Università degli Studi dell'Insubria

FACOLTA' DI SCIENZE MATEMATICHE

FISICHE E NATURALI

Corso di Laurea in Scienze e Tecnologie dell'informazione

SVILUPPO DI FUNZIONALITA' AVANZATE DI INVIO E-MAIL IN UNO STRUMENTO CASE PER IL PROGETTO DI SITI WEB (WebRatio)

Tesi di laurea di:

DAVIDE TAIBI

Relatore:

Prof. SANDRO MORASCA

Correlatore:

Ing. ROBERTO ACERBIS

Anno Accademico 2003-2004

INDICE

Ringraziamenti

Un cordiale ringraziamento va al prof. Sandro Morasca, all'ing. Roberto Acerbis e tutti i componenti del team WebRatio per la professionalità, disponibilità e attenzione con cui hanno seguito questo lavoro di tesi.
Desidero dedicare un caloroso abbraccio ai compagni di corso e agli amici con i quali ho condiviso gli anni di studio all'Università.
Infine ringrazio i miei genitori, mio fratello e Valentina per essermi stati vicini e avermi aiutato ad affrontare tutte le difficoltà incontrate.

Capitolo 1

INTRODUZIONE

La progettazione e lo sviluppo di soluzioni per il web sono processi strettamente dipendenti dal singolo contesto applicativo. Navigando nella rete ci si rende subito conto di come ciascun sito sia stato progettato ad hoc, senza partire da nessuna soluzione preconfezionata. Ciò ha indotto, nel passato, a trascurare uno dei principi cardine dell'Ingegneria del Software: il riutilizzo di moduli precedentemente implementati per altre soluzioni (software reuse). E'opportuno ricordare che, qualora si sia in grado di riutilizzare del codice, questo si possa tradurre in un enorme risparmio dei tempi di realizzazione, nonché dei costi associati ad un progetto informatico. Le attività di adattamento dell'applicazione alle specifiche esigenze sono generalmente supportate da strumenti CASE, che mascherano il codice sottostante consentendo una visione d'alto livello. Nel caso in cui ciò non sia sufficiente, è possibile mettere mano direttamente al codice con un inevitabile innalzamento dei costi. Pertanto, maggiori sono il numero e le potenzialità di personalizzazione a disposizione, minore sarà lo sforzo di realizzazione in termine di tempi e di costi.

WebRatio è uno strumento CASE che supporta la progettazione di applicazioni Web basate sui dati e la generazione automatica di codice a partire dalle specifiche del modello Entità-Relazione e WebML (www.WebML.org), coprendo il ciclo di sviluppo dalla progettazione dei dati e dell'ipertesto fino alla loro implementazione in JSP o ASP.NET.

WebRatio si basa su una complessa architettura software che si avvale delle più recenti tecnologie nell'ambito dello sviluppo di applicazioni per il web quali XML, XSL, Java.

WebRatio SDS può essere utilizzato non solo per la progettazione di applicazioni ex novo (eventualmente basate su dati memorizzati in strutture già

esistenti, ovvero legacy), bensì nuove soluzioni possono efficacemente sfruttare le applicazioni realizzate in precedenza. Questo è vero sia per applicazioni progettate e realizzate per poi essere riutilizzate, i cui requisiti sono stati raccolti facendo riferimento alle best practice, sia con qualche accorgimento in più, per soluzioni generiche: ogni caso applicativo può diventare una base per uno scheletro verticale, una volta che viene privato dei dettagli più fini e specifici della singola applicazione.

1. 2 Obiettivi della tesi

Scopo di questa tesi è l'ampliamento delle funzionalità di WebRatio SDS con l'aggiunta di un nuovo componente per invio di e-mail multiple personalizzate, in formato html/text, prelevando le informazioni di formattazione necessarie da un database relazionale.

L'idea è quella analizzare il problema dal punto di vista del progettista di un sito web, che necessita di inviare un messaggio e-mail a tutti gli utenti registrati su un sito.

Si intende creare un plug-in per WebRatio, in grado di inviare un unico messaggio di posta in formato testo o html a molti utenti, oppure un messaggio diverso per ogni destinatario, con la possibilità di avere allegati diversi: ogni utente deve quindi ricevere un messaggio personalizzato secondo le sue esigenze.

Il lavoro di tesi sarà quindi volto ad analizzare, sviluppare, implementare e verificare le tecnologie software necessarie per gestire questa tematica.

Verranno analizzate diverse soluzioni per la gestione dei messaggi in vari formati.

I requisiti fondamentali che si cercherà di ottenere dal nostro prodotto sono:

- Funzionalità :
    - invio di una e-mail uguale per ogni destinatario, ricevuti gli indirizzi da un database.
    - invio di una e-mail diversa per ogni destinatario.

    - invio di una e-mail con campi personalizzati per ogni destinatario.
- Integrazione: si deve realizzare un plug-in nel linguaggio di programmazione Java, correttamente inserito nel software WebRatio

1. 3 Scenario di riferimento

Il tirocinio svolto si colloca nell'ambito del progetto WebRatio (www.webratio.com) nato da anni di ricerca presso il Dipartimento di Elettronica e Informazione del Politecnico di Milano e confluito in uno spin-off del Politecnico stesso nell'anno 2001.

Con questo lavoro si intende progettare e realizzare un plug-in per la creazione e l'invio di messaggi e-mail personalizzati.

Prima dello sviluppo di tale componente, per la creazione e l'invio di una mail a tutti gli utenti iscritti ad un sito creato con WebRatio SDS, l'amministratore del sito doveva reperire i dati necessari dal database, creare tutti i messaggi e, infine, inviare manualmente tutte le mail con conseguente dispendio di tempo e fatica. Inoltre l'invio di mail personalizzate era molto gravoso da parte dell'operatore che, in tal caso, doveva scorrere tutti i mittenti e, di conseguenza, modificare il testo della mail.

Considerando un applicazione Web di piccole dimensioni e con pochi iscritti, la questione può sembrare relativamente semplice. Ampliando il contesto e considerando un insieme di utenti considerevole, la componente "tempo" ha un peso determinante nella gestione del sito, rendendo lunga e difficoltosa la procedura di invio delle e-mail.

1.3 Approccio al problema

La natura stessa del progetto comporta la necessità di interagire contemporaneamente con diverse attività più o meno complesse, che possono richiedere un accesso attraverso percorsi del tutto differenti.

Principio base di questo studio, è quello di realizzare un prodotto corretto, affidabile e ben integrato nel supersistema.

Visto il tempo a disposizione e, la necessità di consegnare il prodotto nei termini previsti, prima di iniziare il lavoro è stato fatto un attento studio delle attività e delle sottoattività necessarie, in modo da coordinare tutte le fasi di progetto e di verificare in modo continuo lo stato di avanzamento.

ID	Attività:	Durata	Inizio
1	**Obiettivi globali**		
	Definizione degli obiettivi globali del progetto	1,g	21/7/03
	Definizione delle risorse	1,g	22/7/03
2	**Formazione**		
	Definizione metodologie di formazione	2,g	22/7/03
	Studio ed analisi dell'ambiente WebRatio	21,g	24/7/03
	Studio del linguaggio WebML	5,g	24/7/03
	Studio delle funzionalità di WebRatio	16,g	31/7/03
3	**Studio dei linguaggi XML ed XSLT**		
	XML	5,g	24/7/03
	XSLT	15,g	31/7/03
4	**Analisi Requisiti software**		
	Analisi delle esigenze	3,g	22/8/03
	Bozza preliminare delle specifiche software	1,g	27/8/03
	Verifica delle specifiche software	,5g	28/8/03
	Definizione scadenze di consegna	,5g	28/8/03
5	**Progettazione**		
	Verifica delle specifiche software preliminari	,5g	29/8/03
	Sviluppo specifiche funzionali	,5g	29/8/03
	Sviluppo del prototipo in base alle specifiche preliminari	1,g	1/9/03
	Verifica delle specifiche funzionali	1,g	2/9/03
	Progettazione del descrittore	3,g	3/9/03
	Progettazione dei parametri di Input	1,g	7/9/03
	Progettazione dei parametri di Output	0,5g	8/9/03
	Progettazione dell'Interfaccia	2,g	9/9/03

6	**Sviluppo del codice**		
	descriptor.xml	10,g	10/9/03
	input-parameter.xsl	10,g	10/9/03
	output-parameter.xsl	10,g	10/9/03
	operation.xsl	10,g	10/9/03
	type.xsl	10,g	10/9/03
	MultiMailUnit.java	15,g	24/9/03
7	**Testing**		
	Testing dei moduli	13,g	15/10/03
	Testing dei moduli componenti in base alle specifiche del prodotto	2,g	15/10/03
	Identificazione di anomalie in base alle specifiche del prodotto	3,g	17/10/03
	Modifica del codice	3,g	22/10/03
	Testing del codice modificato	5,g	27/10/03
	Testing dei moduli completato		31/10/03
	Test di integrazione	8,g	3/11/03
	Test dell'integrazione dei moduli	1,g	3/11/03
	Modifica del codice	5,g	5/11/03
	Testing del codice modificato	1,g	12/11/03
	Test di integrazione completato		12/11/03
8	**Documentazione**		
	Definizione specifiche manuali per l'utente	2,g	13/11/03
	Stesura manuali per l'utente	3,g	17/11/03
9	**Consegna**		**19/11/03**

I tempi necessari sono stati attentamente stimati creando un diagramma raffigurante tutte le attività di progetto.

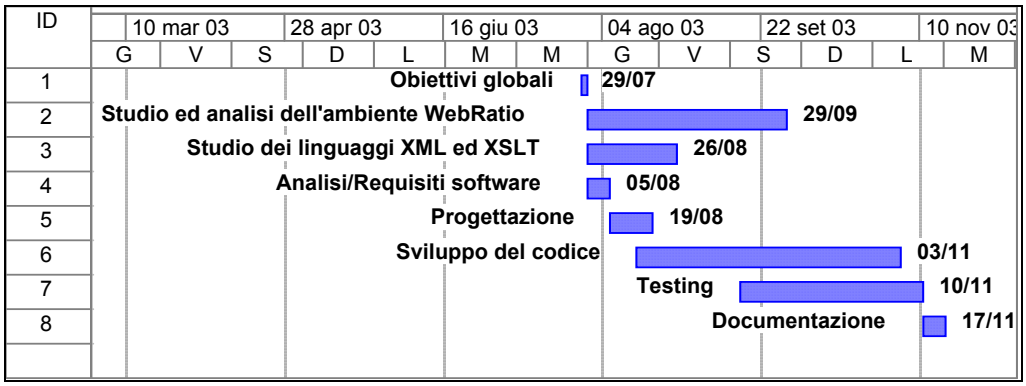


Figura 1.1 Gantt

Le maggiori difficoltà incontrabili sono dovute al tempo di apprendimento delle funzionalità di WebRatio, del linguaggio xml ed xsl. Tali attività vengono stimate per eccesso sovrapponendole alle altre già dall'inizio del progetto, ritenendo molto difficile l'apprendimento di tali linguaggi, fino ad oggi non contemplati nel Corso di Laurea.

1.4 Struttura della tesi

In questo primo capitolo è stata chiarita la teoria sulla quale si basa tutto il lavoro di tesi, indicato gli obiettivi principali da raggiungere, le possibili difficoltà incontrabili, nonchè lo scenario e le tecnologie di riferimento per lo sviluppo del lavoro.

Nel secondo capitolo viene analizzata l'architettura del supersistema di riferimento (WebRatio SDS).

Nel terzo capitolo viene definita l'architettura del plug-in, l'analisi e la specifica dei requisiti, la progettazione di tutti i moduli realizzati.

Nel quarto capitolo viene illustrata dettagliatamente l'implementazione della classe di servizio.

Il quinto capitolo contiene le conclusioni del lavoro svolto ed i possibili sviluppi futuri.

Infine, in appendice A, viene allegato tutto il codice di progetto, mentre in appendice B, il manuale d'uso del prodotto finito.

Capitolo

2

WebRatio

2.1 Introduzione

La progettazione di applicazioni Web data intensive è un compito cruciale; la specifica, il design e l'implementazione sono diventate fasi molto importanti ed allo stesso tempo difficili da realizzare.

WebRatio è un software Case che utilizza un particolare approccio di sviluppo basato su un classico design pattern proposto dagli ingegneri del software, chiamato Modello-Vista-Controllore (MVC).

L'architettura MVC è concepita per meglio separare e isolare le tre funzioni essenziali di un applicazione interattiva:

- La logica di business dell'applicaizone (il Modello)
- L'interfaccia presentata dall'utente (la Vista)
- Il controllo dell'interazione che si determina a seguito delle azioni dell'utente (il Controllore).

La principale caratteristica che distingue WebRatio da tutti i suoi concorrenti, è l'adozione di un linguaggio di specifica formale e la generazione semi automatica del codice. Le specifiche delle applicazioni Web, vengono definite su due livelli: un primo livello (il database) tramite un modello ER (Entità RelationShip), il secondo livello (presentazione) mediante il linguaggio WebML. WebRatio include un interfaccia grafica per la modellazone di schemi ER e WebML, oltre ad un potente e flessibile generatore di codice per trasformare le specifiche ER in un database relazionale tramite una connessione ODBC o JDBC, e le specifiche WebML in templates di pagina JSP o Asp.NET.

Grazie a questo software i principi di WebML possono venire applicati alla pratica in maniera rapida e intuitiva, mediante l'approccio grafico del programma.

Al momento in cui si scrive si è giunti alla versione 3.3 del progetto e attualmente un gruppo di circa 15 sviluppatori sta preparando la nuova versione.

2. 2 Il Liguaggio Web Ml

Le applicazioni Web data-intensive mirano specificamente al trattamento di grandi quantità di informazioni, che sono rappresentate principalmente da dati strutturati, ma anche da dati multimediali come immagini, audio e video. Queste informazioni sono memorizzate e gestite tramite l'utilizzo di database. Per questa categoria di applicazioni, quindi, la possibilità di progettare congiuntamente il sito Web e la base di dati ad esso associata rappresenta un fattore fondamentale per migliorare sviluppo, manutenzione ed evoluzione dell'applicazione nel suo insieme.

Per rendere possibile l'integrazione degli aspetti relativi al Web e di quelli relativi al database, è necessario fornire una descrizione semantica delle informazioni che vengono trattate; è cioè indispensabile riuscire a definire un modello dell'applicazione che coinvolga tutte le dimensioni di progettazione, fornendo una descrizione ad alto livello della struttura dell'applicazione. Questo modello rappresenta il primo passo, a partire dal quale, coerentemente con l'approccio tipico dell'ingegneria del software, deve iniziare il processo di sviluppo che porterà poi a definire l'applicazione completa.

L'utilizzo di notazioni semiformali di specifica e progettazione costituisce un fattore chiave per aumentare la qualità del processo di sviluppo software. Il successo di queste tecniche è legato anche alla presenza di una notazione grafica, più intuitiva e diffusa di quella linguistica.

WebML definisce una notazione per la specifica e la progettazione di siti Web data-intensive e per la pubblicazione dei siti stessi, a partire da specifiche di alto livello. Il linguaggio è formalizzato mediante la definizione dei suoi costrutti tramite il metalinguaggio XML (eXtensible Markup

Language) e permette di modellizzare la realtà di interesse ad un alto livello di astrazione, prescindendo dai dettagli dell'architettura.
Il processo di definizione del modello di un'applicazione tramite WebML è supportato da un insieme di strumenti CASE (Computer Aided Software Engeneering), che utilizzano una notazione grafica intuitiva per rappresentare i costrutti del linguaggio.

