POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione

Corso di Laurea Specialistica in
Ingegneria Informatica

Editor WYSIWYG come supporto per lo sviluppo Model Driven

Relatore: Prof. Marco Brambilla
Correlatore: Dott. Ing. Alessandro Bozzon

Tesi di laurea di: Marco Rivera
matr. 717686

Anno Accademico 2010/11

Indice

Indice delle figure

Indice delle tabelle

Capitolo 1

Introduzione

Le applicazioni Web hanno avuto una enorme diffusione, che si è spinta ben oltre il semplice scambio di documenti per cui il Web era stato concepito. La grande richiesta di questo tipo di software, sia di nuova concezione che come risultato della re-ingegnerizzazione dei sistemi esistenti, associata alla mancanza cronica di esperti IT, ha reso ancora più evidente l'esigenza di migliori pratiche di ingegneria del software. Per migliorare la produttività, è necessario un supporto più ampio al processo di sviluppo di un sito Web, dato che la maggior parte degli strumenti disponibili sul mercato sono concentrati solamente sulle fasi di progettazione e implementazione. Alle fasi di analisi e modellazione concettuale, invece, viene dato poco risalto. Come conseguenza, l'implementazione e la manutenzione di un'applicazione Web è tuttora un'attività che richiede molta manodopera e risulta essere piuttosto propensa agli errori. I benefici introdotti dalla formalizzazione del processo di sviluppo, supportato da specifiche notazioni e strumenti CASE, sono scarsamente sfruttati.

Per far fronte a tali esigenze, la comunità di ricerca ha proposto molti differenti approcci per la cosiddetta progettazione Model Driven di applicazioni Web, che condividono l'idea di sfruttare notazioni semi-

formali per esprimere la struttura dei dati e la topologia degli ipertesti di un sito Web e di utilizzare specifiche concettuali per guidare le fasi di progettazione e implementazione. Tali specifiche, in molti casi, derivano direttamente da rami più maturi dell'ingegneria del software, come gli schemi Entità-Relazione tipici delle basi di dati o i diagrammi UML utilizzati nelle fasi di analisi. La modellazione del back-end di un'applicazione Web è quindi ben supportata da un gran numero di strumenti, che introducono efficacemente l'approccio Model Driven in tutte le sue fasi.

Ciò che risulta più difficile da specificare in modo formale è l'aspetto visuale delle applicazioni: ciò che l'utente vede e che gli permette di interagire con l'intero sistema. Molti approcci alla modellazione delle interfacce utente sono stati proposti, ma esiste tuttora un numero limitato di modelli e di strumenti sufficientemente maturi da essere riconosciuti per un utilizzo commerciale. Il motivo fondamentale di questa situazione risiede nel fatto che gli approcci proposti non sono abbastanza efficaci nel gestire la complessità delle interfacce utente. Tuttavia, la GUI (Graphic User Interface) di un'applicazione è spesso uno dei fattori chiave che ne determinano il successo. Sebbene le funzionalità che un software fornisce agli utenti siano fondamentali, le modalità con cui esso fornisce tali funzionalità sono altrettanto importanti. Un'applicazione difficile da usare non verrà utilizzata; non importa quanto sia tecnicamente superiore o quali funzionalià espone: se non piace agli utenti finali, essi non la utilizzeranno.

Si capisce, quindi, come l'attuale processo di generazione delle applicazioni Web secondo un approccio Model Driven risulta ancora separato in due parti ben distinte. Mentre la creazione della logica di business sfrutta appieno i vantaggi di una simile filosofia, non si può dire altrettanto per quanto riguarda la generazione del front-end ed, in particolare, dell'aspetto visuale del software. Da qui è nata la necessità di studiare un nuovo workflow, che unificasse i due processi attualmente esistenti.

Pertanto, la mancanza di un processo unificato che rendesse possibile un approccio di alto livello anche durante la generazione dell'aspetto visuale degli applicativi, è stato il motivo scatenante che ha dato il via al presente progetto.

1.1 Obiettivi

Alla prima fase di analisi delle problematiche attuali, è seguito un processo che ha portato alla creazione di un nuovo flusso di lavoro. Sono state studiate le modalità con cui integrare l'approccio Model Driven all'interno dell'intero processo di sviluppo di un'applicazione Web, in modo da estenderne i benefici anche alla fase di generazione dell'interfaccia utente.

Le attività di ricerca hanno avuto come sbocco naturale la creazione di un nuovo editor WYSIWYG, orientato alla modellazione diretta dell'interfaccia di un'applicazione Web. L'aspetto innovativo della soluzione proposta risiede nel suo livello di astrazione: esso risulta più elevato rispetto agli editor visuali tipicamente utilizzati per costruire interfacce grafiche. Infatti, l'approccio Model Driven è ciò che la distingue dai prodotti attualmente esistenti: piuttosto che essere orientato alla manipolazione grafica del codice sorgente, l'editor WYSIWYG Model Driven ha come obiettivo la generazione diretta di un modello di presentazione. Le stesse modalità di interazione con l'utente tengono conto di questa differenza: invece che concentrarsi solamente sull'aspetto visuale, lo strumento interagisce direttamente con la specifica della logica di business dell'applicazione. In questo modo, l'anteprima generata all'interno dell'editor WYSIWYG Model Driven risulta coerente con quanto specificato nel back-end. Il risultato finale è una resa molto realistica dell'interfaccia che presenterà l'applicazione finale. In questo modo, l'approccio attualmente utilizzato durante la creazione

del back-end è stato esteso anche alla generazione del front-end.

1.2 Caso di studio

Il linguaggio WebML è una delle proposte per la specifica concettuale e implementazione automatica di applicazioni Web attualmente esistenti. Esso è direttamente supportato dallo strumento di sviluppo WebRatio, che trasforma le specifiche visuali di WebML in template di pagina lato server e query per basi di dati, che costituiscono il sito Web desiderato.

WebML e WebRatio sono stati scelti come casi di studio per il presente progetto, dato che rappresentano un esempio tipico di quanto appena esposto: essi utilizzano un approccio Model Driven durante tutte le fasi che portano dalla descrizione della struttura dei dati di un'applicazione alla definizione degli ipertesti che specificano in quale modo la logica di business manipola le informazioni disponibili. L'ultima fase di generazione dell'aspetto visuale del software, invece, utilizza metodologie di programmazione tradizionali. Tutti i vantaggi del paradigma Model Driven, quindi, vengono meno: le fase di analisi e modellazione concettuale non sono direttamente supportate e l'implementazione assume un ruolo predominante.

Lungo il flusso di sviluppo di un'applicazione Web, lo sviluppatore si trova così ad affrontare due processi ben distinti: in una prima parte, è dato grande risalto all'aspetto progettuale, in modo che lo sviluppatore si possa concentrare sul dominio del problema; nell'ultima fase, invece, il compito preponderante riguarda la risoluzione dei problemi implementativi. La scrittura manuale del codice risulta ancora fondamentale per la creazione dell'interfaccia utente. Inoltre, gli

strumenti a disposizione non aiutano a creare un aspetto visuale coerente con quanto definito nella struttura dei dati e nella topologia degli ipertesti. Tutto il processo si basa totalmente sulle capacità implementative dello sviluppatore.

Il fine ultimo del presente progetto è stato quello di risolvere le problematiche evidenziate. Un primo prototipo di un editor WYSIWYG Model Driven è stato quindi integrato all'interno di WebRatio. In questo modo, è stato possibile generare un nuovo flusso di lavoro che utilizzasse il paradigma Model Driven lungo tutte le fasi del processo di sviluppo. Dalla descrizione della struttura dei dati, alla specifica della topologia degli ipertesti, fino alla progettazione dell'interfaccia grafica, quindi, il processo segue un approccio omogeneo di alto livello, in cui gli aspetti di progettazione sostituiscono quelli implementativi. Infine, il prototipo ha rappresentato un supporto concreto che avvalora e dimostra l'applicabilità pratica delle soluzioni proposte.

Il fatto che si tratti di una prima implementazione non significa che il prototipo creato sia fine a se stesso. Anzi, è stato sviluppato fin dall'inizio come un vero e proprio prodotto destinato alla produzione. Quanto descritto si applica quindi molto bene al mondo reale e non presenta una valenza solamente teorica.

1.3 Organizzazione del documento

Il Capitolo 2 viene introdotto l'argomento cardine dell'intero elaborato: l'utilizzo di un approccio Model Driven per la creazione di interfacce per applicazioni Web. Vengono quindi descritti gli argomenti di base che saranno al centro dell'intera trattazione. In particolare, viene spiegato cosa significa l'espressione WYSIWYG, per quale motivo vengono utilizzati gli editor visuali, cosa si intende per sviluppo Model Driven e in che modo questi mondi possono collaborare. Il Capitolo continua con la descrizione del contesto attuale in cui si inserisce la trattazione. Si comincia con l'introduzione di WebML e di WebRatio, rispettivamente il linguaggio e lo strumento di supporto che sono stati utilizzati come base di partenza per sviluppare il lavoro di ricerca. La trattazione prosegue quindi con l'analisi del workflow attuale, evidenziando quelli che sono risultati essere i punti critici e le modalità con cui si intende risolverli. Il capitolo si conclude con una comparazione delle funzionalità di alcuni editor WYSIWYG disponibili attualmente.

L'analisi dei requisiti e le descrizione degli obiettivi del lavoro vengono esposte nel Capitolo 3. Viene quindi descritto in che modo si sviluppa il nuovo flusso di lavoro, insieme al modello di integrazione studiato appositamente per WebRatio.

Nel Capitolo 4 viene descritta la fase implementativa del prototipo. Particolare attenzione è rivolta alle modalità con cui sono stati risolti i problemi evidenziati. Di pari passo, vengono spiegate le scelte che hanno portato alla creazione di una prima versione del prototipo.

Il Capitolo 5 conclude l'elaborato fornendo diverse considerazioni riguardanti il lavoro svolto e gli eventuali scenari futuri che si sono aperti grazie alle analisi effettuate.

Capitolo 2

Contesto attuale

La prima parte del lavoro svolto si è concentrata sullo studio della situazione attuale. Si è preso in considerazione l'intero processo di sviluppo di un'applicazione Web che segue un approccio Model Driven. Particolare attenzione è stata posta alla fase di creazione delle interfacce. Non sono state nemmeno trascurate le modalità con cui tale processo si inserisce all'interno del più ampio ciclo di sviluppo di un software MDA, sottolineando i punti critici su cui si basa il lavoro svolto.

L'analisi prosegue con la descrizione del linguaggio WebML e del suo strumento di supporto WebRatio. Pertanto, anche se alcune considerazioni sono di carattere generale e applicabili all'intera filosofia Model Driven, la trattazione risulta focalizzata sulle caratteristiche distintive di WebML e di WebRatio.

Prima di tutto, viene introdotto l'argomento cardine dell'intero elaborato, lo sviluppo MDD, e vienedescritto come si presenta il processo di sviluppo nella sua versione attuale, per sottolinearne i punti critici che saranno presi in esame anche nei capitoli successivi.

2.1 Sviluppo Model Driven

La cosiddetta Model Driven Architecture (MDA) rappresenta un nuovo approccio per lo sviluppo software definito dall'Object Management Group. Lo sviluppo di applicazioni segue una filosofia del tutto differente rispetto a quello tradizionale: ci si concentra sulla generazione di modelli e sulla loro trasformazione, in un processo che termina con la creazione del codice vero e proprio del software finale. La specifica dei sistemi software, quindi, risulta del tutto indipendente da uno specifico linguaggio di programmazione: le applicazioni vengono generate in maniera del tutto indipendente dalla tecnologia su cui verranno eseguite. In questo modo, gli investimenti effettuati per la costruzione dei sistemi vengono preservati durante l'evoluzione tecnologica.

L'idea che sta alla base di questo principio è che il modello diventa il "codice sorgente" del sistema da cui, in seguito, vengono semplicemente generati gli eseguibili. In questo modo, i modelli possono ricoprire differenti livelli di astrazione, spaziando dai diagrammi concettuali nello spazio del problema fino ai modelli dettagliati a basso livello specifici di una determinata piattaforma. In generale, il Model Driven Development (MDD) permette di generare applicazioni a partire da modelli formali. Il processo inizia con la creazione di un modello indipendente dalla piattaforma (PIM), che viene poi trasformato in un modello specifico di una determinata piattaforma (PSM) e, infine, tradotto in codice sorgente. Si noti come, in realtà, questi due modelli rappresentano modi differenti di descrivere lo stesso sistema. Il Platform Independent Model è una

rappresentazione della logica di business e del comportamento dell'applicazione e non contiene alcun dettagli relativo alla tecnologia utilizzata per la sua implementazione. Il Platform Specific Model è una rappresentazione di più basso livello del sistema, del tutto dipendente dalla piattaforma tecnologica a cui si riferisce. Pertanto, per implementare un PIM su di una specifica piattaforma è necessario uno strumento capace di trasformare un PIM in un PSM: esso deve conoscere la tecnologia di destinazione ed essere in grado di tradurre gli artefatti contenuti nel PIM in componenti specifici del PSM. Ad esempio, un singolo oggetto della logica di business (e quindi facente parte del PIM) può essere trasformato in una tabella di una base dati o in un Entity Bean, a seconda della tipologia di PSM scelta.

Aspetto fondamentale di MDA è la separazione dei problemi. In questo modo, lo sviluppatore si può concentrare sulle attività di analisi e progettazione, avendo bene a mente il dominio del problema. Le problematiche relative a tutti gli aspetti legati alla fase implementativa possono essere trascurati. Si ottiene così una separazione tra gli aspetti di progettazione e gli aspetti implementativi veri e propri, che rischiano di distogliere l'attenzione dal tema principale che ha spinto alla creazione del software.

MDA è stata fin da subito considerata come lo strumento in grado di migliorare la produttività del processo di sviluppo del software, riducendone i costi e i tempi. Grazie alla notevole diminuzione dello sforzo implementativo, è possibile effettuare rilasci in un periodo di

tempo di gran lunga inferiore rispetto ai metodi tradizionali; anche eventuali attività di prototipazione vengono notevolmente semplificate. Non da meno, il processo di sviluppo risulta maggiormente strutturato, favorendo la divisione dei compiti all'interno del gruppo di lavoro. Attraverso l'utilizzo di un approccio Model Driven, quindi, la qualità delle applicazioni generate, il grado di riutilizzo e, di conseguenza, l'efficienza di sviluppo vengono di gran lunga incrementate. Il successo dell'approccio Model Driven, inoltre, è strettamente associato alla sua usabilità pratica. Se, infatti, la creazione del modello fosse più problematica dell'implementazione delle funzionalità all'interno del codice, difficilmente gli sviluppatori sarebbero interessati al passaggio verso questa filosofia di sviluppo.

Per permettere la generazione automatica del codice a partire da un modello, quest'ultimo deve seguire una sintassi e una semantica ben definite. Il più diffuso linguaggio per raffigurare tali modelli è l'Unified Modeling Language (UML), anche se lo sviluppo Model Driven non è necessariamente legato a tale linguaggio. Come vedremo, infatti, l'intero elaborato si basa su WebML, un linguaggio creato specificatamente per la modellazione di applicazioni Web.

2.2 Editor WYSIWYG

L'espressione WYSIWYG non rappresenta altro che l'acronimo inglese “What You See Is What You Get” ("quello che vedi è quello che ottieni" oppure "ottieni ciò che vedi"). Tale termine è molto utilizzato nel campo dell'informatica, in cui assume sostanzialmente tre significati.

Il primo significato si riferisce al problema di ottenere sulla carta testo o immagini che abbiano una disposizione grafica corrispondente a quella visualizzata sul monitor del computer. Le prime stampanti, infatti, non riuscivano a fornire in stampa risultati pienamente corrispondenti a quelli generati dal software. Con il tempo, il significato dell'acronimo si è esteso per analogia anche ad alcune problematiche nella creazione di pagine Web. Infatti, per una scelta specifica dei suoi creatori, il linguaggio HTML descrive le pagine in maniera logica, senza fornire alcuna informazione sulla disposizione grafica degli elementi al dispositivo o al browser che dovrà interpretare il codice stesso. L'ultimo significato, infine, si riferisce a quegli editor HTML che permettono di modificare pagine Web senza ricorrere alla scrittura del codice, bensì come in un normale word processor.

L'ultima accezione è sicuramente quella che ci interessa maggiormente: si può considerare un editor WYSIWYG come uno strumento che implica in maniera diretta l'utilizzo di una interfaccia utente per permettere la manipolazione diretta della struttura di un documento, senza che l'utilizzatore sia in grado di scrivere le regole che generano la struttura stessa.

Ciò che attrae in modo particolare di un simile strumento è la semplicità di utilizzo e l'immediatezza con cui l'utente è in grado di visualizzare ciò che sta creando. Naturalmente, il livello di aderenza al risultato finale vero e proprio dipende dallo scopo per cui l'editor è stato creato. Infatti, nel caso in cui si voglia ottenere un'anteprima realistica del lavoro svolto, l'obiettivo di un editor WYSIWYG sarà sicuramente quello di generare una resa il più fedele possibile al risultato finale. Al contrario, se lo strumento deve servire da supporto per la creazione o modifica di un documento, la presenza di strumenti utili per la manipolazione del documento stesso (come strumenti di selezione, elementi grafici di guida, …) può comunque essere accettata, anche se può causare una resa visuale meno precisa.

Un simile approccio non presenta solamente aspetti positivi: un editor WYSIWYG si limita a creare il codice che, secondo le sue logiche interne, genera l'aspetto visuale desiderato. Se per un utente con limitate conoscenze tale comportamento può essere accettato, per uno sviluppatore con una certa esperienza può rappresentare un limite. Se, inoltre, ciò che viene generato risulta poco leggibile, si possono creare problemi di manutenibilità del codice stesso. Problemi che posso verificarsi anche nel momento in cui si rende necessario il passaggio da un editor WYSIWYG ad un altro.

2.3 Editor WYSIWYG per sviluppo Model Driven

I modelli, grazie al loro elevato livello di astrazione, si prestano bene ad essere rappresentati per mezzo di schemi o diagrammi, che ne aumentano la leggibilità e la semplicità d'uso. Un esempio molto evidente è rappresentato da un linguaggio di modellazione molto diffuso come UML. Tutti i suoi diagrammi, infatti, possiedono una rappresentazione visuale e sono costruiti componendo elementi grafici (con un significato formalmente definito), elementi testuali formali ed elementi di testo libero.

In un modo del tutto simile, un qualsiasi modello può essere costruito utilizzando alcuni elementi di base; questi ultimi costituiranno quindi i mattoni fondamentali con cui può avvenire la generazione. Naturalmente, affinchè il modello abbia un senso ben definito, è necessario specificare una sintassi e delle regole di interpretazione che ne permettano una semplice comprensione. Grazie alla possibilità di formalizzare una semantica precisa e grazie al grande potere descrittivo intrinseco, i modelli possono essere utilizzati per rappresentare praticamente ogni cosa.

Un approccio complementare è utilizzato dagli editor WYSIWYG: partendo da una lista di componenti di base, mettono a disposizione gli strumenti necessari per creare veri e propri diagrammi che si posizionano ad un livello superiore rispetto agli elementi di cui sono costituiti. Attraverso l'utilizzo di opportuni accorgimenti, come la possibilità di intervenire direttamente sugli elementi grafici con comode operazioni di

trascinamento, si ottiene una grande immediatezza e semplicità d'uso.

Si capisce immediatamente come gli editor WYSIWYG rappresentino uno strumento ideale per la costruzione di modelli. Di conseguenza, essi si prestano molto bene ad essere utilizzati all'interno del flusso di sviluppo software che utilizza un approccio Model Driven. Infatti, esistono molteplici strumenti, WebRatio compreso, che utilizzano un simile paradigma a supporto delle operazioni di modellazione. Inoltre, questo tipo di strumenti è fortemente disaccoppiato dalla piattaforma tecnologica per cui sono utilizzati: i principi di base, infatti, si applicano indistintamente al tipo di diagramma per cui vengono utilizzati. Quello che cambia nella maggior parte dei casi sono gli elementi grafici di base e la loro semantica, ma le operazioni di modellazione risultano per lo più comuni.

Nonostante la filosofia Model Driven e l'approccio degli editor WYSIWYG si sposino molto bene, l'utilizzo dei secondi a supporto dei primi presenta ancora delle problematiche. Gli editor visuali sono stati diffusamente utilizzati a supporto della modellazione, ma il flusso di sviluppo delle applicazioni Web non risulta ancora del tutto omogeneo. Modelli come quelli di WebML contengono al loro interno informazioni sufficienti per la creazione sia del front-end che del back-end di un software. Tuttavia, il supporto visuale alla modellazione della logica di business ha avuto un maggiore impiego rispetto alla creazione delle interfacce. Molto probabilmente ciò è dovuto al fatto che le funzionalità e le procedure tipiche del back-end di un'applicazione seguono modalità

specifiche che sono facilmente riproducibili. Considerando il front-end, invece, si vengono a sommare molteplici fattori che non coinvolgono solamente l'aspetto tecnologico e che risultano difficilmente modellabili. Durante lo sviluppo di un software, ci si trova quindi ad utilizzare strumenti di modellazione visuale per la creazione del back-end; nel momento in cui deve essere generato l'aspetto visuale dell'applicazione, molto spesso si deve ricorrere a metodi di lavoro più tradizionali. Questo flusso di lavoro risulta profondamente separato e va a scontrarsi con quelli che sono i principi dell'approccio Model Driven.

Come abbiamo visto, però, l'uso di MDD è in grado di generare notevoli vantaggi durante l'intera fase di sviluppo di un software. Tali vantaggi spingono alla ricerca di un meccanismo che permetta l'utilizzo degli strumenti di modellazione anche quando si ha a che fare con il front-end di un'applicazione. Una semplice considerazione ha portato all'idea di base del presente elaborato: gli editor WYSIWYG rappresentano un efficace strumento di supporto alla modellazione e per la creazione della grafica delle applicazioni esistono già modelli ben definiti.

L'intero lavoro svolto è stato concentrato sulla creazione di un flusso di lavoro omogeneo, che, tramite l'utilizzo di un editor WYSIWYG appositamente costruito, permettesse l'utilizzo di un approccio Model Driven durante la creazione dell'intera applicazione, interfaccia compresa. Il modello è stato pertanto il collante che ha permesso l'omogeneizzazione di un processo che, attualmente, si presenta profondamente diviso.

2.4 Flusso di lavoro “as-is”

L'utilizzo di un approccio Model Driven per lo sviluppo di un'applicazione Web richiede un processo strutturato, che segue dei passi ben precisi. Infatti, ogni modello risulta in qualche modo associato al precedente: come la logica di business ha bisogno di conoscere i dati che dovrà manipolare, così la logica di presentazione deve sapere quali funzionalità espone il back-end. Tipicamente, in seguito all'analisi dei requisiti e alla progettazione del modello dei dati, il flusso di lavoro prosegue con la creazione del modello di ipertesto, che racchiude in sé la specifica del comportamento e della struttura dell'intera applicazione.

