utilizzare il sistema con disinvoltura.

3. È un conoscitore medio del dominio, ne ha competenze generali e non sufficienti ad utilizzare l'interfaccia senza problemi.

4. Ha conoscenze vaghe e incomplete o parzialmente inesatte, è inesperto e fa un uso sporadico del sistema utilizzandolo ogni volta come fosse la prima.

5. Non ha alucna padronanza dell'ambito dell'applicazione. Deve essere condotto passo passo per ogni operazione da eseguire sull'applicazione

I valori ottenuti da questa valutazione delle caratteristiche vengono riportati in un grafico detto di strategia, come in figura 3.2.

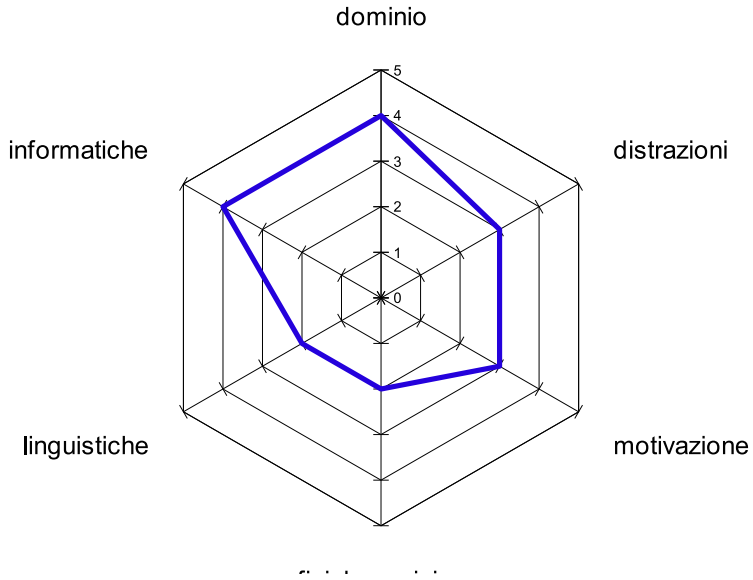

Figura 3.2

Il generatore realizzato in questo elaborato, a partire dalle caratteristiche descritte in CAO=S, realizza un mapping, associando alle caratteristiche specifici widget della struttura dell'interfaccia e della conformazione della navigazione.

3.3 Le componenti dei Concetti, Operazioni e Strutture

Dopo aver analizzato la formalizzazione delle caratteristiche degli Attori in CAO=S, si analizzano anche le altre componenti della progettazione del modello CAO=S, che vengono utilizzate per la generazione dell'applicazione.

3.3.1 C di Concetti

La componente dei Concetti esprime le informazioni trattate dall'applicazione. Rispetto alle classiche definizioni delle entità di un programma, i concetti in CAO=S si focalizzano sul rappresentare il modo in cui l'utente percepisce l'informazioni, questo comporta un lavoro di analisi nel quale è necessario eliminare ambiguità e polisemie. Ad esempio, in un'applicazione per l'università rivolta agli studenti sono presenti diversi attori: gli impiegati amministrativi (che commissionano il software e sono quindi attori indiretti), i docenti (anch'essi indiretti) e gli studenti(gli utenti). Nella fase di analisi dei requisiti si ottengono delle ambiguità semantiche di questo tipo:

La parola Corso indica per gli amministrativi un insieme di insegnamenti organizzati in anni di corso, mentre per i docenti e gli studenti indica ogni singolo insegnamento (nel loro linguaggio gli amministrativi intendono il Corso di Laurea).

La parola Esame indica per l'amministrativo e il docente un'insieme di prove che determinano la valutazione finale dello studente, mentre per lo studente indica ogni singola prova da superare per determinare la valutazione finale, e indica anche, in generale, l'insegnamento di cui seguire le lezioni e svolgere le prove.

La parola Facoltà indica per l'amministrativo e il docente un insieme di docenti di discipline diverse ma riconducibili, che gestiscono gli insegnamenti dei corsi di studi appropriati per queste discipline, mentre per gli studenti rappresenta, egualmente, il corso di studi, la disciplina studiata, e il luogo fisico in cui si svolgono le attività universitare (aule, laboratori, uffici...).

Questa situazione evidenzia dei problemi di normalizzazione, che sono risolvibili dando priorità agli attori diretti, modificando i termini in base alla loro percenzione, in quanto saranno essi gli utenti finali. Olte ad eliminare le ambiguità lunguistiche è necessario adottare altri accorgimenti, alcuni di questi sono:

- *abbreviazioni e codici* non vanno usati codici numerici, abbreviazioni o parole accavallate nelle etichette a meno che non siano usate anche dagli utenti

- *differenze lessicali* se ci sono diversi attori diretti è necessario trovare un termine accettabile da tutti, anche se non è il preferito da nessuno, aggiungendo informazioni (tipo tooltip) che esprimano il rapporto tra il termine scelto e tutti gli altri termini. Va preferito sempre il termine usato dagli utenti con minore competenza di dominio. Nell'esempio insegnamento è una buona soluzione per indicare un corso in modo comprensibile a tutti gli attori. Se non si riesce a trovare un termine condiviso, occorre realizzare interfacce diverse per ogni tipo di utenza, ma ponendo attenzione al problema della permeabilità.

- *differenze concettuali* Se sono presenti attori che attribuiscono un concetto diverso ad una stessa parola questa non va usata in nessun caso, né per un significato né per l'altro. Usare specificazioni e precisazioni per disambiguare il concetto. Ad esempio, la parola Corso non va mai usata. Usare Corso di studi per indicare un concetto e insegnamento per indicare quell'altro.

- *polisemie* Nel caso di parole con più significati è meglio evitare di usarla ma cercare sinonimi accettabili per ciascun significato diverso. In caso non si riescano a trovare soluzioni accettabili a questi problemi si rende necessaria un'ulteriore fase di raccolta di requisiti, dedicata alla riorganizzazione linguistica dell'interfaccia. Prendere decisioni unilaterali in ogni caso non è mai opportuno, è necessario verificare con gli attori.

I Concetti diventano così una descrizione delle informazioni nel linguaggio dell'utente e dall'analisi di essi verrano create le entità dell'applicazione da generare.

3.3.2 O di Operazioni

Le operazioni sono operazioni sui concetti, non sulle strutture dati. Ogni comando, ogni etichetta, ogni widget deve riportare termini connessi con i concetti, e non con i termini di sistema associati. Le operazioni secondo CAO=S sono di quattro tipi (secondo i modelli CRUD e REST):

- Creazione
- Vista
- Aggiornamento
- Eliminazione

La *CREAZIONE* consiste nella generazione di una o più istanze di concetto. Le sue caratteristiche sono:

- Tipo di creazione: manuale, automatica, implicita.
- Valori di Default: sono fondamentali per diminuire il lavoro dell'utente e rassicurarlo sul senso e la destinazione dell'operazione, quindi non devono mai mancare.
- Molteplicità: possibilità di creare solo un'istanza o molte allo stesso tempo.
- Persistenza: se l'operazione è persistente l'istanza del concetto dovrà esistere dopo la fine dell'operazione e deve esistere un operazione di vista associata. Alcune operazioni invece sono transienti e connesse alla mera esecuzione dell'operazione.
- Memoria dell'utente: Oltre al valore di default proposto dal sistema, i valori precedentemente immessi dall'utente sono utili per la creazione rapida ed efficace di istanze del concetto.
- Notifica di insuccesso: nel momento in cui il sistema riscontra un problema, ad esempio: username già usato, posti non disponibili, credito insufficiente, ecc. deve segnalarlo subito non alla fine dell'operazione.

La *VISTA* consiste nella visualizzazione di una o più istanze del concetto, le diverse tipologie sono:

- Vista individuale completa, dove tutte le proprietà associate al concetto sono visibili. Da questa è possibile il passaggio a un update globale se l'utente ne è abilitato;

- Vista individuale ridotta, vengono visualizzate solo alcune proprietà eventualmente modificabili e con la possibilità di passare ad una vista individuale completa;

- Vista multipla (lista) vengono visualizzate poche informazioni di ciascun concetto in un elenco dal quale è possibile passare ad una vista individuale completa. Deve essere possibile eseguire operazioni sulla lista (ordinamenti, raggruppamenti ,filtri, ricerche sul database, ecc.) Se possibile per l'utente, deve essere fornito il componente di navigazione per eseguire una creazione di un'istanza del concetto.

- Vista multipla (lookup) si visualizza un elenco di istanze del concetto con poche informazioni, lo scopo è selezionare una o più istanze del concetto da usare in seguito. Un esempio di lookup può essere l'elenco di concetti ritornati come risultati veloci di una ricerca.

- Vista multipla (ricapitolo) i diversi concetti vengono visualizzati insieme e per ciascuno vengono visualizzate alcune informazioni utili come ad il totale di istanze create per il concetto o altre informazioni "statistiche".

Per rendere facile all'utente la fruizione delle informazioni, ci sono alcuni accorgimenti da adottare nella creazione delle viste, come avere memoria delle configurazone adottate dall'utente (criteri di filtro, di sort, di raggruppamento vanno) e far capire in modo esplicito le distinzioni tra alcune funzionalità che potrebbero non essere chiare (come la distinzione tra ricerca e filtro, la prima è la ricerca di uno specifico individuo la seconda serve a qualificare uno spazio di valori).

