Università di Pisa

Facoltà di Ingegneria

Corso di Laurea Specialistica

in Ingegneria Informatica per la Gestione d'Azienda

Tesi di Laurea

Supporto della piattaforma J2EE nella realizzazione di servizi web ed applicazioni nell'ambito della Pubblica Amministrazione.

Relatori: Prof. Beatrice Lazzerini
Prof. Francesco Marcelloni
Ing. Mario Giovanni C. A. Cimino
Dott. Leonardo Paolino

Candidato: Pasquale Idà

Anno accademico
2006/2007

Indice

A. Obiettivo del lavoro e descrizione dei capitoli successivi

Il presente lavoro di tesi rappresenta uno studio dell'impiego della piattaforma Java 2 Enterprise Edition (J2EE) per la progettazione e realizzazione di applicazioni Web-Oriented nel contesto della Pubblica Amministrazione. In particolare, vengono prese in esame le principali soluzioni applicabili sulla piattaforma, ponendo particolare attenzione a due casi di studio, relativi a sistemi informativi commerciali in effettivo esercizio.

Il lavoro si è articolato in differenti fasi.

La prima fase ha comportato l'analisi del contesto, la definizione del cosiddetto *universo del discorso* e lo studio delle problematiche che tali software sono chiamati a risolvere. In tale contesto, si pone l'accento sul fatto che la Pubblica Amministrazione (nel seguito *PA*) sin dagli inizi del 2000 si sia avvalsa di software con licenza di tipo Open Source (nel seguito *OS*).

Viene mostrato nel dettaglio lo sviluppo di due software commissionati dalla PA, il primo è noto con l'acronimo *BDMC* (Banca Dati delle Misure Cautelari Personali), mentre il secondo è denominato *TurismoPublisher*.

BDMC è un sistema informatico che gestisce le misure cautelari riguardanti le posizioni giuridiche degli indagati e/o degli imputati. Questo supporto informatico è utile sia alla magistratura sia al personale amministrativo degli Uffici Giudiziari.

TurismoPublisher, invece, è uno strumento rivolto a tutte quelle figure (persone o servizi) che esplicano le loro mansioni nell'ambito turistico regionale toscano, perché permette la ricerca delle varie strutture (alberghi, residence,…) presenti sul territorio e consente di ottenerne le relative descrizioni.

Nella seconda fase sono state studiate varie soluzioni per la realizzazione dei software sopra elencati. A tale scopo sono introdotti i concetti fondanti della struttura tecnologica J2EE ed i vari modi per poterla mettere in pratica. In particolare si approfondiscono le due soluzioni adottate, la prima è quella degli Enterprise

JavaBeans (*EJB*), organizzati secondo il pattern Model-View-Controller (nel seguito *MVC*), mentre la seconda è quella dei Web Services.

La terza fase mostra come le due applicazioni siano state vestite con la piattaforma J2EE; nello specifico come BDMC integri il modello MVC in un'architettura *three-tiered* e come invece TurismoPublisher sia stato realizzato tramite un Web Service. In tale sezione del lavoro si mostra come dallo studio dei requisiti si arrivi alla scelta delle metodologie e degli strumenti di sviluppo e manutenzione.

L'ultima parte, infine, contiene alcuni frammenti di codice, mostra alcune soluzioni particolari adottate in sede di implementazione ed illustra le modalità di esecuzione della fase di test.

B. Introduzione al Problema

1 La svolta Open source della PA europea

In informatica, *Open Source* (termine inglese che significa *sorgente aperto*) indica un software rilasciato con un tipo di licenza per la quale il codice sorgente è lasciato alla disponibilità di eventuali sviluppatori, in modo che con la spontanea collaborazione (in genere libera) il prodotto possa raggiungere una qualità maggiore di quella raggiungibile attraverso un singolo gruppo di programmazione gestito dai proprietari del codice. L'Open Source ha ovviamente tratto grande beneficio da Internet.