La logica di business risulta così interamente definita ed è pertanto disponibile per il passo successivo, la creazione dell'aspetto visuale del software. In questa fase successiva, vengono create tutte le interfacce che costituiscono il front-end dell'intero applicativo e permettono all'utente di interagire con il sistema.

L'ultima parte del processo, che riguarda la trasformazione del PIM nel corrispondente PSM e quindi nel codice vero e proprio, comprende regole ben definite che trasformano gli artefatti presenti nei tre modelli in componenti specifici della piattaforma di riferimento. Per questo motivo, quest'ultima parte può essere completamente automatizzata e come risultato finale genera un'applicazione Web pronta per essere distribuita sulla piattaforma tecnologica scelta. Quest'ultimo passo è solitamente gestito da un generatore di codice che sfrutta appieno le possibilità di automazione offerte, implementando le regole di traduzione

in un processo automatico che non richiede l'intervento dello sviluppatore. L'output generato non è altro che l'implementazione delle funzionalità specificate all'interno dell'intero modello.

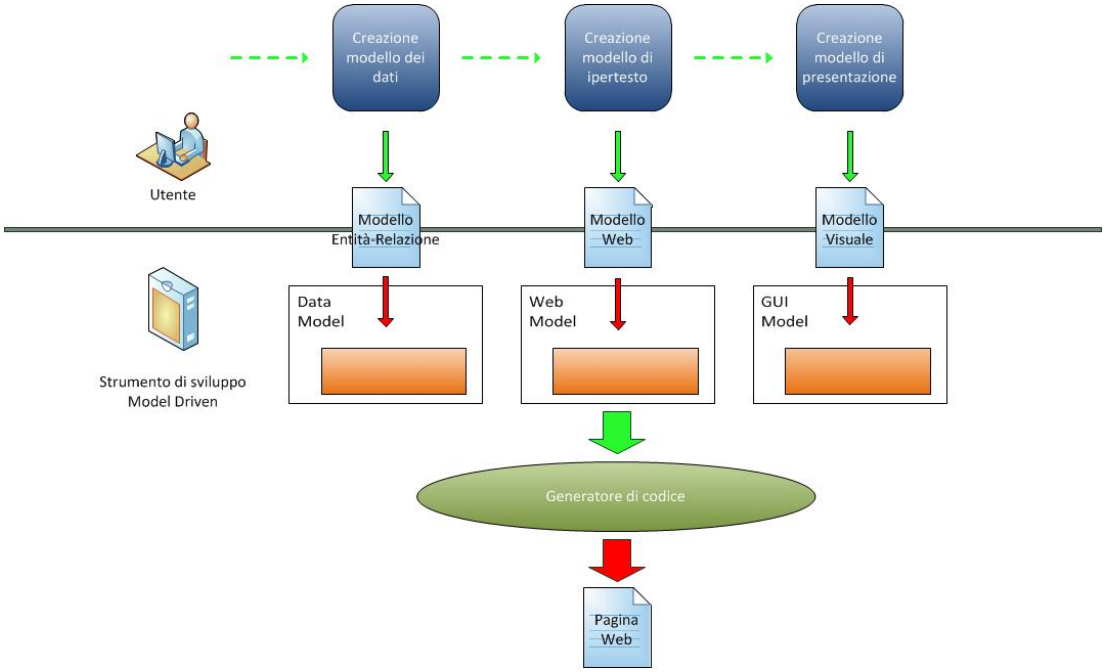

Figura 2.1: Esempio del flusso di creazione di una applicazione Web tramite uno strumento di sviluppo Model Driven

Nel seguire l'intero flusso di lavoro, però, ci si trova ad affrontare due approcci molto differenti tra loro. Nella prima parte, la modellazione dei diagrammi dei dati e della logica di business risulta supportata da strumenti che evitano totalmente il ricorso alla scrittura del codice sorgente. Nel primo caso, sono presenti tutte le funzionalità necessarie per la progettazione di un diagramma dei dati e la sua effettiva implementazione all'interno di una base di dati. Il diagramma che rappresenta il modello di ipertesto può essere direttamente modificato in maniera visuale, per mezzo di interfacce che riprendono i principi di un

editor WYSIWYG. Inoltre, i riferimenti tra i modelli sono automaticamente gestiti dall'applicazione, in modo che il modello di ipertesto abbia accesso completo ai dati che deve utilizzare. L'utilizzo di funzionalità di alto livello, che permettono di intervenire direttamente sulla specifica del modello, semplificano enormemente il compito dello sviluppatore, che può così concentrarsi sugli aspetti di progettazione. L'approccio Model Driven risulta quindi totalmente integrato all'interno del workflow che porta alla creazione dei primi due modelli.

Una situazione del tutto differente si incontra nel momento in cui si passa alla creazione del modello di presentazione. Risulta evidente come esiste una netta spaccatura, che costringe a ricorre a metodologie di sviluppo tradizionali. Infatti, tale fase richiede necessariamente la scrittura manuale del codice, dato che non è presente alcuna interazione diretta con il modello di ipertesto. In questo modo, vengono limitati e in alcuni casi addirittura annullati i vantaggi ottenuti dall'utilizzo di un approccio Model Driven. Infatti, sono del tutto assenti strumenti specifici che permettano la modellazione diretta del diagramma di presentazione. Lo sviluppatore si trova quindi a seguire un flusso di lavoro totalmente basato sulla propria esperienza e ad alta probabilità di errore.

Se per le prime due fasi del processo di sviluppo risulta evidente l'utilizzo della filosofia Model Driven, lo stesso non può essere detto per la parte che riguarda il modello di presentazione. I vantaggi derivati dall'approccio MDD sono sfruttati pienamente solamente durante la progettazione del modello dei dati e del modello di ipertesto. Lo

sviluppatore, infatti, è obbligato a ricorrere alla scrittura del codice che rappresenta la resa visuale dei componenti che costituiscono l'ipertesto. Non esistono nemmeno strumenti specifici per la creazione del modello di presentazione: l'intero processo deve essere gestito dall'utente, che si trova costretto a ricorrere ad una metodologia di sviluppo tradizionale. Le relazioni che intercorrono tra il modello di presentazione e il modello di ipertesto non sono nemmeno considerate: non viene effettuato alcun controllo di consistenza tra quanto creato manualmente per le interfacce e quanto contenuto all'interno degli altri due modelli. L'unico momento di verifica per lo sviluppatore risulta essere la generazione dell'applicazione vera e propria. In caso di errori, quindi, il processo può dilungarsi enormemente, vista la necessità di ricorrere spesso al generatore di codice.

Anche se alcuni aspetti tipici dell'approccio Model Driven sono presenti all'interno dell'intero flusso di lavoro che porta alla generazione di un'applicazione Web, come l'alto grado di riusabilità degli artefatti generati, un approccio omogeneo durante l'intero ciclo di sviluppo risulta senz'altro preferibile. I vantaggi dello sviluppo MDD sono completi solamente se la metodologia si applica all'intero workflow.

2.5 WebML

WebML (Web Modeling Language) è una notazione visuale per la specifica della composizione e della navigazione di applicazioni per il Web, che utilizza un processo di sviluppo di tipo MDA/MDD. Esso si basa su standard diffusi, come il modello Entità-Relazione e il linguaggio UML, e consente di specificare applicativi complessi in modo indipendente dalla piattaforma su cui verranno eseguiti. WebML è composto da un insieme di modelli, indirizzati a specificare aspetti differenti delle applicazioni Web:

- modello dei dati
- modello di ipertesto
- modello di presentazione

Nelle sue recenti estensioni include anche la specifica di processi di business e di integrazione di Web Services.

Fin dalla sua creazione, WebMLè stato pensato per supportare al meglio il processo di sviluppo delle applicazioni orientate al Web e, per questo motivo, si basa su dei principi fondamentali che mirano a semplificare il lavoro degli ingegneri del software e degli sviluppatori in generale:

- Espressività: il modello deve essere capace di descrivere le applicazioni Web ad un livello di complessità paragonabile a quello di sistemi sviluppati manualmente;
- Facilità d'uso: il modello deve essere semplice da imparare per

quegli sviluppatori non esperti di ingegneria del software;

- Implementabilità: il modello deve contenere informazioni sufficienti per permettere la generazione del codice di tutti i tier di un'applicazione Web dinamica; il generatore di codice deve produrre codice ottimizzato.

Allo stesso tempo, però, WebML fa in modo che il suo modello contenga il minor numero possibile di concetti che permettono la generazione del codice. Durante la sua evoluzione, infatti, WebML è stato coinvolto in numerose revisioni orientate all'ottimizzazione delle prestazioni nell'ambito della modellazione, seguendo il principio di base che ogni cosa che può essere dedotta in modo ragionevole dal contesto non deve essere specificata esplicitamente. Per questo motivo, sono state introdotte le tre prospettive di modello dei dati, modello di ipertesto e di presentazione. Quest'ultimo riunisce tutti quegli aspetti che hanno a che fare con l'estetica e l'usabilità delle interfacce e non viene espresso tramite un diagramma.

Una caratteristica distintiva di WebML è l'assenza di un sotto-modello separato specifico per la logica di business. Essa è inclusa in parte nel modello dei dati (sotto forma di specifiche di derivazione dei dati) e in parte nel modello di ipertesto (sotto forma di Content e Operation Unit, considerate come black-box che rappresentano componenti con funzionalità arbitrarie). Questa scelta ha semplificato parecchio il modello, al prezzo di due comportamenti non attesi (non necessariamente negativi):

1. la proliferazione di Custom Unit per includere logiche di business non standard;
2. la crescita di complessità del modello di presentazione che è stato utilizzato per catturare, oltre agli aspetti estetici, anche il comportamento dell applicazione lato client, non esprimibile secondo altre modalità.

Tale restrizione dei confini del modello ha ridotto i concetti necessari per la costruzione di siti Web dinamici. Inizialmente, era possibile realizzare applicazioni Web complete utilizzando solamente diagrammi E-R e quattordici Unit (sei Content Unit: Index, Data, Multi Data, Scroller, Entry, Get; otto Operation Unit: Create, Delete, Modify, Connect, Disconnect, Set, Login, Logout). Naturalmente, non tutti gli applicativi possono essere rappresentati utilizzando solamente tali elementi; nel tempo, quindi, sono stati sviluppati componenti ad-hoc per gestire esplicitamente alcuni specifici requisiti (interazione con Web Services, invio di messaggi, ...). Essi, però, sono stati trattati come delle black-box utilizzabili all'interno dei tre sotto-modelli esistenti, piuttosto che come facenti parte di uno strato di modellazione indipendente.

Una ulteriore semplificazione risiede nel fatto che non esiste un modello architetturale esplicito. Le risorse fisiche (sorgenti di dati, application server, Web Services, ...) non sono modellate in un diagramma, ma sono semplicemente dichiarate come risorse del progetto all'interno degli strumenti di sviluppo e associate agli elementi del modello che le utilizzano. Un altro aspetto cruciale è la rappresentazione compatta e la

gestione efficiente dei riferimenti tra i modelli, che sono presenti ovunque: il diagramma di ipertesto si riferisce al modello dei dati (per l'estrazione delle informazioni), così come il livello di presentazione si riferisce al modello di ipertesto (per il posizionamento dei contenuti all'interno delle pagine).

Un modello WebML presenta un livello di astrazione molto più spinto rispetto ad una rappresentazione orientata agli oggetti. I vari concetti di un modello, infatti, sono solitamente mappati su differenti artefatti software, che molte volte risiedono in tier diversi. Ad esempio, una pagina WebML, assieme alle sue Unit e ai suoi Link, sono una rappresentazione compatta di molteplici artefatti:

1. il codice che si occupa dell'estrazione dei dati nel data tier;
2. gli oggetti che memorizzano i contenuti all'interno del layer di business o di presentazione;
3. le classi che effettuano il disaccoppiamento delle richieste dagli oggetti della logica di business;
4. i servizi di business che si occupano di orchestrare la computazione del contenuto delle pagine;
5. i tag delle pagine Web che traducono il contenuto degli oggetti in markup.

La decisione di mantenere un alto livello di astrazione ha aumentato la compattezza del modello, a discapito del suo realismo. Infatti, agli occhi di uno sviluppatore, il modello WebML di una pagina unisce elementi

che, in realtà, si trovano in strati differenti dell'applicazione. In ogni modo, se lo scopo della modellazione è la generazione del codice, la possibilitàdi creare un modello completo in modo semplice deve prevalere sul realismo della sua rappresentazione. Dato che anche un progetto di piccole dimensioni può tipicamente comprendere decine di pagine contenenti centinaia di componenti, è semplice immaginare come l'innalzamento del livello di astrazione è cruciale per ottenere un certo grado di usabilità.

Un aspetto non ovvio della progettazione basata sui modelli, che distingue WebML dai tradizionali metodi orientati agli oggetti, è l'assenza di un modello separato per la rappresentazione del comportamento dell'applicazione. Tali aspetti sono incorporati nel modello di ipertesto, che presenta una semantica operazionale fissa per tutte le applicazioni WebML e, quindi, non deve essere specificata esplicitamente dal progettista. In pratica, il modello di ipertesto è rappresentato da una macchina a stati finiti, in cui gli eventi catturano le interazioni da parte dell'utente e le transizioni descrivono la propagazione della computazione da un componente all'altro. Il flusso di controllo è espresso per mezzo di condizioni sulle transizioni.

La presenza di una semantica operazionale indipendente dall'applicazione è una delle pietre miliari di WebML: la semantica "in the large" descrive il funzionamento generale dell'applicazione Web come una rete di componenti cooperanti; la semantica "in the small", invece, descrive come i singoli componenti lavorano. La prima è

standardizzata e gli sviluppatori non devono gestirla esplicitamente; la seconda è considerata come un elemento aggiuntivo del modello: i progettisti devono conoscere come lavorano i componenti predefiniti di WebML e definire i loro rispettando poche "regole" richieste dalla semantica "in the large" (essenzialmente, ogni elemento deve dichiarare i suoi parametri di input e di output e, opzionalmente, le sue regole di default).

2.6 WebRatio

Lo sviluppo di applicazioni basato su WebML è supportato da WebRatio, uno strumento che permette la creazione e modifica dei modelli dei dati e di ipertesto di WebML e la conseguente generazione del codice sorgente corrispondente. WebRatio è quindi un ambiente di sviluppo Model Driven ,che permette di esprimere i requisiti di un'applicazione tramite un modello WebML, da cui successivamente viene generato automaticamente il prodotto finale. Il risultato non è altro che un applicativo aderente agli standard per le applicazioni Web scritte in Java, il cui codice sorgente è liberamente consultabile e non contiene alcun componente proprietario. Se il PIM di WebRatio è specificato utilizzando WebML, tutte le restanti logiche che portano alla creazione del PSM sono orientate alla creazione di artefatti tipici della piattaforma Java Enterprise Edition (EE), la piattaforma di riferimento di WebRatio. In particolare, per l'accesso allo stato dei dati, viene utilizzato il framework Hibernate; la logica di business è implementata utilizzando Servlet e Java Bean; il front-end utilizza pagine JSP per la presentazione dei contenuti. Questa netta divisione tra i diversi strati segue il paradigma Model-View-Controller (MVC), tipico del framework Struts ampiamente sfruttato da WebRatio.

L'ambiente di sviluppo di WebRatio è completamente integrato all'interno dell'IDE Eclipse e ne sfrutta tutte le possibilità offerte per quanto riguarda l'estensibilità e la flessibilità di utilizzo. Le stesse funzionalità messe a disposizione da WebRatio sono implementate per

mezzo di plug-in che si integrano all'interno di Eclipse. Tra questi troviamo strumenti specifici per molteplici compiti: dalla dichiarazione delle sorgenti dei dati e dei Web Services utilizzati all'interno del progetto alla creazione e reverse engineering del modello dei dati, dalla generazione dell'aspetto visuale delle applicazioni alla generazione automatica del codice e distribuzione su diverse piattaforme.

2.7 Elementi del modello

Scendendo più nel dettaglio, WebML descrive le applicazioni Web utilizzando tre livelli: gli oggetti che ne costituiscono il contenuto, l'organizzazione delle pagine e l'aspetto visuale. I primi sono specificati utilizzando un modello dei dati basato su uno schema Entità-Relazione (E-R) semplificato, che comprende classi, entità, attributi e alcune espressioni basilari. Il front-end è descritto da un modello di ipertesto, basato su un'organizzazione gerarchica: ogni applicazione corrisponde ad una Site View, strutturata internamente in aree, che a loro volta possono contenere sotto-aree e pagine. Site View differenti possono essere associate allo stesso modello dei dati, ad esempio per generare applicativi progettati per dispositivi diversi oppure per supportare l'accesso da parte di molteplici attori.

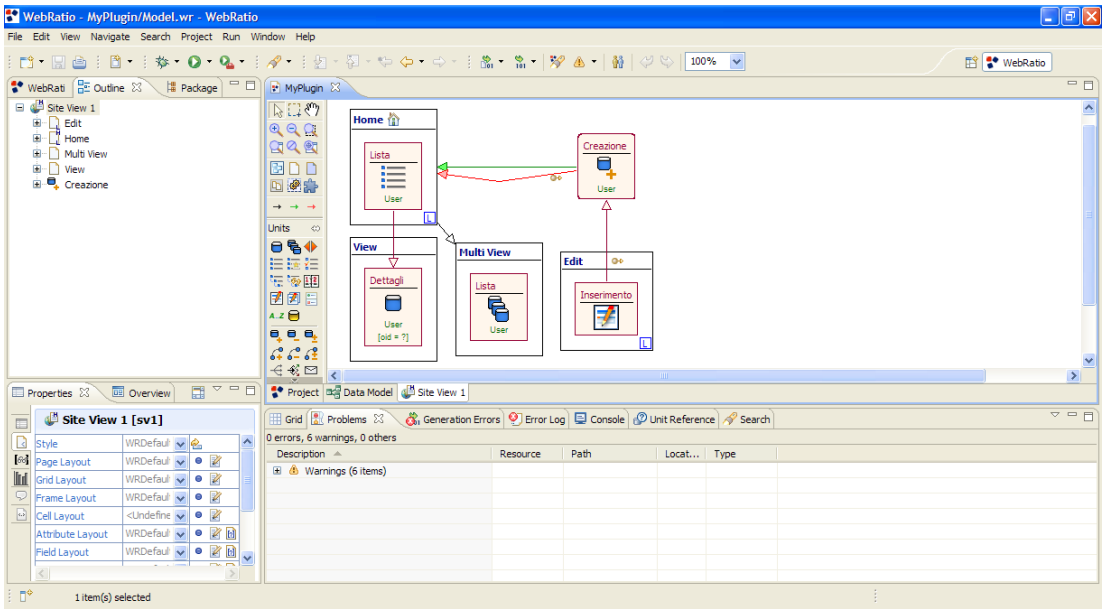

Figura 2.2: Vista del modello di una semplice Site View all'interno dell'ambiente di lavoro di WebRatio

Le pagine sono gli elementi di base delle interfacce: possono essere strutturate in sotto-pagine e contenere Content Unit. Queste ultime rappresentano i componenti fondamentali delle applicazioni e hanno il compito di pubblicare i contenuti all'interno delle pagine; il contenuto visualizzato da una particolare Unit può essere dinamicamente estratto dagli oggetti specificati nel modello dei dati oppure può essere staticamente specificato nel modello di ipertesto. Oltre alle Content Unit, WebML comprende anche le Operation Unit, componenti che si occupano dell'esecuzione della logica di business. Queste Unit non si occupano di pubblicare un contenuto e, per questo motivo, sono posizionate all'esterno delle pagine. Content e Operation Unit possono avere parametri di input e di output. I vari componenti sono connessi tra di loro da Link, che presentano un triplice scopo: permettere la navigazione all'utente, supportare il passaggio di parametri e innescare l'esecuzione dei componenti. Pertanto, un ipertesto WebML può essere descritto essenzialmente come un grafo composto da componenti parametrici, connesso da Link, in cui alcuni componenti stessi pubblicano il contenuto e sono inclusi all'interno delle pagine, altri compiono le azioni che fanno parte della logica di business e sono innescati dai Link uscenti dalle pagine.

Il modello di presentazione è costituito da molteplici Layout Template che, tutti insieme, descrivono l'aspetto visuale dell'intera applicazione. Ogni Layout Template può essere associato ad un componente del modello di ipertesto di WebML; pertanto, esisteranno template specifici

per le pagine, per le Unit e per ogni elemento che contribuisce alla generazione dell'aspetto finale dell'applicazione. Un Layout Template può contenere parti di differenti linguaggi (linguaggi di mark-up, come HTML, espressioni del linguaggio Groovy, componenti lato client oppure tag personalizzati), che rappresentano il codice con cui verrà generato l'aspetto finale dei componenti. Esempi tipici sono i Layout Template associati alle pagine e alle Unit di WebML, che ne definiscono la resa visuale. Al momento della generazione dell'applicazione vera e propria, tutta la gerarchia dei template dei singoli componenti viene combinata per generare il layout finale della singola pagina. Nel caso di WebRatio, il risultato è una pagina il cui contenuto è costituito da tag e scriplet tipici delle tecnologie JSP e JSTL.

Considerando più nel dettaglio il codice che costituisce un Layout Template, va sottolineato come WebRatio utilizzi alcuni tag e placeholder personalizzati per permettere l'associazione tra gli elementi visuali definiti nel template stesso e le informazioni provenienti dal modello WebML. Ad esempio, tramite il tag “Value”, tipico di una Content Unit, è possibile definire in quale punto del template l'effettivo valore di un determinato attributo di una entità verrà visualizzato. In questo modo, si ottiene un forte disaccoppiamento tra gli elementi che concorrono alla creazione dell'interfaccia vera e propria e la logica applicativa che fornisce i dati veri e propri.

Le considerazioni effettuate lungo il corso dell'intero elaborato sono state inizialmente applicate ad un numero limitato di Unit, che, per le loro

caratteristiche rappresentano un sottoinsieme significativo di componenti del modello di ipertesto di WebML. La fase di implementazione è stata pertanto utilizzata per verificare che le tesi proposte si applicassero bene al caso reale delle Data Unit, Multi Data Unit ed Entry Unit.

2.7.1 Data Unit

Le Data Unit si occupano di visualizzare il contenuto proveniente da una determinata sorgente di dati. Più nel dettaglio, questo particolare tipo di Unit ha la funzione di visualizzare le informazioni relative ad una singola istanza di una particolare entità o componente definito nel modello dei dati dell'applicazione. Tipicamente, le Data Unit vengono utilizzate per visualizzare i dettagli di una singola entità, magari dopo la selezione di quest'ultima all'interno di un elenco o come risultato di una ricerca.