L'*UPDATE* è la modifica di uno o più proprietà, di una o più istanze dell'entità, senza creazione di nuove istanze, può essere:

- globale se tutte le proprietà dell'istanza sono modificabili, visualizzate come nella creazione, ma popolate correttamente;

- specifico se dipende dal tipo di valore aggiornato. Ad esempio i valori booleani possono essere aggiornati con un semplice pulsante dentro ad un'altra vista (ad esempio in una lista), in altri casi la modifica non è esplicita (per esempio,

il pulsante "Invia" aggiorna la proprietà "inviato" del messaggio senza che l'utente la manipoli direttamente).

Anche nell'Update sono necessari alcuni accorgimenti di base per realizzare le funzioni minime di usabilità. Gli aggiornamenti multipli degli stessi valori su tutte le istanze vanno gestiti in modo corretto spiegando cosa significano e come realizzarli. Se un aggiornamento è implicito in una funzione automatica, l'operazione da notificare come avvenuta con successo, è la funziona automatica non l'aggiornamento.

La *REMOVE* è la rimozione di una o più entità, può essere di due tipi:

- Eliminazione: l'istanza o le istanze non esistono più e non sono più recuperabili, equivale a una cancellazione totale;

- Archiviazione: l'istanza o le istanze semplicemente non sono più disponibili nelle operazioni di vista generali, e richiedono una operazione di vista particolare, può essere definitiva o temporanea.

In questo tipo di operazione, come negli altri, è opportuna una corretta notifica di successo o insuccesso, nel caso dell'archiviazione, deve essere specificato il percorso per riaccedere ai dati archiviati.

La descrizione delle operazioni viene esaminata dal generatore model-driven per creare l'aggiornamento con le strutture dati permanenti.

3.3.3 S di Strutture

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

Le strutture nel modello CAO=S sono tre:

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

- Viste: modelli di schermata attraverso cui le proprietà delle entità vengono visualizzate. Ogni vista è composta dalla visualizzazione vera e propria, e da comandi attivabili durante la visualizzazione, alcuni di questi comandi sono di navigazione;

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

Il modello CAO=S si basa sulla creazione di un tabella tridimensionale, aventi per assi i concetti, gli attori e le operazioni. Dentro a ciascuna cella vengono inserite annotazioni di come l'attore debba poter eseguire l'operazione sul concetto.

Docente	Insegnamento
Create	no
View	singolo: dati pubblici multiplo: dati ridotti
Update	parziale: programma del corso e pubblicazione del programma
Remove	no

Tabella 3.1: *tabella Concetti x Attori x Operazioni*

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

Si è potuto osservare come la modellazione in concetti, attori, operazioni e strutture di CAO=S, consenta una facile progettazione dell'architettura dell'applicazione. In particolare le caratteristiche degli utenti in questo modello permettono la creazione di componenti dell'interfaccia intercambiabili (i widget), che vengono progettati in modo da facilitare l'utilizzo del programma ad ogni tipologia di utente.

3.4 Il processo di generazione con CAO=S

Il modello di progettazione CAO=S è implementabile come generatore di applicazioni model-driven, sviluppando due applicazioni differenti:

- *CAO=S Analizer*, attraverso la compilazione di report e form da parte del team di sviluppo, estrae le prime tre componenti di CAO=S. Da queste crea la tabella tridimensionale che ha per assi Concetti, Attori e Operazioni utiliz-

zando il diagramma di strategia di CAO=S. Il risultato di questa fase viene passato alla successiva di creazione del codice modellato in un file xml;

- *CAO=S Generator*, a partire dal risultato dell'analisi sviluppa concretamente l'applicazione delineata dal team di sviluppo, generando automaticamente il codice server e client side, l'interfaccia utente e il modulo di persistenza dei dati.

Questo lavoro di tesi si interessa al secondo applicativo, il CAO=S Generator, del quale viene realizzato un prototipo chiamato Anthropos. L'applicazione consiste in un generatore model-driven che prende come input il file xml frutto dell'analisi effettuata con l'applicazione CAO=S Analizer, che descrive il modello di progettazione CAO=S attraverso le strutture che descrivono i Concetti, le Operazioni, e le caratteristiche degli attori dell'applicazione. Dal parsing di queste informazioni contenute nel file xml, Anthropos genera una rappresentazione persistente dei dati con un database e i moduli Server-side che permettono di interfacciarsi con esso, e la logica dell'applicazione lato Client. L'obiettivo è quello di generare un'applicazione che faciliti l'interazione tra l'utente e il sistema rispettando i principali requisiti di usabilità.

Le tecnologie utilizzate e il dettaglio delle fasi di generazione verranno spiegate approfonditamente nel capitolo successivo.

Capitolo 4

Anthropos (CAO=S Generator)

Dallo studio del problema in esame, ovvero la necessità di creare applicazioni che si adattino alle caratteristiche degli utenti che le utilizzano, si è deciso di sviluppare un prototipo di applicazione che fornisca livelli controllabili di usabilità. Sono state studiate le soluzioni proposte finora a livello di architetture model-driven, e viste la loro inadeguatezza, si è deciso di sviluppare un nuovo sistema basandosi sul modello CAO=S.

Questo modello di design può essere implementato suddividendolo in due applicazioni dette CAO=S Analizer e CAO=S Generator. La prima riceve in input, tramite dei form, i questionari prodotti dal team di sviluppo, che a partire dall'analisi del dominio formalizzano i Concetti, gli Attori e le Operazioni dell'applicazione. A partire da queste informazioni CAO=S Analizer genera le Strutture del modello creando un file XML che le descrive. Il secondo applicativo, il CAO=S Generator genera automaticamente l'applicazione a partire dal file XML elaborato dall'Analizer. In questo lavoro è stato implementato un prototipo del secondo applicativo, il CAO=S Generator, chiamato *Anthropos*.

Il diagramma in figura 4.1 rappresenta l'esecuzione di Anthropos, che partendo dall'analisi del file XML contenente le Strutture che descrivono i Concetti, le Viste, la Navigazione e le caratteristiche degli Attori, genera automaticamente un'applicazione con le seguenti componenti:

1. un modulo di Data Storage, collocato su un Server;

2. i moduli della logica dell'applicazione, realizzati utilizzando tecnologie AJAX che permettono di integrare le applicazione lato server con l'esecuzione di al-

cune parti lato client, permettendo di ottenere un'applicazione più dinamica che non richiede un'interazione continua con il server;

3. un'interfaccia dinamica composta da diverse viste navigabili, anch'essa sviluppata lato client; ciascuna singola vista (form, griglia, o altri tipi) può essere visualizzata in modo diverso a seconda dell'utente che la utilizza (vedi figura 4.2).

Lo scopo di questa architettura dell'applicazione prodotta è permettere all'utente un'interazione personalizzata con il sistema, composta dalla visualizzazione dell'interfaccia, la navigazione tra le viste e la struttura della pagina. La caratterizzazione dell'interazione viene realizzata sulla base della descrizione degli attori fornita dal modello in fase di analisi.

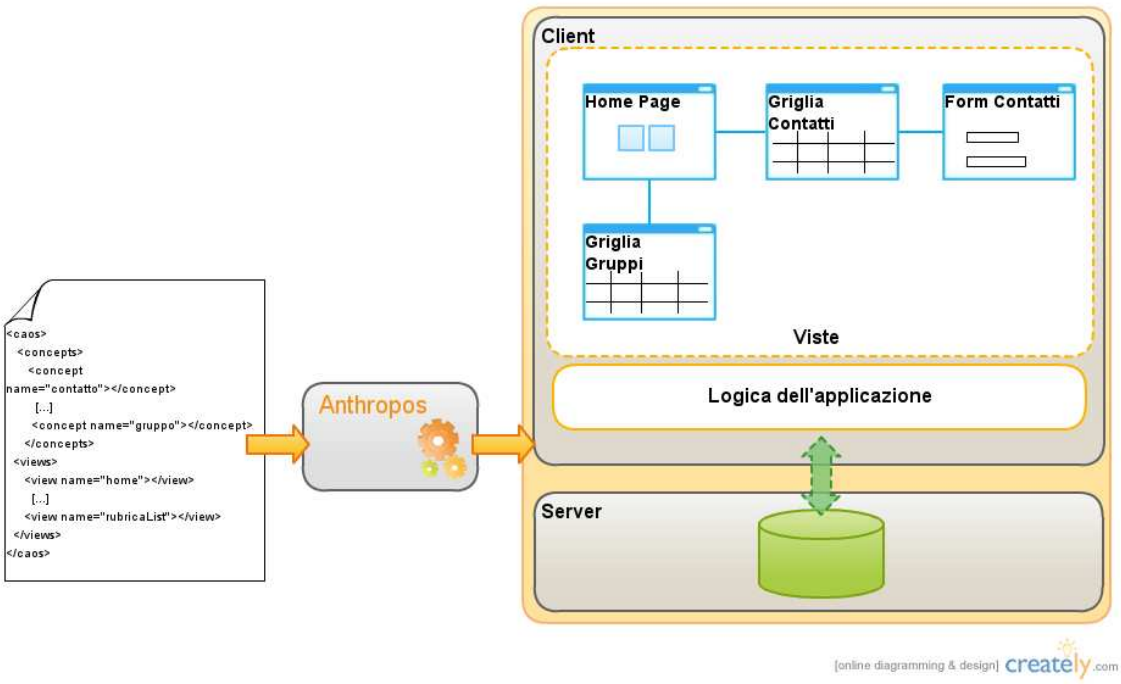

Figura 4.1: *Diagramma di esecuzione di Anthropos*

Per descrivere il processo di generazione di Anthropos si ricorre ad un esempio, descritto nel codice XML del modello CAO=S, e in base agli elementi del modello rappresentato nel file viene descritto come il sistema genera le tre componenti dell'applicazione esaminate precedentemente. La struttura del file XML di CAO=S è quella dell'esempio 4.