La Comunità Europea (CE) come primo segnale concreto ha richiesto all'Unisys Belgium[1] la realizzazione di un portale detto *OSOR*, acronimo di Open Source Observatory and Repository con l'obiettivo di dare un forte impulso alle applicazioni di tipo eGovernment. OSOR non rappresenta uno stop obbligato per le PA europee, in quanto la CE vuole incoraggiare l'utilizzo e lo sviluppo del software Open Source nonché la realizzazione di progetti comuni. Sono state individuate alcune aree di intervento in questo senso quali l'eProcurement (ovvero procedure e sistemi per l'acquisizione di beni e servizi per la PA), l'interoperabilità e formati di documenti di identità elettronica.

Un effetto riscontrato nel breve termine dell'iniziativa è quello della riduzione dei costi: le singole amministrazioni, potendo condividere le proprie competenze, hanno potuto ridurre gli investimenti richiesti sia in fase di ricerca che di sviluppo delle piattaforme applicative.

"Dentro" OSOR trovano posto, oltre al codice sorgente e al codice oggetto, tutte le informazioni sull'uso delle applicazioni, sulle diverse versioni realizzate, sulle relative licenze *open source* e sui contratti.

A tal proposito, Karel De Vriendt, direttore capo dell'European eGovernment Services Unit dell'Unione Europea, chiarisce che l'interesse delle pubbliche amministrazioni nei confronti dell'open source non nasce con il proposito di

[1]Società leader nella fornitura di servizi e soluzioni di Information Technology.

sostituire gradualmente le soluzioni proprietarie, ma si fonda sull'esigenza di realizzare applicazioni personalizzate basate su software aperto e licenze libere, risultato di una stretta collaborazione progettuale. L'iniziativa OSOR dovrebbe diventare uno strumento privilegiato per rendere più veloce e semplice la condivisione dei software tra gli Stati membri.

1.1 eGovernment in Italia

Per *eGovernment* si intende il processo di informatizzazione della pubblica amministrazione, il quale unitamente ad azioni di cambiamento organizzativo consente di trattare la documentazione e di gestire i procedimenti con sistemi digitali, grazie all'uso delle tecnologie dell'informazione e della comunicazione (ICT), allo scopo di ottimizzare il lavoro degli enti e di offrire agli utenti (cittadini ed imprese) sia servizi più rapidi che nuovi servizi, attraverso ad esempio i siti web delle amministrazioni interessate.

Il termine inglese eGovernment deriva da "government", che può significare sia "governo" che "amministrazione", mentre il prefisso "e" sta per "electronic" e viene utilizzato per designare determinate attività svolte con l'ausilio di Internet. La traduzione più fedele di eGovernment, pertanto, sarebbe amministrazione elettronica, anziché governo elettronico anche se normalmente è preferibile non tradurre affatto i neologismi legati ad ambiti tecnologici.

L'introduzione della tecnologia, generalmente, porta benefici concreti all'attività amministrativa, quali l'interoperabilità, la deburocratizzazione delle procedure, la riduzione degli sprechi (quindi maggiore efficienza).

Oggi esistono programmi di eGovernment, verso le piccole e medie imprese, che modernizzano le procedure degli adempimenti riducendo i tempi della burocrazia. In questo modo è stato realizzato un canale preferenziale di contatto tra lo Stato e il cittadino attraverso Internet. Ad esempio la compilazione della dichiarazione dei redditi (Unico, Iva, 770) avviene con invio telematico attraverso l'utilizzo di particolari software; i servizi online disponibili per le piccole e medie imprese sono diversi, possiamo ricordare ad esempio quelli riguardanti:

- l'apertura e l'esercizio d'impresa;
- i dati sulla fiscalità;

- gli sportelli unici per le imprese;
- le risorse umane.

L'eGovernment per le imprese si indica tramite il cosiddetto *G2B*, ovvero il Government To Business, una stretta connessione tra imprese e Pubblica Amministrazione; una connessione che riguarda gli aspetti fiscali, economici, legislativi, ambientali che toccano da vicino le attività produttive.