2.2.1 Visione d'insieme di WebML

La metodologia WebML individua cinque parti in cui suddividere la progettazione:

- *Struttura:* descrizione dell'organizzazione dei dati contenuti nel sito.
- *Derivazione:* estensione della struttura con informazioni ridondanti, utili a rendere più chiara la progettazione e più efficiente l'implementazione.
- *Composizione:* allocazione dei contenuti all'interno delle pagine del sito.
- *Navigazione:* collegamento delle pagine in un ipertesto e definizione del comportamento degli elementi interni ad una pagina complessa.
- *Gestione dei dati:* definizione della parte operativa dell'applicazione, indicando le modalità di gestione e modifica dei dati a Run-Time

In aggiunta a queste cinque parti, vi è la progettazione della presentazione, cioè del modo in cui appaiono le pagine del sito usando HTML, WML, XSL o qualsiasi altro linguaggio di rendering. WebML lascia ai tools grafici appositamente studiati il compito di supportare quest'ultima parte della progettazione.
L'idea fondamentale da cui si parte è la separazione tra i dati che un sito Web deve visualizzare, l'ipertesto che definisce come vengono esplorati i dati attraverso la navigazione dell'utente e la presentazione grafica e multimediale dei dati. Questa separazione viene poi affinata, nella fase di progettazione, nel seguente modo:

- Dati: Struttura + Derivazione

- Ipertesto: Composizione + Navigazione + Gestione dei dati
- Presentazione: Presentazione

2.2.2 Struttura dei dati e generalizzazione

La struttura dei dati su cui si basa il sito Web viene definita tramite un modello Entità-Relazione (ER) semplificato. Questo modello consente di definire con un alto livello di astrazione le entità che dovranno essere rappresentate nell'applicativo Web, e le relazioni esistenti tra le entità stesse. L'alto livello di astrazione implica una maggiore facilità di passaggio dalle specifiche allo schema risolutivo.

Un notevole vantaggio che questo modello strutturale permette di raggiungere, è quello di avere strumenti di generazione automatica dell'implementazione. Dallo schema è possibile, ad esempio, generare una base di dati relazionale che conterrà tutte le informazioni del sito; infine dopo la successiva fase di definizione dell'ipertesto, è possibile ottenere automaticamente il codice di navigazione dei dati in un linguaggio di scripting come JSP o ASP .NET.

L'astrazione introdotta utilizzando il modello ER permette di ottenere un ulteriore vantaggio: l'indipendenza dell'applicazione anche dalla struttura fisica con cui sono memorizzati i dati. Questi ultimi possono infatti essere contenuti in un database relazionale, in una serie di documenti XML o in una base di dati legacy preesistente. Per definire il legame tra lo schema astratto e i dati, WebML prevede la definizione di semplici corrispondenze di mapping.

I concetti principali con cui si definisce la struttura sono le entità e le relazioni. Ogni entità rappresenta una classe di oggetti, reali o meno, che vengono rappresentati nell'applicazione. Ogni oggetto richiede la memorizzazione di informazioni suddivise in più attributi. L'entità è quindi caratterizzata da un nome e dai suoi attributi. La relazione è una connessione semantica tra entità. Essa definisce un legame tra le istanze dell'entità e può porre alcuni vincoli di cardinalità del legame. Le entità possono essere organizzate in gerarchie di generalizzazione.

Questi concetti dovrebbero essere usati per definire una struttura non ridondante. Ciò permette di evitare più facilmente inconsistenze tra i dati e sprechi di spazio fisico di memorizzazione. Tuttavia è spesso utile avere informazioni ridondanti per realizzare in modo più chiaro ed efficiente pagine contenenti dati provenienti da diverse entità. Per questo WebML definisce un modello di derivazione che permette di specificare:

- Attributi importati in una entità da un'altra entità.
- Entità derivate da altre entità sotto specifiche condizioni e con possibili attributi aggiuntivi.
- Relazioni derivate da una catena di due o più relazioni esistenti.

La derivazione è definita tramite il linguaggio WebML-OQL, che non interessa approfondire in questo elaborato.

2.2.3 L'ipertesto di navigazione dei dati

WebML permette la definizione di più modalità di navigazione dei dati per una stessa applicazione tramite il concetto di SiteView. Una SiteView è un modulo che definisce una modalità di accesso ai dati. Tramite più SiteView è possibile distinguere l'utente non registrato dall'amministratore del sito o da altri tipi di utente, fornendo una visione del sito completamente diversa, anche se basata sugli stessi dati. Le SiteView possono inoltre venire utilizzate per creare modalità di navigazione specificatamente pensate per un client di navigazione, ad esempio per un normale browser su PC, per un cellulare Wap o per un browser a sintesi vocale.

Una SiteView è composta da pagine, cui corrispondono elementi complessi di visualizzazione del sito: frame HTML, deck WML, ecc. Vi sono poi altre componenti, chiamate Unit, che possono stare all'interno delle pagine o all'esterno delle pagine. Le Unit interne aggiungono alla pagina dei contenuti visibili, tipicamente prelevati dalla struttura dati(content unit). Le Unit fuori pagina, come ad esempio la MultiMailUnit sviluppata, eseguono operazioni di modifica delle informazioni o altre operazioni legate alla navigazione del sito.

I collegamenti tra le pagine e tra le Unit sono realizzati tramite i Link.

Un link è la generalizzazione del concetto di anchor e rappresenta lo strumento con il quale l'utente può navigare tra le pagine del sito. Un link può anche trasportare delle informazioni (per esempio dei parametri a una Unit): in questo caso il link viene denominato contestuale.
In generale però non tutti i link sono riconducibili a un anchor: esistono infatti i link di trasporto che sono dedicati esclusivamente al trasporto di parametri e non rappresentano un collegamento fisico navigabile dall'utente

2.3 L'architettura MVC

Le applicazioni prodotte con WebRatio adottano un organizzazione basata sull'architettura MVC (Model View Control) in grado di separare gli aspetti di controllo, stato e interfaccia, atte a favorire riusabilità e manutenibilità, pertanto il codice generato soddisfa i requisiti più stringenti delle applicazioni Web basate sui dati di livello industriale.
L'aspetto chiave dell'architettura di WebRatio è il mapping delle primitive WebML (pagine, content units ed operation units) nell'architettura MVC.
Ogni pagina WebML viene tradotta in cinque elementi:

- Un "page action" nel modello
- Un servizio di pagina
- Un template JSP nella vista
- Un "page action" nel file di configurazione del controller

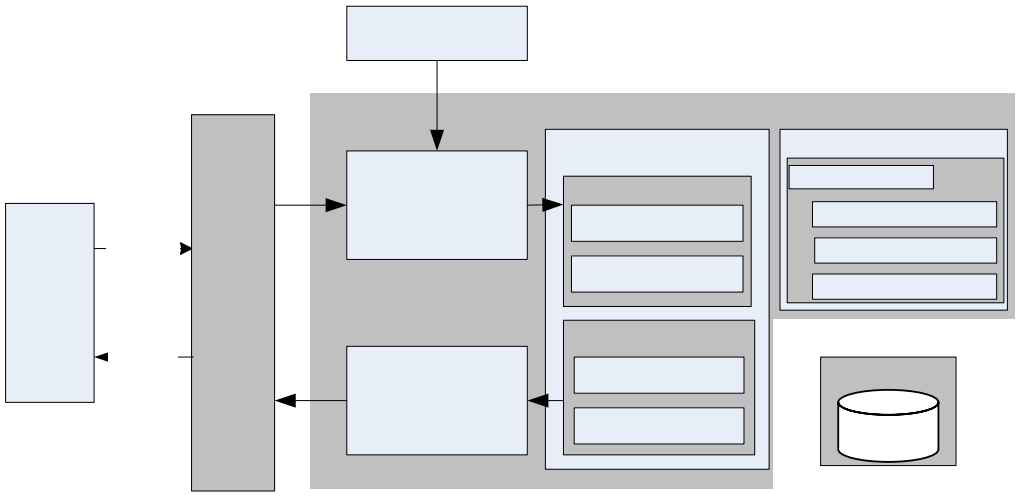


Figura 2.3.1: L'architettura MVC collocata nell'architettura con application server

Quando un applicazione ha dimensioni veramente rilevanti, la soluzione MVC non allevia il problema associato alla dimensione dell'applicazione:

- Ogni unit e operazione richiede un servizio dedicato nello strato di business. Se vi sono molte unit, occorre sviluppare e mantenere un elevato numero di servizi.
- Ogni pagina richiede un servizio di pagina distinto. Tali servizi, sono numerosi e simili, in quanto differiscono solamente per i parametri estratti dalla richiesta HTTP e per la sequenza con cui sono invocati
- I servizi di business sno implementati come programmi eseguiti all'interno del servlet container.
- Il look-and-feel dell'applicazione è cablato nei template JSP. Cambiare lo stile di presentazione richiede un intervento manuale su un numeroso umero di files.

Per evitare la proliferazione di servizi di pagina e di unit, è possibile sfruttare la genericità, un principio classico di progettazione del software. I servizi associati alle unit, si possono riorganizzare in accordo al pattern mostrato in figura 2.3.2.

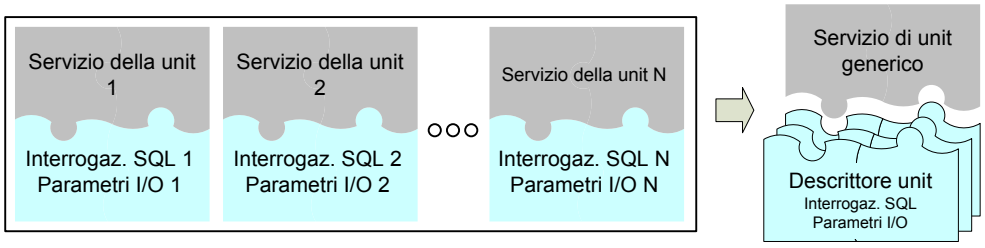


figura 2.3.2.: Comparazione tra servizi associati a istanza di unit e un servizio di unit generico affiancato da un descrittore

2.4 Il processo di sviluppo di WebRatio

Come già accennato precedentemente, WebRatio, copre le fasi di progettazione dei dati dalla e dell'ipertesto, supporta la fase di implementazione attraverso la generazione automatica della base di dati rezionale e dei template di pagina dell'applicazione. Più precisamente, WebRatio si focalizza su cinque aspetti fondamentali:

- *Progettazione dei dati:* supporta la progettazione di schemi di dati ER, con un interfacci agrafica che consente di disegnare e specificare le proprietà di entità, relazioni, attributi e gerarchie di generalizzazione.
- *Progettazione dell'Ipertesto:* assiste la progettazione delle *site view*, consentendo di disegnare e specificare le proprietà di aree, pagine, unit e link.
- *Mapping dei dati:* permette di dichiarare le sorgenti dati a cui lo schema concettuale dei dati deve essere associato, e traduce automaticamente i diagrammi Entità-Relazione e le espressioni OQL in basi di dati e viste.
- *Progettazione della presentazione:* offre funzionalità per definire lo sile di presentazione dell'applicaizione, consentendo allo sviluppatore di creare fogli di stile XSL e associarli alle pagine; permette inoltre di distribuire i contenuti nella pagina, specificando le posizioni relative delle unit di contenuto della pagina stessa.
- *Generazione del codice:* traduce automaticamente le site view in applicazioni Web complete, basate sulle piattaforme J2EE, Struts e .Net

Il diagramma seguente, descrive il processo di sviluppo di WebRatio, evidenziando tutte le fasi di progettazione con i rispettivi ingressi ed uscite.

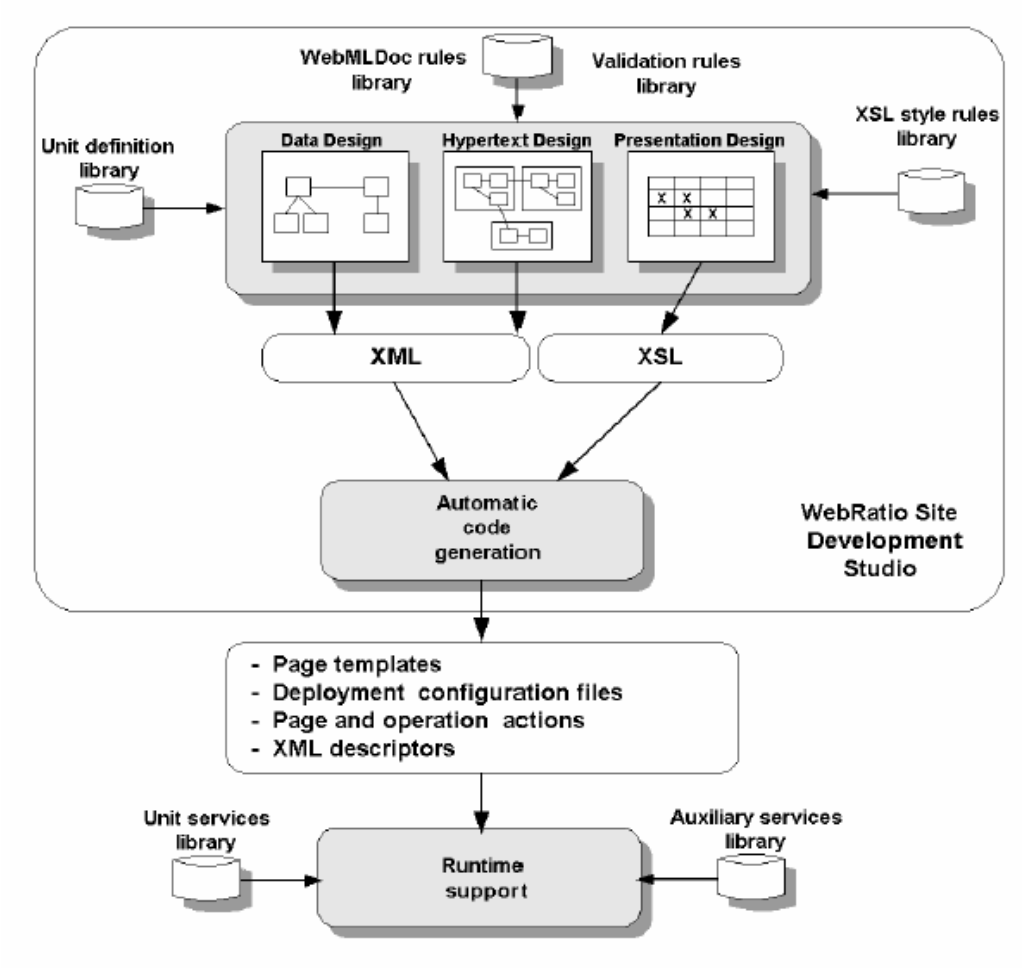


Figura 2.4.1: Diagramma del processo di sviluppo di WebRatio

2.5 Site Development Studio

Il programma presenta un'interfaccia grafica nella quale si possono individuare facilmente le varie funzionalità offerte:

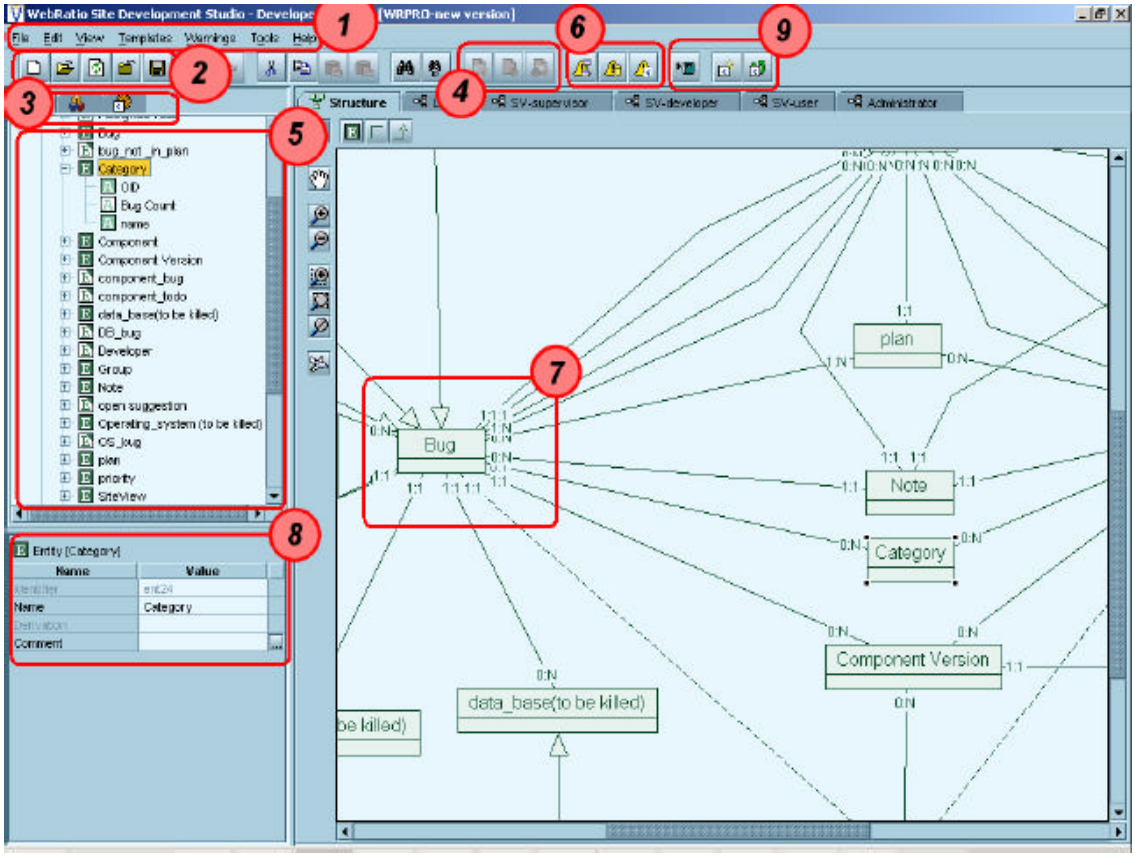


Figura 2.5.1: l'interfaccia di WebRatio SDS

1. I classici menu a tendina di ogni programma visuale da cui si accede a tutte le funzionalità della suite.
2. Icone rappresentanti i comandi per la gestione dei progetti (apertura, salvataggio…).
3. Comandi per cambiare ambiente.
4. Comandi par la generazione automatica delle pagine Web e dei descrittori XML necessari al modulo di run-time per operare.
5. Albero che rappresenta in maniera grafica il progetto WebML.
6. Strumenti di controllo errori e debug.
7. Un esempio di entità.
8. Rappresentazione grafica delle proprietà di ogni singolo oggetto.
9. Strumenti dedicati a Easy Styler, strumento per costruire l'aspetto grafico delle pagine Web.

I passi per procedere alla costruzione di un applicativo Web sono molto semplici e immediati:

1. Apertura di un nuovo progetto.
2. Inserimento della struttura dati sottostante l'applicativo e successiva mappatura su una base dati esistente. Da sottolineare il fatto che WebRatio è scritto in Java, quindi occorre inserire nel programma le librerie ponte verso altre piattaforme diverse da JDBC.
3. Inserimento della struttura delle pagine dell'applicativo e delle Unit in esse contenute, con il settaggio opportuno delle varie opzioni. In questo momento si costruiscono i pattern di navigazione necessari a svolgere i requisiti funzionali del sito Web.
4. Creazione dei link di navigazione. In questa fase l'attenzione viene posta soprattutto sui parametri trasportati dai link, che devono correttamente combaciare con i parametri di ingresso delle Unit di destinazione del link.
5. Associazione di ogni Unit con il rispettivo modulo (core) che si occupa della sua rappresentazione grafica e posizionamento della Unit nella pagina di riferimento. In questa fase viene deciso l'aspetto che l'applicativo avrà una volta generato.
6. Generazione delle pagine, operazione svolta completamente in automatico dal modulo run-time della suite.

Si ricorda che WebRatio viene distribuito con un servlet container freeware per gestire la visualizzazione delle pagine .jsp generate e l'utilizzo dei JavaBean. L'applicativo in questione è Tomcat (www.apache.org).

L'applicativo dispone di molti moduli per ottimizzare alcune operazioni. Fra questi vale la pena di ricordare il Wizard per le query SQL, intuitivo e di facile uso, e Easy Styler, un vero e proprio programma a se stante con il compito di creare gli stili grafici per le pagine del sito Web.

Un progetto WebRatio è composto da un singolo diagramma Entità-Relazione e da un insieme di site view (viste). A ciascun progetto viene automaticamente aggiunto un modello dei dati predefinito, costituito dalle entità Utente e Gruppo, necessario per la creazione multiutente personalizzate.

Capitolo

3

PROGETTAZIONE DELLA UNIT

3. 1 Introduzione

L'architettura del plug-in consiste in un'applicazione scritta in Java, un descrittore xml e tre moduli xsl per inserire l'applicazione creata nell'ambiente WebRatio. La definizione dell'architettura viene svolta seguendo i criteri dell'ingegneria del software, cioè sviluppando l'analisi, la definizione e la specifica dei requisiti, e sviluppando il disegno software dei moduli principali. L'architettura ma soprattutto l'implementazione poggia su alcuni elementi utilizzati:

- Il tool XML SPY, per facilitare l'implementazione del descrittore e dei moduli XML;
- Il tool JBuilder 9, per l'implementazione e la compilazione delle classi Java;
- La libreria javax.mail, sviluppata da SUN, per la manipolazione di messaggi in formato mime;

3. 2 Il processo di design

Il progetto della unit, viene organizzato nelle fasi illustrate in figura

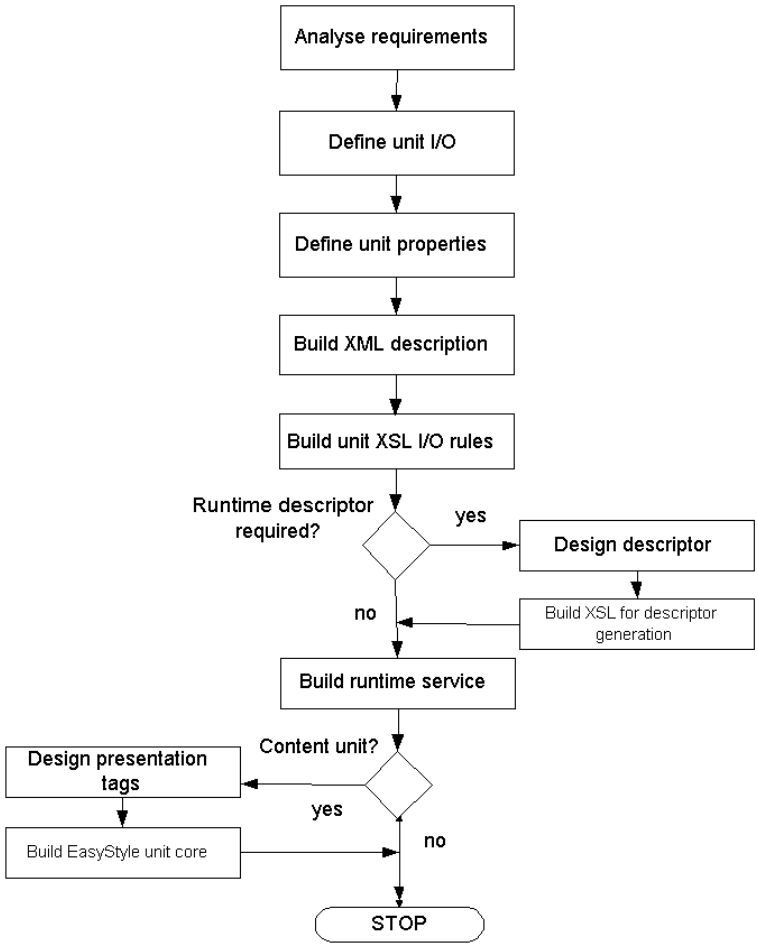


Figura 3.1 Il processo di progettazione

3.2.1 Visione d'insieme dell'albero delle directory

La creazione di una "Custom Unit" in WebRatio SDS, consiste nella creazione di alcuni pezzi di codice e specifiche, quindi copiare tutto il codice in una specifica directory.

Tutto il codice che costituirà il plug-in, andrà organizzato in una struttura di directory predefinita, quindi copiato nella cartella di installazione di WebRatioSDS (SDS/Conf/-Units Directory).

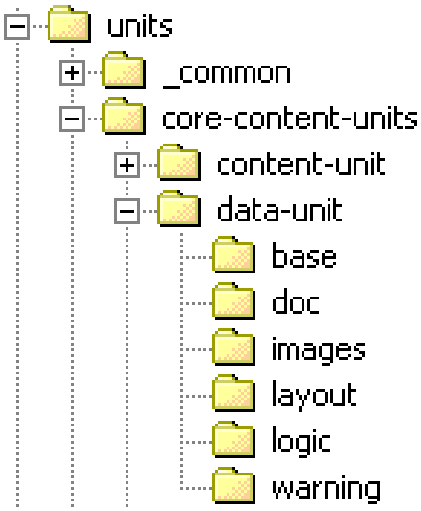


Figura 3.2 Struttura delle cartelle delle units

Ogni unit, contiene tre sottocartelle, con i file associati ai differenti aspetti del plug-in.

- Base: Contiene un file XML (descriptor.xml), usato da Webratio SDS per istanziare il servizio implementato.
- Images: Contiene le immagini usate per rappresentare la unit nell'interfaccia grafica di WebRatio
- Logic: Contiene tre files: due di questi (input-parameters.xsl e output-parameters.xsl) per specificare i parametri di input e output, un ultimo file (operation.xsl) che produce il descrittore XML di run-time per il servizio istanziato nella unit.

3.3 Documento dei requisiti: Definizione dei requisiti

Vogliamo creare una nuova unit per inviare delle e-mail

I messaggi dovranno essere inviati utilizzando un server SMTP standard, specificato nelle proprietà.

Il plug-in deve accettare in ingresso i seguenti parametri:

- L'indirizzo e-mail del mittente (tale indirizzo deve essere specificabile sia in modo statico che dinamico)
- Un set di indirizzi e-mail di destinazione
- Un set di indirizzi e-mail in Copia Carbone (CC)
- Un set di indirizzi e-mail (BCC)
- Una Stringa indicante l'oggetto del messaggio (deve essere specificabile in modo dinamico o statico)
- Un file xhtml contenente il corpo del messaggio o, in caso non sia specificato nessun file html, un campo testo dove inserire il testo della mail

In caso di messaggio html, contenenti specifici tag, la unit deve mostrare in ingresso un numero di campi pari al numero di tag indicati.

3. 4 Definizione delle proprietà

La unit necessita solo di due proprietà, che potranno essere editate nel pannello delle Proprietà di WebRatio SDS quando un istanza di tale unit viene selezionata:

- smtp-server: l' URL dell'SMTP server in uso.
- default-from: un indirizzo e-mail di default utilizzato nel campo from nel caso in cui quest'ultimo non sia disponibile in fase di esecuzione

3. 5 Creazione del descrittore XML

In un primo momento, è stata creata la struttura ad albero delle cartelle, come specificato nel paragrafo 3.2.

Viene successivamente creata la cartella SDS/conf/units/MultiMailUnits con le relative sottodirectory. Quindi si crea un file descriptor.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE unit-descriptor SYSTEM "unit-descriptor.dtd">
<unit-descriptor
  type="operation"
```

```
  version="1.0.0"
      tag-name="MULTIMAILUNIT"
      id-prefix="Mmlu"
      name-prefix="Send Multi Mail Unit "
      label="Send Multi Mail Unit "
      link-source="no"
      link-target="yes"      ko-link-source="yes"      ko-link-
target="yes"
      ok-link-source="yes" ok-link-target="yes"
      tree-icon-file="MultiMailUnit.gif"
      action-icon-file="MultiMailUnit.gif"
      diagram-icon-file="MultiMailUnitBig.gif">
      <string-property    id="MultiMailUnit.smtp"    label="SMTP
server"
            attribute-name="smtp-server"/>
      <string-property           id="MultiMailUnit.defaultFrom"
label="Default from"
            attribute-name="default-from"/>
<string-property id="MultiMailUnit.attachementCount" label="Nr.
of attachements" attribute-name="attCount"/>
      <file-property id="MultiMailUnit.html" label="File HTML"
            attribute-name="html"                  destination-
directory="descr"/>