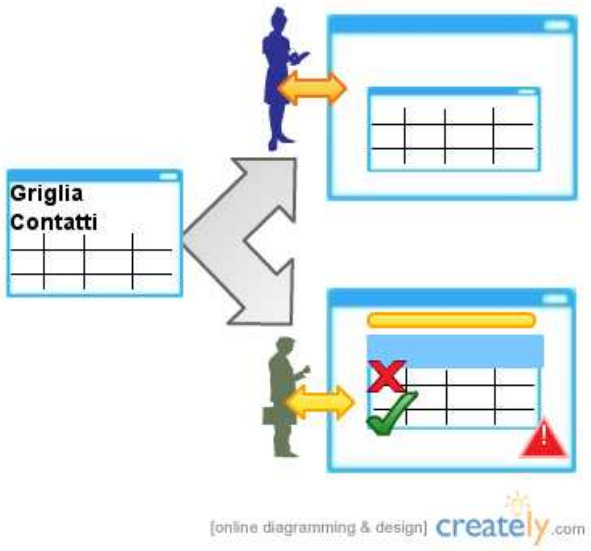

Figura 4.2: *Diagramma della diversità dell'interfaccia in base all'utente in Anthropos*

```
<caos>
   <concepts>
      <concept name="contatto"></concept>
      <concept name="gruppo"></concept>
   </concepts>
   <views>
      <view name="home"></view>
      <view name="rubricaCompact"></view>
       [...]
      <view name="gruppoForm"></view>
   </views>
   <actors>
      <actor role="UtenteEsperto(liv2)"></actor>
      <actor role="UtenteMedio(liv3)"></actor>
   </actors>
</caos>
```

Esempio 4.1: *Struttura del file XML di CAO=S*

4.1 Modulo di Data Storage

Il modello rappresentato nel codice XML d'esempio contiene due Concetti distinti, i *Contatti* della rubrica e i *Gruppi* di contatti, che come è possibile vedere nel frammento di file nell'esempio4.1, sono collegati da una relazione 1-N che associa a ciascun contatto un gruppo.

```
<concepts>
  <concept name="contatto">
     <automaticField name="codicecontatto" type="integer" id="true"
      autoincrement="true"/>
     <editableField name="nome" type="string" queryable="true"
      listable="true"/>
     <editableField name="cognome" type="string" queryable="true"
      listable="true"/>
     <editableField name="email" type="string" queryable="true" listable="true"
       />
     <editableField name="telefono" type="string" queryable="false"
      listable="true"/>
     <joinField concept="gruppo" name="codicegruppo" type="integer"/>
     <lookupField concept="gruppo" name="nomegruppo" type="string"
      listable="true"/>
  </concept>
  <concept name="gruppo">
     <automaticField name="codicegruppo" type="integer" id="true"
      autoincrement="true"/>
     <editableField name="nomegruppo" type="string" listable="true"
      queryable="true"/>
     <editableField name="descrizione" type="string" listable="true"
      queryable="true"/>
  </concept>
</concepts>
```

Esempio 4.2: *I Concetti di CAO=S*

I Concetti descritti nel file contengono diversi tipi di field, ciascuno dei quali richiede un appropriato mapping nel database:

- **AutomaticField:** rappresenta un elemento di identificazione univoca del concetto, generalmente non viene gestito dall'utente ma dal sistema; questo campo viene mappato come la chiave primaria della tabella rappresentante il concetto.

- **JoinField:** serve a specificare la relazione che vi è fra il concetto corrente ed un'altro e viene mappata come fereign key della tabella nel database;

- **LookupField:** identifica un riferimento a un campo di un altro concetto di

tipo AutomaticField o EditableField, quindi non viene mappato come colonna nella tabella del Concetto;

- **EditableField:** è un campo del concetto modificabile dall'utente, viene mappato come colonna del concetto.
- **ListField:** identifica un riferimento a un altro concetto come il LookupField ma a differenza di esso rende fruibile una lista di informazioni reperibili dagli altri concetti.

Anthropos mappa ciascun concetto come un'entità del Database Relazionale utilizzato per il Data Storage, quindi nell'esempio vengono realizzate due tabelle, una per i contatti 4.1 e l'altra per i gruppi 4.1, contententi i campi appropriati rispetto ai field presenti nel modello.

tabella Contatto	
int (primary key)	codicecontatto
string	nome
string	cognome
string	e-mail
string	telefono
int (foreign key)	codicegruppo

Tabella 4.1: *tabella dei Contatti realizzata da Anthropos*

tabella Gruppo	
int (primary key)	codicegruppo
string	nomegruppo
string	descrizione

Tabella 4.2: *tabella dei Gruppi realizzata da Anthropos*

4.2 Viste dell'interfaccia

Come si può vedere nel codice di esempio 4.2, in CAO=S vengono definiti diversi tipi di viste:

- **List:** fa riferimento ad un Concetto e definisce una struttura di visualizzazione dinamica di elementi dello stesso tipo (una griglia di istanze del medesimo Concetto), e mostra come colonne tutti gli elementi del Concetto contenuti in esso (solo quelli con l'attributo listable uguale a true) come in figura 4.4;

- **Summary:** viene utilizzato per esprimere informazioni di ricapitolo riguardanti il Concetto a cui si riferisce; In figura 4.5 si può vedere un esempio di come viene generata la vista summary.

- **Form:** indica la realizzazione della omonima struttura HTML, che dia la possibilità all'utente di interagire con la vista stessa per realizzare creazioni o modifiche ai concetti; In figura 4.3 viene mostrato un esempio di Form di modifica dei contatti della rubrica;

- **Subview:** permette di includere View all'interno di altre View, costruendone di tipo più complesso.