La Pubblica Amministrazione si avvicina al mondo delle imprese anche tramite un portale nazionale per le imprese: *impresa.gov.it* che riunisce tutti i servizi online forniti dalla PA per il mondo delle imprese e per il commercio.

Il Portale per i Servizi integrati alle imprese si presenta come uno sportello telematico, tramite il quale è possibile compilare la modulistica per svolgere i diversi obblighi amministrativi, senza doversi recare agli sportelli tradizionali. Dopo aver portato a termine la richiesta di certificazione o documentazione, si ottiene una ricevuta di avvenuta consegna con il corrispondente numero di protocollo, da utilizzare per controllare lo stato di avanzamento della pratica.

Attualmente lo sviluppo dell'eGovernment, sia a livello centrale, per quanto riguarda i servizi tributari, sia a livello locale, nel supporto allo sviluppo delle imprese, è in continua crescita.

2 Banca Dati delle Misure Cautelari Personali (BDMC)

2.1 Normative e iniziative

Tra i numerosi settori dove la rete Internet sta assumendo il ruolo di infrastruttura primaria di interconnessione, quello che sta producendo gli effetti maggiormente innovativi è l'eGovernment, cioè la possibilità per tutti gli utenti, cittadini e imprenditori, di gestire il rapporto con la Pubblica Amministrazione anche on-line. È in atto una vera e propria rivoluzione che sta trasformando i siti internet pubblici da vetrine istituzionali a strumenti per interagire in maniera diretta con lo Stato.

Questa rivoluzione vedrà la piena esplosione quando, attraverso la firma digitale, i cittadini potranno accedere ai siti Internet pubblici non solo come anonimi

visitatori, ma come persone legalmente identificate, consentendo perciò l'attivazione di una miriade di nuovi servizi[1].

Si deve sottolineare però che negli ultimi anni la Pubblica Amministrazione italiana ha vissuto un'intensa stagione di riforme normative e di modernizzazione ed ha cercato di recuperare il ritardo sia infrastrutturale che culturale che la caratterizzava.

La Rete sta diventando un mezzo importante sia per accrescere la produttività del lavoro all'interno degli uffici pubblici, sia per migliorare la qualità dei servizi che essi devono offrire ai cittadini. Certo sono ancora molti i problemi da risolvere a partire dalla firma digitale fino ad arrivare all'interoperabilità telematica tra tutte le Amministrazioni pubbliche. Gli sforzi sono comunque evidenti e, con le ovvie differenze, i nostri servizi pubblici hanno oggi in Rete una presenza paragonabile a quella dei Paesi europei più avanzati tecnologicamente.

Il Governo Italiano ha emanato da diversi anni le "*Linee Guida per l'organizzazione, l'usabilità e l'accessibilità dei siti web delle pubbliche amministrazioni*"[2] il cui obiettivo è di fornire a chiunque, all'interno delle amministrazioni, si occupi di progettazione, realizzazione e manutenzione di sistemi informativi basati sulle tecnologie del Web, indicazioni sugli aspetti più importanti che riguardano le reali fruizioni dei siti Web nelle amministrazioni pubbliche, con particolare riferimento al contesto organizzativo, all'usabilità del Web, all'accessibilità delle informazioni.

Questa direttiva è nata nell'ambito del Piano di Azione di eGovernment che rappresenta la prima proposta del Governo Italiano per il sostegno ai processi di innovazione realizzati dalle pubbliche amministrazioni mediante l'utilizzo delle nuove tecnologie dell'informazione e della comunicazione.

Il progetto Banca Dati delle Misure Cautelari (BDMC) Personali nasce da questi presupposti nel 2002 ed oggi è in funzione in diverse città italiane.

2.2 Introduzione a BDMC

BDMC è un sistema informatico che contiene e consente di gestire le misure cautelari riguardanti le posizioni giuridiche degli indagati e/o degli imputati.

Questo supporto informatico è utile sia alla magistratura che al personale amministrativo degli Uffici giudiziari.