</unit-descriptor>
```

Le prime due linee, dichiarano che si tratta di un fil XML e che è associato ad una DTD, detta unit-descriptor.dtd. tale file permette di specificare alcune proprietà della custom unit.

Dopo la dichiarazione XML, ci sono gli elementi che descrivono la unit contenenti i seguenti attributi:

- type: specifica il tipo della unit (operation o content)
- version: indica la versione corrente della unit
- tag-name: indica il nome della unit che verrà visualizzato per ogni istanza nel file di progetto. Il file di progetto è un file XML che rapresenta le specifiche WebML dell'applicazione, tale file viene salvato nella cartella DataRepository/<YourProject> and named SDSProject.xml.
- id-prefix: specifica il prefisso usato da WebRatio Site Development Studio per determinare l'XML ID dell'istanza di ogni unit.
- name-prefix: specific ail prefisso usato per generare il nome di default quando si istanzia una unit Site Development Studio.
- label: indica il nome dell'unità visualizzata WebRatio Site Development Studio nella toolbar.

- link-source: è una prorietà booleana che imposta il comportamento di WebRatio Site Development Studio, quando viene creato un link uscente dalla unit. Se è impostato a yes, la unit può essere fonte di informazioni per un link uscente (normal, automatic or transport). Altrimenti, WebRatio nega la possibilità di creare link in uscita dalla unit.
- link-target: se impostato a yes, la unit può avere link in entrata (normal, automatic or transport). Altrimenti WebRatio nega la possibilità di creare link in entrata.
- ko-link/ok-link source/target: simile alle proprietà sopra ma riferiti a OK e KO links.
- tree-icon-file: un icona di 16 x 16 pixel gif or jpeg usata per rappresentare la unit nell'albero di progetto di WebRatio Site Development Studio. L'immagine creata, è stata inserita nella relativa cartella.
- action-icon-file: un icona di 16 x 16 pixel gif o jpeg usata per rappresentare la unit nella barra degli strumenti di WebRatio Site Development Studio.
- diagram-icon-file: : un icona di 32 x 32 pixel gif o jpeg usata per rappresentare la unit nel diagramma delle viste del sito (site view diagram).

Oltre alle proprietà standard sopraindicate, utilizzate da WebRatio Site Development Studio per il rendering e la manipolazione di custom unit, vengono aggiunte delle proprietà che verranno visualizzate nel pannelo dela Proprietà quando un istanza della unit viene selezionata:

- id: L'identificatore unico di proprietà
- label: Il nome della proprietà indicate.
- attribute-name: Il nome dell'attributo XML usato nel project.file per salvare il valore di tale proprietà.

3.6 Visualizzazione della unit in WebRatio SDS

Dopo aver salvato il file descriptor.xml ed averlo correttamente inserito nella relative cartella, vengono creati i due file di immagine (MailUnit.gif e MailUnit- Big.gif) inserendoli nella directory images, lanciando in esecuzione WebRatio Site Development Studio, la nuova unit viene correttamente visualizzata sulla barra degli strumenti. Inserendo un'istanza della unit in una pagina web, vengono correttamente visualizzate le proprietà richieste nelle specifiche.

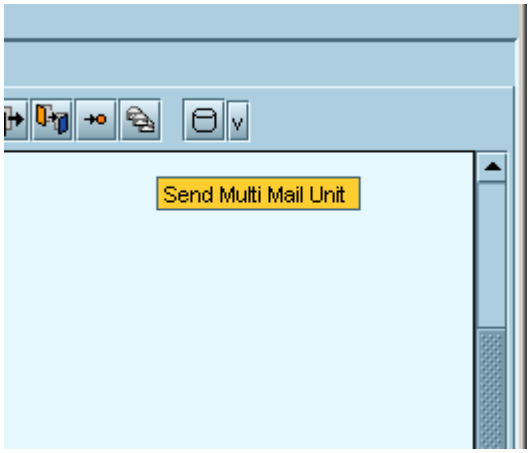


Figura 3.6.1: L'icona della barra degli strumenti

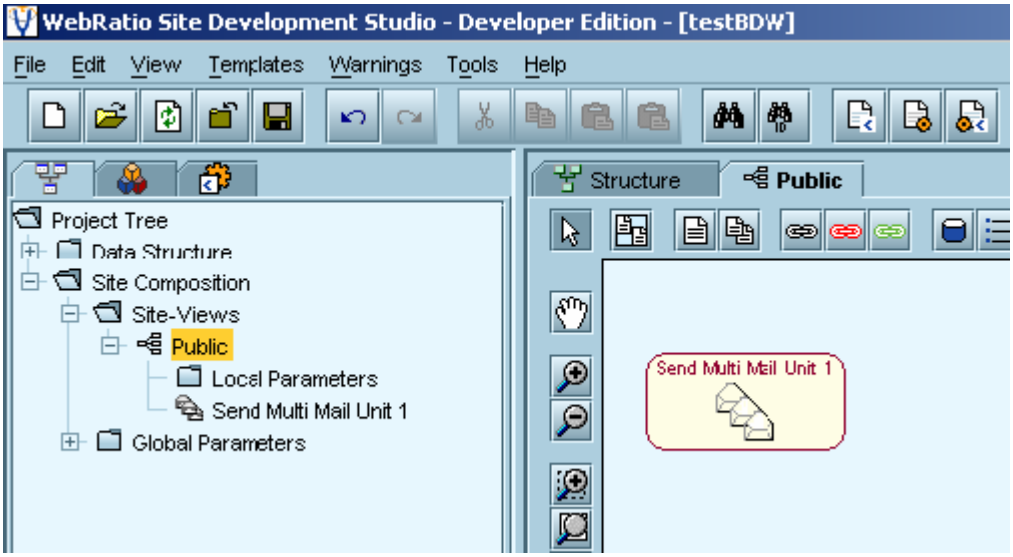


Figura 3.6.2: L'icona nel workspace

3.7 Specifiche di input / output

Lo step successivo del progetto consiste nella specifica dei parametri accettati in input o restituiti in output della unit.

Per tale motivo, vengono utilizzati due fogli di stile contenenti alcune regole XSLT, che una volta applicate al file XML, producono un frammento di file XML usato da WebRatio per costruire la finestra di dialogo dei parametri di accoppiamento (Couple source and target parameters)

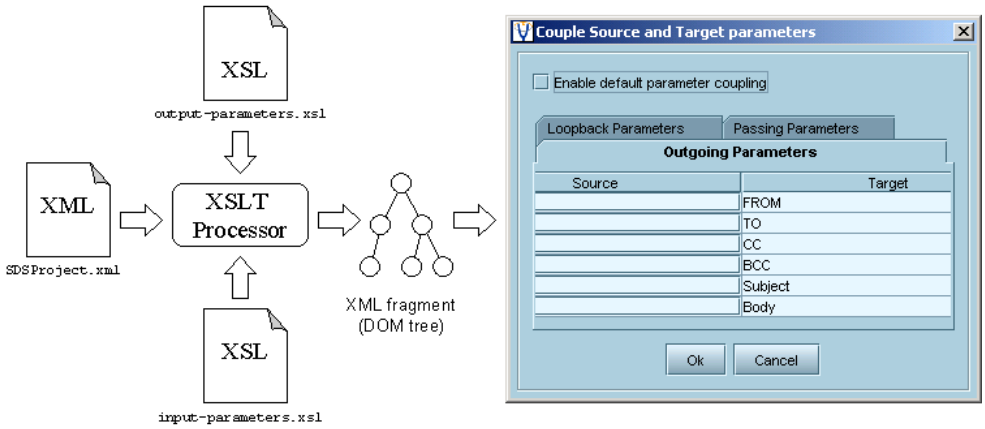


Figura 3.7.1: Il processo di computazione dei parametri esposti dalla unit

3.8 Specifica dei link di Input ed Output

L'operation unit "MultiMail", potrà essere collegata nel modo illustrato in figura:

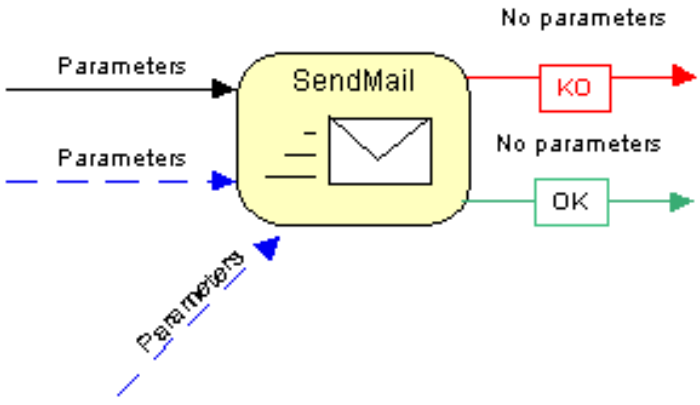


Figura 3.8.1: Possibili collegamenti della MultiMailUnit

La unit deve mostrare i seguenti parametri di input:

- From: indirizzo e-mail del mittente caricabile in modo statico o dinamico (String)
- To: Un set di indirizzi di destinazione (String[])
- CC: Un set di indirizzi CC (String[])
- BCC: Un set di indirizzi BCC (String[])
- Subject: L'oggetto del messaggio (String)
- Body: Il corpo del messaggio in formato txt, in caso non venga specificato nessun file html.
- File-HTML: Il corpo del messaggio in formato HTML
- Attachment n°: deve essere possibile specificare il numero di allegati e, in fase di accoppiamento visualizzare un campo per ogni allegato.

Per semplicità, la unit, non ha parametri di output: l'OK-link sarà invocato se il messaggio verrà inviato correttamente, e il KO-link se il server SMTP segnala un errore.

3.8.1 Specifiche del file html

Deve essere possibile inviare un messaggio personalizzato in formato html: ogni utente deve ricevere un messaggio con dei campi diversi, prelevati da un database.

Dopo un attento studio delle tecnologie disponibili, si è deciso di utilizzare un file in formato xhtml, con un nuovo tag "hole" che identifica un campo modificabile all'interno del documento. Tale tag potrà avere un attributo "name", utile per la gestione dell'accoppiamento dei dati in fase di sostituzione.

Una volta definito il file html e creati i giusti accoppiamenti, è possibile mandare in esecuzione il sistema che, sostituirà all'interno del documento html tutti i tag "hole" con i relativi dati contenuti nel database.

In fase di runtime, il file html viene caricato in memoria, vengono cercati i tag <hole> e i relativi accoppiamenti, quindi si crea un file html per ogni destinatario, infine ogni file viene inviato al mittente indicato DB.

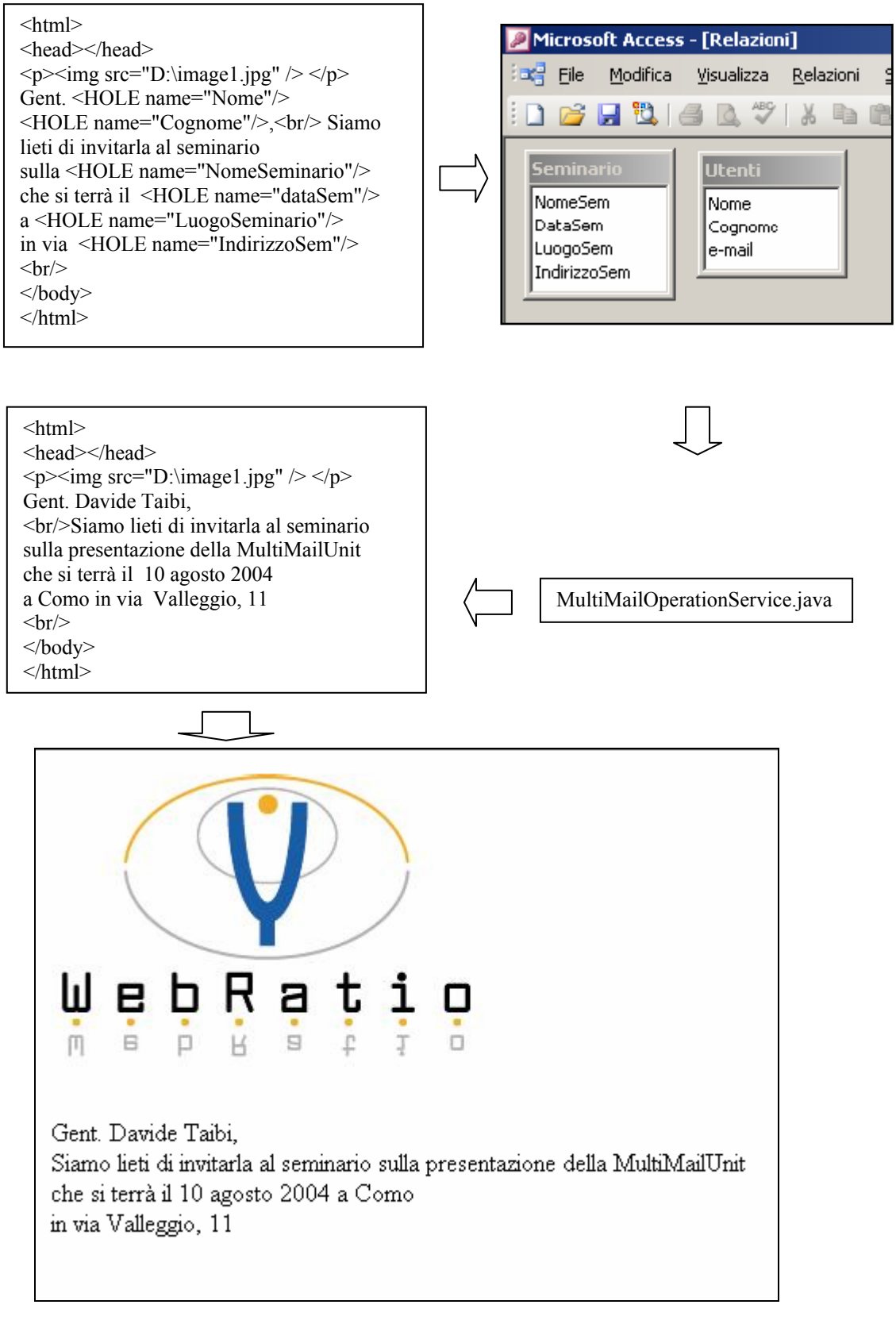


Figura 3.8.1 Processo di gestione del file html

3.9 Definizione dei parametri di Input

La specifica dei parametri di input viene fatta mediante la creazione di un file XSL, dove si andranno ad indicare delle regola XSL che produrranno in output un file XML, contenente i parametri di input tra cui: un elemento "logic:input-parameters", che comprende uno o più sotto elementi. Ogni elemento descrive un parametro di input, utilizzando gli attributi seguenti:

- name: L'identificatore del parametro.
- type: Il tipo del parametro.
- label: L'etichetta che mostrerà WebRatio Site Development Studio in fase di visualizzazione dei parametri di input nella finestra di coupling

Tutti i parametri di input sono stati implementati nel seguente file, quindi salvati nella sottodirectory "conf/units/common/logic".

input-parameters.xsl

```
<?xml version="1.0"?>
<!-- ================================================= -->
<!-- input-parameters -->
<!-- lists input parameters accepted by this unit -->
<!-- ================================================= -->
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:logic="http://www.WebML.org/logic"
xmlns:auxiliary="http://www.WebML.org/auxiliary"
version="1.0">
<xsl:template match="MULTIMAILUNIT"mode="input-parameters">
<logic:input-parameters>
<logic:input-parameter name="{@id}.from"
type="string" label="FROM"/>
<logic:input-parameter name="{@id}.to"
type="set-of(string)" label="TO"/>
<logic:input-parameter name="{@id}.cc"
type="set-of(string)" label="CC"/>
<logic:input-parameter name="{@id}.bcc"
type="set-of(string)" label="BCC"/>
<logic:input-parameter name="{@id}.subject"
type="string" label="Subject"/>

<xsl:if test="not (@html or @html != '')">
      <logic:input-parameter name="{@id}.body" type="text"
label="Body"/>
</xsl:if>
<xsl:if test="string(number(@attCount)) != 'NaN'">
  <xsl:call-template name="count-attachments">
    <xsl:with-param name="n" select="number(@attCount)"/>
  </xsl:call-template>
</xsl:if>
```

```
<xsl:variable name="path"><xsl:value-of
select="/WebML/@auxiliary:project-directory"/></xsl:variable>
<xsl:message><xsl:value-of
select="concat('file:///',$path,'/descr/',
@html)"/></xsl:message>
<xsl:if test="(@html and @html != '')">
      <xsl:apply-templates
select="document(concat('file:///',$path,'/descr/',
@html))//HOLE">
            <xsl:with-param name="id" select="@id"/>
      </xsl:apply-templates>
</xsl:if>
</logic:input-parameters>
</xsl:template>

<xsl:template name="count-attachments">
  <xsl:param name="n"/>
  <xsl:if test="$n &gt; 0">
    <xsl:call-template name="count-attachments">
      <xsl:with-param name="n" select="$n - 1"/>
    </xsl:call-template>
    <logic:input-parameter name="{@id}.att{$n}" type="string"
label="Attachment nr. {$n}"/>
  </xsl:if>
</xsl:template>

<xsl:template match="PROPERTY[(@name!='') and (
                                  (@value='subject') or
(@value='Subject') or (@value='SUBJECT')
                                  or (@value='body') or
(@value='Body') or (@value='BODY')
                                  or (@value='attachment') or
(@value='Attachment') or (@value='ATTACHMENT')
                                  ) and (@name!='smtp-server')
and (@name!='debug') and (@name!='subject-separator')
                                  and (@name!='body-separator')
and (@name!='validate-addresses')]" mode="input-parameters">
    <xsl:param name="unitID"/>
    <logic:input-parameter name="{$unitID}.{@name}"
type="attribute" label="{@name}"/>
  </xsl:template>

<xsl:template match="HOLE">
      <xsl:param name="id"/>
      <logic:input-parameter label="{@name}" type="string"
name="{$id}.{@name}"/>
</xsl:template>
</xsl:stylesheet>
```

La mail unit non ha parametri di output, restituisce solamente un ok-link, in caso di invio corretto di un messaggio.mail.

A tale scopo viene creato il file output-parameters.xsl:

```
<?xml version="1.0"?>
<!-- ========================================== -->
<!-- output-parameters -->
<!-- lists output parameters exposed by this unit -->
<!-- ========================================== -->
```

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:logic="http://www.WebML.org/logic"
version="1.0">
<xsl:template match="MAILUNIT" mode="output-parameters">
<logic:output-parameters/>
</xsl:template>
</xsl:stylesheet>
```

Come è possible notare, tale file contiene solamente un elemento logic:output-parameters vuoto.