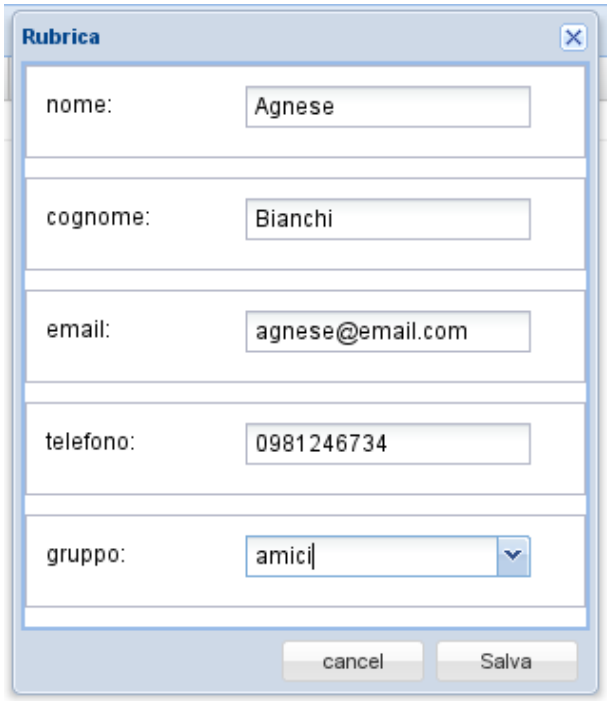

Figura 4.3: *Esempio di Form di modifica dei contatti della rubrica.*

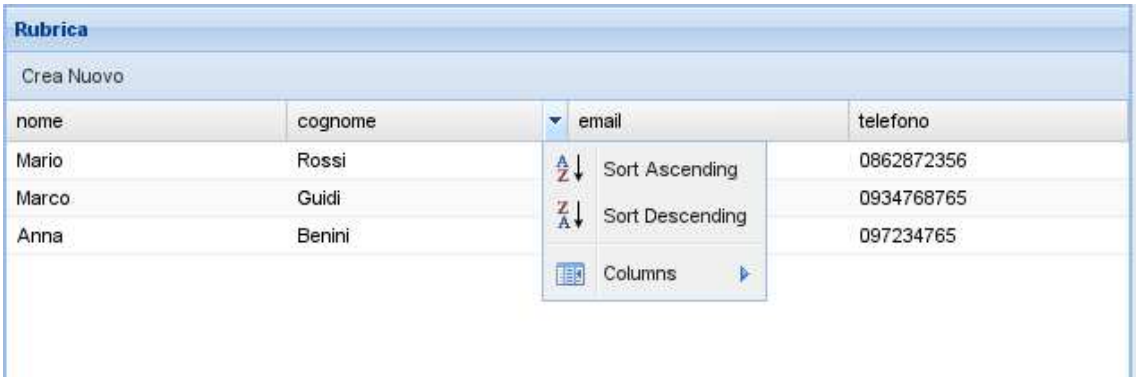

Figura 4.4: *Esempio di una vista di tipo List.*

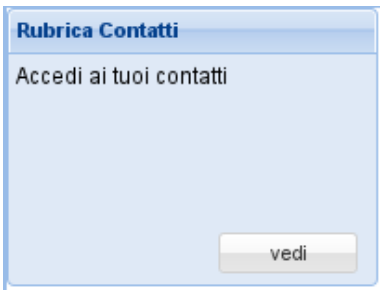

Figura 4.5: *Esempio di una vista di tipo Summary con relativo comando di navigazione creata da Athropos.*

```
<views>
   <view name="home">
      <subview>
         <view ref="rubricaCompact"/>
      </subview>
      <navigation/>
   </view>
   <view name="rubricaCompact">
      <summary concept="contatto">
         <heading>Rubrica Contatti</heading>
         <content>Accedi ai tuoi contatti</content>
      </summary>
      <navigation>
         <link op="list" concept="contatto" view="rubricaList">Visualizza</link>
      </navigation>
   </view>
   <view name="rubricaList">
      <list concept="contatto">
         <heading>Rubrica</heading>
         <operations>
            <show view="contattoForm"/>
            <edit view="contattoForm"/>
            <sort/>
            <group/>
            <filter/>
         </operations>
         <navigation>
            <link op="create" concept="contatto" view="contattoForm">Crea
             Nuovo</link>
         </navigation>
      </list>
   </view>
   <view name="contattoForm">
      <form concept="contatto">
         <heading>Contatto Rubrica</heading>
         <operations>
            <remove/>
            <archive/>
            <create/>
         </operations>
      </form>
   </view>
</views>
```

Esempio 4.3: *Le viste di CAO=S*

4.3 Logica dell'applicazione

Per realizzare la logica dell'applicazione in modo ottimale Anthropos, come verrà descritto dettagliatamente in seguito, utilizza il pattern Model-View-Controller. Questo pattern prevede la separazione tra i componenti di gestione dei dati, visualizzazione e controllo dell'applicazione. L'utilizzo di MVC permette ad Anthropos di effettuare un mapping molto funzionale tra le componenti di CAO=S e l'architettura dell'applicazione suddivisa in modelli, viste e controller.

Anthropos analizza i Concetti di CAO=S e per ciascuno produce un *Modello*. Nell'applicazione d'esempio quindi saranno presenti due modelli rappresentanti i Contatti e i Gruppi, e questi modelli saranno collegati da un'associazione che rappresenta la relazione tra i due concetti.

La seconda componente della logica dell'applicazione, il *Controller*, viene composto analizzando tutti i componenti di Navigazione presenti nelle viste all'interno del modello e per ciascuno di essi vengono realizzate le opportune operazioni di passaggio da una vista all'altra. Nell'esempio 4.3 è presente il comando "Crea Nuovo", che verrà realizzato nell'interfaccia con un bottone nella vista della lista dei gruppi, all'evento click sul pulsante il controller gestirà il passaggio alla vista del rispettivo Form di inserimento.

4.4 La caratterizzazione dell'interazione sugli utenti

Come rappresentato in figura 4.2 in Anthropos le viste vengono visualizzate cambiando dinamicamente in base all'utente che le utilizza. La caratterizzazione delle viste viene realizzata basandosi sulle caratteristiche degli *Attori* descritte in CAO=S, di seguito ne vediamo un esempio in cui i due *Attori* dell'applicazione si differenziano unicamente sulle competenze di dominio. Nel primo caso, in cui l'utente è più esperto, Anthropos visualizza un'interfaccia semplice e con le sole informazioni essenziali, mentre nel secondo caso fornisce descrizioni dettagliate dei concetti per facilitare l'utente nella comprensione del dominio. Questo principio viene applicato, per esempio, all'interfaccia dei form aggiungendo ad ogni campo una nota che ne spiega meglio il significato.

```
<actors>
   <actor role="UtenteEsperto(liv2)">
      <domain>2</domain>
```

```
<view name="gruppiList">
   <list concept="gruppo">
      <heading>Gruppi di contatti</heading>
      <operations>
         <select/>
         <multipleSelect/>
         <show view="gruppoForm"/>
         <edit view="gruppoForm"/>
         <sort/>
         <group/>
         <filter/>
      </operations>
      <navigation>
         <link op="create" concept="gruppo" view="gruppoForm">Crea Nuovo</link>
      </navigation>
   </list>
</view>
<view name="gruppoForm">
   <form concept="gruppo" >
      <heading>Gruppo di Contatti</heading>
      <operations>
         <remove/>
         <create/>
      </operations>
   </form>
</view>
```

Esempio 4.4: *La Navigazione tra le Viste di CAO=S*

```
        <technological>3</technological>
        <linguistic>3</linguistic>
        <physical>3</physical>
        <motivation>3</motivation>
        <distraction>3</distraction>
    </actor>
    <actor role="UtenteMedio(liv3)">
        <domain>3</domain>
        <technological>3</technological>
        <linguistic>3</linguistic>
        <physical>3</physical>
        <motivation>3</motivation>
        <distraction>3</distraction>
    </actor>
</actors>
```

Capitolo 5

L'implementazione di Anthropos

Dopo aver esaminato a livello concettuale il funzionamento di Anthropos, in questo capitolo, vediamo come questo avvenga concretamente. Anthropos è un'applicazione sviluppata nel linguaggio PHP che viene eseguita da un browser e prende in input, attraverso l'inserimento in un'apposita directory, un file XML che rappresenta il modello CAO=S, dal quale genera l'applicazione descritta. Come si può vedere in figura 5.1, Anthropos, a partire dal file di input che descrive Concetti, Viste e Attori, genera un'applicazione suddivisa in diverse componenti. L'analisi dei Concetti porta alla realizzazione del modulo di Data Storage, viene poi generata la logica dell'applicazione Client-side, suddivisa in un Modello che si occupa della rappresentazione dei dati, un Controller che gestisce l'interazione e le Viste che mostrano le informazioni tramite delle interfacce. Per fornire l'adattabilità dell'interfaccia al tipo di utente, Anthropos fornisce un set di widget che sono delle componenti di visualizzazione, che possono essere applicate alle viste modificandole in modo dinamico.

5.1 Tecnologie utilizzate

La progettazione del generatore realizzato inizia con una fase di analisi del problema e la scelta delle tecnologie da utilizzare per la realizzazione che sono le seguenti:

- **REST**: Fielding [2000] è l'acronimo di Representational Transfer State, ed è un paradigma per la realizzazione di applicazioni Web che permette la manipolazione delle risorse per mezzo dei metodi GET, POST, PUT e DELETE del protocollo HTTP. Basando le proprie fondamenta sul protocollo HTTP, il

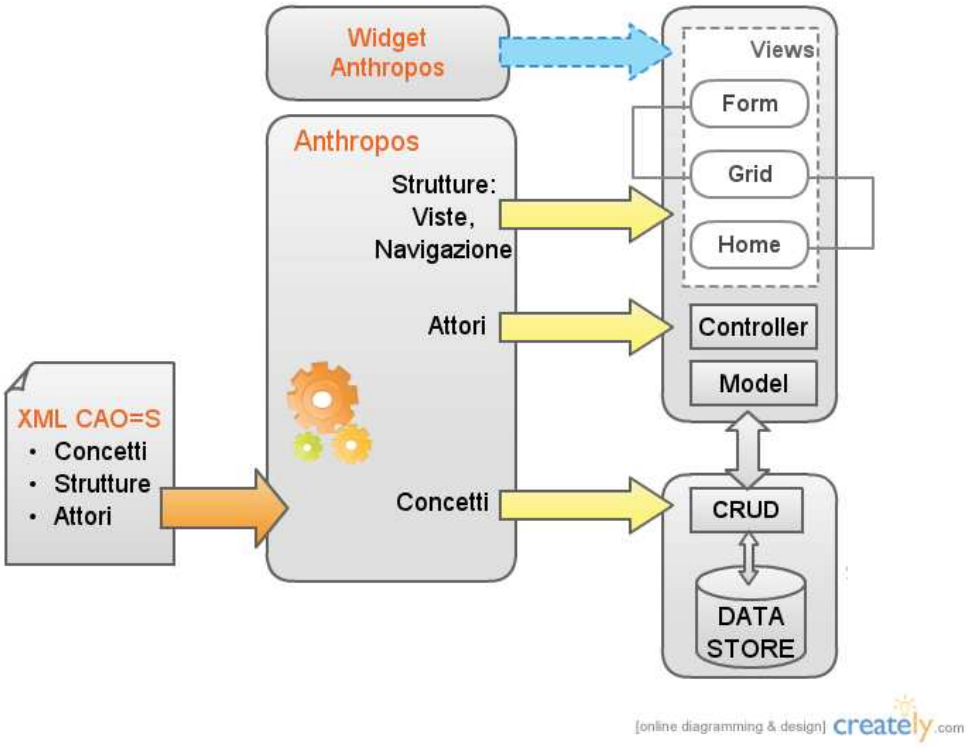

Figura 5.1: *Diagramma di esecuzione di Anthropos*

paradigma REST restringe il proprio campo d'interesse alle applicazioni che utilizzano questo protocollo per la comunicazione con altri sistemi. Un concetto fondamentale sono le Risorse, ogni risorsa è unica e indirizzabile usando sintassi universale per uso nei link ipertestuali (URI).