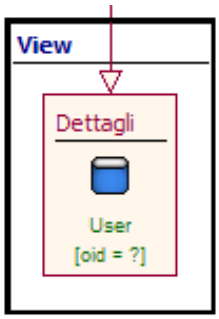

Figura 2.3: Dettaglio della rappresentazione visuale di una Data Unit

Per quanto riguarda il Layout Template, nel caso delle Data Unit, WebRatio utilizza due tag personalizzati: il tag “Value” agisce come placeholder per il posizionamento a tempo di esecuzione del valore degli attributi definiti nell'entità associata alla Unit; il tag “Label”, invece, permette l'inserimento di una stringa testuale che descrive l'attributo stesso.

2.7.2 Multi Data Unit

Le Multi Data Unit, anch'esse inserite tra le Content Unit, sono molto

simili alle Data Unit: contrariamente a queste ultime, però, sono utilizzate per la visualizzazione di un insieme di istanze di un'entità o di componenti di un oggetto composito. Un utilizzo tipico di questa tipologia di Unit è nelle liste e nelle tabelle: permettono, infatti, di visualizzare i dettagli relativi a molteplici istanze della stessa entità.

Se si considera il Layout Template associato alle Multi Data Unit, si nota subito una certa somiglianza: anche questa tipologia di Unit, come le Data Unit, prevede l'utilizzo dei tag personalizzati “Value” e “Label”, con un significato equivalente a quanto descritto in precedenza. Ciò che distingue le Multi Data Unit e ne aumenta la complessità, però, è il tag “Iterate”: esso ha la funzione di associare gli stessi componenti visuali che si trovano al suo interno ad ogni differente istanza dell'entità a cui la Unit è collegata. In questo modo, si otterrà una resa visuale simile per tutte le istanze dell'entità stessa.

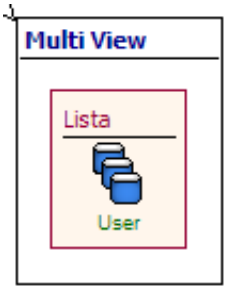

Figura 2.4: Dettaglio della rappresentazione visuale di una Multi Data Unit

2.7.3 Entry Unit

Infine, l'altro tipo di Content Unit supportato dalla prima implementazione è rappresentato dalle Entry Unit, la cui funzione è di presentare una serie di campi che tipicamente costituiscono una form HTML. In altre parole, le Entry Unit sono solitamente utilizzate per permettere l'inserimento e l'invio di dati da parte dell'utente. Contrariamente alle prime due tipologie di

Figura 2.5: Dettaglio della rappresentazione visuale di una Entry Unit

Unit già viste, le Entry Unit non sono direttamente associate ad una entità o un componente del modello dei dati.

Il Layout Template associato a questo tipo di Unit è simile a quello delle Data Unit (comprende, infatti, i tag personalizzati “Value” e “Label”): l'unica differenza si ritrova nel tag aggiuntivo denominato “Link”. Dato che la funzione delle Entry Unit è appunto quella di consentire all'utente la possibilità di inviare dei dati, lo scopo di questo tag aggiuntivo è quello di permettere l'inserimento di quegli elementi visuali che danno il via al trasferimento dei dati stessi (tipicamente un pulsante che effettua il submit di una form HTML).

2.8 Strumenti di sviluppo Model Driven

Per dimostrare come le limitazioni sottolineate non siano presenti solamente all'interno di WebRatio, ma siano in realtà un problema comune agli strumenti Model Driven esistenti, sono stati presi in esame alcuni prodotti che fanno di MDD il loro punto di forza.

Instant Developer è stato scelto come primo esempio. Esso non rappresenta semplicemente uno strumento CASE, ma un vero e proprio sistema di sviluppo che gestisce tutti gli aspetti del ciclo di vita del software, dall'analisi all'installazione. Il cuore del funzionamento di In.de è la "programmazione relazionale". Infatti, esso permette di descrivere il comportamento del software tramite la composizione di un grafo di relazioni piuttosto che con la scrittura di tanti file di testo, come avviene nei sistemi tradizionali. All'interno di Instant Developer, il codice non viene memorizzato in un file di testo, ma direttamente in un grafo le cui relazioni vengono tracciate automaticamente dall'IDE. Gli oggetti contenuti nel grafo delle relazioni sono tali da non dipendere da una specifica tecnologia: è il compilatore che è in grado di generare il codice sorgente relativo alla specifica piattaforma. La novità della programmazione relazionale risiede più nel modo con cui vengono dichiarati i comportamenti dell'applicazione più che la logica degli stessi; infatti, essa permette di utilizzare gli stessi costrutti e le stesse regole di costruzione del software presenti nei linguaggi di programmazione tradizionali.

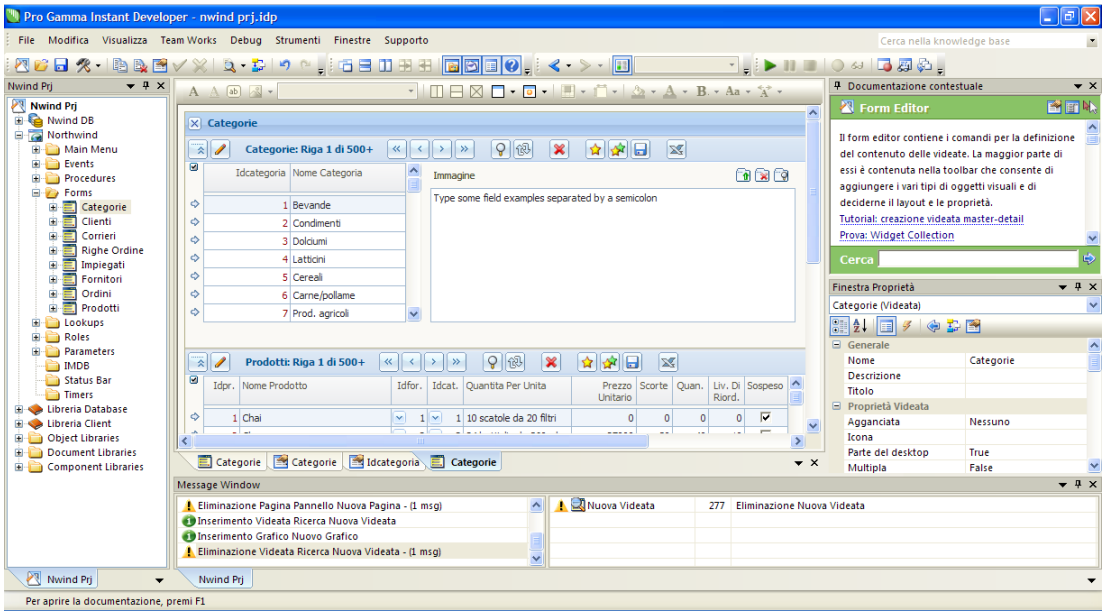

Figura 2.6: Instant Developer

Il limite fondamentale riscontrato in questo approccio riguarda il suo basso livello di astrazione: la programmazione relazionale non è altro che una generalizzazione dei linguaggi Java e C#, che la rendono indipendente dalla piattaforma di riferimento, ma costringono comunque lo sviluppatore a scrivere una sorta di codice sorgente. Il modello utilizzato da In.de è semplicemente un nuovo linguaggio che si trova ad un livello leggermente superiore rispetto ai linguaggi di programmazione tradizionale. I vantaggi dell'approccio Model Driven, quindi, sono molto limitati: la fase implementativa è ancora al centro del processo di sviluppo, mentre le fasi di analisi e progettazione non sono per nulla supportate.

Per quanto riguarda la creazione delle interfacce, invece, il supporto dello strumento appare completo: attraverso un semplice editor è possibile costruire l'aspetto visuale delle applicazioni. Anche se la

semplicità d'uso non è paragonabile a quella di un editor WYSIWYG, permette ugualmente la creazione di interfacce complete utilizzando strumenti di alto livello. Il limite dell'approccio di In.de, però, risulta evidente anche in questa fase: ogni interazione con la parte di back-end richiede il ricorso alla programmazione relazionale e, quindi, alla scrittura del codice.

Risulta quindi evidente come una simile soluzione sia molto più vicina ad uno strumento di sviluppo tradizionale con un buon editor WYSIWYG: obbliga ad imparare un nuovo linguaggio che, a conti fatti, costringe ugualmente alla gestione degli aspetti di basso livello del software.

Un approccio più innovativo è quello proposto da Concrete Editor: un vero e proprio editor di modelli che può essere configurato per lavorare con differenti DSL (Domain Specific Language). Tale strumento può essere eseguito all'interno di un browser. La sua caratteristica fondamentale risiede nel fatto di poter essere personalizzato per un particolare DSL con una grammatica basata su JSON, mentre l'aspetto visuale dei suoi componenti può essere specificato utilizzando HTML, JavaScript e CSS. Il risultato è un editor che permette di creare direttamente un particolare modello; la struttura di tale modello e la sua resa visuale sono descritti per mezzo di un DSL.

Concrete Editor nasce come un semplice editor di tipo testuale; sta all'utente trasformarlo in un editor visuale, procedendo alla personalizzazione dell'aspetto dei componenti. Tale processo richiede la

scrittura del codice HTML, JavaScript e CSS che ne rappresenta l'aspetto visuale.

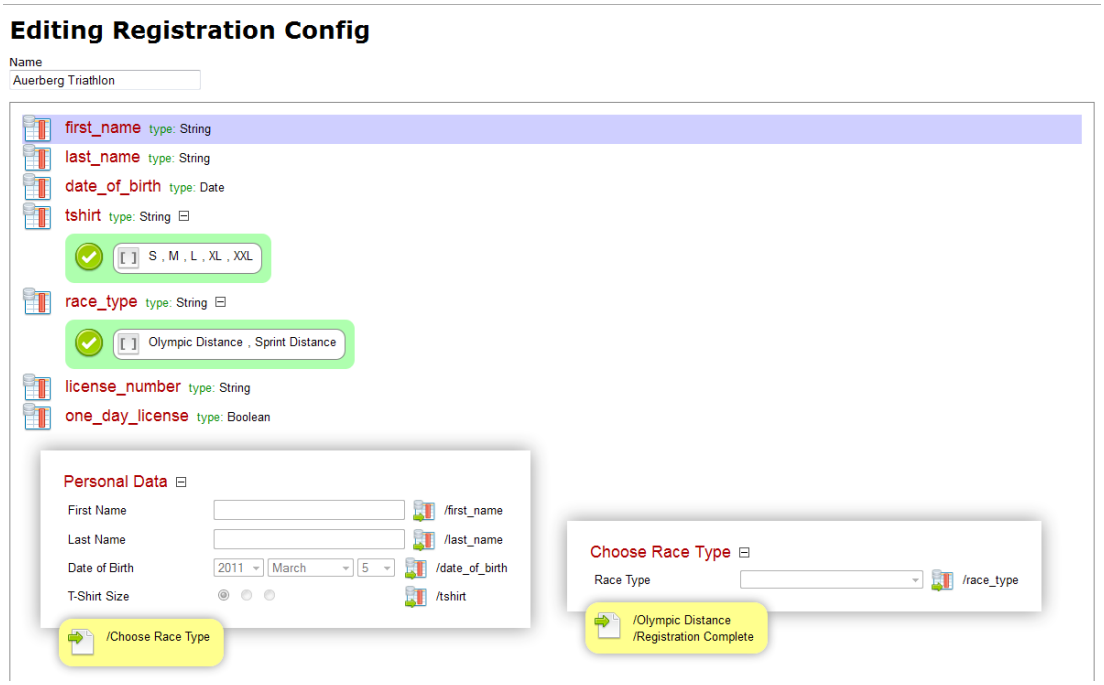

Figura 2.7: Esempio di un editor DSL per la generazione di una semplice applicazione Web creato utilizzando Concrete

Il suo punto di forza, quindi, rappresenta allo stesso tempo un punto di debolezza: Concrete Editor, infatti, è orientato verso uno specifico problema, per cui ogni soluzione manca di generalità. Infatti, piuttosto che concentrarsi sulla specifica formale di un modello, esso si concentra su come risolvere una particolare problematica. La specificità dei modelli generati, quindi, ne limita fortemente il riutilizzo al di fuori del caso in esame. Prendendo in considerazione il processo più ampio della creazione dell'aspetto visuale di un'applicazione Web, inoltre, il rischio è quello di generare un modello che risulta più complesso del problema che si intende risolvere.

In questo caso, inoltre, manca del tutto una specifica formale del modello. Solitamente, infatti, gli strumenti Model Driven basano le proprie funzionalità sulla sintassi e sulla semantica del modello stesso, in modo che gli strumenti di modellazione supportino al meglio l'utente nelle fasi di progettazione. Utilizzando Concrete Editor, invece, le modalità di interazione con lo sviluppatore dipendono dal DSL e sono difficilmente standardizzabili.

Infine, per utilizzare Concrete Editor come editor di un modello già esistente, si rende necessaria la costruzione di uno strato software aggiuntivo, che si occupi della traduzione tra le diverse rappresentazioni interne: dalla specifica JSON di Concrete Editor al formato del particolare modello considerato e viceversa.

2.9 Editor WYSIWYG di mercato

Prima di procedere con l'implementazione del prototipo, è stata effettuata un'attività di ricerca degli editor WYSIWYG attualmente disponibili sul mercato. Le motivazioni che hanno spinto a procedere in tale direzione sono fondamentalmente di due tipi: prima di tutto avere un'idea di quali funzionalità offrono gli strumenti già esistenti avrebbe aiutato nel processo di stesura dei requisiti; in secondo luogo, si confidava di riuscire ad individuare un prodotto da utilizzare come base di partenza per i nostri scopi, evitando così l'implementazione delle funzionalità di base di un editor WYSIWYG.

Durante la fase di ricerca vera e propria, sono stati presi in considerazione tutti i principali strumenti di sviluppo software che contenessero al proprio interno un componente per la costruzione di interfacce Web in maniera visuale. Per creare un elenco esaustivo, in un primo tempo non si è badato né alle licenze che accompagnano i prodotti, né alle modalità di installazione di questi ultimi, per evitare di escludere dall'analisi alcune applicazioni commerciali molto diffuse. Lo scopo di questa ricerca, infatti, era quello di creare un elenco di quelle che sono le caratteristiche fondamentali che ci si aspetta da un editor WYSIWYG ideale.

Tutti gli applicativi trovati sono pensati per la creazione di interfacce Web, ognuno orientato verso un particolare paradigma o tecnologia. Pertanto, come metro di paragone si è preferito utilizzare alcuni parametri che si concentrassero più sull'interazione uomo-macchina che

sul risultato generato. Per questo motivo, è stato dato un grande peso all'usabilità del prodotto e alla facilità d'uso da parte dell'utilizzatore, in modo che la curva di apprendimento non risultasse troppo gravosa. In questa fase, quindi, si è tenuto conto dei seguenti aspetti:

- Familiarità dell'interfaccia dellapplicazione: risulta senz'altro preferibile utilizzare strumenti che utilizzano un'interfaccia conosciuta o che segua alcune linee guida comuni; un'interfaccia innovativa o addirittura rivoluzionaria che però presenta una curva di apprendimento piuttosto lunga non è sicuramente desiderabile.
- Intuitività degli strumenti messi a disposizione: un'interfaccia semplice che mette a disposizione dell'utente tutti gli strumenti disponibili in modo chiaro favorisce l'adozione dello strumento stesso.
- Completezza delle funzionalità disponibili: la mancanza di strumenti fondamentali che rendano necessario il ricorso alla scrittura del codice vero e proprio che costituisce l'interfaccia non favorisce l'adozione di un simile strumento; anzi, nel lungo periodo, può favorirne l'abbandono.

In un secondo momento, i vari strumenti individuati sono stati suddivisi in due macro-categorie: quelli che si integrano all'interno dell'ambiente di sviluppo Eclipse come veri e propri plug-in e quelli che, invece, sono del tutto indipendenti e vengono distribuiti come software stand-alone proprietario. La motivazione è molto semplice: il prototipo utilizza come piattaforma di riferimento WebRatio. Essendo quest'ultimo

completamente integrato all'interno di Eclipe, i prodotti appartenenti alla seconda categoria sono da escludere a priori per un eventuale utilizzo come base di partenza del prototipo.

2.9.1 Editor stand-alone

La prima applicazione stand-alone considerata è stata EXT Designer, sviluppata da Sencha Inc. come supporto alla creazione di interfacce Web utilizzando la propria libreria JavaScript proprietaria EXT JS. Tale applicazione si presenta con un'interfaccia piuttosto standard, che presenta l'elenco dei componenti disponibili che è possibile trascinare all'interno dell'area di editing visuale vero e proprio. Un apposito pannello permette la configurazione delle proprietà di base del componente attualmente selezionato all'interno della finestra principale. La resa visuale dell'area di editing non è del tutto aderente al risultato finale effettivo; per questo motivo, si rende necessario il ricorso ad una ulteriore pagina di anteprima, che provoca inutili rallentamenti nel corso dello sviluppo. Quella che può essere vista come una forte limitazione dell'applicativo deriva dal fatto che non è possibile in alcun modo intervenire sul codice generato, ma solo averne un'anteprima non modificabile.

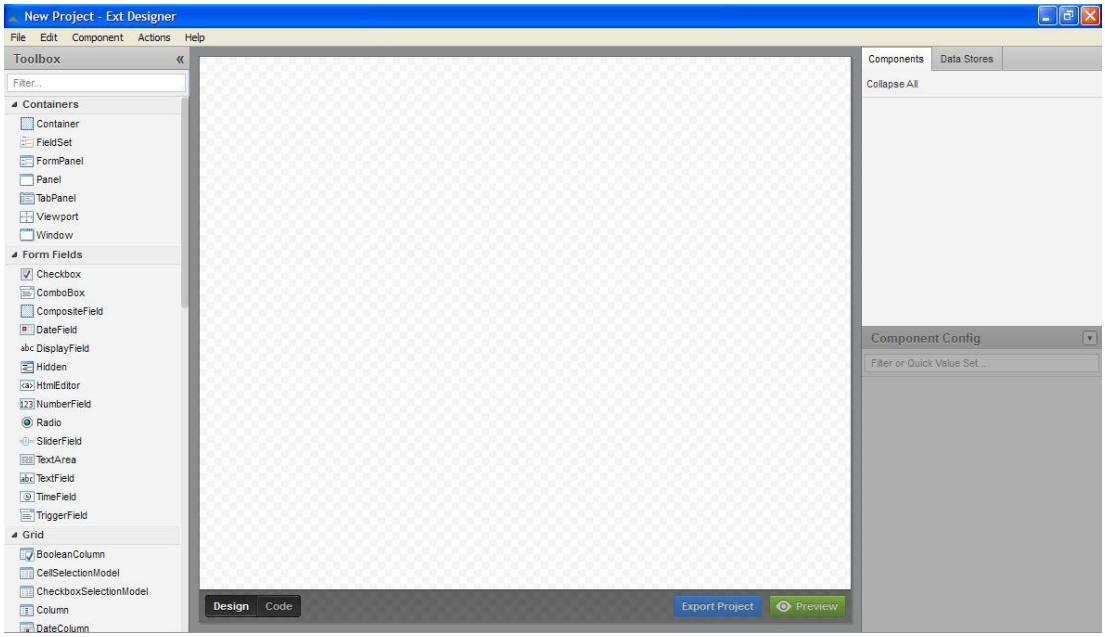

Figura 2.8: EXT Designer

Fin dal primo utilizzo, si capisce che tale applicazione è stata pensata principalmente come supporto alla libreria JavaScript della compagnia, piuttosto che come strumento di sviluppo vero e proprio. In ogni modo, è stata inserita nell'analisi in quanto rappresenta un tipico esempio di editor WYSIWYG, con alcuni elementi di base che risultano comuni anche a software più corposi e complessi.

Come seconda applicazione, è stata presa in considerazione Microsoft Expression Web 4, che costituisce uno strumento orientato alla creazione di pagine Web chiaramente derivato da Microsoft Visual Studio, da cui eredita gran parte delle funzionalità. Questo software presenta un'interfaccia piuttosto chiara, con tutti gli strumenti disposti in pannelli posizionati accanto alla finestra di editing visuale. Quest'ultima permette

di visualizzare contemporaneamente il codice generato e l'editor WYSIWYG; di particolare utilità è risultata essere la sincronizzazione tra le due diverse viste: selezionando una parte di codice, il relativo componente visuale viene immediatamente evidenziato e viceversa, permettendo un rapido riscontro di quanto si sta costruendo. Gli altri pannelli visualizzano tutte le proprietà dei componenti su cui è possibile intervenire, fornendo, ove possibile, anche utili suggerimenti sui valori permessi, in modo da limitare o praticamente annullare il ricorso alla scrittura del codice. Codice che peraltro viene generato con una corretta indentazione e una resa visuale che ne aumenta la leggibilità. L'interazione con l'utente è semplificata per mezzo di alcuni semplici ma efficaci accorgimenti: quando viene selezionato un particolare componente, questo viene evidenziato con un colore differente, viene applicata un'etichetta con il suo nome per un più semplice riconoscimento e viene visualizzata la gerarchia all'interno della quale il componente stesso si trova. Tutti i pannelli che costituiscono l'interfaccia possono essere impostati in modalità “a scomparsa”, in modo da rendere disponibile un'area più ampia possibile per l'editing visuale. Una funzionalità molto utile risulta essere la possibilità di visualizzare un'anteprima del codice generato direttamente all'interno di differenti versioni dei browser più comuni, in modo da poter immediatamente verificare che il codice generi il risultato voluto con differenti motori di rendering.

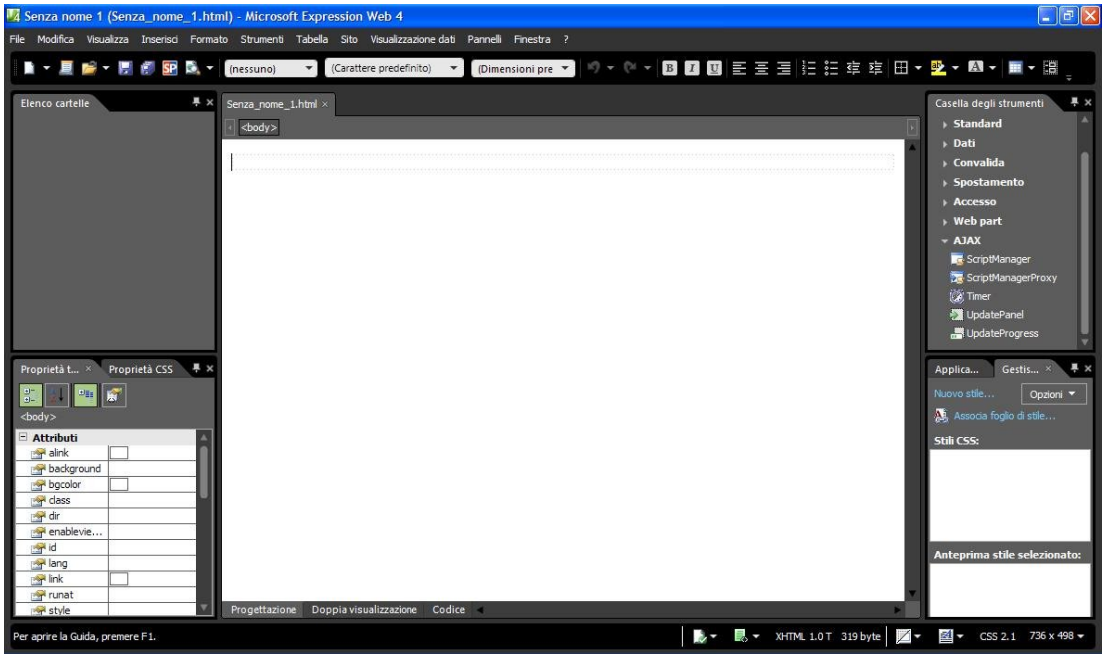

Figura 2.9: Microsoft Expression Web 4

Questa applicazione ci ha positivamente sorpreso. Nonostante non presentasse grosse novità per quanto riguarda l'interazione con l'utente, tutti gli strumenti sono da subito disponibili e presentati in modo chiaro. Non ci è voluto molto per trovarsi a proprio agio con questo prodotto Microsoft. È anche risultato essere l'applicativo che genera il codice più pulito e leggibile tra tutti quelli provati.