Dopo tali modifiche, è possible visualizzare correttamente tutti I parametri della unit in WebRatio SDS

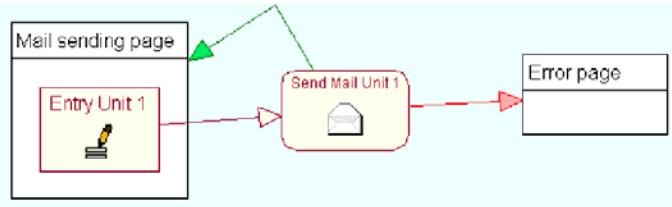


Figura 3.9.1

Selezionando il link di ingresso, nel caso non venga specificato nessun file html, vengono visualizzati i seguenti parametri di input.

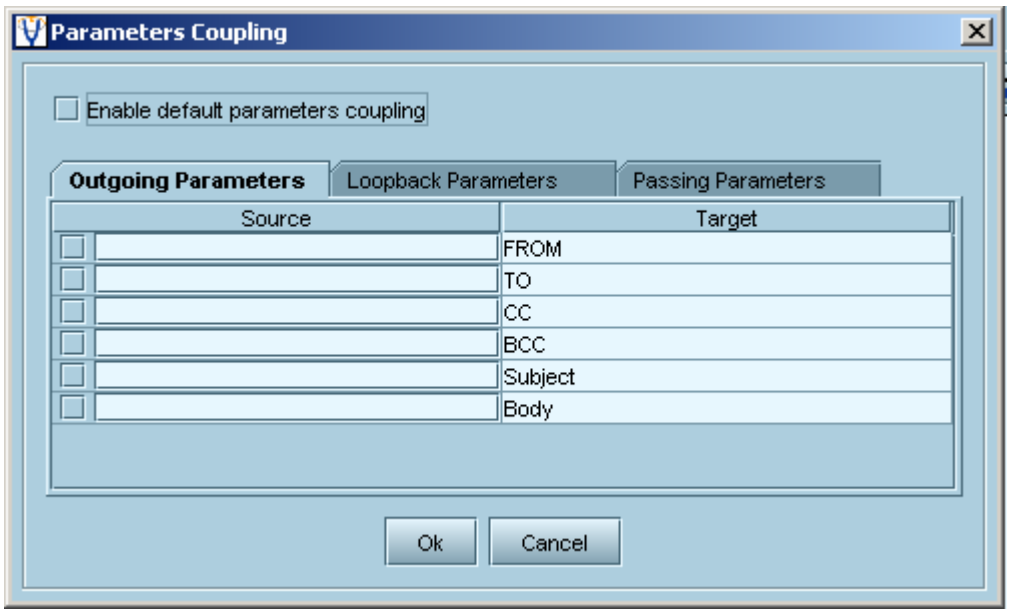


Figura 3.9.2: Finestra di Coupling nel caso di messaggio in formato text

Nel caso, invece, venga specificato un file html (in questo caso con quattro campi vuoti, selezionando il link in ingresso viene visualizzata la seguente finestra:

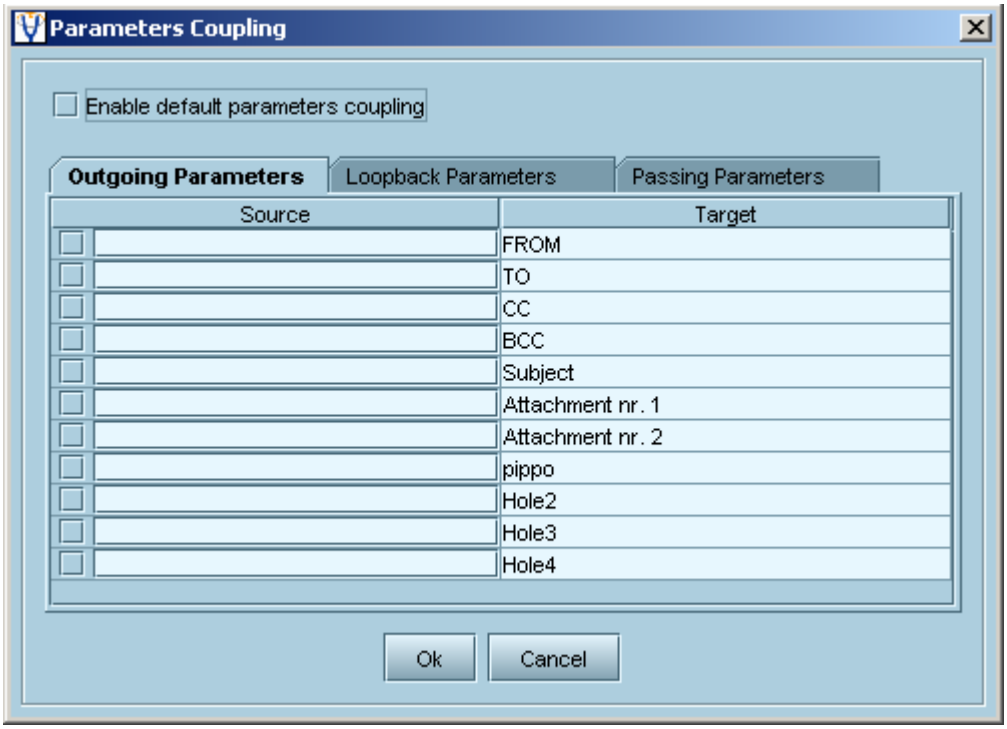


Figura 3.9.3: Finestra di Coupling nel caso di messaggio in formato html

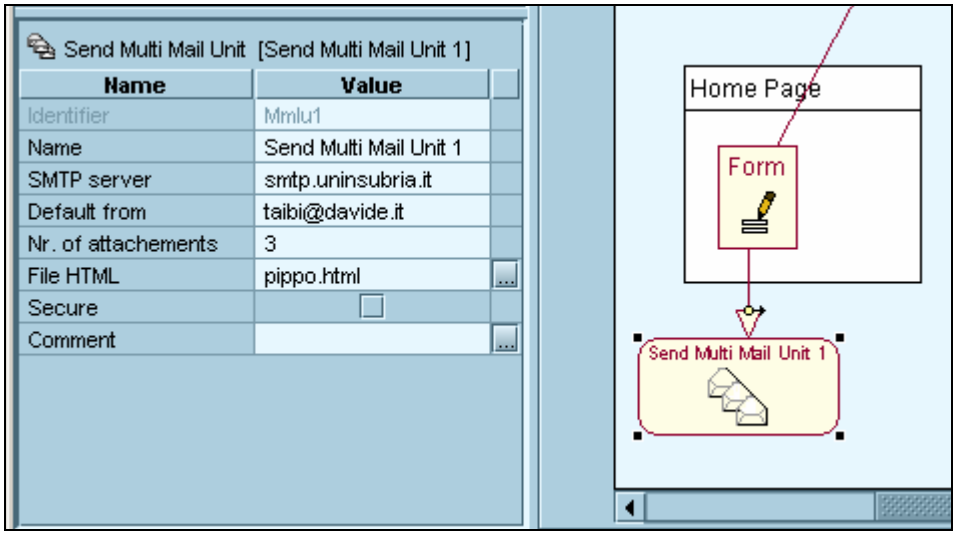


Figura 3.9.4: Visualizzazione della Unit in WebRatio SDS

3.10 Progettazione del descrittore di runtime

Per utilizzare le funzionalità della unit, bisogna implementare un servizio per la piattaforma J2EE; tale servizio utilizzerà le API javamail (www.java.sun.com) per istanziare e inviare messaggi di email.

Il servizio utilizza due tipi di input:

- I parametri di input (subject, body, recipients) forniti dall'utente in fase di runtime, quindi istanzia una richiesta HTTP
- I parametri di input forniti dal progettista in fase di design come valori di default, da utilizzare nel caso non siano presenti I valori sopraindicati.

A tale scopo, era possibile creare due differenti classi, una per ogni tipo di operazione. Tale soluzione porta ad una duplicazione del codice, con una scarsa manutenibilità.

Quindi si preferisce sfruttare il Runtime Support framework di WebRatio, e creare un'unica classe generica, utilizzando un descrittore XML per ricevere i valori del mittente e del server smtp di default di una specifica istanza della unit, quindi utilizzare tali informazioni per creare concretamente il servizio associato alla singola istanza della unit.

Il processo di sviluppo viene suddiviso in due fasi:

- Progettazione del formato del descrittore del servizio e delle regole XSLT per permettere a WebRatio di creare automaticamente il file di progetto.
- Implementazione del servizio generico della unit.

In un primo momento, viene definito il contenuto del descrittore del servizio che chiameremo genericamente servizio di sendmail.

Tale documento dovrà semplicemente copiare nei campi "default-from" e "smtp-server" i valori delle proprietà corrispondenti ad ogni istanza delle unit create dal progettista WebML.

Una volta stabilito il formato del descrittore del servizio, è possible utilizzare WebRatio per produrre il descrittore di runtime per tutte le istanza delle unit ma, per maggior completezza, si preferisce editare tale file manualmente, inserendolo nella sottodirectory WEBINF/descr

Quando vine fatto il build dell'applicazione completa dalla "Presentation View", WebRatio applica al file di progetto delle regole XSL richiamate dal file operation.xsl associato ad ogni unit e istanziato per ogni occorrenza della unit in fase di runtime.

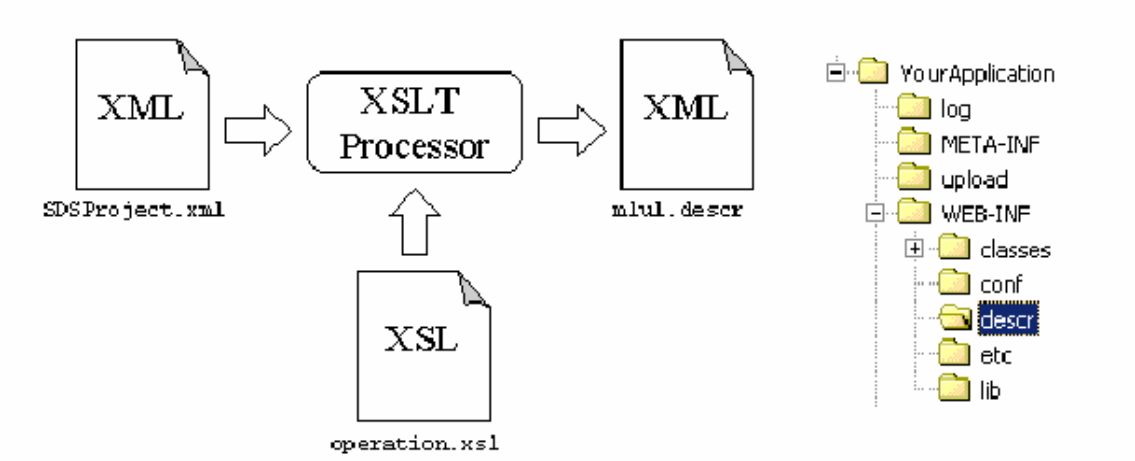


Figura 3.10.1: Il processo di generazione del descrittore di runtime

3.11 Progettazione della classe di servizio

Per semplificare la creazione delle classi di servizio nelle nuove unit, WebRatio richiede di rispettare alcune interfaccie Java.

Una "unit service" è una classe che viene invocata dal Runtime Support framework di WebRatio per eseguire le computazioni richieste dal contenuto della operation unit.

La classe di servizio deve rispettare il nome dato nel descrittore XML creato in precedenza, si decide quindi di chiamare la classe MultiMailOperationService.

Il prossimo passo consiste nel progetto e nell'implementazione della classe Java, rispettando tutte le funzioni necessarie al funzionamento della unit come specificato nel documento di specifica dei requisiti.

Tutto ciò richiede i seguenti passi:

- Implementazione del cuore del servizio, con la creazione dei messaggi di posta elettronica
- Ricerca delle informazioni dal descrittore di runtime XML di ogni unit associate al servizio.
- Ricerca dei parametri di input della richiesta http
- Notifica del risultato dell'operazione

Il nocciolo dello sviluppo del servizio è il progetto e l'implementazione della classe che rappresenta l'operation unit service. Per poter essere utilizzata all'interno di WebRatio, tale classe deve rispettare alcune regole:

- La classe deve implementare un interfaccia chiamata RTXOperationUnitService che, a sua volta estende un altra interfaccia chiamata RTXService. Il listato dell'interfaccia viene qui riportato:

```
public interface RTXService  {
public  String  getID();
public  void  dispose();
}

public  interface  RTXOperationUnitService
                extends  RTXService  {
          public Object execute(Map
                   operationContext Map
                   sessionContext)
      throws  RTXException;
}
```

- La classe deve avere un costruttore con i parametri appropriati.

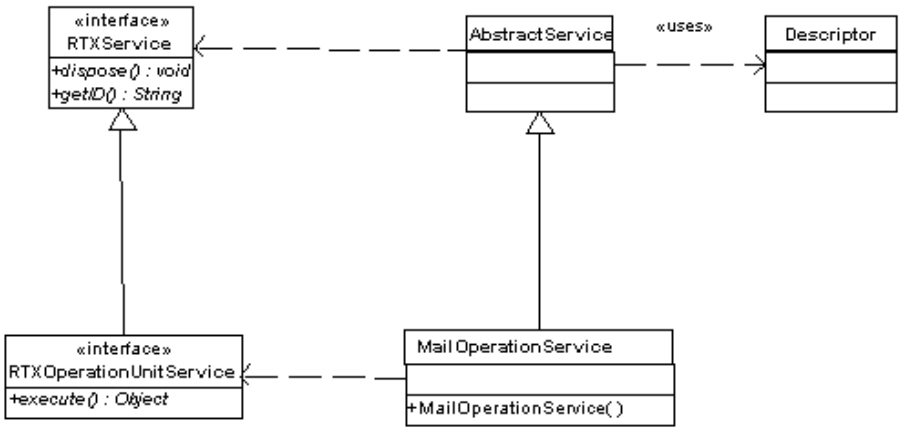


Figura 3.11.1: UML class diagram

Nella prossima sezione, verranno descritte tutte le implementazioni dei metodi che costituiscono la classe di servizio.