- **Model-View-Controller**: Leff and Rayfield [2001] è un pattern di progettazione introdotto originariamente con Smalltalk (1980 Xerox), si basa su astrazioni presenti in tutte le applicazioni dotate di interfaccia grafica. Porta importanti migliorie in termini di riusabilità, manutenibilità ed estendibilità del software. Il principio su cui si basa questo pattern è la separazione di un'applicazione in componenti che:

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

  – producono ed elaborano i contenuti (Controller);

  – presentano i contenuti (View)

  Il principale vantaggio è dato dalla separazione delle viste dai modelli facendoli interagire tramite il controller. Questo permette di creare e modificare

viste senza dover cambiare il modello e rappresentare un modello per mezzo di viste multiple indipendenti. L'utilizzo di questo pattern in Anthropos permette di mappare in modo semplice le componenti del modello CAO=S con l'architettura dell'applicazione MVC.

- **AJAX** (Asynchronous JavaScript and XML): è una tecnica di sviluppo per la realizzazione di applicazioni web interattive (Rich Internet Application). Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è necessario che le richieste di caricamento debbano essere necessariamente asincrone. Questa tecnonologia offre dei vantaggi nell'interazione dell'utente con l'applicazione, in quanto permette di ottenere modifiche dinamiche e leggere sulle pagine, come ad esempio i suggerimenti sui campi di ricerca, in grado di suggerire le parole presenti e velocizzare l'inserimento. La scelta di utilizzare questa tecnica è dovuta proprio a questi vantaggi, in quanto permettono di raggiungere lo scopo di rendere l'interazione dell'utente col sistema il più fluida e ed efficace possibile.

Il generatore CAO=S Anthropos è sviluppato nel linguaggio PHP, e genera applicazioni strutturate con:

- Database Mysql;
- Componenti PHP che permettono la comunicazione tra database e logica dell'applicazione;
- Logica dell'applicazione in Javascript.

Il linguaggio Javascript ha lo svantaggio di non avere un'unica sintassi che valga per qualsiasi browser. Per questo motivo si è scelto di utilizzare un framework, ExtJs, in quanto questa soluzione offre diversi vantaggi:

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

- strutturazione il codice, supporto fondamentale per la creazione automatica del codice;
- supporto cross-browser;
- API per la creazione e la gestione di richieste asincrone al server;
- librerie di widget disponibili per realizzare l'interfaccia.

Nello specifico è stato scelto il Framework ExtJs, in quanto fornisce due strumenti fondamentali per realizzare Anthropos:

- librerie di widget, ovvero classi che realizzano componenti standard dell'interfaccia come Griglie, Form e layout particolari, che sono facilmente estendibili creandone di nuove;
- supporto alla strutturazione del codice secondo il pattern Model-View-Controller che permette di gestire semplicemente la dinamicità dell'applicazione nella navigazione e nel cambiamento dell'interfaccia.

5.2 Fasi della generazione

La generazione, in Anthropos, avviene attraverso le seguenti fasi:

La prima fase si occupa di inizializzare il sistema, analizzando il file di configurazione che permette di instanziare il database e creare le directory della nuova applicazione da generare. L'architettura viene realizzata creando la struttura di directory e file dell'applicazione seguendo il pattern MVC proposto dal Framework javascript Ext Js.

Nella seconda fase vengono analizzati i Concetti, descritti nel file xml, che rappresentano le informazioni percepite dall'utente e ne rappresentano le strutture dati; attraverso la loro analisi si estraggono le informazioni che servono a creare il modulo Server-side di gesione del database e interfaccia dello stesso con la logica dell'applicazione. .

La terza fase, la più complessa, ha lo scopo di creare la logica dell'applicazione, che viene interamente sviluppata lato Client con javascript. In questa fase si realizzano innanzitutto i Modelli dei dati che rappresentano i Concetti elaborati dal modello CAO=S e gli Store che gestisceno le comunicazioni AJAX

con il server per il recupero dei dati. Successivamente dall'analisi degli Attori e delle strutture View e Navigation, vengono creati i moduli delle View che si occupa di presentare le informazioni dei concetti all'utente attraverso l'interfaccia. Infine si realizza il controllo dell'applicazione che è il modulo che permette di gestire concretamente la navigazione all'interno dell'applicazione e viene costruito sull'analisi delle operazioni inserite nelle Strutture di CAO=S.

5.2.1 Inizializzazione

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 (un esempio di file di configurazione è 5.2.1):

- projectRoot: il PATH dove l'applicazione viene sviluppata
- projectName: il nome del progetto da sviluppare
- database: tutte le informazioni necessarie alla connessione col database

```
<config>
  <projectRoot>/system/public_html/rubrica/</projectRoot>
  <projectName>rubrica</projectName>
  <database>
     <type>mysql</type>
     <hostname>localhost</hostname>
     <port></port>
     <dbname>rubrica</dbname>
     <user>root</user>
     <password></password>
  </database>
</config>
```

Esempio 5.1: *Il file di configurazione di Anthropos*

Anthropos esamina il file ed instanzia l'ambiente di sviluppo configurando la connessione con il database e creando la struttura dell'applicazione, attraverso la creazione delle cartelle necessarie per implemtere l'applicazione secondo il pattern MVC supportato da EXT Js. L'utilizzo del pattern Model-View-Controller permette una gestione del codice che semplifica la generazione automatica e permette una buona scalabilità. In figura 5.2 si può vedere come viene strutturato il codice. I file

dell'applicazione vengono suddivisi nelle cartelle *app* e *data*, nella prima è contenuto il codice javascript dell'applicazione mentre in data ci sono i moduli php che si occupano della comunicazione con il database. La terza cartella presente al primo livello, *ext*, contiene i file della libreria javascript Ext Js.

Il file di lancio dell'applicazione è l'*index.html* che contiene l'inclusione della libreria EXT Js e il file javascript *app.js* che contiene le configurazioni globali dell'applicazione e mantiene i riferimenti a tutti i modelli i controller e le viste utilizzate dall'applicazione, definiti tutti all'interno della cartella app. Infine il file app.js contiene una funzione di launch che lancia l'esecuzione dell'applicazione.

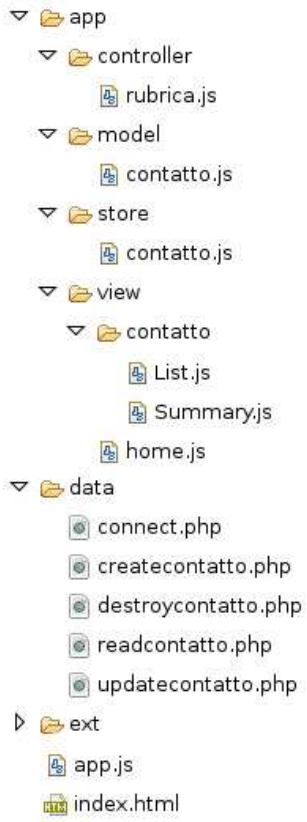

Figura 5.2: *Struttura dell'applicazione*

5.2.2 modulo Server-side

Ultimata la fase di startup, il sistema è pronto a iniziare le varie fasi utili alla realizzazione del progetto; Anthropos, in questa seconda fase, analizza inizialmente i concetti descritti all'interno del file xml rappresentante il modello CAO=S preso in input per poterne ricavare informazioni utili allo sviluppo del modello di persistenza

dei dati. I Concetti descritti sono le informazioni espresse nel linguaggio 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.

Dall'analisi dei Concetti, e dei rispettivi elementi, Anthropos utilizza un mapping per passare dalla struttura CAO=S a quella relazionale del database. I Concetti vengono mappati come Entità del database e gli elementi del Concetto come record della rispettiva entità. Esistono differenti elementi che possono descrivere le caratteristiche di un Concetto ed a ciascuno di essi corrisponde ad un diverso tipo di elemento nel modello relazionale.

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.
- Autoincrement: indica un indice numerico incrementato per ogni nuovo inserimento dal database.

Per creare il modulo di persistenza dei dati Anthropos crea, tramite delle funzioni a linea di comando di PHP, una tabella per ogni Concetto da cui prende anche il nome. Le colonne della tabella rappresentano quegli elementi che hanno un legame forte col Concetto e sono quindi tutti quelli indicati nel file xml con tagname: AutomaticField, EditableField, JoinField. 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. Successivamente alla realizzazione del database dell'applicazione vengono creati anche i moduli Server-side che si occupano della comunicazione tra il database e l'applicazione Javascript. Vengono realizzati 4 file PHP, secondo il modello CRUD, che si osccupano rispettivamente di comunicare al Client i risultati delle operazioni di Creazione, Lettura, Modifica ed Eliminazione.