La Banca Dati delle Misure Cautelari è nata per dare completa esecuzione sia alla normativa che dispone la comunicazione dell'applicazione di una qualsiasi misura cautelare al servizio informatico all'uopo istituito da un decreto del ministro della giustizia, sia alla normativa sui Registri penali informatici.

Questo sistema informativo/informatico si distingue per aver portato nel mondo dell'informatica giudiziaria penale la novità della gestione delle Misure Cautelari, gestione intesa dal punto di vista sia del personale di magistratura che del personale amministrativo.

C'è sempre stata molta attenzione da parte degli Uffici Giudiziari sul controllo delle posizioni giuridiche degli indagati e/o degli imputati; in questo senso si pensi alle numerose circolari emesse dal ministero, dai presidenti delle Corti di Appello, dai presidenti dei Tribunali, tutte raccomandanti la massima cura nella trattazione delle misure cautelari. Sono stati molteplici i tentativi di gestire gli scadenziari delle misure stesse attraverso la creazione di schede personali usate come promemoria, gestire gli scadenziari attraverso la creazione di fogli elettronici realizzati in casa da qualche utente della giustizia più esperto. Pari discorso può essere fatto per il Tribunale del Riesame, dove sembravano insormontabili i problemi collegati alla conoscenza immediata dei provvedimenti, nonché la gestione del registro di questo Tribunale.

Da questa quantità di tentativi di gestire realtà così strutturate, è nata la volontà di creare un supporto valido ed efficace sia per il personale di magistratura che per il personale amministrativo, uno strumento che potesse accompagnare l'attività dell'Ufficio Giudiziario, rendendo meno difficoltosa la gestione delle misure cautelari.

BDMC è di supporto al personale di magistratura con lo scadenziario poiché consente di avere sempre sotto controllo la posizione dei soggetti sottoposti a misure cautelari dal proprio ufficio. Ciò implica che il personale amministrativo abbia le notizie sufficienti e necessarie per l'aggiornamento della banca dati, affinché questo sistema possa essere al massimo efficace nella funzione di controllo che gli è stata demandata.

La Banca Dati è di supporto al personale della magistratura con lo scadenziario perché consente di avere un immediato riscontro sull'attività del Tribunale del Riesame. Per la prima volta si è sviluppato un sistema che prevede la completa integrazione con il Tribunale del Riesame, il quale lavorando sul suo registro informatico, ora uguale per tutti e ufficiale, e annotando gli esiti di riesami ed appelli sulle ordinanze del giudice, aggiorna in tempo reale anche la singola posizione del soggetto sottoposto a misura cautelare.

La presenza di una serie di “allarmi” consente di evidenziare i termini delle misure cautelari, in quanto è presente una pagina iniziale di apertura del sistema che mostra l'elenco delle posizioni "allarmate", che necessitano di un controllo da parte del giudice.

Vi è un vasto sistema di ricerche che consente di poter verificare, in tempo reale, se vi siano posizioni che “*rischiano*” perché vi sono degli adempimenti da fare con scadenze ravvicinate. Si pensi, ad esempio, alla fissazione dell'interrogatorio di un soggetto sottoposto alla custodia cautelare in carcere, da effettuarsi entro cinque giorni dall'esecuzione della misura stessa, come pure i due giorni per l'ordinanza di convalida, o i venti giorni per la conferma della misura cautelare.

Notevoli sono anche i vantaggi, nella fase dell'emissione e/o della richiesta della misura cautelare, forniti dalla consultazione della base dati nazionale. Si pensi alla possibilità di verificare se un soggetto abbia altre misure cautelari in procedimenti penali, in qualsiasi ufficio giudiziario d'Italia, verificando, altresì, anche i capi di imputazione delle eventuali misure riscontrate. Anche per il giudice si hanno altrettanti vantaggi, quali ad esempio avere un quadro completo delle imputazioni richieste per ogni singolo soggetto dal Pubblico Ministero, consentendo un’analisi completa anche nei casi di “mega ordinanze” di applicazione delle misure cautelari.