In particolare, vi sono quattro metodi che richiedono di essere implementati:

- Il costruttore: riceve un unico identificatore che può essere associato ad un sottoservizio.
- execute(): è il metodo che inizia la computazione, riceve due parametri (maps) contenenti la richiesta http e i dati relativi alla sessione. Ritorna un oggetto con una proprietà detta resultCode. Se tale proprietà è impostata a "success" ritorna un OK-Link, se impostata ad "error" ritorna un KO-Link
- dispose(): rilascia tutte le risorse allocate in precedenza.
- getID(): ritorna l'identificatore dell'istanza del servizio.

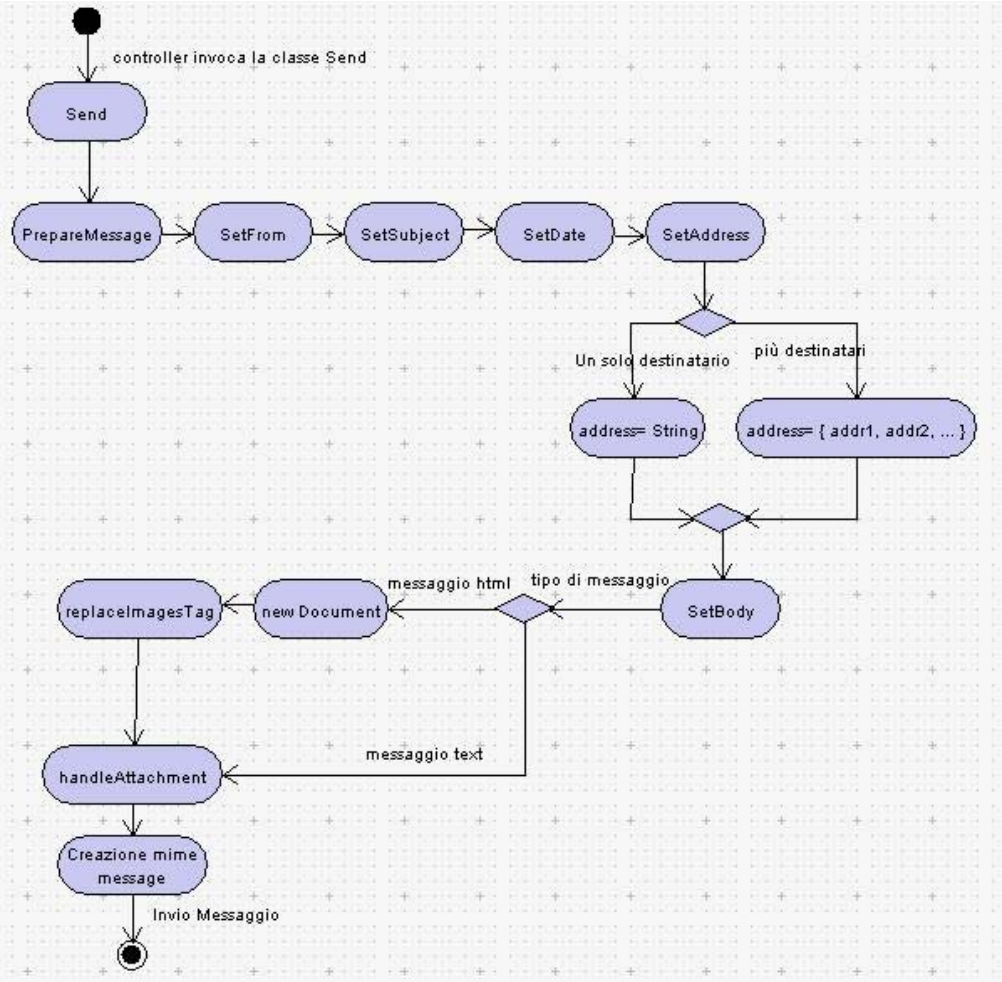


Figura 3.11.2: Flusso dei dati della unit

Quando, in fase di runtime, viene richiamata un'istanza della unit, si invoca il metodo execute della classe MultiMailOperationService che richiama il proprio metodo prepareMessage dove si crea un nuovo MimeMessage, si imposta l'intestazione del messaggio (mittente, destinatario, data e oggetto), quindi se il messaggio è in formato testo, imposta il corpo del messaggio in formato testo, altrimenti, il metodo setBody crea un Documento XML dato dal file HTML specificato nelle opzioni dell'istanza della unit e sostituisce i tag <hole> con i parametri corrispondenti, ricevuti dal database.

Infine, in caso di allegati li aggiunge al messaggio e lo invia al destinatario specificato.

Capitolo

4

IMPLEMENTAZIONE DEL SERVIZIO

4.1 Introduzione

L'implementazione della classe di servizio è stata fatta partendo dalle specifiche.

Per la creazione del messaggio ci si è avvalsi delle funzionalità fornite dalle API javax.mail fornite da SUN.

Si inizia dall'illustrazione del costruttore della classe, tale metodo è un istanza del servizio creato.

Il costruttore è chiamato dalla classe ServiceProvider, quando si chiama l'istanza specifica della unit per la prima volta.

In seguito viene illustrato il costruttore della classe:

```
public MultiMailOperationService(
        String id,
        RTXManager mgr,
        Descriptor descr) throws RTXException {
        super(id, mgr.getLog());
        this.mgr = mgr;
        DescriptorElement fr = getChild(descr, "default-from", true);
        this.defaultFrom = fr.getValue();
        DescriptorElement svr = getChild(descr, "smtp-server", true);
        this.server = svr.getValue();
        DescriptorElement htm = getChild(descr, "file-html", true);
        this.htmlPath = htm.getValue();
        DescriptorElement attach=getChild(descr, "attachment-count", true);
        try {
            attachementNumber = Integer.parseInt(attach.getValue());
        } catch (NumberFormatException e) {
            attachementNumber = 0;
        }

        DescriptorElement dbg = getChild(descr, "debug", false);
        if (dbg != null) {
            if (dbg.getValue().equals("true")) {
                this.debug = true;
            }
        }
    }
```

Come già indicato, la classe estende AbstractService e implementa l'interfaccia RTXOperationUnitService.

La classe contiene due variabili (defaultFrom e server), che corrispodono al mittenta di default ed al server di default che vengono associati alla sendmail unit.

Ogni istanza del servizio, ha valori propri per tali variabili, è quindi possibile utilizzare nella stessa applicazione server SMTP e mittenti di default diversi.

Il costruttore della classe riceve tre parametri:

- id: una stringa indicante l'identificatore univoco della unit
- mgr: un oggetto di tipo RTXManager, che può essere utilizzato per invocare il Runtime Manager, nel caso che la unit necessita di sotto servizi.
- Descr: un oggetto di tipo Descriptor che rappresenta il risultato del parsing del servizio.

Il costruttore di default, invoca il costruttore dell'AbstractService che inizializza le variabili e imposta l'identificatore del servizio. Quindi riceve dal descrittore le informazioni necessarie per inizializzare i campi membro.

Dopo aver istanziato tutti i membri, questi non potranno essere più modificati per tutta la durata del servizio.

4.2 Implementazione del metodo execute()

Il metodo execute, riportato sotto, è invocato dall'operation ogni volta che l'utente in fase di runtime, utilizza il servizio.

```
public MultiMailOperationService(
        String id,
        RTXManager mgr,
        Descriptor descr) throws RTXException {
        super(id, mgr.getLog());
        this.mgr = mgr;
        DescriptorElement fr = getChild(descr, "default-from", true);
        this.defaultFrom = fr.getValue();
        DescriptorElement svr = getChild(descr, "smtp-server", true);
        this.server = svr.getValue();
        DescriptorElement htm = getChild(descr, "file-html", true);
        this.htmlPath = htm.getValue();
        DescriptorElement attach=getChild(descr, "attachment-count", true);
        try {
            attachementNumber = Integer.parseInt(attach.getValue());
        } catch (NumberFormatException e) {
            attachementNumber = 0;
        }

        DescriptorElement dbg = getChild(descr, "debug", false);
        if (dbg != null) {
            if (dbg.getValue().equals("true")) {
                this.debug = true;
            }
        }
    }
```

Il metodo riceve in input due oggetti necessari all'esecuzione:

- La mappa operationContext contiene tutti i parametri della richiesta, inclusi i parametri inviati dalla http request. Il contenuto di questo oggetto è utilizzabile per tutta la durata della risposta HTTP. Il servizio, estrarrà dalla mappa operationContext i parametri necessari.
- La mappa sessionContext contiene tuti i parametri relativi alla sessione

Le due mappe (operationContext e sessionContext) verrano utilizzate sia per ricevere i dati che per salvarli.

Il metodo execute( ) verifica il numero dei destinatari, crea un MimeMessage per ogni destinatario, invoca il metodo prepareMessage( ) e ritorna uno stato, (una proprietà resultCode) utilizzato della unit per restituire il risultato dell'operazione.

4.3 Implementazione del metodo prepareMessage()

Il metodo prepareMessage, riceve in input due parametri map (operationContext e sessionContext) e estrae dalla mappa operationContext i dati necessary per creare un messaggio object email (from, to, cc, bcc, subject, and body)

Il codice del metodo prepareMessage è riportato sotto:

```
private MimeMessage prepareMessage(
        Map operationContext,
        Map sessionContext, int counter, ArrayList
tempFiles) throws
        RTXException {

        // creates some properties and get the default
Session
        Properties props = new Properties();
        props.put("mail.smtp.host", server);

        Session session = Session.getDefaultInstance(props,
null);
        session.setDebug(debug);
        // creates a message
        MimeMessage msg = new MimeMessage(session);

        // sets the from field
        setFrom(operationContext, sessionContext, msg);

        // sets its recipient addresses
        setAddresses(operationContext, sessionContext, msg,
TO_SUFFIX, counter);
        setAddresses(operationContext, sessionContext, msg,
CC_SUFFIX, counter);
        setAddresses(operationContext, sessionContext, msg,
BCC_SUFFIX, counter);

        // sets the subject
        setSubject(operationContext, sessionContext, msg,
counter);

        // sets the body
        setBody(operationContext, sessionContext, msg,
counter, tempFiles);

        return msg;
    }
```

Il metodo crea ed inizializza una sessione JavaMail, quindi richiama tutte le funzioni ausiliarie necessarie per l'impostazione dell'intestazione del messaggio.

A scopo illustrativo viene riportato il codice di uno dei metodi creati:

```
private void setSubject(
        Map operationContext,
        Map sessionContext,
        MimeMessage msg, int counter) throws RTXException {
        Object sub = operationContext.get(id +
SUBJECT_SUFFIX);
        if (sub != null) {
            try {
                if (sub instanceof String) {
                    msg.setSubject( (String) sub);
                }
                if (sub instanceof String[]) {
                    msg.setSubject( ( (String[])
sub)[counter]);
                }
            } catch (MessagingException ex) {
            }
        }
    }
```

I metodi setAdresses( ) e setFrom( ) sono simili al metodo sopra mentre il metodo setBody() merita una trattazione a parte.

4.4 Implementazione del metodo setBody()

Il metodo imposta il corpo del messaggio in formato html o testo inserendo gli allegati del messaggio.

In caso di messaggio in formato html, setBody( ) crea un Documento XML in memoria contenente il file HTML specificato, quindi richiama il metodo replaceHoles che si occupa di recuperare tutti i tag <hole> nel documento html e di sostituirli con quelli relevati dalla mappa operationContext.

Infine, se ci sono uno o più allegati, richiama il metodo handleAttachment() per inserire correttamente tutti gli allegati.

4.5 Implementazione del metodo send()

L'ultima parte di codice necessaria per il funzionamento del servizio è il corpo del metodo send(), indicato sotto:

```
private void send(MimeMessage msg) throws RTXException {
        try {

            Transport.send(msg);
            System.out.println("Ok, messaggio inviato");
        } catch (MessagingException mse) {
            mse.printStackTrace();
            throw new RTXException(mse);
        }
    }
```

Send, usa un metodo statico per inviare il messaggio costruito dal metodo prepareMessage() e verifica le possibili eccezioni.

Capitolo

5

TESTING

Capitolo

6

CONCLUSIONI E SVILUPPI FUTURI

Il lavoro svolto nel corso della tesina ha prodotto dei risultati conformi alle aspettative ed agli obiettivi prefissati.
Attraverso la progettazione della MultiMailUnit, si è potuto venire a contatto con uno strumento CASE estremamente potente come Site Development Studio, che permette di creare applicazioni web data–intensive a partire da un modello basato su WebML, quindi molto intuitivo e di facile comprensione anche ad utenti meno esperti.
Lavorando a stretto contatto con il team di sviluppatori di WebRatio, che sono poi gli utilizzatori del componente sviluppato, si è potuti venire a conoscenza delle loro esigenze per quanto riguarda la struttura e i contenuti della unit stessa. Basandoci quindi sulle richieste degli sviluppatori stessi sono stati messi a punto dei requisiti da sviluppare, relativi a nuove funzionalità da implementare ma anche alla correzione di possibili imperfezioni rilevate.
L'enfasi di questo lavoro è stata posta sull'implementazione delle funzionalità piuttosto che sull'aspetto della presentazione all'utente.
Uno degli sviluppi futuri di questo lavoro potrebbe essere quindi il miglioramento dell'interfaccia grafica, e la possibilità in inviare video messaggi.