La comunicazione tra Server e Client avviene attraverso i seguenti passaggi: il client invia una richiesta AJAX al server che effettua la query al database e ritorna una risposta di successo. Le comunicazioni AJAX avvengono utilizzando il linguaggio JSON per trasmettere le notifiche di successo e i dati.

5.2.3 Moduli Client-side

Questa fase particolarmente complessa e articolata si divide in tre parti: la creazione dei modelli di rappresentazione dei dati, la creazione delle Viste sui concetti e l'implementazione della logica dell'applicazione realizzando le strutture di controllo adeguate.

I Modelli dei Concetti

ExtJs fornisce due strumenti per rappresentare le informazioni gestite dall'applicazione, i Modelli e gli Store. I primi contengono la definizione dei campi che rappresentano le entità e tramite gli Store gestiscono i rispettivi dati persistenti. I modelli possono essere collegati tra di loro tramite delle associazioni che rappresentano le relazioni esistenti tra le entità che rappresentano.

Questa struttura permette di mappare ciascun Concetto di CAO=S in un Model ExtJs che contiene la descrizione dei fields che lo compongono. Le relazioni di join tra i Concetti vengono rappresentate nel modello e gestite in maniera completamente automatica.

Viene poi generato anche uno Store per ciascun Concetto, che viene associato al rispettivo Model, e gestisce le comunicazioni con il Server per il recupero dei dati. Ciascuno Store contiene un Proxy Ajax con i riferimenti ai moduli PHP con i quali interagire, categorizzati secondo il modello CRUD in: create, read, update e destroy. Le comunicazioni Ajax avvengono tramite Json.

Dato che solo una parte degli elementi che compongono un modello vengono realmente salvati all'interno della tabella del database che lo rappresenta, nel momento in cui il controller dell'applicazione richiama una View, ad esempio un Form, lo Store, grazie alla definizione dei campi nel Model, si occupa 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. Lo Store, nel momento di creazione del componente Form, va a ricercare le informazioni degli elementi lookup presso altri Concetti seguendo le associazioni che vi sono fra i Modelli, 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.

L'Interfaccia

La creazione dell'interfaccia dell'applicazione si costruisce sulla base dell'analisi congiunta di due elementi di CAO=S nel file xml, le Strutture, composte da Viste e Navigazione, e la descrizione degli Attori e delle loro caratteristiche.

Le Viste sono rappresentate in una struttura Views (come in esempio 5.2.3) che è composta da diversi elementi *View*, che rappresentano per l'utente una specifica interfaccia che gli consente di effettuare operazioni sui dati.

```
<views>
   <view name="home">
    [...]
   </view>
   <view name="contattoForm">
    [...]
   </view>
</views>
```

Esempio 5.2: *Struttura delle viste di CAO=S*

Queste informazioni vengono utilizzate da Antropos per realizzare la rappresentazione dei dati nelle Viste che sono 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 di diverso tipo, in base agli elementi/strutture che la compongono, come: liste, sommari, form, navigazione, sottoviste, ecc. Di seguito analizziamo le tipologie gestite da Anthropos.

List Questa vista fa riferimento ad un concetto e definisce una struttura di visualizzazione dinamica di elementi dello stesso tipo (una griglia di istanze del medesimo concetto), e mostra come colonne tutti gli elementi del concetto contenuti in esso (solo quelli con l'attributo listable uguale a true). Una vista di tipo List viene

generata da Anthropos come un elemento Grid di ExtJs. Nel file xml del modello CAO=S la lista viene rappresentata nel modo come nell'esempio 5.2.3:

```
<view name="rubricaList">
   <list concept="contatto">
      <heading>Rubrica</heading>
      <operations>
         <select/>
         <multipleSelect/>
         <show view="contattoForm"/>
         <edit view="contattoForm"/>
         <sort/>
         <group/>
         <filter/>
      </operations>
      <navigation>
         <link op="create" concept="contatto" view="contattoForm">Crea
          Nuovo</link>
      </navigation>
   </list>
</view>
```

Esempio 5.3: *Struttura delle Liste di CAO=S*

Nella List sono presenti dei sotto elementi che la descrivono:

Heading: rappresenta l'intestazione della tabella

Operations: è un elemento che serve a definire le operazioni che possono essere effettuate su quella specifica lista. Anthropos effettuando il parsing di questo elemento, aggiunge alla lista gli elementi di navigazione necessari per compiere le operazioni (bottoni, link...) e al controller dell'applicazione la gestione delle operazioni richieste. 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 da 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 (un esempio 5.2.3).

```
<view name="rubricaCompact">
   <summary concept="contatto">
      <heading>Rubrica Contatti</heading>
      <content>Accedi ai tuoi contatti</content>
   </summary>
   <navigation>
      <link op="list" concept="contatto" view="rubricaList">Visualizza</link>
   </navigation>
</view>
```

Esempio 5.4: *Struttura Summary di CAO=S*

Form L'elemento form indica la realizzazione della omonima struttura HTML, che dia la possibilità all'utente di interagire con la vista stessa, è caratterizzato da un sotto elemento Heading, che ne definisce il contenuto dell'intestazione e dall'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 cancellarla. Un'ulteriore operazione è stata aggiunta con archive, 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. La struttura che rappresenta il form nel file xml è nell'esempio 5.2.3.

```
<view name="contattoForm">
   <form concept="contatto">
      <heading>Contatto Rubrica</heading>
      <operations>
         <create/>
      </operations>
   </form>
</view>
```

Esempio 5.5: *Struttura dell'elemento Form di CAO=S*

Subview Questo elemento permette di includere View all'interno di altre View, costruendone di tipo più complesso. Anthropos, grazie all'adozione dell'architettura

MVC, realizza un sistema di sviluppo dell'interfaccia a moduli intercambiabili in modo da poter utilizzare a proprio piacimento le varie View del sistema, ottenendo dei frammenti componibili in modo semplice. Un esempio di inclusione di Viste annidate è presente nell'esempio 5.2.3 frammento di XML CAO=S che descrive l'homepage di un'applicazione.

```
<view name="home">
   <subview>
      <view ref="rubricaCompact"/>
      <view ref="gruppiCompact"/>
   </subview>
</view>
```

Esempio 5.6: *Struttura dell'elemento SubView di CAO=S*

Navigation La navigazione è il meccanismo utilizzato per passare da una Vista ad un'altra; questa struttura non identifica solamente la barra di navigazione dell'interfaccia ma tutti gli elementi presenti nelle View a cui è associato un cambio di vista. Nella fase di creazione delle viste dell'applicazione, l'analisi della navigazione serve a creare all'interno delle viste gli elementi adeguati (bottoni, link, ecc.) per permettere di eseguire le operazioni.

Quello che caratterizza il lavoro di questa tesi è generare applicazioni che siano modificabili dinamicamente negli aspetti di interazione (interfaccia, navigazione e struttura), in base alle caratteristiche degli utenti che la utilizzano.

La dinamicità dell'interfaccia consiste nel modificare la struttura delle viste, aggiungendo ad esempio degli elementi che facilitino gli utenti ad utilizzarle. Per fare questo ci si avvale dell'utilizzo di widget, sviluppati grazie alla possibilità offerta dal framework ExtJs di definire tutte le componenti dell'applicazione secondo il paradigma object oriented. I widget infatti sono semplicemente delle classi che estendono le classi delle componenti di visualizzazione di ExtJs (come le Grid, i form, i fieldset ecc.).

Anthropos fornisce una libreria di widget organizzati in gruppi, in cui ciascun gruppo si focalizza su una caratteristica dell'utente (ad esempio la competenza nel dominio o l'abilità linguistica). All'interno dello stesso gruppo sono presenti widget che estendono diverse componenti dell'interfaccia, ad esempio Form e Grid. Ciascun widget è identificato in base ad un codice che lo associa a una precisa combinazione delle caratteristiche dell'utente e ogni widget quindi è realizzato in modo da facilitare l'utente che ha quelle precise caratteristiche.

Vengono prese in considerazione le caratteristiche dell'Attore definite in CAO=S, si valutano su una scala da 1 a 5 e si studia quale interfaccia sia più adatta all'utente con quelle caratteristiche. Ad esempio per la Competenza di dominio si possono identificare questi tipi di utenti:

1. Specialista della materia;
2. Conoscitore esperto;
3. Conoscitore medio del campo;
4. Persona con conoscenze vaghe e incomplete;
5. Completamente inesperto.

In questo caso il widget della vista Form che si rivolge all'Attore con valore 1 o 2 nelle competenze di dominio, dovrà avere un'interfaccia molto ridotta nei contenuti, perché questo tipo di utente non ha bisogno di spiegazioni sul significato dei campi che deve inserire, mentre l'Attore con valore maggiore di 3 viene aiutato con l'aggiunta di elementi di spiegazione.