Considerato da molti il punto di riferimento per la creazione di siti Web, Adobe Dreamweaver era già stato utilizzato in passato da WebRatio per la costruzione delle interfacce utente. Come già spiegato, però, l'utilizzo di un applicativo esterno, il cui unico compito sia quello di generare i componenti visuali di un'applicazione, complica oltremodo il ciclo di vita dell'applicazione stessa. Già al primo avvio si nota subito come

Adobe sia intervenuta pesantemente sull'interfaccia del suo software per cercare di ottimizzare al massimo gli spazi: la stessa barra dell'applicazione è stata modificata per includere il menù, in modo da massimizzare lo spazio disponibile per le barre degli strumenti e l'editor WYSIWYG. Quest'ultimo presenta interessanti funzionalità: oltre ad alcune caratteristiche già incontrate in altri prodotti, il menù contestuale risulta ricco di opzioni che completano quanto già messo a disposizione dalle diverse barre degli strumenti e aiuta l'utente a ricorrere il meno possibile alla scrittura del codice. Risulta utile anche la possibilità di trascinare un componente indifferentemente all'interno della vista del codice sorgente o dell'editor WYSIWYG; sorprendetemene, tale caratteristica è risultata assente in molti dei prodotti analizzati.

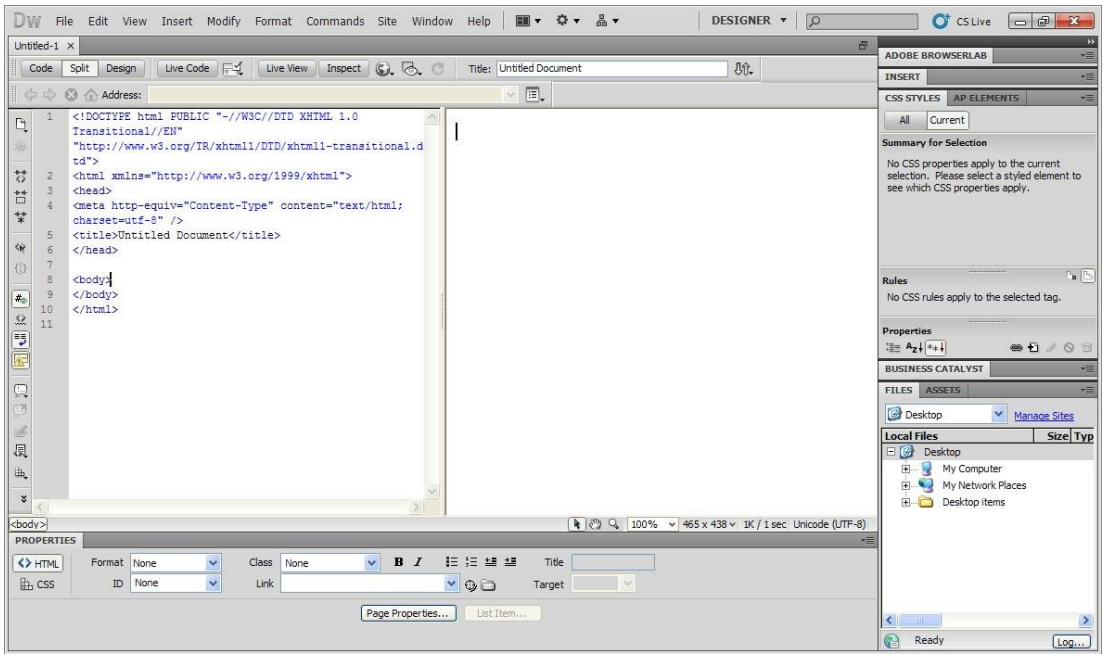

Figura 2.10: Adobe Dreamweaver CS5

Non si può notare, utilizzando questa applicazione, la cura nei minimi particolari che, insieme alla grande esperienza maturata da Adobe nelle numerose versioni create, fanno di Dreamweaver un prodotto completo e davvero ricco di funzioni utili, che lascia poco spazio a fronzoli estetici e permette di semplificare enormemente il lavoro di uno sviluppatore. Se proprio si vuole trovare una mancanza, il feedback visuale dell'editor WYSIWYG potrebbe essere ulteriormente migliorato dando una maggiore evidenza dei vari componenti di cui è costituita la pagina, soprattutto nel momento in cui uno di questi risulta selezionato.

2.9.2 Editor integrati in Eclipse

Il primo prodotto preso in esame è un componente integrato nella versione studiata per gli sviluppatori di applicazioni Java EE (Enterprise Edition) di Eclipse, in particolare il Web Page Editor incluso nel progetto WTP (Web Tools Project). Essendo uno strumento sviluppato internamente dalla stessa Eclipse Foundation, si nota immediatamente come tutti i suoi strumenti risultino perfettamente integrati nell'IDE. Anche se non presenta un'interfaccia con soluzioni particolarmente innovative, l'integrazione con un ambiente di sviluppo così largamente conosciuto aiuta l'utente ad essere immediatamente produttivo. Si nota subito che la sua semplicità non deriva dalla mancanza di strumenti o funzionalità, ma, piuttosto, sembra un punto cardine su cui si sono concentrati i suoi creatori. Troviamo, infatti, tutte le principali funzioni messe a disposizione dagli altri strumenti, con qualche piccolo dettaglio che, anche se non stravolge l'interazione con l'utente, ne facilita

sicuramente il lavoro. Prima di tutto, come da tradizione di Eclipse, questo plug-in è facilmente estendibile per supportare librerie di tag aggiuntive e personalizzate. Inoltre, la finestra di editing visuale si distingue per alcune utili soluzioni: il menù contestuale permette di intervenire sui principali aspetti di visualizzazione dei componenti per avere un feedback visuale immediato; al passaggio del puntatore del mouse, dopo una breve pausa, viene evidenziato il contorno del componente su cui ci si trova e un'etichetta ne permette una semplice identificazione; al trascinamento di un componente all'interno dell'editor WYSIWYG, un'ulteriore etichetta testuale informa l'utente circa la posizione in cui il componente stesso sta per essere inserito, in modo da evitare errori e inutili tentativi andati a vuoto.

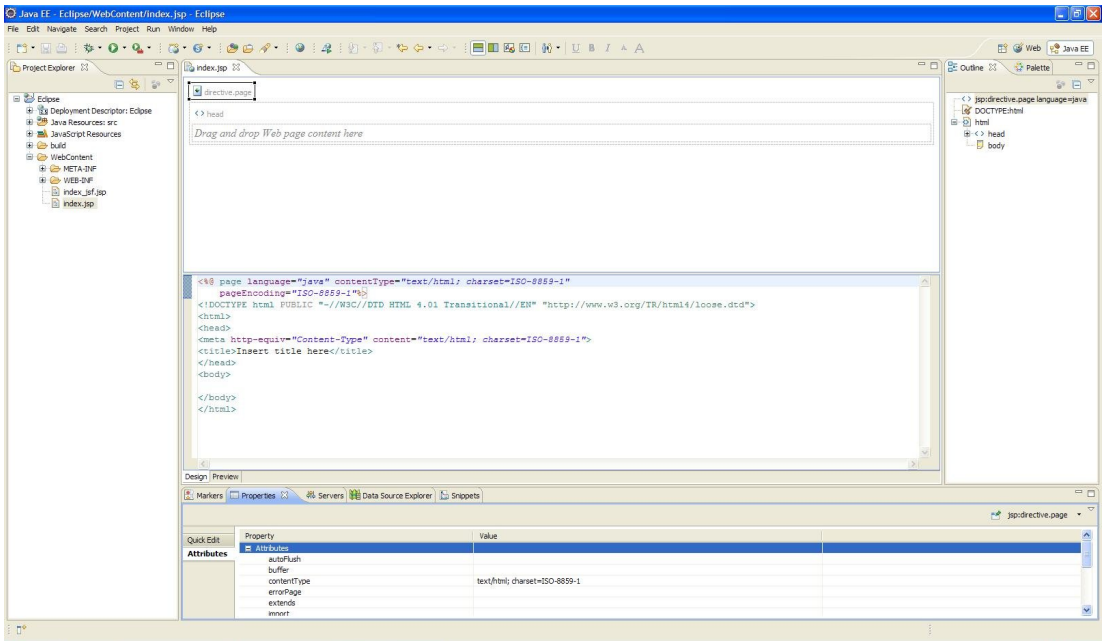

Figura 2.11: Eclipse Web Tools Platform Web Page Editor

La sua interfaccia chiara e pulita, oltre alla sua completa integrazione all'interno di uno strumento di sviluppo così diffuso come Eclipse, ne fanno uno strumento molto interessante. In particolare,come vedremo, la relativa semplicità con cui tale plug-in può essere esteso ha giocato un ruolo fondamentale in quella che è stata la scelta finale.

Il Web Page Editor incluso nel pacchetto di plug-in denominato OEPE (Oracle Enterprise Pack for Eclipse) non è altro che un'estensione del Web Page Editor appena analizzato. È stato ugualmente incluso nell'analisi in quanto presenta alcune funzionalità particolarmente utili. Si nota subito come, al trascinamento di un componente all'interno della pagina di editing visuale, viene visualizzata una finestra che guida l'utente nella creazione del componente stesso e nella scelta di quelli che sono i suoi attributi principali. Inoltre, oltre al pannello che elenca le proprietà del componente attualmente selezionato che è possibile modificare, ne è stata creata una versione che elenca gli attributi più utilizzati, in modo che l'utente possa avere una vista immediata di quelli di uso più comune.

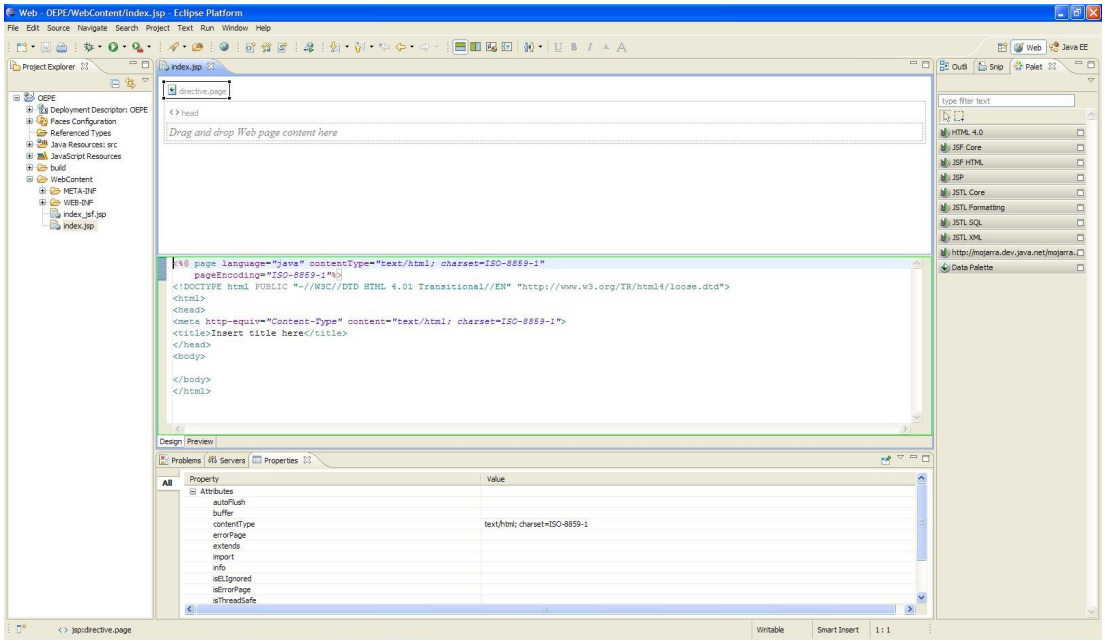

Figura 2.12: Oracle Enterprise Pack for Eclipse Web Page Editor

Se non fosse stato per alcuni particolari accorgimenti, questa applicazione molto probabilmente non sarebbe stata presa in considerazione. Come vedremo in seguito, infatti, le particolarità inserite da Oracle nel suo Web Page Editor hanno ispirato profondamente alcune soluzioni implementate nella soluzione finale.

Bravo JSP Editor si distingue da tutti gli altri prodotti inseriti in questa analisi per una caratteristica unica: in un angolo della finestra dell'editor WYSIWYG appare un'etichetta che informa l'utente circa il componente che si trova attualmente sotto il puntatore del mouse. Non è sicuramente nulla di rivoluzionario, ma ci ha colpito per la sua immediatezza e il modo in cui velocizza le operazioni nella finestra di editing visuale. Al contrario, gli altri applicativi richiedono una pausa o l'esplicita selezione

da parte dell'utente prima di visualizzare un'informazione tanto semplice quanto utile, rendendo l'interazione uomo-macchina meno fluida.

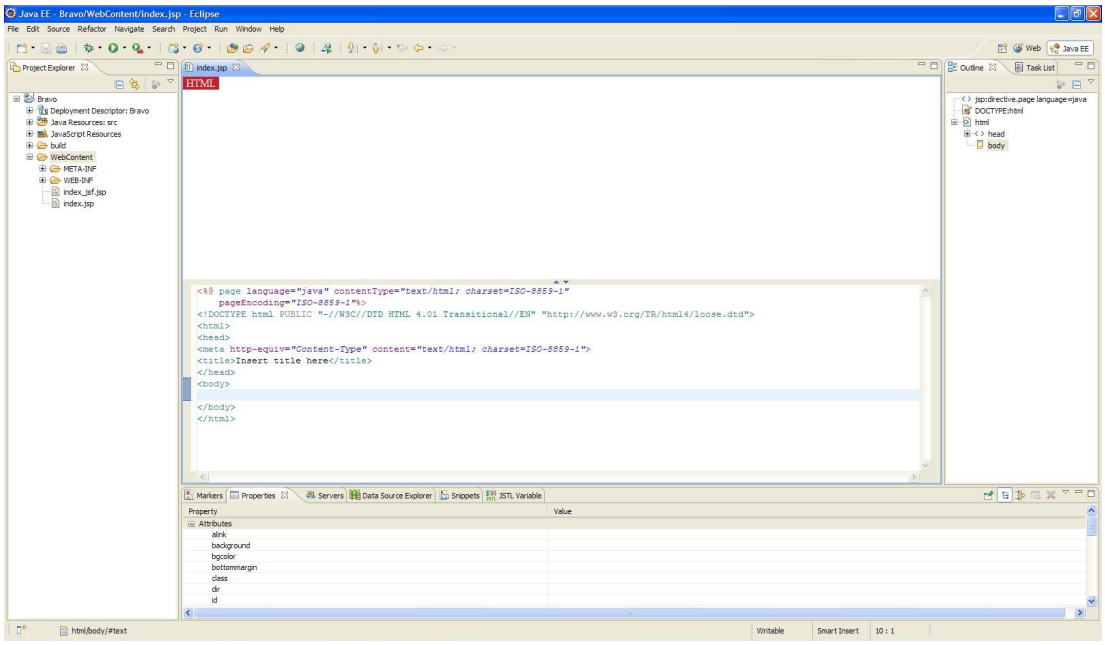

Figura 2.13: Bravo JSP Editor

Purtroppo, questo strumento si è verificato essere particolarmente limitato per quanto riguarda gli altri aspetti di editing visuale, nonostante la sua integrazione con Eclipse facesse ben sperare.

L'ultima applicazione inserita in questa macro-categoria è Adobe Flash Builder 4. Una prima nota positiva di questo applicativo si riscontra nel rendering effettuato all'interno dell'editor WYSIWYG: senza ombra di dubbio risulta essere il prodotto che ha fornito la resa visuale migliore; non c'è però da trascurare la motivazione di questo risultato: Flash Builder è orientato alla creazione di interfacce per applicazioni RIA (Rich Internet Application) basate sul linguaggio Flex, a sua volta basato

su componenti di più alto livello rispetto ai semplici tag HTML. Nonostante questo, però, la finestra di editing visuale fallisce nella visualizzazione dei contenuti dinamici che sembrano essere ignorati dall'editor WYSIWYG.

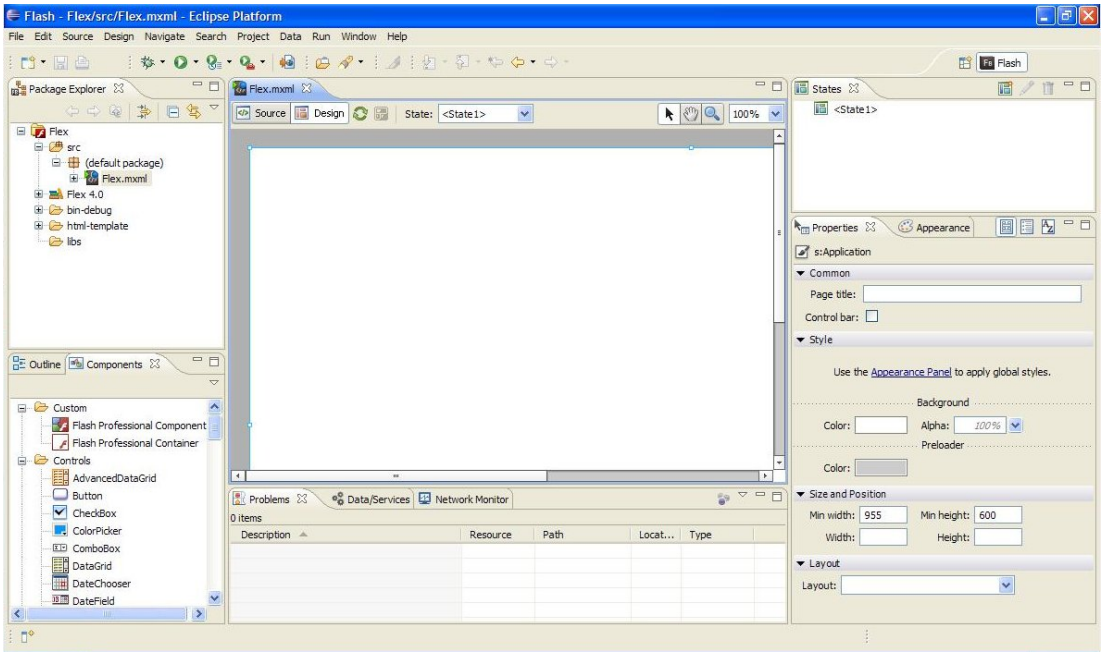

Figura 2.14: Adobe Flash Builder 4

Nonostante la buona resa visiva dei componenti trascinati all'interno della finestra, l'editor visuale, durante il suo utilizzo, fornisce una sensazione di inadeguatezza che costringe spesso a ricorre alla stesura manuale del codice: gli attributi presentati nei pannelli rappresentano sempre un sottoinsieme di tutti quelli disponibili per ogni componente e la loro scelta a volte appare addirittura discutibile. Uno sviluppatore con una certa esperienza, insomma, si ritroverà più spesso a scrivere il codice manualmente piuttosto che a ricorrere all'editor WYSIWYG. Senza

dubbio positiva la scelta di Adobe nella costruzione del pannello per la modifica degli attributi del componente attualmente selezionato nella pagina di editing visuale: piuttosto che presentare una semplice lista, con al più un suggerimento circa i valori permessi, è stata implementata un'interfaccia che presenta in modo intuitivo le principali proprietà del componente e, per ognuna, mette a disposizione il metodo più efficace per la modifica del suo valore; una sorta, quindi, di editor visuale per le proprietà del componente. Probabilmente, come editor WYSIWYG è l'applicazione che ha più deluso le aspettative: l'espressività del linguaggio Flex non è stata sfruttata a dovere per creare un editor visuale che aveva tutte le potenzialità per distinguersi rispetto a tutti gli altri.

2.9.3 Risultato dell'analisi

In una fase successiva, sono stati presi in considerazione tutti quegli elementi che sono risultati essenziali o perlomeno importanti per un editor visuale. Per ogni prodotto analizzato, sono stati elencati quelli che si sono dimostrati essere sia i punti di forza sia quelli di debolezza. In questo modo, è risultato semplice riassumere quelle che sono le caratteristiche che ci si aspetta da un editor WYSIWYG e quelle che sono le mancanze che assolutamente devono essere evitate.

Nella Tabella 1 sono riassunte le caratteristiche peculiari di ognuno dei prodotti analizzati:

Prodotto	**Sincronizzazione tra la finestra di editing visuale e il codice**	**Feedback visuale al passaggio del mouse**	**Modifica visuale degli attributi**	**Menù contestuale**	**Visualizzazione della gerarchia dei componenti**
EXT Designer	Non possibile dal codice alla finestra di editing visuale	Componente evidenziato	Tramite una lista in un pannello	Funzioni di base / Modifica limitate dei componenti	✔
Microsoft Expression Web 4	✔	Nome e componente evidenziati solo dopo la selezione	Tramite una lista in un pannello	Funzioni di base / Modifica di base dei componenti	✘
Adobe Dreamweaver CS5	✔	Componente evidenziato solo dopo una pausa	Tramite una lista in un pannello o il menù contestua-	Funzioni di base / Modifica completa dei componenti	✔

Prodotto	Sincronizza-zione tra la finestra di editing visuale e il codice	Feedback visuale al passaggio del mouse	Modifica visuale degli attributi	Menù contestua-le	Visualizza-zione della gerarchia dei componenti
			le	e dello stile	
Eclipse Web Tools Platform Web Page Editor	✔	Nome e componen-te evidenziati solo dopo una pausa	Tramite una lista in un pannello	Funzioni di base / Modifica di base dei componenti e dello stile	✔
Oracle Enterprise Pack for Eclipse Web Page Editor	✔	Nome e componen-te evidenziati solo dopo una pausa	Tramite una lista in un pannello o il menù contestua-le	Funzioni di base / Modifica completa dei componenti e dello stile	✔
Bravo JSP Editor	✔	Nome e componen-te evidenziati solo dopo la selezione	Tramite una lista in un pannello	Modifica di base dei componenti	✔
Adobe Flash Builder 4	✔	Componen-te evidenziato solo la selezione	Tramite una lista in un pannello	Funzioni di base	✔

Tabella 2.1: Risultato dell'analisi comparativa

2.9.4 Analisi delle licenze software

Dopo aver considerato le caratteristiche di ogni applicazione, è stato necessario procedere con una veloce analisi di quelle che sono le licenze che accompagnano i vari software. La motivazione è semplice: prima di procedere con lo sviluppo di un nuovo prototipo che presentasse le

caratteristiche desiderate, era utile conoscere se qualcuno degli applicativi poteva essere utilizzato come base di partenza per i nostri scopi. Come vedremo in seguito, tale scelta è risultata essere un passo fondamentale che ha permesso di concentrare gli sforzi sulle tematiche centrali del lavoro.