Inoltre BDMC è di supporto al personale amministrativo, perché fornisce uno strumento che interagisce con i servizi di cancelleria in maniera continua e puntuale.

Nella pagina iniziale è presente un cruscotto del sistema che fornisce gli elenchi di tutti i fascicoli che devono essere oggetto di controllo, come, ad esempio, quelli con allarmi sulla scadenza dei termini, o quelli da prendere in carico perché trasmessi da altri uffici o dalla procura.

Ad ogni compito che deve essere eseguito, sono stati collegati gli stampati per quella medesima funzione, così da avere delle notevoli economie di scala sul lavoro della cancelleria stessa, come ad esempio tutti i moduli che riguardano la fissazione degli interrogatori o delle udienze di convalida, i vari tipi di verbali, gli ordini di traduzione ecc. La maggior cura nell'annotazione delle informazioni che riguardano il soggetto sottoposto a misura cautelare porta a vantaggi quasi immediati nella gestione quotidiana delle posizioni giuridiche dei soggetti. Ulteriore prova ne è il fatto che in molti tribunali si erano sviluppati dei sistemi informatici "artigianali" che consentissero questa gestione a livelli molto più bassi del sistema BDMC e con un'opera di input dei dati molto più onerosa rispetto, sempre, al sistema BDMC.

BDMC consente al personale dei vari uffici, come quelli degli inquirenti, dei giudicanti, degli appellanti ed altri ancora, di avere il primo registro informatico ufficiale, che sostituisce a tutti gli effetti quello cartaceo il quale era sino a poco tempo fa l'unico vigente. Consente, inoltre, al personale di quest'ultimo ufficio di avere un sistema inserito a livello distrettuale con gli altri uffici, in cui sono immediatamente visibili tutte le informazioni sui soggetti per i quali è stata eseguita una misura cautelare personale. Lo stesso vantaggio, come già detto, si ha anche per il personale che lavora con i giudici e con la Procura che ha la visibilità immediata sui provvedimenti del Tribunale del Riesame.

Si è tentato, per ogni funzione, di garantire una certa navigabilità così da rendere chiaro, in qualsiasi fase di utilizzo, l'avanzamento delle operazioni che si stanno compiendo.

Il sistema, nella versione ultima ed a pieno regime, permette al personale, di estrarre statistiche molto più precise e puntuali, al fine di rispondere a quanto richiesto in sede ministeriale, in vista delle periodiche ispezioni; ad esempio grazie ad una funzione di valutazione dei carichi di lavoro, consente una migliore valutazione della allocazione del personale amministrativo.

Le funzioni di ricerca consentono anche al personale amministrativo di avere il controllo dei servizi di cancelleria connessi agli adempimenti sulle misure cautelari, soprattutto riguardo alla completezza ed all'esattezza delle informazioni annotate nel sistema. Tutto questo si è ottenuto senza abbassare il livello di sicurezza nell'accesso alle informazioni. Infatti, la gestione delle richieste delle misure, fino

all'esecuzione delle misure stesse, porta ad una segregazione del fascicolo informatico, visibile soltanto a chi lavora a stretto contatto con il giudice ed il PM, escludendo tutto il personale non direttamente interessato.

Questo sistema si fonda sull'interazione dei diversi uffici e, all'interno di questi, sulla massima collaborazione tra personale di cancelleria e personale di magistratura. Quanto più vi sarà un completo concorso di contributi tra tutti gli attori, tanto più il sistema sarà affidabile e privo di falle.

BDMC è stato realizzato tramite un'architettura a tre livelli (vedi paragrafo C.1.1.1), composta da uno strato di presentazione, uno di logica di business ed uno dedicato alla banca dati.

3 TurismoPublisher

3.1 eGovernment per la Regione Toscana

La Regione Toscana ha dato inizio al progetto e.Toscana che prevede un programma straordinario pluriennale degli investimenti strategici stanziati dalla Regione stessa. Raccoglie gli investimenti per lo sviluppo dell'eGovernment.