L'analisi degli utenti effettuata da CAO=S produce la componente degli Attori come definiti dal modello stesso. Quindi Anthropos riceve in input la definizione degli attori in formato xml con la sintassi come nell'esempio 5.2.3: A partire da

```
<actor role="UtenteEsperto">
   <domain>2</domain>
   <technological>3</technological>
   <linguistic>3</linguistic>
   <physical>3</physical>
   <motivation>3</motivation>
   <distraction>3</distraction>
</actor>
```

Esempio 5.7: *Struttura dell'elemento Attore di CAO=S*

queste definizioni di Attori, Anthropos genera le View applicandoci i widget adeguati alle caratteristiche dell'Attore. Anthropos durante la generazione analizza gli Attori e le caratteristiche che li rappresentano e crea una View dell'applicazione per ogni Vista di ciascun tipo (Form, Grid, ecc.) e per ogni possibile utente. Le viste dello stesso tipo (ad esempio tutti i Form) e relative agli stessi tipi di utente, ma associate a Concetti diversi, estendono tutte lo stessa classe, ovvero lo stesso widget che contiene le configurazioni adeguate all'utente. Ne è visibile un esempio in figura 5.3.

Anthropos permette di gestire diversi tipi di utenti nella stessa applicazione e di cambiare dinamicamente l'interfaccia a seconda dell'Utente che la sta utilizzando.

Figura 5.3: *Cambiamento della visualizzazione dei campi di un form in base alle caratteristiche dell'utente al quale si rivolge: nel caso in cui non abbia molte competenze del dominio dell'applicazione viene mostrata un'interfaccia che fornisce delle spiegazioni aggiuntive e una visualizzazione più completa degli errori*

Controllo dell'applicazione

L'architettura MVC permette di separare le operazioni di gestione dei dati e visualizzazione degli stessi e affida al Controller il compito di fare interagire queste due componenti. All'interno del controller vengono implementate tutte le funzioni di Navigazione definite nelle Strutture di CAO=S e le operazioni definite all'interno delle singole View che richiedono l'apertura di un'altra View o l'interazione con lo Store (ad esempio create, show, update...).

Il Controller dell'applicazione, nel momento in cui viene richiesta la visualizzazione delle informazioni riguardanti un Concetto, prima reperisce tutte le informazioni interrogando lo Store e successivamente le trasmette alla View. La classe Controller in ExtJs implementa la funzione control che serve a catturare tutti gli eventi scatenati dall'interfaccia e associargli la corretta funzione di gestione.

L'avere questo meccanismo centralizzato di controllo permette di implementare in modo semplice una funzione molto importante delle applicazioni generate con Anthropos, ovvero la possibilità di cambiare dinamicamente l'aspetto delle Viste in base all'Utente che sta utilizzando l'applicazione. Durante la generazione vengono create le Viste adeguate a ciascun tipo di Utente e queste vengono nominate attraverso dei codici che rappresentano la combinazione di caratteristiche degli Attori che soffisfano. Nell'applicazione generata vengono memorizzati adeguatamente gli Attori che utilizzano l'applicazione ed è sempre possibile a run-time sapere quale

Utente sta interagendo con il programma. Il controller monitora ad ogni evento che viene lanciato la tipologia di Utente che sta utilizzando l'applicazione e carica la vista adeguata alle sue caratteristiche.

Capitolo 6

Valutazione

6.1 Strutturazione dei test

Per testare la generazione model-driven di applicazioni con Anthropos è stata realizzato un modello CAO=S che rappresenta un'applicazione di gestione di una rubrica di contatti. Il documento xml del modello di rubrica è stato costruito ad hoc perché non è stata ancora implementata l'applicazione CAO=S Analizer, che dovrebbe generarlo. Il modello di rubrica contiene due Concetti collegati tra loro da una relazione:

- il Contatto della rubrica, rappresentato tramite i campi: Nome, Cognome, e-mail, telefono e gruppo;
- e i Gruppi di contatti che hanno un nome e una descrizione.

Le Strutture da realizzare nell'applicazione sono le Viste seguenti, con le relative operazioni di Navigazione:

- l'Homepage dell'applicazione che contiene i Summary dei Contatti e dei Gruppi;
- i Summary dei Contatti e dei Gruppi, dai quali si accede alla visualizzazione dell'elenco dei medesimi concetti;
- le List dei due Concetti dell'applicazione, nelle quali vengono visualizzate le relative istanze ed è possibile crearne di nuove, modificare le esistenti;
- il Form dei Gruppi, dal quale si può creare un nuovo gruppo o modificarne uno esistente;

- il Form dei Contatto, dove si possono inserire o modificare i campi del Contatto e selezionare il Gruppo al quale esso appartiene.

I test sono stati eseguiti generando l'applicazione su un Server Apache/2.2.8 (Ubuntu) su un sistema Linux 2.6 con CPU Intel(R) Pentium(R) M (1.73GHz) 1024MB di RAM, ed utilizzando come Browser Firefox 3.6 eseguito anch'esso sulla stessa macchina Linux. Un ulteriore ambiente sul quale è stata testata l'applicazione è il server del dipartimento di computer science dell'università di Bologna, in questo caso l'applicazione generata è stata eseguita sul Browser Firefox nella medesima macchina Linux del test precedente e sui browser Firefox3.6 su una macchina WINDOWS XP HOME EDITION 2002 - SP3 con CPU : VIA NANO U2250 1440MHz e RAM: 2GB.

Lo scopo del test realizzato è quello di dimostrare la capacità di Anthropos di generare, a partire dal modello CAO=S, applicazioni che modificano dinamicamente le modalità di interazione dell'utente con l'applicazione, sulla base delle caratteristiche dell'utente. Per questo il modello dell'applicazione di rubrica include la definizione di due Attori distinti che rappresentano due tipologie di utenti dell'applicazione: uno con delle buone competenze di dominio e il secondo più inesperto.

6.2 Valutazione dell'efficienza

Il primo risultato che si può apprezzare dall'esecuzione di Anthropos con il modello di test rappresentate la rubrica è l'efficienza e la velocità di esecuzione del generatore. Risulta invece essere meno efficente l'esecuzione dell'applicazione della rubrica prodotta, il che è probabilmente dovuto al framework ExtJs che include nell'applicazione una libreria molto estesa, questa inefficenza si manifesta particolarmente nell'esecuzione sul server dell'università.

6.3 Valutazione dell'efficacia di Anthropos

Le prime considerazioni qualitative apprezzabili dai test eseguiti su Anthropos sono le caratteristiche di usabilità apprezzabili nell'interfaccia, grazie all'adozione del modello CAO=S.

L'adozione delle tecnologie Model-Driven porta a una semplificazione della pro-

gettazione e dell'implementazione richiesta a un team che utilizza CAO=S, per il quale le fasi di sviluppo si riducono a:

- Fase di analisi: nella quale il team di sviluppo descrive i concetti, gli attori e le operazioni del sistema da realizzare.
- Fase di generazione CAO=S con Anthropos: nella quale viene utilizzato Anthropos per realizzare automaticamente l'applicazione descritta nella precedente fase di analisi.
- Fase di monitoraggio: nella quale viene valutata l'applicazione generate e si identificano eventuali problemi, e possibili migliorie.

Grazie a questo strumento di sviluppo la progettazione di un'applicazione di complessità pari a quella della rubrica prodotta, che per quanto semplice richiederebbe a un programmatore di impiegare almeno un'intera giornata di lavoro e il supporto di una consulenza sull'usabilità, può essere generata in pochissime ore e in completa autonomia.

Infatti CAO=S, grazie alla realizzazione di un generatore model-driven come Anthropos, permette un'astrazione del lavoro di sviluppo, richiedendo solo la progettazione dell'applicazione tramite la modellazione di Concetti, Attori e Operazioni. Questo semplifica la progettazione di applicazioni usabili e lo rende alla portata di team di sviluppo medio-piccoli.

Anthropos utilizza pattern e tecnologie che permettono di renderlo un sistema facilmente ampliabile:

- il paradigma MVC realizzato nell'applicazione finale, la rendono facilmente scalabile;
- l'architettura model-driven permette la generazione automatica delle applicazioni riducendo gli errori di sviluppo del codice;
- il modello CAO=S permette di produrre un'applicazione che rispetti le principali caratteristiche di usabilità ed accessibilità;
- l'utilizzo delle tecnologie Web più moderne (AJAX, REST, PHP e ExtJs) rendono le applicazione realizzate da Anthropos facilmente fruibili ed efficienti;
- l'implementazione dei widget javascript intercambiabili rende facilmente ampliabile il software, con la gestione di ulteriori combinazioni di caratteristiche degli utenti.

Il risultato più evidente nel testing di Anthropos è la sua capacità di adattare dinamicamente l'interfaccia, aggiungendo degli elementi che facilitino in modo adeguato ciascun tipo di utente che la utilizza, in base alle sue caratteristiche. Per ciascun tipo di Attore definito, l'interazione dell'utente subisce dei cambiamenti molto evidenti nell'interfaccia come si può vedere in figura 6.1. La gestione degli attori in Anthropos comporta poche righe di codice e produce invece cambiamenti notevoli, questo è possibile grazie all'adozione dell'architettura Model View Controller, che rende facilmente applicabili all'interfaccia i widget di Anthropos.