Nella Tabella 2 sono riassunte le caratteristiche principali delle licenze dei prodotti analizzati:

Prodotto	Licenza	Free software	Compatibile GPL	Copyleft	Utilizzo da parte di un software con licenza differente
EXT Designer	EXT Designer Software License	✗	✗	✗	✗
Microsoft Expression Web 4	Licenza software Microsoft – Software di progettazione Microsoft Expression Studio 4 e versioni di valutazione	✗	✗	✗	✗
Adobe Dreamweaver CS5	Adobe Software License Agreement	✗	✗	✗	✗
Eclipse Web Tools Platform Web Page Editor	Eclipse Public License	✓	✗	Limitato	✓
Oracle Enterprise Pack for Eclipse Web Page Editor	Oracle Enterprise Pack for Eclipse License	✗	✗	✗	✗
Bravo JSP Editor	Bravo JPS Editor License	✗	✗	✗	✗
Adobe Flash Builder 4	Adobe Software License Agreement	✗	✗	✗	✗

Tabella 2.2: Risultato dell'analisi delle licenze

Risulta subito chiaro come l'unico applicativo la cui licenza ne permettesse l'integrazione all'interno di un altro prodotto si è rivelato essere il Web Page Editor di Eclipse WTP. Inoltre, il fatto che tale applicazione sia a tutti gli effetti da considerare come “software libero”, ha influito notevolmente nel processo decisionale che ha seguito le attività di analisi.

2.10 Eclipse WTP

WTP (Web Tools Platform) è un progetto che estende l'IDE (Integrated Development Environment) Eclipse e ha lo scopo di permettere lo sviluppo di applicazioni Web basate sulle specifiche di Java EE (Enterprise Edition). Lo stesso WTP non è altro che un plug-in di Eclipse che viene installato di default nella versione enterprise di quest'ultimo.

Oltre a tutte le funzionalità di base incluse nell'IDE per sviluppare applicativi Java, WTP mette a disposizione diversi strumenti orientati ad una piattaforma enterprise:

- funzionalità per la gestione dei numerosi application server Java esistenti e la distribuzione delle applicazioni all'interno di questi ultimi
- debugger per applicativi in esecuzione all'interno di un server
- editor di pagine JSP e JSF
- funzionalità specifiche per tecnologie Web, come i Web Services
- supporto alle più diffuse basi di dati

2.10.1 Web Page Editor

Una delle caratteristiche distintive che introduce WTP rispetto alla versione standard di Eclipse è WPE (Web Page Editor), un editor studiato appositamente per la creazione di pagine Web. In particolare, esso fornisce tutti quegli strumenti che lo rendono un vero e proprio editor WYSIWYG, come la possibilità di trascinare i componenti

direttamente all'interno dell'anteprima della pagina o la possibilità di inserire e modificare gli attributi di un elemento senza ricorrere alla scrittura del codice.

WPE sfrutta il meccanismo delle estensioni tipico di Eclipse per permettere a chiunque di ampliare le funzionalità presenti o personalizzare alcuni aspetti del plug-in. In particolare, è stata prevista la possibilità di ampliare la lista di tag disponibili per mezzo dell'importazione di Tag Library opportunamente configurate. Inoltre, come accade per l'intero progetto WTP, trattandosi di un'applicazione open source, è possibile analizzare il codice sorgente di WPE, per meglio comprendere i meccanismi che sono stati adottati dai suoi creatori.

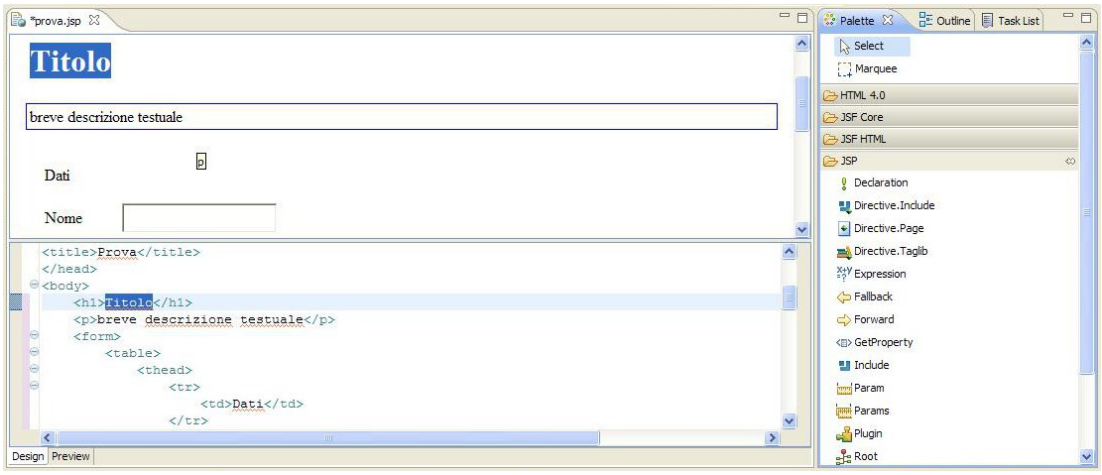

Figura 3.15: Dettaglio del Web Page Editor e del pannello Palette

Come è possibile notare nella Figura 3.15, l'area destinata all'editor WYSIWYG vero e proprio fornisce la possibilità di visualizzare solamente il codice sorgente o l'area di editing visuale oppure una finestra divisa in due parti con una vista su entrambe; tale finestra,

inoltre, può essere impostata per essere separata in maniera orizzontale o verticale, in modo da adattarsi al meglio ad ogni esigenza. Il pannello "Palette" è sicuramente il più importante per quanto riguarda l'inserimento dei componenti in maniera visuale: esso fornisce la lista di elementi che è possibile trascinare nell'area dell'editor WYSIWYG; come già anticipato, WTP fornisce già gli elementi corrispondenti ai tag di base dei linguaggi HTML e JSP, ma può essere ulteriormente esteso tramite l'importazione di Tag Library personalizzate.

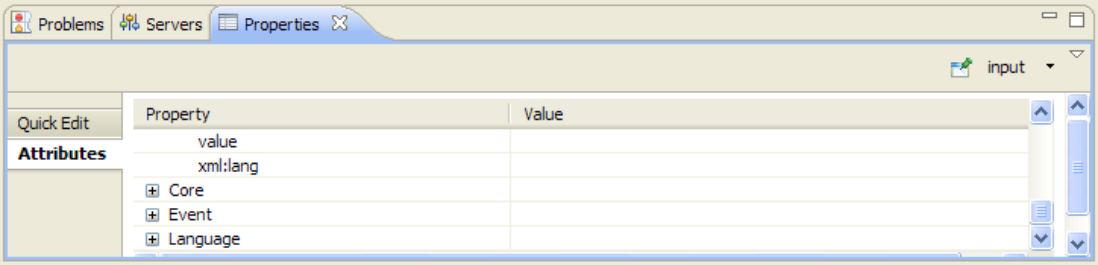

Figura 3.16: Dettaglio del pannello Properties e della scheda Attributes

Nella Figura 3.16 è presente un dettaglio di un altro strumento molto utile per limitare al massimo il ricorso alla scrittura del codice, il pannello "Properties". Come si può notare, esso presenta due schede: la prima, "Attributes", presenta una lista degli attributi che è possibile inserire oppure modificare per il componente attualmente selezionato.

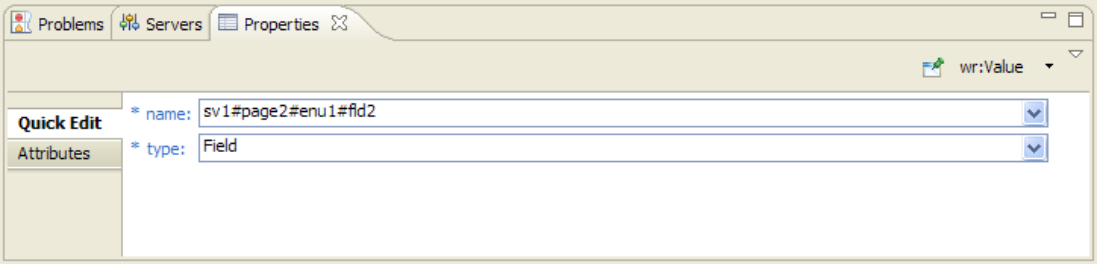

Figura 3.17: Dettaglio del pannello Properties e della scheda Quick Edit

La seconda scheda, denominata "Quick Edit", invece, può sembrare una

funzione ridondante, ma, dopo un breve periodo di utilizzo, dimostra subito la sua utilità: essa è infatti pensata per visualizzare solamente le proprietà di uso più comune, in modo da velocizzare le operazioni svolte con maggiore frequenza.

Capitolo 3

Obiettivi e requisiti

Partendo dalle criticità individuate durante la fase di analisi, è stata studiata una soluzione che permettesse di affrontare in maniera omogenea l'intero flusso di lavoro tipico dello sviluppo di un'applicazione.

L'obiettivo fondamentale che è stato perseguito, infatti, mira ad estendere i vantaggi e le peculiarità dell'approccio Model Driven all'intero processo di sviluppo. Per ottenere un simile risultato, ci si è concentrati su una completa omogeneizzazione di tutte le fasi affrontate da un sviluppatore per giungere alla creazione di un modello completo di un'applicazione Web. Le considerazioni effettuate si basano sul caso specifico di WebML e di WebRatio, in quanto sono stati utilizzati come casi di studio per l'intero elaborato. Si noti comunque che, a meno dei dettagli che coinvolgono i due strumenti precedenti, molti degli aspetti citati si adattano bene al caso più generale del processo di sviluppo di applicazioni Web secondo una filosofia Model Driven.

In seguito alla specifica degli obiettivi, vengono proposti i requisiti che una successiva di implementazione dovrà rispettare per rispettare le specifiche del nuovo processo introdotto.

3.1 Obiettivi

Prima di procedere con la descrizione dei dettagli della particolare soluzione adottata, è necessario comprendere in che modo si è giunti all'integrazione di un editor WYSIWYG Model Driven all'interno del processo di sviluppo di un'applicazione Web.

3.1.1 Un nuovo flusso di lavoro integrato

La fase di analisi ha evidenziato come il processo attuale di creazione di un'applicazione risulta nettamente separato in due fasi distinte. Nella prima parte, la creazione del back-end, sono evidenti i vantaggi derivanti dall'utilizzo di un approccio Model Driven. Quando si tratta, invece, della costruzione del front-end delle applicazioni, è necessario ripiegare su una metodologia di sviluppo più tradizionale. Si è reso pertanto necessario lo studio di un processo unificato, che ampliasse i vantaggi di MDD all'intero flusso di lavoro.

Come già sottolineato, la creazione del modello di presentazione di WebML non è supportata da strumenti specifici, il che obbliga alla scrittura di tutto il codice sorgente a mano. Risulta perciò naturale l'utilizzo di un editor WYSIWYG a supporto di tale operazione. In particolare, nel caso specifico delle applicazioni Web, i linguaggi utilizzati per la creazione dell'aspetto visuale si prestano molto bene ad essere manipolati per mezzo di strumenti di questo tipo. Il loro utilizzo all'interno di un ambiente come quello di WebRatio risulta particolarmente conveniente, dato che la modellazione visuale dei

linguaggi HTML e JSP è piuttosto diffusa e ampiamente collaudata. In questo modo è possibile ottenere un livello di astrazione superiore, che limita notevolmente e può addirittura annullare gli interventi a livello del codice.

A questo punto, però, le due fasi risultano ancora separate: l'utilizzo dell'editor WYSIWYG semplifica il lavoro dello sviluppatore, ma, senza una vera integrazione con il modello, i vantaggi apportati risultano limitati. Si ha perciò la necessità di individuare uno strumento o una metodologia che si comporti come un vero e proprio collante tra il processo di creazione del back-end e il processo di generazione del front-end corrispondente. Abbiamo già osservato come la prima parte del flusso di lavoro sfrutti efficacemente il modello dei dati e quello di ipertesto e come i riferimenti tra questi due siano parte integrante del processo di sviluppo della logica di business. Pertanto, per omogeneizzare l'intero workflow, è sufficiente fare in modo che tutti e tre i modelli di WebML collaborino in un processo unificato.

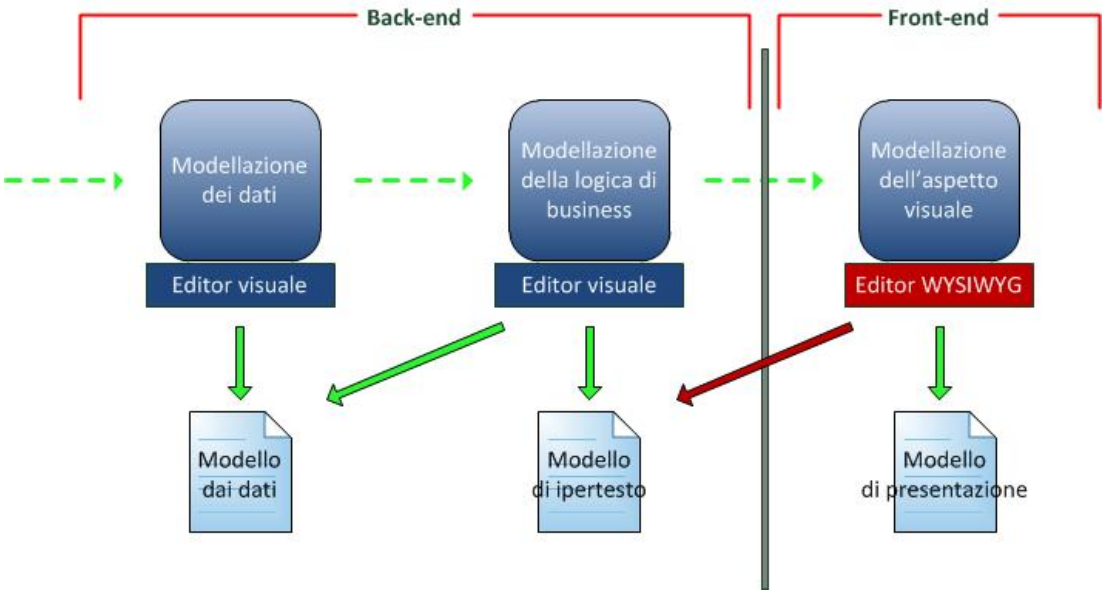

Figura 3.18: Il modello come collante tra la modellazione di back-end e front-end

Le funzionalità di modellazione possono essere estese anche alle fasi di creazione del front-end: come illustrato nella Figura 3.18, è sufficiente la costruzione di un editor WYSIWYG in grado di accedere alle informazioni contenute nel modello complessivo. Così, come il processo attuale fa interagire il modello di ipertesto con il modello dei dati, il modello di presentazione si può appoggiare ai primi due per permettere un più alto livello di astrazione e la generazione di interfacce coerenti con il back-end.

Si capisce, quindi, come l'utilizzo di un editor WYSIWYG di tipo Model Driven non solo fornisce un notevole aiuto in termini di semplicità d'uso e immediatezza, ma è parte integrante di un più ampio processo. In questo modo, è il modello stesso a fare da collante e spingere l'intero workflow verso un approccio unificato di tipo Model Driven. Ogni passo del flusso di lavoro entra a far parte di un più grande processo strutturato

in cui ogni strumento messo a disposizione ha la conoscenza necessaria per supportare al meglio lo sviluppatore, che ha la possibilità di concentrarsi sugli aspetti progettuali piuttosto che sui dettagli implementativi.

3.1.1 Un editor WYSIWYG Model Driven per WebRatio

Un approccio simile a quello descritto, oltre alla sua valenza teorica, deve dimostrare di avere un'applicabilità pratica. Prendendo in considerazione il caso di studio, WebRatio, è necessario studiare in che modo l'integrazione di simili modalità modifica il flusso di lavoro e come l'utilizzo dei nuovi strumenti introdotti può integrarsi al suo interno.

Prima di tutto, sono state studiate le modalità con cui un editor WYSIWYG può essere inserito all'interno del workflow di WebRatio; in particolare, si è posta grande attenzione su come l'utilizzo di questo nuovo strumento può entrare a far parte del ciclo di sviluppo di un'applicazione Web. Per ovvi motivi, si è preferito seguire un approccio che generasse un impatto che fosse il più limitato possibile.

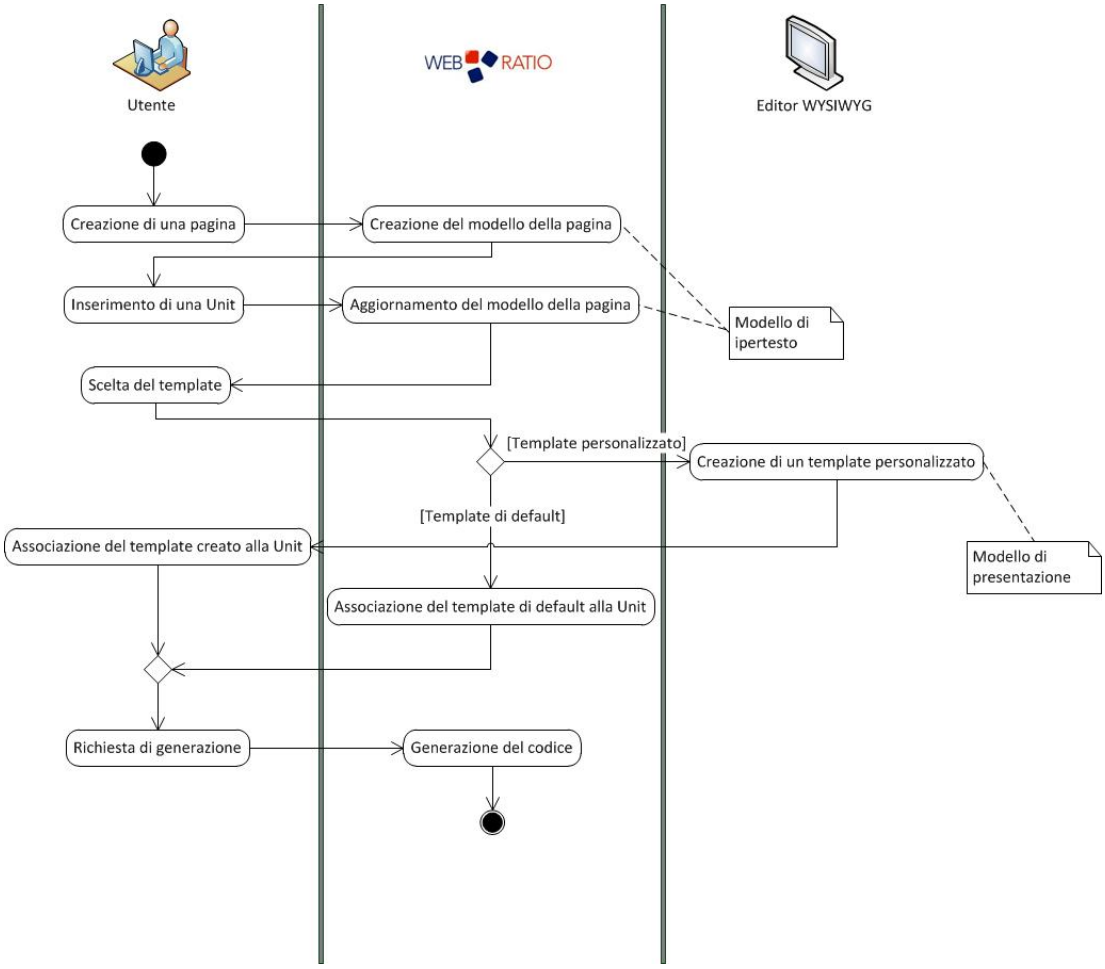

Figura 3.19: Activity Diagram del processo

Come si può osservare nella Figura 3.19, è bastato introdurre l'utilizzo di tale strumento nel momento della creazione di un Layout Template personalizzato per raggiungere l'obiettivo prefissato. In questo modo, l'editor WYSIWYG si sostituisce integralmente all'editor testuale attualmente presente. Si noti, comunque, come tale sostituzione non pregiudichi le funzionalità attualmente presenti: l'utente è ugualmente in grado di intervenire manualmente sul codice sorgente per mezzo dell'apposita vista disponibile all'interno della pagina di editing visuale.

Quello appena descritto è solamente il primo passo che porta alla reale integrazione dell'editor WYSIWYG nel workflow di WebRatio: infatti, esso deve poter utilizzare il modello WebML per permettere la modellazione delle interfacce. L'editor visuale, infatti, non deve limitarsi a supportare la creazione di pagine Web, ma deve interagire con il modello dei dati ed il modello di ipertesto per supportare l'utente nella creazione del un modello di ipertesto. Da non sottovalutare, poi, il fatto che quanto visualizzato nella finestra di editing visuale deve essere un'anteprima realistica di quello che sarai poi il risultato effettivo. Ciò rende assolutamente necessaria l'interazione da parte dell'editor WYSIWYG con l'intero modello, dato che le informazioni necessarie per generare l'aspetto visuale di un'applicazione sono contenute al suo interno.

Si nota immediatamente come il flusso di lavoro attuale sia stato modificato il meno possibile: l'utilizzo di un approccio Model Driven mira principalmente a semplificare quelle fasi che possono risultare particolarmente macchinose o molto dispendiose in termini di tempo. L'obiettivo, infatti, non è quello di stravolgere un processo ormai collaudato, ma piuttosto di risolvere l'evidente separazione tra i sottoprocessi di creazione del back-end di un'applicazione Web e del suo front-end. L'introduzione di un editor WYSIWYG di tipo Model Driven soddisfa in pieno tale esigenza.

3.2 Requisiti

Per modificare il flusso di lavoro attuale e consentire un approccio omogeneo durante l'intero processo di sviluppo, è stato scelto di integrare al suo interno un editor WYSIWYG Model Driven. Prima di tutto, è quindi necessario capire le modalità con cui il nuovo strumento deve interagire con l'architettura esistente.