Per la Regione Toscana, lo sviluppo dell'innovazione tecnologica è percepito come un processo evolutivo continuo che richiede una sempre più attenta politica di indirizzo, coordinamento e verifica a tutti i livelli. Inoltre è visto come condizione abilitante per il cambiamento della Pubblica Amministrazione verso un modello maggiormente orientato ai risultati.

In relazione a questo convincimento, e all'inizio del nuovo ciclo di programmazione regionale 2007-2010 sui temi dello sviluppo della Società dell'Informazione, a livello amministrativo è stato avviato, in collaborazione con le Province toscane, un percorso di confronto con gli enti locali, le associazioni di categoria e le imprese che operano nel settore ICT.

Gli argomenti trattati riguardano la programmazione regionale e locale sui temi della semplificazione amministrativa e dell'eGovernment e per ciascuno degli incontri è stato individuato uno specifico argomento oggetto di una tavola rotonda alla quale hanno partecipato amministratori, esperti, rappresentanti delle università, delle categorie economiche, del mondo produttivo e del consumo.

Questi incontri hanno costituito un momento importante di confronto e con l'occasione è stato fatto il punto sui risultati del precedente ciclo di programmazione e sono state definite le nuove priorità, individuando le leve strategiche di sostegno e di attuazione degli indirizzi politici.

In questo contesto è nato il nuovo progetto sul Turismo che ha consentito di tracciare un secondo scenario all'interno della presente tesi.

3.1.1 Open source per la Regione Toscana

La crescente diffusione della società dell'informazione ed il costante aumento degli utenti connessi alla rete impongono un'approfondita riflessione sulle logiche che regolano l'effettiva possibilità d'accesso alle informazioni, nonché sul livello di concorrenza presente nel mercato delle Tecnologie dell'Informazione e della Comunicazione (settore internazionalmente noto come ICT) e sui costi (sia in termini di risorse monetarie che di tempo) che la PA è chiamata a sostenere per usufruire dei benefici offerti dalle nuove tecnologie.

In particolare la Regione si propone di attivare un percorso di diffusione delle conoscenze riferite alle applicazioni a "codice sorgente aperto" nella PA, per accrescere le possibilità di riuso e di abbattimento dei costi della produzione e diffusione del software, per valutare e/o confermare se tali applicazioni siano adatte a risolvere in tutto o in parte le criticità esistenti, per sviluppare nuove possibilità di accesso alle applicazioni ed ai servizi da parte degli utenti, e per aumentare il livello di concorrenza.

Per supportare l'iniziativa regionale è stato attivato uno specifico Centro di Competenza di risorse universitarie, nazionali e private, per l'analisi degli impatti, delle possibilità e criticità dell'open source, e per sostenere le azioni legate alla sua promozione nella PA, con particolare riferimento agli strumenti per il lavoro di ufficio, alla diffusione di formati aperti, all'adozione di sistemi operativi non proprietari.

Pertanto la Regione ha previsto interventi per favorire:

- la diffusione delle conoscenze sulle opportunità offerte dal software di tipo Open Source in termini di capacità delle applicazioni, fiducia, riservatezza, scalabilità, interoperabilità delle soluzioni;

- la diffusione di strumenti d'ufficio Open Source e di sistemi operativi non proprietari;
- l'aggiornamento su attività in corso, progetti e applicazioni esistenti e riusabili;
- la valutazione degli aspetti positivi e di criticità dell'Open Source nella fase di progettazione, diffusione ed assistenza dei prodotti;
- la valutazione del costo totale richiesto all'utente per passare da sistemi di tipo "proprietario" a sistemi OS, a seconda del tipo di applicazione considerata.

3.2 TurismoPublisher

La Regione Toscana censisce le varie tipologie di strutture turistiche presenti sul territorio regionale ed il supporto informatico richiesto ha lo scopo di automatizzare la comunicazione tra la regione stessa e le singole province.