Figura 6.1: *Screenshot della una vista Form generata da Anthropos per due diverse tipologie di utente; nel primo caso un utente esperto del dominio e nel secondo caso uno inesperto*

Capitolo 7

Conclusioni

In questa dissertazione sono stati esaminati gli aspetti di usabilità nell'interazione degli utenti con le applicazioni, utilizzando le tecniche di generazione model-driven. Sono stati esaminati gli studi realizzati fin'ora su questo ambito, constatando la presenza di diversi strumenti di generazione model-driven che però non si focalizzano efficacemente sugli aspetti di usabilità, oppure li affrontano solo a livello concettuale e di progettazione. D'altro canto i modelli di progettazione user-centered, sia Task-Oriented che Goal-Oriented, non forniscono delle soluzioni pratiche per lo sviluppo del software che possano essere di supporto agli sviluppatori. Il design delle applicazioni utilizzando il modello Goal-Oriented comporta un lavoro molto complesso di analisi degli utenti, focalizzandosi sui loro obiettivi. Questo richiede la consulenza di esperti di usabilità e quindi risulta troppo dispendioso per piccoli team di sviluppo.

A questo scopo è stato scelto di sviluppare un software model-driven che risolva queste mancanze, Anthropos, utilizzando un modello di sviluppo Goal-oriented innovativo chiamato CAO=S, permette di realizzare applicazioni che abbiano i requisiti essenziali di usabilità. Come visto nella dissertazione, CAO=S modifica il tradizione modello di sviluppo riducendo l'analisi degli utenti alle sole caratteristiche che hanno un impatto effettivo sull'interazione.

Attraverso questo modello si riescono a produrre strutture dati analizzando i tre componenti chiave che sono gli Attori, le Operazioni e i Concetti. Anthropos rappresenta il modulo di generazione automatica dell'applicazione nel modello CAO=S e permette attraverso l'acquisizione di un documento XML di sviluppare un'applicazione completa che permetta un'efficace interazione dell'utente con essa.

Tramite il test con il modello di un'applicazione di una rubrica viene dimostrato come Anthropos, grazie all'utilizzo dell'architettura model-driven, permette la

generazione dell'applicazione in modo automatizzato, fornendo le caratteristiche di usabilità essenziali per l'interazione.

Anthropos è stato sviluppato con una particolare attenzione ai linguaggi e ai pattern di sviluppo, scegliendo i più utili a rendere le applicazioni realizzate nel processo di generazione molto dinamiche e scalabili (Javascript e l'architettura Model View Controller in particolare). L'aspetto più innovativo introdotto da Anthropos è la realizzazione di un'interfaccia che si adatta dinamicamente alle caratteristiche dell'utente che la utilizza. Dai test effettuati si può apprezzare come realizzi un cambiamento sostanziale nell'interfaccia basandosi su poche righe di codice che descrivono le caratteristiche dell'utente.

Il modello di progettazione CAO=S attualmente è implementato solo nella parte di generazione Model-Driven e manca 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 Anthropos. Inoltre bisognerebbe ampliare e formalizzare lo schema del documento XML utilizzato come comunicazione tra CAO=S Analizer e CAO=S Generator, in modo da permettere lo sviluppo di nuovi generatori fornendo specifiche meglio definite.

Riguardo al prototipo di CAO=S Generator realizzato, Anthropos, questo sviluppa automaticamente applicazioni AJAX utilizzando un Framework Javascript, ExtJs, che fornisce due elementi fondamentali:

- le librerie di widget, ovvero classi che realizzano componenti standard dell'interfaccia come Griglie o Form, che sono facilmente estendibili creandone di nuove;

- il supporto alla strutturazione del codice secondo il pattern Model-View-Controller che permette di gestire semplicemente la dinamicità dell'applicazione nella navigazione e nel cambiamento dell'interfaccia.

Nello sviluppo di Anthropos sono stati realizzati alcuni widget relativi a particolari caratteristiche degli Attori definiti in CAO=S ma viene fornita la possibilità di crearne ancora moltissimi in modo semplice. Grazie alla struttura delle applicazioni generate i nuovi widget possono essere inseriti all'interno del prototipo realizzato in modo molto semplice.

Anthropos riesce a gestire diverse tipologie di utente all'interno della stessa applicazione e permette di realizzare dei cambiamenti nell'interfaccia in base all'utente

che la utilizza a run-time in modo dinamico. Un'ulteriore funzionalità che può essere introdotta è la gestione dell'evoluzione dell'utente, che se ha competenze basse su alcune caratteristiche, come ad esempio le competenze informatiche, progredendo nell'utilizzo del sistema queste potrebbero migliorare e si potrebbe quindi adeguare la descrizione dell'utente a questo miglioramento.

Bibliografia

Silvia Abrahao and Emilio Insfran. Early usability evaluation in model driven architecture environments. In *Proceedings of the Sixth International Conference on Quality Software 26-28 October 2006, Beijing, China.*, pages 287–294, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2718-3. doi: 10.1109/QSIC.2006.26. URL `http://dl.acm.org/citation.cfm?id=1190618.1191343`.

K. Andrews. Human-computer interaction lecture notes. July 1999. URL `http://www.iicm.edu/hci`.

T. Atkinson, C.; Kuhne. Model-driven development: a metamodeling foundation. *IEEE Software*, 20:36 – 41, 2003.

Cédric Bach and Dominique Scapin. Ergonomic criteria adapted to human virtual environment interaction. In *Proceedings of the 15th French-speaking conference on human-computer interaction*, IHM 2003, pages 24–31, New York, NY, USA, 2003. ACM. ISBN 1-58113-803-2. doi: http://doi.acm.org/10.1145/1063669.1063674. URL `http://doi.acm.org/10.1145/1063669.1063674`.

Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005. ISBN 0321267974.

Stefano Ceri, Piero Fraternali, and Maristella Matera. Conceptual modeling of data-intensive web applications. *IEEE Internet Computing*, 6:20–30, July 2002. ISSN 1089-7801. doi: 10.1109/MIC.2002.1020321. URL `http://dl.acm.org/citation.cfm?id=613355.613720`.

Alan Cooper, Robert Reimann, and Dave Cronin. *About Face 3: the essentials of interaction design*. John Wiley & Sons, Inc., New York, NY, USA, 2007. ISBN 9780470084113.

Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. URL `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`. University of California, Irvine.

Krzysztof Z. Gajos, Daniel S. Weld, and Jacob O. Wobbrock. Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, 174:910–950, August 2010. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/j.artint.2010.05.005. URL `http://dx.doi.org/10.1016/j.artint.2010.05.005`.

Avraham Leff and James T. Rayfield. Web-application development using the model/view/controller design pattern. In *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing 4-7 September 2001, Seattle, WA, USA*, EDOC '01, pages 118–130, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1345-X. URL `http://dl.acm.org/citation.cfm?id=645344.650161`.

C. Lewis and J. Rieman. *Task-Centered User Interface Design: A Practical Introduction*. 1993. URL `ftp://ftp.cs.colorado.edu/pub/cs/distribs/clewis/HCI-Design-Book/`.

Esteban Robles Luna, José Ignacio Panach, Julián Grigera, Gustavo Rossi, and OSCAR PASTOR. Incorporating usability requirements in a test/model-driven web engineering approach. *Journal of Web Engineering*, 9:132–156, June 2010. ISSN 1540-9589. URL `http://dl.acm.org/citation.cfm?id=2011309.2011312`.

Pedro J. Molina. User interface generation with olivanova model execution system. In *Proceedings of the 9th international conference on Intelligent user interfaces, Funchal, Portugal — January 13 - 16*, IUI '04, pages 358–359, New York, NY, USA, 2004. ACM. ISBN 1-58113-815-6. doi: http://doi.acm.org/10.1145/964442.964533. URL `http://doi.acm.org/10.1145/964442.964533`.

Fabio Paternò, Carmen Santoro, Jani Mantyjarvi, Giulio Mori, and Sandro Sansone. Authoring pervasive multimodal user interfaces. *Journal of Web Engineering*, 4:235–261, May 2008a. ISSN 1476-1289. doi: 10.1504/IJWET.2008.018099. URL `http://dl.acm.org/citation.cfm?id=1366965.1366970`.

Fabio Paternò, Carmen Santoro, and Antonio Scorcia. User interface migration between mobile devices and digital tv. In *Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on*

Task Models and Diagrams, Pisa, Italy, September 25-26, HCSE-TAMODIA '08, pages 287–292, Berlin, Heidelberg, 2008b. Springer-Verlag. ISBN 978-3-540-85991-8. doi: http://dx.doi.org/10.1007/978-3-540-85992-5_28. URL `http://dx.doi.org/10.1007/978-3-540-85992-5_28`.

Schmidt. Model-driven engineering. *IEEE Computer Society*, 1:1, 2006. URL `www.cs.wustl.edu/~schmidt/GEI.pdf`.

Jean-Sébastien Sottet, Gaëlle Calvary, Joëlle Coutaz, and Jean-Marie Favre. A model-driven engineering approach for the usability of plastic user interfaces. In Jan Gulliksen, Morton Harning, Philippe Palanque, Gerrit van der Veer, and Janet Wesson, editors, *Engineering Interactive Systems*, Lecture Notes in Computer Science, pages 140–157. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-92697-9. doi: http://dx.doi.org/10.1007/978-3-540-92698-6_9. URL `http://dx.doi.org/10.1007/978-3-540-92698-6_9`.