3.2.1 Le modalità di integrazione

Il modello dei dati e il modello di ipertesto vengono creati nella prima fase del processo, utilizzando gli strumenti visuali messi a disposizione da WebRatio. Come risultato, si ha la generazione dei PIM, che rappresentano la logica di business e i dati su cui questa si appoggia. La seconda parte del processo riguarda la creazione del modello di presentazione, che, nel nuovo flusso di lavoro, viene realizzata per mezzo dell'editor WYSIWYG Model Driven.

Il nuovo strumento viene introdotto proprio in questa fase. Esso permette la modellazione diretta dell'aspetto visuale dell'applicazione e genera in output il modello di presentazione. Tuttavia, ciò non è sufficiente: quanto creato deve risultare coerente con i due modelli precedenti, il modello dei dati e il modello di ipertesto. In questo punto del processo, tali artefatti sono già disponibili. Pertanto, l'editor WYSIWYG, oltre ad occuparsi della generazione del modello di presentazione, deve supportare gli sviluppatori nella creazione di un aspetto grafico coerente con la logica di business. L'accesso alle informazioni presenti nei primi

due modelli risulta quindi fondamentale.

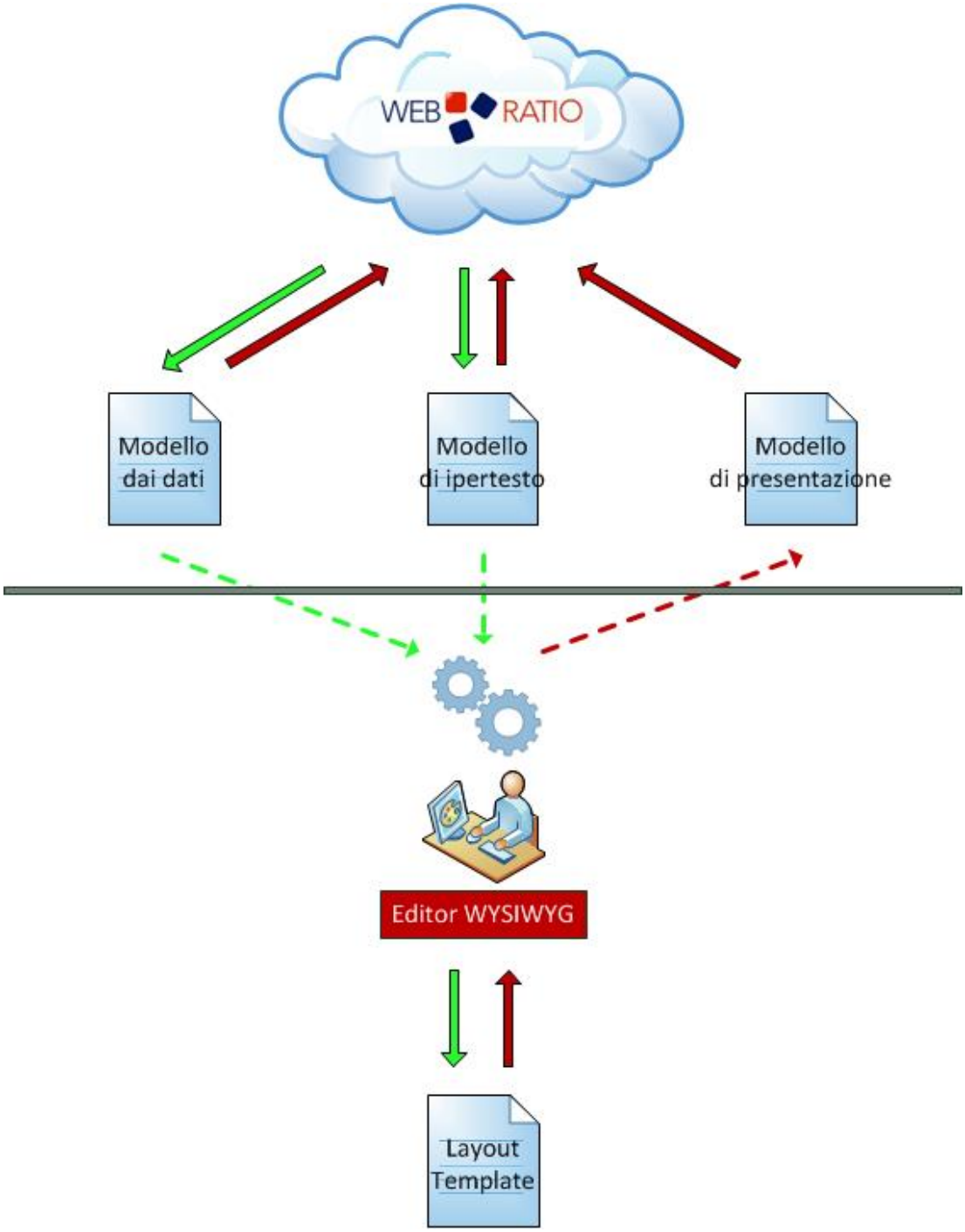

Figura 4.20: Schema architetturale di insieme; gli artefatti scambiati tra WebRatio e il prototipo

Come si nota nella Figura 4.20, il flusso di informazioni generato dall'editor non segue una sola direzione: esso ottiene tutte le informazioni di cui ha bisogno dal modello dei dati e dal modello di ipertesto, espone le proprie funzionalità in modo coerente a quando estrapolato e, infine, genera il modello di presentazione corrispondente a quanto creato dall'utente finale.

Scendendo più nel dettaglio, l'editor WYSIWYG agisce in realtà ad un livello più basso del modello di presentazione: i Layout Template di WebRatio. Ciò che viene generato, quindi, rappresenta l'aspetto visuale dei singoli componenti di cui è costituito il modello della logica di business dell'applicazione. Pertanto, a seconda dell'elemento su cui sta operando, il prototipo necessita solamente di un sottoinsieme delle informazioni contenute all'interno del modello dei dati e del modello di ipertesto.

Il modello di presentazione, quindi, non è altro che l'insieme di tutti i Layout Template corrispondenti a tutti gli elementi di cui sono costituite le pagine del modello di ipertesto. Sarà, poi, il generatore di codice ad occuparsi di effettuare la giusta combinazione di ognuno di questi componenti, in modo da generare l'interfaccia vera e propria del software.

Le funzionalità che permettono l'associazione tra un'istanza di un componente e la sua rappresentazione visuale, il Layout Template, sono già disponibili all'interno di WebRatio. Pertanto, per completare il ciclo di sviluppo di un'applicazione Web utilizzando un approccio Model

Driven in ognuna delle sue fasi, è sufficiente l'introduzione di uno strumento che supporti la modellazione ad alto livello di tutti gli aspetti grafici delle applicazioni, eliminando la necessità di ricorrere alla scrittura del codice sorgente. Il prototipo dovrà rispondere a tale necessità.

3.2.2 Le funzionalità

Dopo aver analizzato in che modo l'editor WYSIWYG Model Driven si inserisce all'interno del flusso di lavoro, è necessario comprendere quali funzionalità esso deve esporre per supportare l'utente nella generazione del modello di presentazione.

Prima di tutto, per consentire la manipolazione del modello del front-end, l'editor deve permettere la gestione di tutti gli elementi visuali di cui quest'ultimo è costituito. Nel caso di WebRatio, quindi, oltre ai tag tipici dei linguaggi HTML e JSP, è necessario introdurre il supporto ai tag specifici dei Layout Template. Come accade già per le tabelle, i campi di input e i bottoni, essi devono essere gestiti come elementi veri e propri elementi dell'interfaccia. Deve quindi essere possibile sfruttare tutte le funzionalità dell'editor WYSIWYG anche per tali componenti.

L'associazione tra il modello di presentazione e il modello della logica di business deve essere gestita interamente dal nuovo strumento: al momento, infatti, tale funzionalità non è prevista da WebRatio. L'editor WYSIWYG Model Driven deve occuparsi quindi dell'estrapolazione delle informazioni di cui ha bisogno direttamente dal modello dei dati e dal modello di ipertesto. Tali informazioni riguardano, in particolare, la

generazione dell'aspetto grafico dei tag dei Layout Template. Infatti, va ricordato che la fase di creazione del modello di ipertesto è solamente l'ultimo passo di un processo strutturato, in cui le funzionalità dell'applicazione sono già state specificate. L'editor WYSIWYG deve permettere la pubblicazione dei dati in maniera coerente a quanto descritto nella logica di business. Quali contenuti devono essere visualizzati dipende dal modello del back-end, ma in che modo tali contenti vengono resi disponibili all'utente finale viene descritto nel modello del front-end, di cui si occupa l'editor WYSIWYG Model Driven.

Infine, per generare un'anteprima realistica delle interfacce delle applicazioni, è stato scelto di generare alcuni dati casuali che sostituissero quelli reali disponibili solo a tempo di esecuzione. La tipologia di dati visualizzati all'interno dell'editor dipende direttamente dalla tipologia dell'attributo del modello dei dati che si sta visualizzando: pertanto, nel caso di un campo di tipo “password”, viene visualizzata una sequenza di asterischi; nel caso di un campo di tipo “data”, invece, viene visualizzata una stringa contenente una rappresentazione della data attuale.

- estrapolazione delle informazioni dal modello dei dati e di ipertesto
- associazione tra i tag di WR e gli elementi dei 2 modelli precedenti
- anteprima consistente

3.2.3 Il prototipo

In seguito alla comparativa effettuata, sono state prese in considerazione le possibili alternative da seguire per l'implementazione di un prototipo. Le strade percorribili sono risultate essere essenzialmente due: sviluppare una soluzione ad-hoc che aderisse ai requisiti individuati oppure appoggiarsi ad un prodotto esistente e riadattarlo secondo le nostre esigenze.

La prima strada, senza dubbio, avrebbe richiesto uno sforzo maggiore e avrebbe notevolmente aumentato i tempi di realizzazione. Inoltre, per la parte riguardante gli strumenti tipici di un editor WYSIWYG, una grossa fetta del lavoro avrebbe riguardato lo sviluppo di funzionalità già incontrate in molti prodotti e prive di caratteristiche innovative. Da non sottovalutare, poi, il fatto che l'aumento della complessità realizzativa sarebbe stato accompagnato da un aumento dei rischi associati al progetto.

Fortunatamente, è stato possibile intraprendere il secondo cammino. Infatti, tra i prodotti oggetto dell'analisi, ne è stato individuato uno che, per alcuni fattori distintivi, ben si presta per essere utilizzato come punto di partenza per lo sviluppo del prototipo, Eclipse WTP. Sotto il profilo delle funzionalità di editing visuale, esso presenta alcune caratteristiche comuni agli editor WYSIWYG oggetto dell'analisi. Inoltre, non sono state riscontrate particolari carenze che ne mettessero in dubbio il possibile utilizzo. Ciò che però ha fatto propendere per la sua adozione sono alcuni punti chiave che l'hanno distinto dagli altri.

In primo luogo, si tratta di un plug-in per Eclipse. Questo fatto rappresenta un vantaggio, dato che la piattaforma di riferimento, WebRatio, si appoggia completamente allo stesso ambiente di sviluppo. Inoltre, WTP sfrutta in modo massivo il meccanismo delle estensioni tipico di Eclipse, permettendo un alto grado di personalizzazione delle sue funzionalità di base. Infine, la licenza di questo prodotto, di tipo open source, ne permette l'utilizzo all'interno di un software commerciale. Quest'ultimo aspetto risulta di grande interesse: la disponibilità del codice sorgente può rivelarsi fondamentale nella fase implementativa, nel caso in cui gli aspetti di personalizzazione già previsti dallo strumento non risultino sufficienti per la realizzazione della soluzione finale.

3.2.4 il Web Page Editor di WTP

Avendo scelto Eclipse WTP come base di partenza per la realizzazione del prototipo, è stato necessario approfondire le funzionalità già presenti e gli interventi necessari per trasformarlo in un editor WYSIWYG Model Driven. In particolare, sono state prese in considerazione le possibilità di personalizzazione che permette il Web Page Editor. Per quanto riguarda gli strumenti di modellazione visuale, la copertura funzionale dell'editor di pagine Web di WTP si è dimostrata completa: pertanto, da questo punto di vista, non si è resa necessaria alcuna modifica. Ciò ha permesso di concentrare l'attenzione sugli aspetti del paradigma Model Driven che mancano del tutto all'interno di WPE e che quindi devono essere introdotti.

La scelta effettuata ha permesso di semplificare la parte iniziale del lavoro: piuttosto che cominciare da zero con lo sviluppo di una nuova soluzione, si è partiti da un editor WYSIWYG che offre pieno supporto ai linguaggi HTML e JSP. Nel pannello “Palette”, infatti, si possono trovare i tag tipici di questi due linguaggi, pronti per essere trascinati all'interno della finestra di editing visuale.

Per la realizzazione del prototipo, sono state sfruttate appieno tutte le funzionalità che rendono WPE un vero e proprio editor WYSIWYG. In particolare, il motore che si occupa del rendering dei componenti a video è risultato del tutto adatto alle necessità.

Ciò che, invece, manca del tutto all'interno di WPE sono gli strumenti necessari per la manipolazione diretta di un modello. Come vedremo, la fase di implementazione ha avuto come obiettivo fondamentale proprio tale modifica: la trasformazione di un semplice editor WYSIWYG in un editor Model Driven in grado di permettere la manipolazione diretta del modello di presentazione di WebRatio.

3.2.5 Gli “Extension Point” di WPE

Eclipse è basato su un'architettura modulare che utilizza componenti aggiuntivi che hanno lo scopo di estenderne le funzionalità di base. La maggior parte dei componenti che formano tale piattaforma sono essi stessi dei plug-in.

I plug-in di Eclipse non sono necessariamente del tutto indipendenti l'uno dall'altro, ma possono creare tra di loro una gerarchia: ognuno di

essi può definire uno o più “Extension Point” per permettere ad altri componenti di aggiungere nuove funzionalità. Il componente che definisce un “Extension Point” decide con quali modalità le sue funzionalità possono essere estese ed arricchite, ma, allo stesso tempo, ha la responsabilità di valutare e validare il contributo degli altri plug-in. Al contrario, un componente che intende estenderne un altro deve dichiararlo esplicitamente all'interno dei suoi file di configurazione. In questo modo, si crea un accoppiamento debole che permette di creare un'associazione tra plug-in differenti; il fatto che sia un legame debole garantisce comunque la massima flessibilità all'utente, che può decidere liberamente a quali funzionalità è interessato al momento dell'installazione. In questo modo, ogni plug-in si troverà ad avere delle dipendenze che devono essere necessariamente soddisfatte per il suo corretto funzionamento.

Eclipse incorpora al suo interno tutti gli strumenti necessari per la creazione di un plug-in. Come è possibile osservare nella Figura 4.21, tramite una semplice interfaccia grafica viene data la possibilità di configurare tutti i parametri disponibili. In particolare, è possibile specificare quali altri plug-in vengono estesi da quello attuale e quali nuovi “Extension Point” quest'ultimo intende mettere a disposizione.

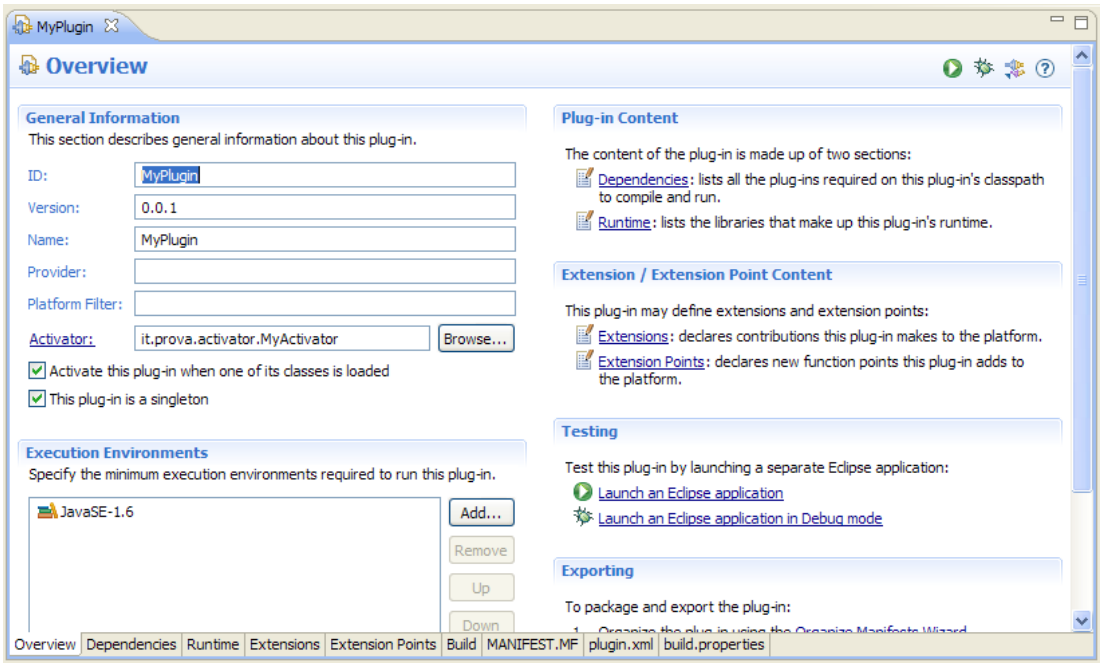

Figura 4.21: Dettaglio della vista di configurazione di un plug-in di Eclipse

WPE definisce numerosi punti di estensione, che permettono una profonda personalizzazione delle funzionalità disponibili. In particolare, per la realizzazione del prototipo, sono stati sfruttati i seguenti "Extension Point" di WTP:

- *org.eclipse.jst.jsf.common.standardMetaDataFiles*: permette di aggiungere il supporto a Tag Library non incluse all'interno di WPE di default; viene tipicamente sfruttato per l'inserimento di nuovi tag all'interno del pannello "Palette";
- *org.eclipse.jst.pagedesigner.pageDesignerExtension*: permette di definire quali classi Java del plug-in si occuperanno di gestire i vari aspetti dell'editor WYSIWYG, come la resa visuale dei tag, il popolamento dei menù o le azioni da intraprendere in seguito ad

alcuni eventi;

- *org.eclipse.jst.jsf.core.AttributeValueRuntimeTypes*: permette di definire quali componenti si occuperanno di verificare se un particolare valore è valido o meno per l'attributo considerato; in particolare, è sfruttato dal pannello “Properties” per validare l'input dell'utente oppure per fornire la lista di valori permessi per un determinato attributo.

Risulta subito evidente come gli “Extension Point” messi a disposizione da WTP sono risultati più che sufficienti per estendere le funzionalità WYSIWYG presenti in WPE, in modo da supportare appieno la generazione dei Layout Template di WebRatio.

Capitolo 4

Editor WYSIWYG per sviluppo Model Driven

L'ultima parte del lavoro svolto ha riguardato la realizzazione del prototipo a supporto delle tesi proposte. Quest'ultima fase si è concentrata sul caso di studio e, in particolare, sulla descrizione delle scelte implementative che hanno permesso di risolvere le problematiche incontrate.

Come già anticipato, la realizzazione del prototipo segue fedelmente tutte le considerazioni effettuate in precedenza e fornisce un riscontro pratico alle soluzioni che sono state individuate. Per questo motivo, esso non si configura come un'applicazione a se stante, ma nasce come un vero e proprio componente integrato all'interno del flusso di lavoro di WebRatio.

La trattazione, volutamente, non si dilunga troppo sui dettagli di basso livello: dove possibile, si è cercato di fornire indicazioni di carattere generale, che non si applicano quindi al solo caso in esame. Nella maggior parte dei casi, ai fini della comprensione delle scelte effettuate, vengono forniti esempi che si riferiscono in modo specifico all'implementazione effettuata.

4.1 Design

Prima di procedere con l'implementazione vera e propria, è stata affrontata la fase di progettazione, che ha permesso di verificare che le possibilità di personalizzazione offerte da WPE fossero sufficienti per le esigenze individuate. Nello specifico, è stato appurato che tutte le considerazioni teoriche fossero effettivamente implementabili all'interno di una soluzione reale.

Come già anticipato, il prototipo di basa completamente su WPE e ha come obiettivo l'estensione delle funzionalità offerte per giungere alla creazione di un editor WYSIWYG Model Driven. Il punto di partenza di questa fase è rappresentato dagli Extension Point definiti all'interno di WPE. Questi ultimi, infatti, sono il punto di ingeresso da cui è possibile generare una nuova soluzione, dotata di un insieme più ampio di funzionalità.

4.1.1 org.eclipse.jst.jsf.common.standardMetaDataFiles

L'Extension Point *org.eclipse.jst.jsf.common.standardMetaDataFiles* permette di aggiungere il supporto a Tag Library personalizzate all'interno di WPE. Sfruttando questo meccanismo, è quindi possibile inserire nuovi tag, oltre a quelli già presenti, in modo da estendere il supporto da parte dell'editor WYSIWYG ad altri elementi visuali. Tale possibilità può essere ampiamente sfruttata dal prototipo per aggiungere il supporto ai tag tipici dei Layout Template di WebRatio. Per ognuna delle Unit che si vogliono supportare, quindi, è necessario creare una

nuova Tag Library e fornire a WPE i relativi dettagli, specificati all'interno di un file contenente i relativi metadati, in modo che questa possa essere opportunamente gestita. All'interno del file dei matadati, infine, è possibile indicare, per ogni tag, quale particolare istanza degli Extension Point *org.eclipse.jst.pagedesigner.pageDesignerExtension* e *org.eclipse.jst.jsf.core.AttributeValueRuntimeTypes* sarà utilizzata per gestire tutti gli altri aspetti dell'editor WYSIWYG.

4.1.2 org.eclipse.jst.pagedesigner.pageDesignerExtension

Il cuore dell'editor WYSIWYG di WPE può essere modificando usando l'Extension Point org.eclipse.jst.pagedesigner.pageDesignerExtension. Esso permette di estendere la classe astratta *AbstractEditFactory* e generare così una nuova Factory, che si occupa della gestione delle azioni intraprese dall'utente. In particolare, come si può osservare nella Figura 4.22, attraverso l'estensione delle classi *AbstractElementEdit* e *AbstractDropCustomizer*, è possibile personalizzare il comportamento del prototipo in seguito alla creazione di un nuovo elemento visuale o alla sua successiva modifica all'interno dell'editor WYSIWYG. È proprio in questo momento che può avvenire il collegamento tra il modello del front-end e il modello del back-end: dopo il trascinamento di un tag all'interno della finestra oppure dopo la modifica di qualche suo attributo, infatti, il prototipo può richiedere all'utente a quale componente del modello della logica di business il tag corrente deve essere associato. In base alla scelta effettuata, saranno i componenti che gestiscono la resa visuale a dover creare un aspetto grafico coerente con

la selezione effettuata.