APPENDICE A

Sommario dell'implementazione

Descriptor.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE unit-descriptor SYSTEM "unit-descriptor.dtd">
<unit-descriptor
  type="operation"
  version="1.0.0"
      tag-name="MULTIMAILUNIT"
      id-prefix="Mmlu"
      name-prefix="Send Multi Mail Unit "
      label="Send Multi Mail Unit "
      link-source="no"
      link-target="yes" ko-link-source="yes" ko-link-
target="yes"
      ok-link-source="yes" ok-link-target="yes"
      tree-icon-file="MultiMailUnit.gif"
      action-icon-file="MultiMailUnit.gif"
      diagram-icon-file="MultiMailUnitBig.gif">
      <string-property id="MultiMailUnit.smtp" label="SMTP
server"
            attribute-name="smtp-server"/>
      <string-property id="MultiMailUnit.defaultFrom"
label="Default from"
            attribute-name="default-from"/>
<string-property id="MultiMailUnit.attachementCount" label="Nr.
of attachements"
            attribute-name="attCount"/>

      <file-property id="MultiMailUnit.html" label="File HTML"
            attribute-name="html" destination-
directory="descr"/>

</unit-descriptor>
```

input-parameters.xsl

```
<?xml version="1.0"?>
<!--============================================ -->
<!-- input-parameters -->
<!-- lists input parameters accepted by this unit -->
<!-- ============================================ -->
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:logic="http://www.WebML.org/logic"
xmlns:auxiliary="http://www.WebML.org/auxiliary"
version="1.0">
<xsl:template match="MULTIMAILUNIT" mode="input-parameters">
<logic:input-parameters>
<logic:input-parameter name="{@id}.from"
type="string" label="FROM"/>
<logic:input-parameter name="{@id}.to"
type="set-of(string)" label="TO"/>
<logic:input-parameter name="{@id}.cc"
type="set-of(string)" label="CC"/>
<logic:input-parameter name="{@id}.bcc"
type="set-of(string)" label="BCC"/>
<logic:input-parameter name="{@id}.subject"
type="string" label="Subject"/>

<xsl:if test="not (@html or @html != '')">
      <logic:input-parameter name="{@id}.body" type="text"
label="Body"/>
</xsl:if>
<xsl:if test="string(number(@attCount)) != 'NaN'">
  <xsl:call-template name="count-attachments">
    <xsl:with-param name="n" select="number(@attCount)"/>
  </xsl:call-template>
</xsl:if>
<xsl:variable name="path"><xsl:value-of
select="/WebML/@auxiliary:project-directory"/></xsl:variable>
<xsl:message><xsl:value-of
select="concat('file:///',$path,'/descr/',
@html)"/></xsl:message>
<xsl:if test="(@html and @html != '')">
      <xsl:apply-templates
select="document(concat('file:///',$path,'/descr/',
@html))//HOLE">
            <xsl:with-param name="id" select="@id"/>
      </xsl:apply-templates>
</xsl:if>
</logic:input-parameters>
</xsl:template>

<xsl:template name="count-attachments">
  <xsl:param name="n"/>
  <xsl:if test="$n &gt; 0">
    <xsl:call-template name="count-attachments">
      <xsl:with-param name="n" select="$n - 1"/>
    </xsl:call-template>
    <logic:input-parameter name="{@id}.att{$n}" type="string"
label="Attachment nr. {$n}"/>
  </xsl:if>
</xsl:template>
```

```
<xsl:template match="PROPERTY[(@name!='') and (
                                    (@value='subject') or
(@value='Subject') or (@value='SUBJECT')
                                    or (@value='body') or
(@value='Body') or (@value='BODY')
                                    or (@value='attachment') or
(@value='Attachment') or (@value='ATTACHMENT')
                                    ) and (@name!='smtp-server')
and (@name!='debug') and (@name!='subject-separator')
                                    and (@name!='body-separator')
and (@name!='validate-addresses')]" mode="input-parameters">
    <xsl:param name="unitID"/>
    <logic:input-parameter name="{$unitID}.{@name}"
type="attribute" label="{@name}"/>
  </xsl:template>

<xsl:template match="HOLE">
      <xsl:param name="id"/>
      <logic:input-parameter label="{@name}" type="string"
name="{$id}.{@name}"/>
</xsl:template>
</xsl:stylesheet>
```

output-parameters.xsl

```
<?xml version="1.0"?>

<!-- ========================================== -->
<!-- output-parameters
-->
<!-- lists output parameters exposed by this unit -->
<!-- ========================================== -->
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:logic="http://www.WebML.org/logic"
                version="1.0">

  <xsl:template match="MULTIMAILUNIT" mode="output-parameters">
    <logic:output-parameters/>
  </xsl:template>

</xsl:stylesheet>
```

operation.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:template match="MULTIMAILUNIT" mode="descriptor">
    <descriptor service="mail.MultiMailOperationService">
      <default-from><xsl:value-of select="@default-
from"/></default-from>
      <smtp-server><xsl:value-of select="@smtp-server"/></smtp-
server>
      <file-html><xsl:value-of select="@html"/></file-html>
```

```
<attachment-count><xsl:value-of
select="@attCount"/></attachment-count>
    </descriptor>
  </xsl:template>

</xsl:stylesheet>
```

type.xsl

```
<?xml version="1.0"?>
<!-- ========================================== -->
<!-- type                                       -->
<!--      provides information about the unit   -->
<!-- ========================================== -->
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:logic="http://www.WebML.org/logic"
version="1.0">
<!-- ========================================== -->
<!--Indicates whether the matched element is a unit->
<!--========================================== -->
  <xsl:template match="MULTIMAILUNIT" mode="is-unit">
    <xsl:text>yes</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

MultiMailOperation.java

```
package mail;

import java.io.*;
import java.util.*;
import javax.activation.*;
import javax.mail.*;
import javax.mail.internet.*;

import com.webratio.WebML.rtx.*;
import com.webratio.WebML.rtx.core.*;
import com.webratio.WebML.rtx.core.beans.*;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Attr;

import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;

public class MultiMailOperationService
    extends AbstractService
    implements RTXOperationUnitService {
```

```
    // static strings used for parameter retrieval
    private static final String FROM_SUFFIX = ".from";
    private static final String TO_SUFFIX = ".to";
    private static final String CC_SUFFIX = ".cc";
    private static final String BCC_SUFFIX = ".bcc";
    private static final String SUBJECT_SUFFIX = ".subject";
    private static final String BODY_SUFFIX = ".body";

    // member fields initialized from descriptor file
    private String defaultFrom;
    private String server;
    private String htmlPath;
    private RTXManager mgr;
    private int attachementNumber;
    boolean debug = false;

    private TreeSet bodyParts;
    private TreeSet subjectParts;
    private TreeSet attachParts;

    public MultiMailOperationService(
        String id,
        RTXManager mgr,
        Descriptor descr) throws RTXException {
        super(id, mgr.getLog());
        this.mgr = mgr;
        DescriptorElement fr = getChild(descr, "default-from", true);
        this.defaultFrom = fr.getValue();
        DescriptorElement svr = getChild(descr, "smtp-server", true);
        this.server = svr.getValue();
        DescriptorElement htm = getChild(descr, "file-html", true);
        this.htmlPath = htm.getValue();
        DescriptorElement attach=getChild(descr,"attachment-count",
true);
        try {
            attachementNumber = Integer.parseInt(attach.getValue());
        } catch (NumberFormatException e) {
            attachementNumber = 0;
        }

        DescriptorElement dbg = getChild(descr, "debug", false);
        if (dbg != null) {
            if (dbg.getValue().equals("true")) {
                this.debug = true;
            }
        }

    }

    public Object execute(Map operationContext, Map sessionContext)
throws
        RTXException {
        BasicOperationUnitBean bean = new BasicOperationUnitBean();

        System.out.println("---- Start Of Send Mail Service ---");

        try {
            Object counter = operationContext.get(this.id +
this.TO_SUFFIX);
            ArrayList tempFiles = new ArrayList();
            if (counter instanceof String[]) {

                for (int i = 0; i < ( (String[]) counter).length; i++)
{
                    MimeMessage msg = prepareMessage(operationContext,
                        sessionContext, i, tempFiles);
                    send(msg);
```

```
                }

bean.setResultCode(BasicOperationUnitBean.SUCCESS_CODE);
            }
            if (counter instanceof String) {
                MimeMessage msg = prepareMessage(operationContext,
                                                 sessionContext, 0,
tempFiles);
                send(msg);

bean.setResultCode(BasicOperationUnitBean.SUCCESS_CODE);
                // finally deletes temporary files
                for (int i = 0; i < tempFiles.size(); i++) {
                    ( (File) tempFiles.get(i)).delete();
                    if (debug) {
                        System.out.println("Temp file erased");
                    }
                }

             System.out.println("---- End Of Send Mail Service ---");

            }

        } catch (RTXException e) {
            e.printStackTrace();
            // logs the wrapped exception
            logError("Unable to send mail", e.getException());
            bean.setResultCode(BasicOperationUnitBean.ERROR_CODE);
        }
        return bean;
    }

    private MimeMessage prepareMessage(
        Map operationContext,
        Map sessionContext, int counter, ArrayList tempFiles) throws
        RTXException {

        // creates some properties and get the default Session
        Properties props = new Properties();
        props.put("mail.smtp.host", server);

        Session session = Session.getDefaultInstance(props, null);
        session.setDebug(debug);
        // creates a message
        MimeMessage msg = new MimeMessage(session);

        // sets the from field
        setFrom(operationContext, sessionContext, msg);

        // sets its recipient addresses
        setAddresses(operationContext, sessionContext, msg, TO_SUFFIX,
counter);
        setAddresses(operationContext, sessionContext, msg, CC_SUFFIX,
counter);
        setAddresses(operationContext, sessionContext, msg,
BCC_SUFFIX, counter);

        // sets the subject
        setSubject(operationContext, sessionContext, msg, counter);

        // sets the body
        setBody(operationContext, sessionContext, msg, counter,
tempFiles);

        return msg;
    }
```

```
    private void setFrom(
        Map operationContext,
        Map sessionContext,
        MimeMessage msg) throws RTXException {
        String from = defaultFrom;
        Object obj = operationContext.get(id + FROM_SUFFIX);
        if (obj != null) {
            if (obj instanceof String) {
                if ( ( (String) obj).length() > 0) {
                    from = ( (String) obj).trim();
                }
            }
        }
        InternetAddress msgFrom = null;
        try {
            msgFrom = new InternetAddress(from);
        } catch (AddressException ae) {
            ae.printStackTrace();
            throw new RTXException(ae);
        }
        try {
            msg.setFrom(msgFrom);
        } catch (MessagingException me) {
            me.printStackTrace();
            throw new RTXException(me);
        }
    }

    private void setAddresses(
        Map operationContext,
        Map sessionContext,
        MimeMessage msg,
        String type, int counter) throws RTXException {

        String address = null;
        Object value = operationContext.get(id + type);
        if (value != null) {
            if (value instanceof String) {
// parameter is a plain string
                String addressLine = ( (String) value).trim();
                StringTokenizer tokenizer =
                    new StringTokenizer(addressLine, ",;");
                while (tokenizer.hasMoreTokens()) {
                    address = tokenizer.nextToken().trim();
                }
            }
            if (value instanceof String[]) {
// parameter is a array of plain string

                String addressLine = ( (String[])
value)[counter].trim();
                StringTokenizer tokenizer =
                    new StringTokenizer(addressLine, ",;");
                while (tokenizer.hasMoreTokens()) {
                    address = tokenizer.nextToken().trim();
                }
            }

            try {
                if (type.equals(TO_SUFFIX)) {
                 msg.addRecipients(Message.RecipientType.TO, address);
                } else if (type.equals(CC_SUFFIX)) {
                 msg.addRecipients(Message.RecipientType.CC, address);
                } else if (type.equals(BCC_SUFFIX)) {
                msg.addRecipients(Message.RecipientType.BCC, address);
                }
            } catch (MessagingException mse) {
```

```
                throw new RTXException(mse);
            }
        }
    }

    private void setSubject(
        Map operationContext,
        Map sessionContext,
        MimeMessage msg, int counter) throws RTXException {
        Object sub = operationContext.get(id + SUBJECT_SUFFIX);
        if (sub != null) {
            try {
                if (sub instanceof String) {
                    msg.setSubject( (String) sub);
                }
                if (sub instanceof String[]) {
                    msg.setSubject( ( (String[]) sub)[counter]);
                }
            } catch (MessagingException ex) {
            }
        }
    }

    private DataSource handleAttachment(
        Map operationContext,
        Map sessionContext,
        ArrayList tempFiles,
        String attachName,
        MimeMultipart mp, int counter) throws MessagingException {
        Object input = operationContext.get(attachName);
        System.out.println("-----" + input);
        if (input != null) {
            try {
                InputStream inStream = null;
                FileOutputStream outStream = null;
                File tempFile = null;
                String name = null;
                if (input instanceof RTXBLOBFileData) {

                    RTXBLOBFileData fileData = (RTXBLOBFileData)
input;
                    name = fileData.name;
                    if (fileData.length != 0) {
                        if (debug) {
                            System.out.println("Found Attachment");
                            System.out.println("Name: " + name);
                        }

                        inStream = fileData.inputStream;
                    }

                }

                if (input instanceof RTXBLOBFileData[]) {

                    RTXBLOBFileData fileData = ( (RTXBLOBFileData[])
input)[
                        counter];
                    name = fileData.name;
                    if (fileData.length != 0) {
                        if (debug) {
                            System.out.println("Found Attachment");
                            System.out.println("Name: " + name);
                        }

                        inStream = fileData.inputStream;
                    }
```

```
                    }

                    else if (input instanceof String) {
                        String relPath = (String) input;

                        if (relPath.length() != 0) {
                            try {
                                inStream =

mgr.getResourceLocator().getInputStream(
                                        "..//" + relPath);
                            } catch (IOException ioe) {
                                System.out.println(
                                    "Exception for resource locator");
                                ioe.printStackTrace();
                            }
                            name =
                                relPath.substring(
                                relPath.lastIndexOf('/') + 1,
                                relPath.length());
                            if (debug) {
                                System.out.println("Found Attachment");
                                System.out.println("Name: " + name);
                                System.out.println("RelPath: " + relPath);
                            }

                        }
                    } else if (input instanceof String[]) {
                        String relPath = ( (String[]) input)[counter];
                        System.out.println(relPath);
                        if (relPath.length() != 0) {
                            try {
                                inStream =

mgr.getResourceLocator().getInputStream(
                                        "..//" + relPath);
                            } catch (IOException ioe) {
                                System.out.println(
                                    "Exception for resource locator");
                                ioe.printStackTrace();
                            }
                            name =
                                relPath.substring(
                                relPath.lastIndexOf('/') + 1,
                                relPath.length());
                            if (debug) {
                                System.out.println("Found Attachment");
                                System.out.println("Name: " + name);
                                System.out.println("RelPath: " + relPath);
                            }

                        }
                    }

                    if (inStream != null) {
                        // serializza

                        try {
                            // crea il file temporaneo
                            tempFile = File.createTempFile("WebML", null);
                            tempFile.deleteOnExit();
                            // GARANZIA DI CANCELLAZIONE
                            tempFiles.add(tempFile);

                            outStream = new FileOutputStream(tempFile);
                            byte bytes[] = new byte[8192];
```

```
                        int flength = 0;
                        do {
                            outStream.write(bytes, 0, flength);
                            flength = inStream.read(bytes, 0,
bytes.length);
                        } while (flength != -1);

                    } catch (FileNotFoundException fnfe) {
                    } finally {
                        if (inStream != null) {
                            inStream.close();
                        }
                        if (outStream != null) {
                            outStream.flush();
                            outStream.close();
                        }
                    }

                    if (tempFile != null) {
                        FileDataSource fds = new
FileDataSource(tempFile);
                        MimeBodyPart bp = new MimeBodyPart();
                        bp.setDataHandler(new DataHandler(fds));
                        bp.setFileName(name);
                        mp.addBodyPart(bp);
                        System.out.println("DataSource ");
                        return fds;
                    }
                }
            } catch (IOException ioe) {
                ioe.printStackTrace();
            }
        }
        return null;
    }

    private void setBody(
        Map operationContext,
        Map sessionContext,
        MimeMessage message, int counter, ArrayList tempFiles) throws
        RTXException {
        // Create your new message part
        BodyPart messageBodyPart = new MimeBodyPart();

        try {
            // Create a related multi-part to combine the parts
            MimeMultipart multipart = new MimeMultipart("related");
            if ( (htmlPath != null) && (htmlPath.length() > 0)) {

                System.out.println("Html format");

                InputStream in =
mgr.getResourceLocator().getInputStream(
                    "descr" + File.separatorChar + htmlPath);

                Document document = loadDocument(in);

                //replace the holes
                replaceHoles(operationContext, document, counter);

         //chiamo replaceImages che sostituisce i tag con cid:name...
                replaceImages(operationContext, document);

         // Set the content of the body part
            messageBodyPart.setContent(print(document), "text/html");

                // Add body part to multipart
```

```
                    multipart.addBodyPart(messageBodyPart);

                    setImage(operationContext, multipart);

                    for (int i = 0; i < attachementNumber; i++) {
                        String name = getID() + ".att" + (i + 1);