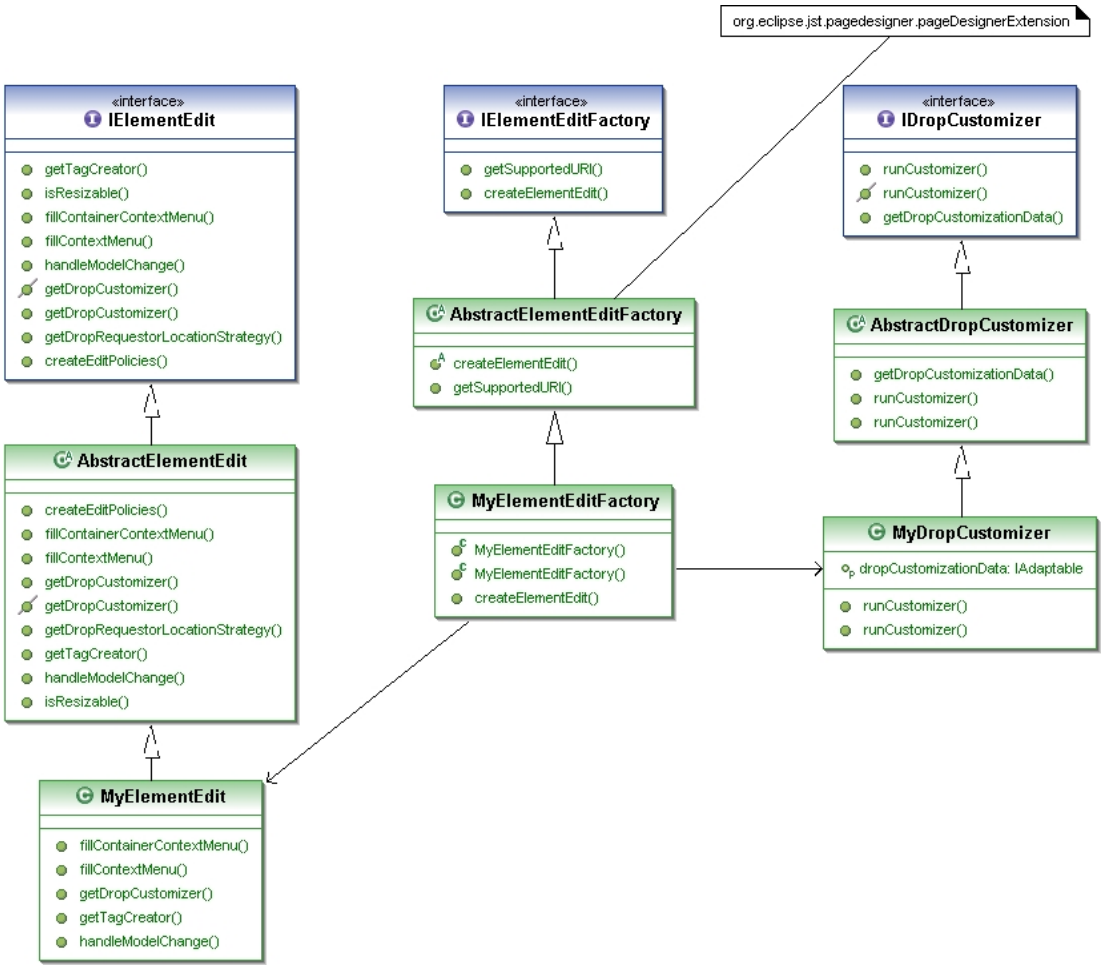

Figura 4.22: Class Diagram dell'Extension Point org.eclipse.jst.pagedesigner.pageDesignerExtension: dettaglio riguardante le funzionalità di gestione visuale dei tag

Gestione della resa visuale che viene gestita da particolari sottoclassi della classe astratta *AbstractTransformOperation*. All'interno di tali componenti, vengono gestiti tutti gli aspetti visuali di ogni tag presente all'interno dell'editor WYSIWYG. Come detto, tale meccanismo può essere sfruttato dal prototipo per generare una resa visuale coerente con la selezione effettuata dall'utente e, in particolar modo, con quanto

specificato nel modello dei dati e di ipertesto.

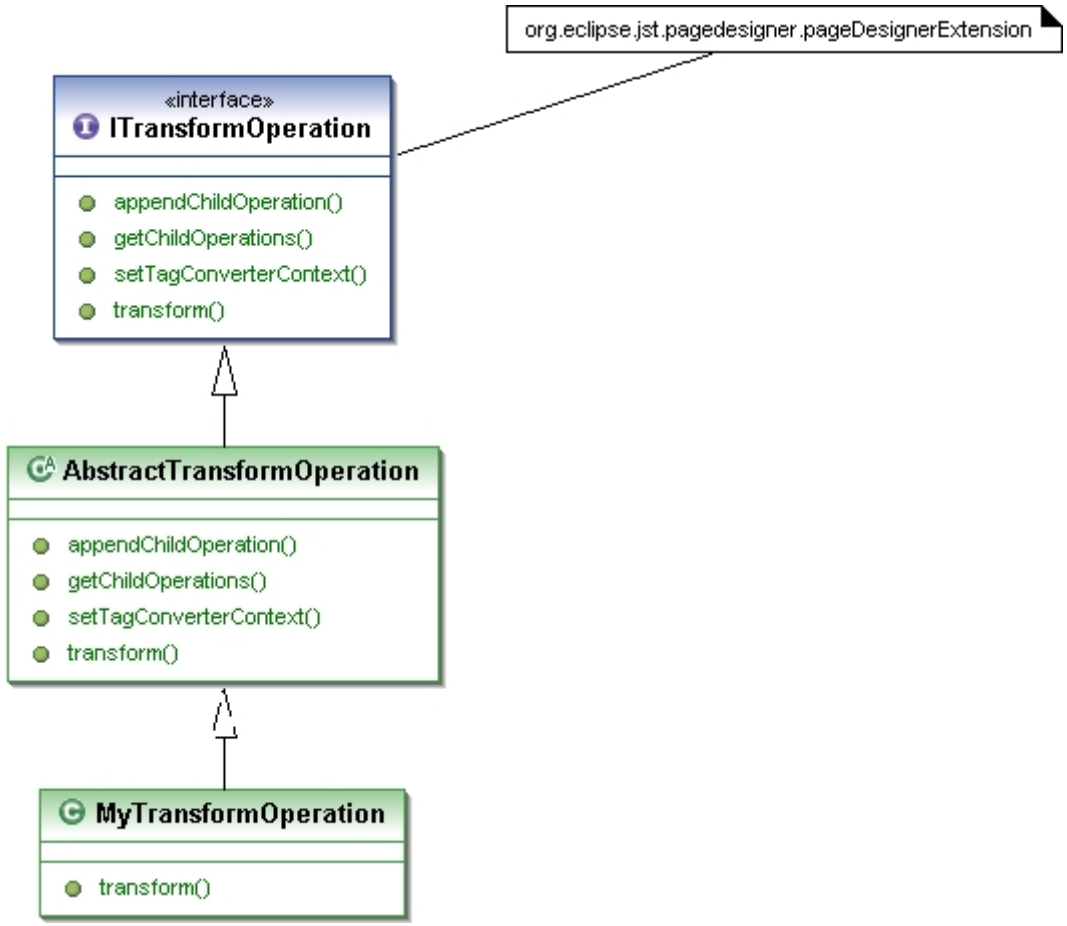

Figura 4.23: Class Diagram dell'Extension Point org.eclipse.jst.pagedesigner.pageDesignerExtension: dettaglio riguardante le funzionalità di rendering dei tag di WebRatio

4.1.3 org.eclipse.jst.jsf.core.AttributeValueRuntimeTypes

L'Extension Point *org.eclipse.jst.jsf.core.AttributeValueRuntimeTypes* e, in particolare, la classe astratta *EnumerationType* risultano di grande utilità per migliorare l'usabilità ed impedire errori dovuti ad input errati da parte dell'utente. Tali componenti, infatti, si occupano di verificare la validità dei valori specificati dallo sviluppatore a partire da una lista di valori ammessi. Il prototipo può sfruttare questo meccanismo per

supportare l'utente nel momento in cui deve essere effettuata l'associazione tra un tag e il corrispondente elemento del modello del back-end. Semplicemente accedendo alle informazioni contenute all'interno del modello della logica di business, esso può informare lo sviluppatore circa i componenti presenti all'interno di tale modello. In questo modo, l'associazione tra front-end e back-end viene notevolmente semplificata.

4.1.4 Interazione con l'utente

Come risulta evidente nella Figura 4.24, tutti gli elementi analizzati concorrono alla generazione del risultato atteso: un editor WYSIWYG che agisca secondo un approccio Model Driven. L'utente, durante l'utilizzo del prototipo, si limita ad interagire con un editor visuale che permette la manipolazione diretta del modello di ipertesto. In questo modo, lo sviluppatore si trova a gestire solo gli aspetti di alto livello. Tutti gli altri aspetti che non riguardano strettamente l'interfaccia grafica del software sono nascosti. Infatti, è la logica implementata all'interno del prototipo che si occupa di interagire con il modello del back-end e generare di conseguenza un'anteprima realistica dell'aspetto visuale dell'applicazione. Tutto quello che lo sviluppatore era chiamato a gestire esplicitamente, ora è gestito direttamente dall'editor WYSIWYG.

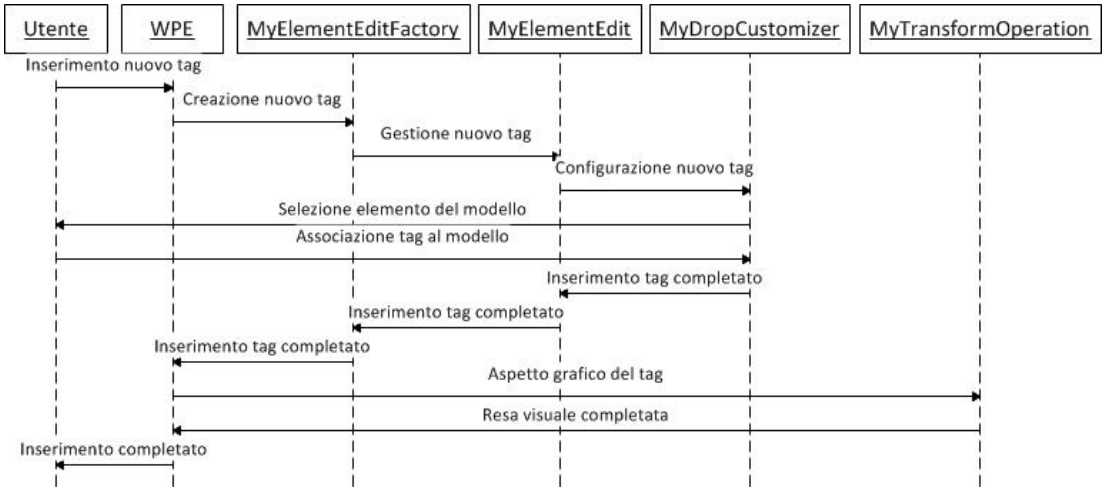

Figura 4.24: Sequence Diagram della fase di inserimento di un nuovo tag all'interno dell'editor WYSIWYG

4.2 Implementazione

La prima parte delle attività ha portato alla creazione di un nuovo plug-in per Eclipse che basa le sue funzionalità su quanto offerto da WPE e le estende ulteriormente per ottenere un editor WYSIWYG Model Driven. Per prima cosa, sono state poste le basi per trasformare l'editor visuale in un software di progettazione basato su un modello.

4.2.1 Associazione con il modello

È evidente come non sia sufficiente l'utilizzo di un semplice editor WYSIWYG per poter costruire un modello di presentazione coerente con il modello dei dati e il modello di ipertesto. Il prototipo, quindi, per poter generare una corretta resa visuale dei tag specifici dei Layout Template, deve avere una conoscenza approfondita della specifica del back-end dell'applicazione.

Il modello, nel caso in esame, è costituito da diversi file XML che descrivono la logica di business e i dati a cui essa può accedere. Attraverso l'utilizzo dei linguaggi XPath e XQuery, il prototipo è in grado di estrapolare dal modello dei dati e di ipertesto tutte le informazioni di cui l'editor visuale avrà bisogno per la generazione del modello di presentazione e, in particolar modo, dell'anteprima della resa grafica dell'applicazione. In questo modo, l'editor WYSIWYG espone tutte le funzionalità che permettono la modellazione dell'aspetto grafico, senza avere una conoscenza diretta delle logiche del modello.

Questo tipo di accoppiamento debole tra le logiche di WebRatio e le

logiche dell'editor WYSIWYG permette una grande flessibilità. Infatti, eventuali modifiche di uno dei due componenti non si riflettono necessariamente sull'altro. Nel caso in cui venga modificato il modello dei dati o il modello di ipertesto, è possibile applicare gli stessi cambiamenti allo strato software del prototipo che si occupa di estrarre i dati, senza intervenire sugli aspetti di modellazione grafica. Nel caso contrario, invece, il vantaggio è ancora più evidente: le funzionalità dell'editor WYSIWYG non influenzano minimamente l'operato di WebRatio; è quindi sufficiente che l'output da esso generato continui a rispettare le specifiche del modello di presentazione.

WPE nasce come un editor WYSIWYG per la creazione di pagine Web. Le funzionalità di alto livello che mette a disposizione sono principalmente incentrate sulla modellazione dell'aspetto grafico. Ciò che risulta limitato o del tutto assente è il supporto alla costruzione del front-end in maniera integrata con il contenuto del relativo back-end.

Come più volte sottolineato, un simile aspetto risulta fondamentale ai fini della realizzazione del prototipo. Per poter ottenere un approccio completamente integrato, infatti, l'editor visuale deve permettere la manipolazione degli elementi grafici in modo coerente con quanto specificato nella logica di business dell'applicazione. Una volta ottenuto l'accesso alle informazioni contenute nel modello dei dati e di ipertesto di WebRatio, la fase di implementazione si è quindi concentrata sugli aspetti di modellazione visuale e sulle modalità con cui tali operazioni devono ricollegarsi al contenuto di questi due modelli. La creazione dei

Layout Template, infatti, non può prescindere dal risultato dalle fasi precedenti del flusso di lavoro; anzi, dipende direttamente da queste.

Si noti come questa fase ha generato un effetto collaterale molto positivo: grazie agli interventi effettuati, il ricorso alla scrittura del codice è risultato del tutto superfluo.

4.2.2 Supporto ai tag di WebRatio

WPE, come impostazione di partenza, offre il supporto ai tag principali dei linguaggi HTML e JSP. Attraverso un semplice meccanismo, permette di espandere ulteriormente la lista di tag a disposizione dello sviluppatore. Inoltre, come conseguenza di tale operazione, il pannello "Palette" subisce una modifica: al suo interno vengono inseriti i tag di cui è composta la Tag Library appena importata.

Tale meccanismo è stato ampiamente utilizzato per aggiungere il supporto ai tag specifici dei Layout Template di WebRatio all'interno del prototipo. È stata sufficiente la creazione di una libreria per ognuna delle Unit di riferimento, che comprendesse al suo interno i tag necessari per la costruzione dell'aspetto visuale della Unit stessa.

Una Tag Library non è altro che un archivio nel formato Jar formattato secondo regole ben precise: oltre a contenere le classi che gestiscono la logica corrispondente ai tag, comprende un descrittore, il cosiddetto TLD (Tag Library Description). Quest'ultimo è un file XML che descrive la libreria e i tag in essa contenuti, fornendone nome, descrizione e altre utili informazioni.

Il prototipo, nella sua prima versione, fornisce il supporto ai tag relativi alle Data Unit, Multi Data Unit ed Entry Unit; è stata quindi necessaria la creazione di altrettante Tag Library. Naturalmente, cambiando la Unit, cambia anche la lista di tag che la relativa libreria deve contenere. Come descritto in precedenza, per le Data Unit sono sufficienti i tag "Value" e "Label"; per le Multi Data Unit è necessario il supporto al tag aggiuntivo "Iterate", mentre per le Entry Unit è sufficiente l'aggiunta del tag "Link". Una volta effettuata l'importazione di questi tre componenti, il risultato ottenuto a livello del pannello "Palette" è stato quello osservabile nella Figura 4.25.

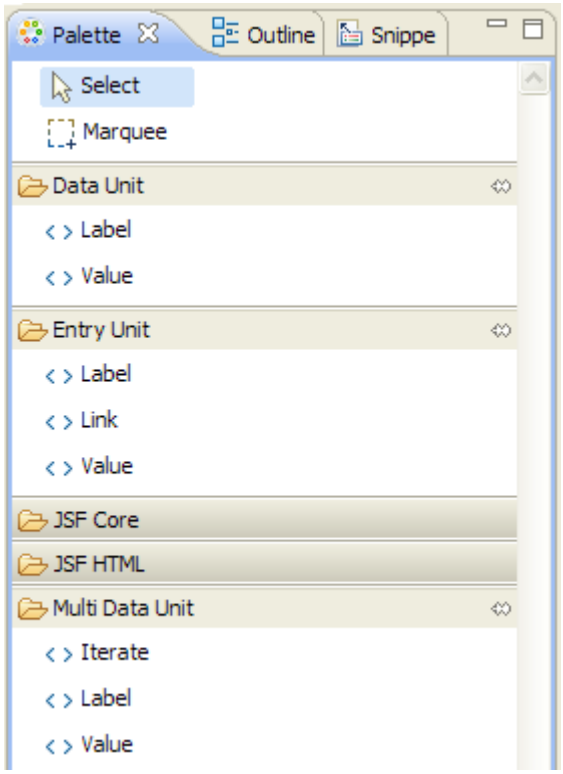

Figura 4.25: Dettaglio del pannello Palette in seguito all'importazione della libreria di tag personalizzata

Per ovvie ragioni, non è sufficiente l'aggiunta di una libreria di tag all'interno di WPE per poter sfruttare appieno le funzionalità messe a disposizione dall'editor WYSIWYG. Infatti, per ottenere il pieno supporto per una Tag Library personalizzata, è necessario ricorrere all'Extension Point *org.eclipse.jst.jsf.common.standardMetaDataFiles*. Quest'ultimo permette di specificare un file XML di configurazione per

ogni libreria di tag. Esso non contiene altro che tutti i metadati necessari per una corretta configurazione della Tag Library e, di conseguenza, un corretto utilizzo di quest'ultima da parte del prototipo.

I file dei metadati rappresentano il punto nevralgico per la personalizzazione dell'editor visuale e, pertanto, risultano fondamentali per il prototipo. Essi, infatti, permettono di modificare il comportamento dell'editor in modo da adattarlo il più possibile alle esigenze di differenti soluzioni. Nel caso in esame, è stato possibile personalizzare le funzionalità dell'editor WYSIWYG, modificandone il comportamento nella gestione dei tag specifici di WebRatio.

La funzionalità più importante ai fini della resa grafica della libreria di tag è quella che, all'interno del file dei metadati, permette di specificare il componente che si occuperà del rendimento visuale di ogni tag.

4.2.3 Rendering dei tag di WebRatio

Per ognuno dei tag che costituiscono una libreria, WPE permette di personalizzare molti fattori, che vanno dalla semplice gestione delle operazioni di manipolazione visuale, agli aspetti di basso livello che riguardano la resa grafica vera e propria.

Ai fini della realizzazione del prototipo, sono state pesantemente modificate le logiche che riguardano il rendering a video dei tag delle Unit di WebRatio. In particolare, gli interventi effettuati hanno avuto come scopo principale il collegamento tra i singoli tag e i componenti del modello di ipertesto. In questo modo, la creazione del modello di

presentazione dipende direttamente dal modello dei dati e di ipertesto e risulta così completamente integrata all'interno del flusso di sviluppo di un'applicazione.

Per ogni tag di WebRatio, il prototipo estende le funzionalità di WPE per permettere la configurazione sia degli aspetti di carattere visuale, sia dell'associazione con gli altri due modelli. Non solo esso permette di specificare l'aspetto visuale del tag stesso, ma anche a quale componente del modello di ipertesto esso si riferisce. Ogni tag, quindi, ha associati sia gli elementi grafici che ne costituiranno la resa visuale, sia il componente del back-end di cui visualizzerà il contenuto. Naturalmente, lo sviluppatore deve poter decidere dove deve essere posizionato all'interno dell'interfaccia il valore ricevuto dalla parte di back-end. L'editor WYSIWYG è stato modificato per permette di specificare dove tale contenuto verrà visualizzato per mezzo di un particolare placeholder: in questo modo, allo sviluppatore è lasciata la massima flessibilità sia per quanto riguarda la creazione dell'aspetto visuale, sia per il posizionamento dei contenuti al suo interno. Infatti, il placeholder può essere inserito in qualunque posizione dell'interfaccia e il suo contenuto sarà opportunamente sostituito a tempo di esecuzione con i dati provenienti dalle entità specificate nel modello dei dati e opportunamente rielaborati dalla logica di business. Naturalmente, per fornire un'anteprima completa all'interno dell'editor visuale, è stato previsto un meccanismo che genera una resa visuale anche per tale componente, in maniera coerente con quanto specificato all'interno del

modello dei dati e di ipertesto. A seconda della tipologia di attributo a cui è stato associato il tag, il contenuto del relativo placeholder viene sostituito da una stringa contenente dei dati di esempio, in modo che l'interfaccia visualizzata dall'editor risulti verosimile rispetto a quella reale. Ad esempio, se si considera un attributo di tipo testuale, all'interno della finestra di editing visuale esso sarà visualizzato come una stringa di caratteri; se, invece, l'attributo è di tipo "password", verrà visualizzata una sequenza di asterischi. La resa grafica di un tag restituita dall'editor WYSIWYG dipende, quindi, da due fattori: gli elementi grafici con cui è stato costruito e le proprietà del componente ad esso associato all'interno del modello di ipertesto.

In questo modo, l'anteprima visualizzata dall'editor WYSIWYG risulta compatibile con quanto specificato all'interno del modello di ipertesto di WebRatio e genera così un risultato molto vicino a quello reale. Un simile approccio si sposa molto bene con il paradigma Model Driven: permette la creazione dei prototipi delle interfacce in tempo reale; inoltre, il riscontro immediato che viene dato, evita allo sviluppatore di dover ricorrere al generatore di codice per avere un'idea dell'aspetto della soluzione generata.

4.2.4 Creazione del Layout Template

Per completare l'integrazione dell'editor WYSIWYG Model Driven all'interno del flusso di lavoro, è necessario che quest'ultimo sia in grado di generare in output Layout Template che rispettino le specifiche del modello di presentazione di WebRatio. In questo modo, il generatore di

codice è poi in grado di procedere alla costruzione dell'intera soluzione.

L'ultimo passo implementativo, quindi, ha avuto come obiettivo la creazione del Layout Template vero e proprio. La sua generazione deve essere effettuata in maniera esplicita: l'editor WYSIWYG, infatti, opera su pagine Web complete, contenenti i tag specifici dei linguaggi HTML e JSP, oltre a quelli introdotti nel corso degli sviluppi. Il risultato atteso, invece, è il file contenente il template che rappresenta il codice che verrà utilizzato per generare l'aspetto grafico di una specifica Unit. Per questo motivo, esso rappresenta solamente un frammento di codice che verrà incluso all'interno di una pagina al momento della generazione dell'applicazione Web vera e propria. Inoltre, l'editor WYSIWYG supporta alcuni tag specifici che non possono essere semplicemente copiati all'interno del template, ma devono essere opportunamente gestiti.

Per mantenere un debole accoppiamento tra l'editor WYSIWYG e il modello vero e proprio dell'applicazione, è stata studiata una soluzione che permettesse la generazione del Layout Template senza dover dipendere dalle logiche di WebRatio. Fortunatamente, in questo caso, le logiche stesse di quest'ultimo hanno semplificato il lavoro: il file di un Layout Template, infatti, risulta già indipendente rispetto al modello di ipertesto e la sua associazione ad una Unit deve essere esplicitamente richiesta da parte dell'utente per mezzo di una apposita interfaccia grafica.

Il template generato per mezzo dell'editor visuale non è direttamente

utilizzabile da WebRatio. Esso va opportunamente sfoltito di quegli elementi che non possono far parte di un Layout Template. Per prima cosa si procede con l'eliminazione di tutte le intestazioni tipiche di una pagina HTML, come il tag “head” e tutti i suoi discendenti. Successivamente, viene estrapolato il contenuto del tag “body” e quest'ultimo, dopo una opportuna trasformazione, viene utilizzato per generare il Layout Template corretto. L'operazione di trasformazione si occupa solamente di effettuare una traduzione: infatti, i tag specifici di WebRatio, come “Value” e “Label”, vengono tradotti nei corrispondenti tag JSTL, in modo che il server sia in grado di gestirli e generare così l'output corretto.

Per limitare l'intervento da parte dell'utente, la creazione del Layout Template avviene in maniera automatica ad ogni salvataggio. Sfruttando il meccanismo dei listener messo a disposizione da Eclipse, è stato possibile configurare un componente che viene attivato ogni volta che l'utente invoca un salvataggio all'interno della pagina di editing visuale. Tale componente è proprio quello che si occupa di estrarre le informazioni dall'editor WYSIWYG, trasformarle nel modo opportuno ed infine generare il file del Layout Template.

In questo modo, senza che l'utente debba intervenire in maniera diretta, WebRatio sarà in grado di generare un'applicazione Web dotata di un'interfaccia del tutto simile a quanto visualizzato nella finestra di editing visuale.

4.2.5 Risultato finale

La Figura 4.26 dimostra le capacità del prototipo di fornire un riscontro immediato allo sviluppatore. Senza scrivere una riga di codice, è stato possibile generare l'interfaccia di una semplice form di login, la cui logica è specificata nel modello di ipertesto, generato utilizzando gli strumenti disponibili all'interno di WebRatio.

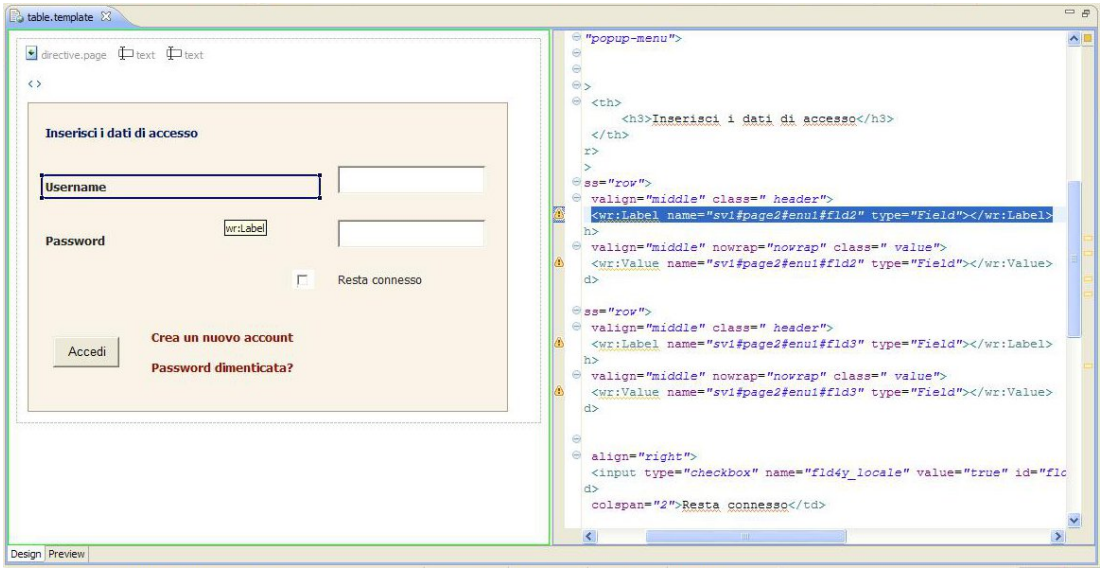

Figura 4.26: Esempio di una form di login creata tramite il plugin WYSIWYG

Si nota immediatamente come l'anteprima visualizzata dall'editor WYSIWYG risulta del tutto comparabile all'aspetto grafico finale che avrà l'applicazione vera e propria. In questo modo, risulta del tutto inutile il ricorso al generatore di codice, fintanto che l'intero processo non è stato completato e si rende necessaria la generazione del codice sorgente dell'intero progetto.

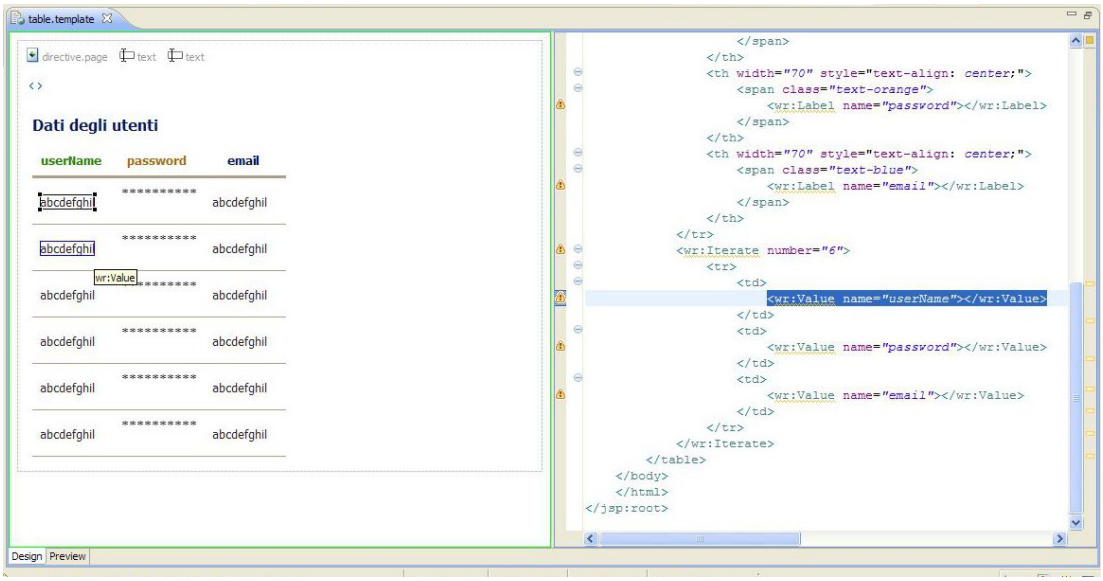

Figura 4.27: Esempio di utilizzo del tag Iterate per simulare la generazione del contenuto di una tabella

Nella Figura 4.27 è visualizzato un ulteriore esempio delle potenzialità del prototipo: in questo caso, il contenuto dell'anteprima è generato dinamicamente in modo da fornire un'idea ancora più precisa di come cambia l'interfaccia grafica in base al contenuto disponibile a tempo di esecuzione. All'interno dell'editor visuale, infatti, nel caso delle Multi Data Unit, è possibile selezionare per quante istanze della stessa entità deve essere generato l'aspetto visuale. Lo sviluppatore ottiene così un ulteriore supporto durante la fase di progettazione, che può risultare molto prezioso per risparmiare parecchi passaggi inutili.

4.3 Ottimizzazione

Alcuni interventi di personalizzazione di WPE sono stati effettuati per migliorare l'usabilità del prototipo finale. In particolare, sono stati realizzati alcuni affinamenti che mirano a migliorare l'esperienza utente e a semplificare il compito dello sviluppatore.

Durante la fase di analisi, uno dei prodotti che ha maggiormente colpito per la presenza di alcune soluzioni particolarmente efficaci è OEPE. Esso, infatti, in seguito al trascinamento di un tag all'interno dell'area di editing visuale, visualizza una nuova finestra che presenta un comodo percorso guidato che permette di specificare i parametri principali del componente che sta per essere creato. Ad esempio, nel caso di una tabella, è possibile selezionare il numero di colonne e di righe di cui è composta.

Benché tale strumento non rivoluzioni l'editor WYSIWYG nella sua essenza, è risultato subito evidente come questo migliori significativamente l'esperienza utente. In molti casi, esso diminuisce i passi necessari per ottenere il risultato voluto, semplificando così il lavoro dello sviluppatore. Si è pertanto deciso di adottare un approccio analogo: il trascinamento di uno dei tag dei Layout Template all'interno dell'editor visuale causa l'apertura di una nuova finestra. Essa ha lo scopo di aiutare l'utente nella creazione dei componenti personalizzati, in modo che questi risultino da subito coerenti con il modello dell'applicazione. Questo comportamento, inoltre, migliora la resa visuale complessiva dell'editor WYSIWYG: quest'ultimo, infatti, non è in grado di generare

un aspetto grafico per quegli elementi non ancora associati al modello di WebRatio. Il risultato, seppur temporaneo, è un'interfaccia spoglia, in cui, al posto dei componenti personalizzati, viene visualizzato il nome del tag che li rappresenta.

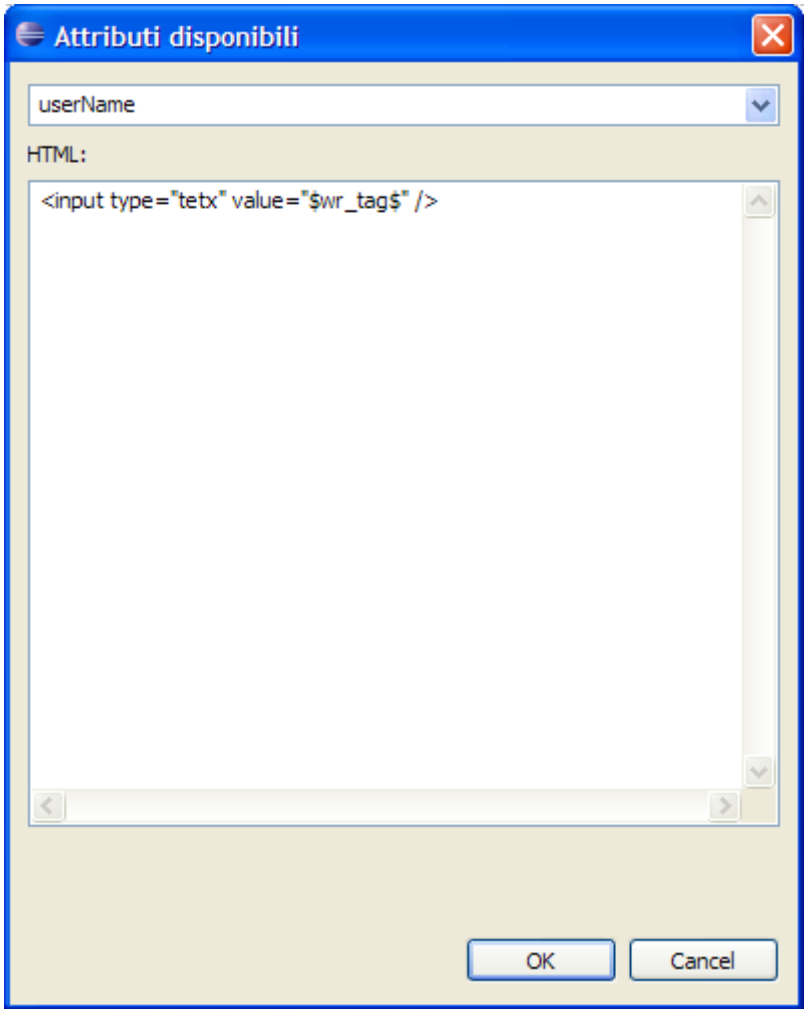

Figura 4.28: Dettaglio della finestra che guida nella creazione di un nuovo tag

Molto spesso i valori inseriti dall'utente possono presentare delle imprecisioni e generare così degli errori in fase di compilazione dell'intera applicazione. Per evitare che l'utente si accorga di tali

situazioni solamente nella parte finale del processo di sviluppo, quando interviene il generatore di codice, sono stati introdotti alcuni accorgimenti specifici. In particolare, per ogni attributo dei tag direttamente associato ad un elemento del modello di WebRatio, viene messo a disposizione un menù all'interno del pannelo “Properties” da cui l'utente può scegliere il valore corretto. Nel caso in cui tale associazione non fosse presente, viene comunque eseguito un processo di validazione che controlla l'effettiva validità dei dati inseriti.

Capitolo 5

Conclusioni

L'analisi del processo attuale di sviluppo di un'applicazione Web secondo un approccio Model Driven ha posto le basi per l'intero lavoro svolto. Si è notato come la fase di creazione del back-end avviene seguendo modalità ben distinte rispetto alla parte che riguarda la generazione del font-end. Partendo da questa considerazione, sono stati evidenziati i punti critici che generano tale spaccatura. La metodologia MDD, infatti, risulta completamente applicabile solamente durante le fasi di creazione della logica di business: lo sviluppatore può concentrare i suoi sforzi sulle attività di progettazione e modellazione, senza doversi preoccupare degli aspettii implementativi. Nel momento in cui si passa alla generazione dell'aspetto visuale del software, invece, l'approccio Model Driven risulta scarsamente utilizzato e in molti casi del tutto assente. I metodi di progettazione di alto livello vengono sostituiti da metodologie di programmazione più tradizionali, che richiedono la scrittua manuale del codice sorgente e distolgono l'attenzione dal dominio del problema. È stato quindi proposto un nuovo processo, che prevedesse un approccio omogeneo lungo tutto il ciclo di sviluppo di un'applicazione Web. Le criticità individuate durante la fase di analisi sono state risolte introducendo la filosofia Model Driven anche durante la creazione delle interfacce degli applicativi. Le fasi ancora basate su operazioni ripetitive,

time-consuming e ad alto tasso di errore sono state sostituite da uno strumento che ha permesso di semplificare motli compiti: un editor WYSIWYG. La completa integrazione tra la generazione del back-end e del front-end è stata ottenuta utilizzando il modello esistente come un vero e proprio collante, in modo da omogeneizzare l'intero flusso di lavoro. Il risultato finale è stato raggiunto elevando il livello di astrazione durante l'ultima fase del processo: l'editor visuale è stato trasformato in un editor WYSIWYG Model Driven, uno strumento in grado di supportare la modellazione diretta dell'aspetto visuale delle applicazioni Web. Per dimostrare l'applicabilità pratica di quanto proposto, è stato realizzato un prototipo che introduce il nuovo processo all'interno di WebRatio, uno strumento di sviluppo Model Driven che è stato utilizzato come caso di studio. Le soluzioni proposte sono state così direttamente implementate all'interno di un prodotto vero e proprio. Questo tipo di approccio ha permesso di verificare la validità pratica delle considerazioni teoriche: il nuovo processo ha effettivamente permesso la creazione di un'applicazione Web seguendo il paradigma Model Driven durante tutte le sue fasi. In questo modo, le criticità incontrate sono state eliminate e i vantaggi della metodologia MDD sono risultati immediatamente evidenti.

5.1 Considerazioni finali

La realizzazione di un prototipo funzionante a supporto delle tesi proposte rappresenta la dimostrazione pratica di come l'attuale processo di sviluppo di un'applicazione Web secondo un approccio Model Driven può essere effettivamente migliorato. L'evidente spaccatura esistente tra il processo di creazione del back-end di un'applicazione ed il relativo front-end può essere risolta con una semplice considerazione: il modello stesso può agire da collante tra i due approcci e portare alla creazione di un processo omogeneo in cui la filosofia Model Driven guida tutte le fasi dello sviluppo.

Partendo da una metodologia di programmazione tradizionale, il cambiamento di approccio nella fase di generazione del modello di presentazione è stato guidato dall'integrazione di un editor WYSIWYG, appositamente modificato per integrare anche in questa parte del flusso di lavoro i vantaggi offerti dal paradigma Model Driven.

Lo sviluppatore, in questo modo, si trova ad affrontare un workflow omogeneo, basato principalmente sulla progettazione di software ad un livello di astrazione superiore rispetto ai metodi tradizionali. L'onere della gestione dei dettagli implementativi è lasciato agli strumenti di sviluppo che, da semplici editor di testo si sono trasformati in veri e propri generatori di codice. Tutte le fasi ripetitive, ad alta probabilità di errore, macchinose e soprattutto time-consuming sono affidate a strumenti automatici, in grado di gestirle con una maggiore velocità. Il valore aggiunto di uno sviluppatore, quindi, non è più rappresentato dalla

sua capacità di risolvere i problemi implementativi; la sue esperienza va totalmente a supporto delle fasi di progettazione, in cui ci si può concentrare sul dominio del problema.

5.2 Prospettive future

La versione attuale del prototipo è indirizzata a fornire le funzionalità di un editor WYSIWYG Model Driven per un numero limitato di Unit di WebRatio. Naturalmente, sono previsti ulteriori sviluppi, che prima di tutto si occupino di estendere il numero di componenti supportati. Un ulteriore passo verso la completa integrazione degli strumenti messi a disposizione, riguarderà la completa integrazione dello strumento all'interno dell'ambiente di sviluppo di WebRatio.

Attualmente, infatti, le interfacce delle applicazioni generate tramite WebRatio possono essere costruite per mezzo di un particolare pannello, denominato “Grid”. Tale pannello mette a disposizione un'area all'interno dell'ambiente di sviluppo in cui viene visualizzata, in forma tabellare, la struttura della pagina attualmente selezionata all'interno del modello di ipertesto di WebML. Per mezzo di un elenco, è poi possibile trascinare all'interno della griglia tutti gli elementi di cui sono costituite le Unit, in modo da posizionare ogni componente in una zona specifica della pagina stessa. Il rendimento visuale di ogni singolo componente dipende solamente dal Layout Template associato a quest'ultimo e, attualmente, non si ha alcun riscontro visuale finché l'applicazione non viene visualizzata all'interno di un browser.

Il pannello "Grid" fornisce soltanto un'idea della struttura generale della pagina stessa, senza fornire informazioni sulla resa finale dell'applicazione. Per migliorare le funzionalità di questo pannello, è possibile trasformarlo in una vera e propria anteprima che renda l'idea di

come l'intera pagina sarà visualizzata all'interno di un browser. Tale risultato può essere ottenuto combinando il rendimento visuale di tutte le singole Unit e i singoli componenti contenuti all'interno della pagina, opportunamente disposti a seconda della struttura selezionata. Per realizzare effettivamente una simile funzionalità, è possibile sfruttare l'anteprima generata dall'editor WYSIWYG, in modo che questa risulti il più aderente possibile al risultato atteso. Per ovvie ragioni, non si è voluto stravolgere la funzione principale del pannello, lasciando il compito della modifica del rendimento grafico di ogni singolo componente alla relativa finestra di editing visuale.

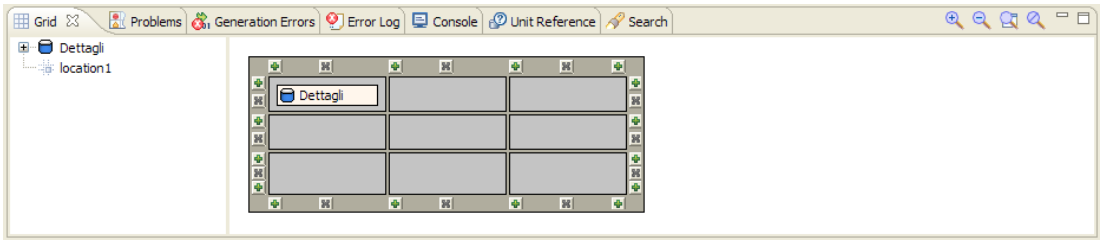

Figura 5.29: Dettaglio del pannello Grid di WebRatio

Inoltre, per migliorare e semplificare l'interazione dell'utente, è possibile permettere l'apertura dell'editor WYSIWYG direttamente dal pannello "Grid". Attualmente, infatti, la finestra per la creazione di un Layout Template personalizzato viene visualizzata solamente al momento della creazione del template stesso. Per ogni ulteriore modifica, è necessario ricercare il file del Layout Template all'interno del progetto in cui è stato inserito; come si può notare, tale operazione, anche se non particolarmente gravosa, richiede diversi passaggi e può risultare relativamente complicata nel caso di applicazioni molto complesse,

composte da molteplici progetti. Limitando il numero di passi necessari per l'apertura dell'editor WYSIWYG, si ottiene una modalità più semplice e immediata: l'invocazione di quest'ultimo viene effettuata direttamente dall'editor del modello di ipertesto di WebRatio. Dato che la lista dei componenti e la loro disposizione all'interno della pagina sono già disponibili all'interno del pannello “Grid”, è possibile sfruttare tale strumento per i nostri scopi. Un semplice doppio click sul componente su cui si vuole intervenire scatena l'apertura dell'editor WISIWYG, già impostato per intervenire sul Layout Template associato al componente stesso.

Si noti, infine, come l'intervento a livello del pannello “Grid” ha permesso di migliorare sensibilmente l'esperienza utente, che, in questo modo, con un numero minore di passi, si trova a disposizione tutti gli strumenti necessari per la costruzione di interfacce più ricche in un tempo inferiore.

Bibliografia

- Ceri, Fraternali, Bongio, Brambilla, Comai, Matera – “Designing Data-Intensive Web Applications”
- Moreno, Fraternali, Vallecillo - “WebML modelling in UML”
- Comai, Fraternali - “A semantic model for specifying data-intensive Web applications using WebML”
- Lu, Wan - “Model Driven Development of Complex User Interface”
- Jespersen, Linvald - “Investigating User Interface Engineering in the Model Driven Architecture”
- Wu, Shin, Chien, Chao, Hsieh - “An extended MDA method for User Interface Modeling and Transformation”
- Sun, Gray, Langer - “A WYSIWYG Approach for Configuring Model Layout using Model Transformations”
- Sousa, Mondonça, Vanderdonckt - “Towards Method Engineering of Model-Driven User Interface Development”
- Coutaz - “User Interface Plasticity: Model Driven Engineering to the Limit!”
- Rech, Bunse - “Model-Driven software development – Integrating

Quality Assurance”

- http://www.webml.org/
- http://www.webratio.com/