                        System.out.println(name);
                        handleAttachment(
                            operationContext,
                            sessionContext,
                            tempFiles,
                            name,
                            multipart, counter);
                    }
                } else {
                    Object bod = operationContext.get(id + BODY_SUFFIX);
                    if (bod instanceof String) {
                        messageBodyPart.setContent( ( (String) bod),
                                                    "text/plain");
                        // Add body part to multipart
                        multipart.addBodyPart(messageBodyPart);

                        for (int i = 0; i < attachementNumber; i++) {
                            String name = getID() + ".att" + (i + 1);

                            System.out.println(name);
                            handleAttachment(
                                operationContext,
                                sessionContext,
                                tempFiles,
                                name,
                                multipart, counter);
                        }
                    }
                    if (bod instanceof String[]) {

                        messageBodyPart.setContent( ( (String[])
bod)[counter],
                                                    "text/plain");

                        // Add body part to multipart
                        multipart.addBodyPart(messageBodyPart);

                        for (int i = 0; i < attachementNumber; i++) {
                            String name = getID() + ".att" + (i + 1);

                            System.out.println(name);
                            handleAttachment(
                                operationContext,
                                sessionContext,
                                tempFiles,
                                name,
                                multipart, counter);
                        }

                    }

                }
                message.setContent(multipart);
            } catch (MessagingException ex) {
                ex.printStackTrace();
            } catch (IOException ex1) {
                ex1.printStackTrace();
            }
        }
```

```
    private void replaceHoles(Map operationContext, Document doc, int
counter) {

        NodeList holeList = doc.getElementsByTagName("HOLE");

        List list = new ArrayList();
        for (int i = 0; i < holeList.getLength(); i++) {
            list.add(holeList.item(i));
        }
        for (int i = 0; i < list.size(); i++) {
            Node node = (Node) list.get(i);
            NamedNodeMap attributes = node.getAttributes();
            Attr attr = (Attr) attributes.getNamedItem("name");

            String name = attr.getValue();

            Object value = operationContext.get(this.id + "." + name);
            if (value instanceof String) {
                //       System.out.println(this.id + "." + name);
                //    System.out.println(operationContext);
                if (value == null) {
                    value = "";
                }
                Node text = doc.createTextNode( (String) value);
                Node parent = node.getParentNode();
                parent.removeChild(node);
                parent.insertBefore(text, null);
            } else if (value instanceof String[]) {
                //  System.out.println(this.id + "." + name);
                //    System.out.println(operationContext);
                if ( ( (String[]) value)[counter] == null) {
                    value = "";
                }
                Node text = doc.createTextNode( ( (String[])
value)[counter]);
                Node parent = node.getParentNode();
                parent.removeChild(node);
                parent.insertBefore(text, null);

            }
        }
        // System.out.println("replaceHoles effettuato
Correttamente");
    }

    private void replaceImages(Map operationContext, Document doc) {
        //    System.out.println("sto entrando nel metodo
replaceImages");
        NodeList list = doc.getElementsByTagName("img");
        //  System.out.println("\n\n\nFind " + list.getLength() + "
images");
        for (int i = 0; i < list.getLength(); i++) {

            Node node = list.item(i);
            NamedNodeMap attributes = node.getAttributes();
            Attr attr = (Attr) attributes.getNamedItem("src");
            String path = attr.getValue();

            //inserisco il path nella mappa passandogli come chiave
image[i];
            //passata come parametro da webratio
            String imageName = "image" + i;
            operationContext.put(imageName, path);

            String attrName = "cid:" + imageName;
            attr.setValue(attrName);
        }
```

```
        System.out.println("fine del metodo replaceImages");
    }

    private Document loadDocument(InputStream in) {

        try {
            //       System.out.println("entro?");
            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = null;
            builder = factory.newDocumentBuilder();

            InputSource is = new InputSource(in);
            Document doc = builder.parse(is);
            // System.out.println("\n\n\nDocumento creato con
successo");
            return doc;

        } catch (IOException ex) {
            ex.printStackTrace();
            //System.out.println("Documento non creato");
        } catch (SAXException ex) {
            ex.printStackTrace();
            // System.out.println("Documento non creato");
        } catch (ParserConfigurationException ex) {
            ex.printStackTrace();
            //System.out.println("Documento non creato");
        }
        return null;
    }

    private void setImage(Map operationContext, MimeMultipart
multipart) {
        try {
            for (int i = 0; ; i++) {
                // Create part for the image
                MimeBodyPart messageBodyPart = new MimeBodyPart();

                // Fetch the image and associate to part
                String path = (String) operationContext.remove("image"
+ i);
                if (path == null) {
                    return;
                }
                DataSource fds = new FileDataSource(path);
                messageBodyPart.setDataHandler(new DataHandler(fds));
                // Add a header to connect to the HTML
                messageBodyPart.setHeader("Content-ID", "<image" + i +
">");

                // Add part to multi-part
                multipart.addBodyPart(messageBodyPart);
            }
        } catch (MessagingException ex) {
            ex.printStackTrace();
        }
    }

    private void send(MimeMessage msg) throws RTXException {
        try {

            Transport.send(msg);
            System.out.println("Ok, messaggio inviato");
        } catch (MessagingException mse) {
            mse.printStackTrace();
            throw new RTXException(mse);
```

```
        }
    }

    private String print(Document doc) {

        StringWriter stringWriter = null;
        try {
            // Create dom source for the document
            DOMSource domSource = new
DOMSource(doc.getDocumentElement());

            // Create a string writer
            stringWriter = new StringWriter();

            // Create the result stream for the transform
            StreamResult result = new StreamResult(stringWriter);

            // Create a Transformer to serialize the document
            TransformerFactory tFactory =
TransformerFactory.newInstance();
            Transformer transformer = tFactory.newTransformer();
            transformer.setOutputProperty("indent", "yes");

            // Transform the document to the result stream

            transformer.transform(domSource, result);

            return stringWriter.toString();
        } catch (TransformerException ex) {
            ex.getException().printStackTrace();
            ex.printStackTrace();
        } catch (IllegalArgumentException ex) {
            ex.printStackTrace();
        } catch (TransformerFactoryConfigurationError ex) {
            ex.printStackTrace();
        }

        return null;
    }

    public void dispose() {
    }
}
```

APPENDICE B

Manuale Utente
Multi Mail Unit

WebRatio 3.3

Davide Taibi

10 Ottobre 2003

Introduzione

MultiMailUnti è un plugin per WebRatio con funzionalità avanzate di invio email

E' possibile inviare messaggi in formato testo ed in formato html

Si possono inviare messaggi singoli tramite una data-unit o inviare contemporaneamente più messaggi a destinatari diversi tramite una multi-data unit.

E' possibile inviare mail personalizzate in formato html.

Installazione

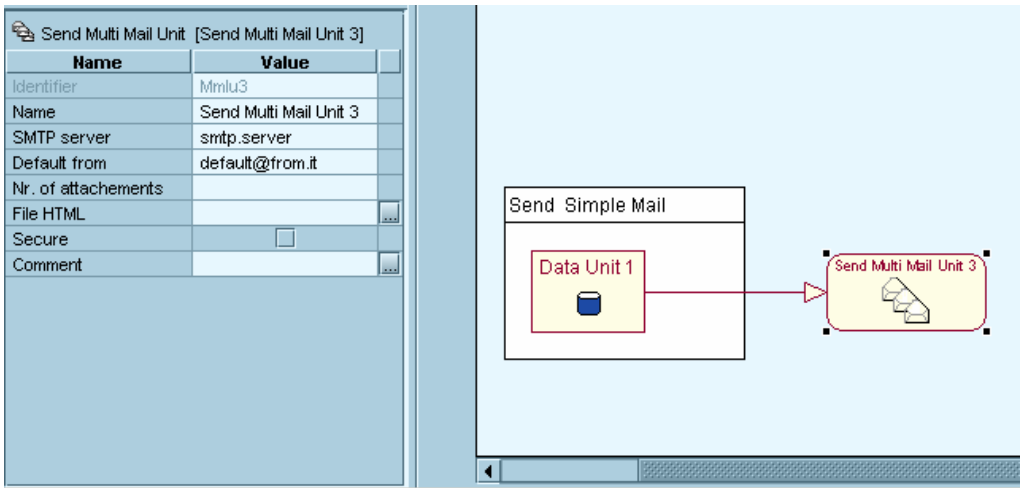
Copiare la cartella multiMailUnit nella directory ..\WebRatio-3.3\SDS\conf\units\extended-units

Copiare la cartella mail nella directory class del progetto corrente
(es ..\WebRatio-3.3\tomcat\webapps\condivisionedisegni\WEB-INF\classes)

Copiare il file saxon.jar, mail.jar e activation.jar nella directory lib sostituendone le vecchie copie
(es ..\WebRatio-3.3\tomcat\webapps\condivisionedisegni\WEB-INF\lib)

Utilizzo

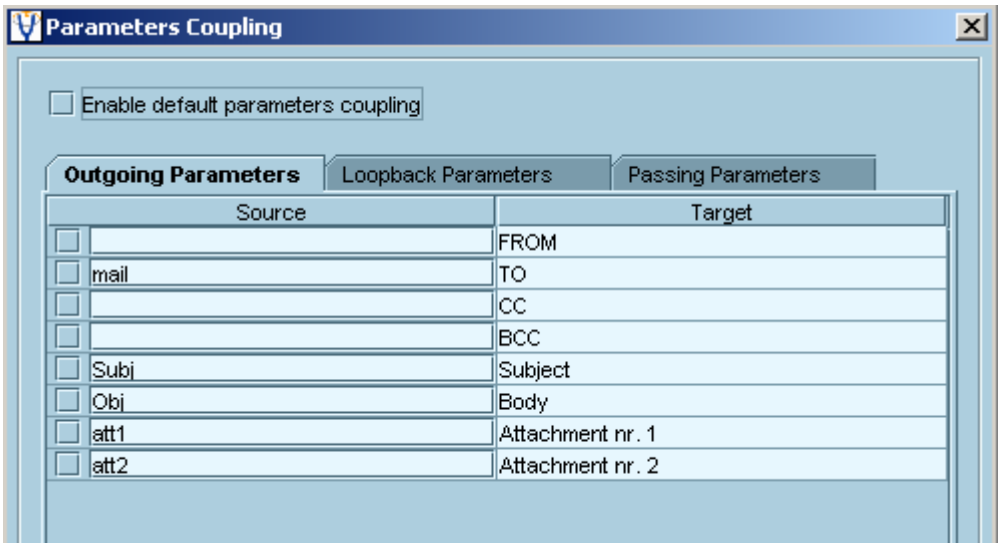
Creare una data-unit (per inviare un solo messaggio o , una multi-data-unit per inviarne più di uno contemporaneamente) consistente all'interno di una pagina, creare una Multi Mail Unit esternamente alla pagina creata, creare un link dalla data-unit alla Multi Mail Unit.



Invio in formato testuale:

Inserire il server smtp di posta in uscita, il mittente di default e, in caso di allegati, inserirne il numero. Lasciare vuoto il campo File–HTML

Selezionare il link, quindi aprire la finestra di coupling
Apparirà un numero di attributi pari a quanti specificati nel campo nr. Of Attachments.



Accoppiare correttamente i campi contenuti nella data-unit a quelli richiesti.

Invio in formato HTML:

Per inviare uno o più messaggi in formato-html, è necessario creare precedentemente un unico file in formato xhtml.
Non è possibile inviare messaggi con frame.
E'possibile inviare un messaggio uguale per tutti gli utenti o creare i messaggi inserendo dei campi personalizzati.

Inserimento di campi personalizzati:

E' possibile inserire campi personalizzati in una mail creando il file xhtml correttamente: si possono inserire campi personalizzati in qualunque punto del file
L'inserimento va effettuato inserendo un tag <HOLE name = " "> specificando il nome dell'attributo.

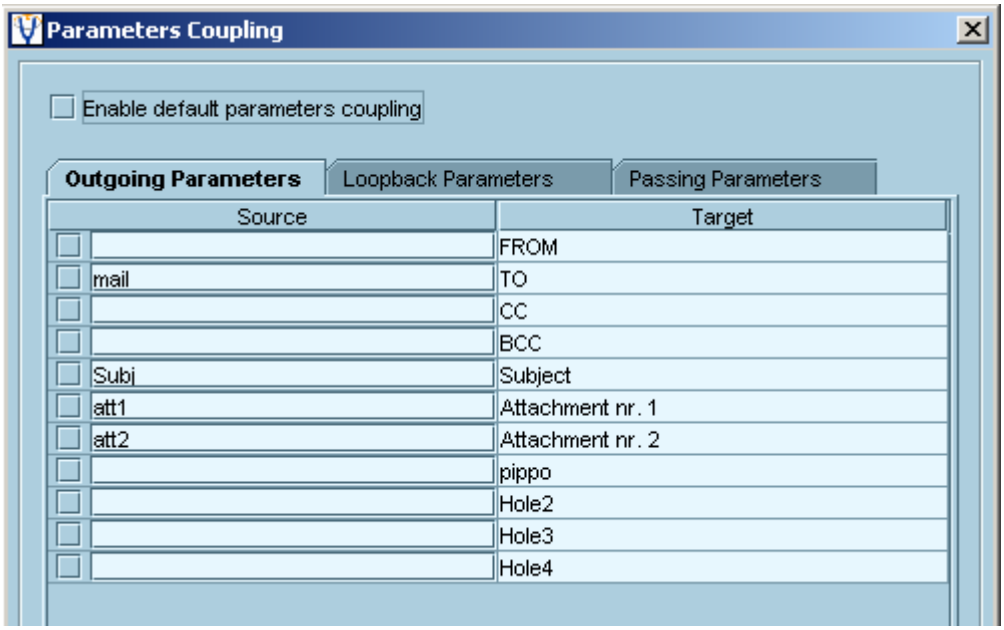
Esempio di creazione di una pagina con quattro campi personalizzati

```
<html>
<head>

</head>

<body bgcolor="#0033FF">

<p><HOLE name="pippo"/></p>
<p><HOLE name="Hole2"/></p>
<p><HOLE name="Hole3"/></p>
<p><HOLE name="Hole4"/></p>
<p>Fine della pagina</p>
<p></p>

<p><img src="D:\image1.jpg" /> </p>
<p></p>
<p><img src="D:\image2.jpg" /></p>

</body>
</html>
```

Selezionare la multi mail unit creata, inserire il server smtp di posta in uscita, il mittente di default , incaso di allegati, inserirne il numero e selezionare il file HTML

Selezionare il link che collega la data-unit alla multi mail unit, quindi aprire la finestra di coupling.

Apparirà un numero di attributi pari a quanti specificati nel campo nr. Of Attachments così come per ogni tag <HOLE> inserito apparirà un campo a cui associare un valore presente nel database.



Per terminare, accoppiare correttamente i campi contenuti nella data-unit a quelli richiesti.

BIBLIOGRAFIA

[1] P.Pialorsi, "XML Il nuovo linguaggio del web", Mondatori Informatica

[2] WebRatio Team "WEBRATIO DOCUMENTATION" www.webratio.com

[3] WebML "WEBML DOCUMENTATION" www.WebML.org

[4] S.Ceri, P.Fraternali, A.Bongio, M.Brambilla, S.Comai, M.Matera "Progettazione di DATI E APPLICAZIONI PER IL WEB", McGrawHill 2003

[5] http://jakarta.apache.org/struts, 2002

[6] SUN, http://java.sun.com/J2EE, 2003

INDICE DELLE FIGURE