Prima della realizzazione di tale supporto informatico, la registrazione delle varie strutture avveniva per mezzo di documenti cartacei o di file XML (vedi paragrafo B.5.1.1) inviati mediante posta elettronica. Tale sistema comportava diversi disagi quali l'accumularsi di tali documenti, l'alta percentuale di errori umani o addirittura la perdita di materiale.

Oltre alla registrazione delle varie strutture, i dipendenti dovevano effettuare delle ricerche e questa procedura veniva effettuata sempre manualmente all'interno di grossi scaffali.

L'introduzione del supporto informatico deve permettere di poter continuare a svolgere tutte quelle attività che venivano sostenute, ma con maggiore efficienza.

Il software commissionato dalla Regione prevede lo sviluppo di due pacchetti indipendenti che debbano utilizzare la stessa base dati. Il primo, detto *TurismoPublisher*, riguarda lo svolgimento dell'attività di ricerca delle strutture turistiche, mentre il secondo, *TurismoSubscriber*, ne consente l'archiviazione all'interno della banca dati.

Tale banca dati è di supporto al personale della regione ed a tutte quelle figure che hanno un rapporto di collaborazione, in quanto consente di avere un immediato riscontro sulla gestione dell'informazione connessa alle strutture turistiche. Per la prima volta si è dato vita ad un sistema in grado di gestire in modo integrato i dati di tali soggetti, di lavorare su di un proprio registro informatico, ora

standardizzato, e di annotare i nuovi dati, aggiornando in tempo reale ogni singola informazione sulla singola struttura.

Inoltre si potranno riscontrare notevoli vantaggi, anche nella fase della ricerca e/o dell'inserimento delle strutture. Si pensi alla possibilità di verificare se una struttura abbia o meno dei requisiti (la provincia di ubicazione, il numero di stelle, il numero di camere, il tariffario e tanti altri).

Il prodotto richiesto dalla regione, come detto in precedenza, si divide in due pacchetti software. TurismoPublisher, permette alla regione di rintracciare le strutture, mentre TurismoSubscriber consente alle province di inserirle o di modificare i dati esistenti.

Il TurismoPublisher è implementato in termini di Web Service e come tale codifica i messaggi di richiesta e risposta in XML (messaggi contenenti i criteri di ricerca delle strutture, tramite i quali interrogare la banca dati) e risponde con un messaggio sempre di tipo XML, contenente le relative strutture che soddisfano i criteri richiesti.

Analogamente TurismoSubscriber, è realizzato tramite Web Service e come tale codifica le informazioni da inserire nella base dati tramite uno schema XML.

4 Infrastruttura J2EE (Java 2 Enterprise Edition)

La versione della piattaforma Java 2 Enterprise Edition[3] che verrà approfondita nel corso della tesi è la 1.4.

J2EE è una piattaforma di sviluppo e di deployment[2] di applicazioni distribuite "multilivello" secondo un paradigma orientato al ciclo di vita delle applicazioni all'interno delle organizzazioni, ossia organismi con una certa complessità strutturale. Pertanto, rispetto alla piattaforma base J2SE (Standard Edition), la quale è orientata allo sviluppo di applicazioni desktop tradizionali, contiene ulteriori livelli di funzionalità di livello superiore.

Con il termine "Enterprise" si vuole quindi identificare una generica organizzazione, che al suo interno possiede varie unità organizzative con diverse attività, come ad esempio la gestione degli ordini di acquisto o di vendita, la gestione del personale, del controllo della produzione, e così via.

Come tale, in termini infrastrutturali l'espressione *applicazione enterprise* è, soprattutto, sinonimo di *applicazione distribuita*, ossia dislocata su vari nodi di una rete informatica aziendale.

4.1 Architettura multilivello

J2EE è fondata su un'architettura multilivello (Figura 1) nella quale le varie componenti che formano l'applicazione sono divise logicamente.

[2] Messa in campo o in atto.

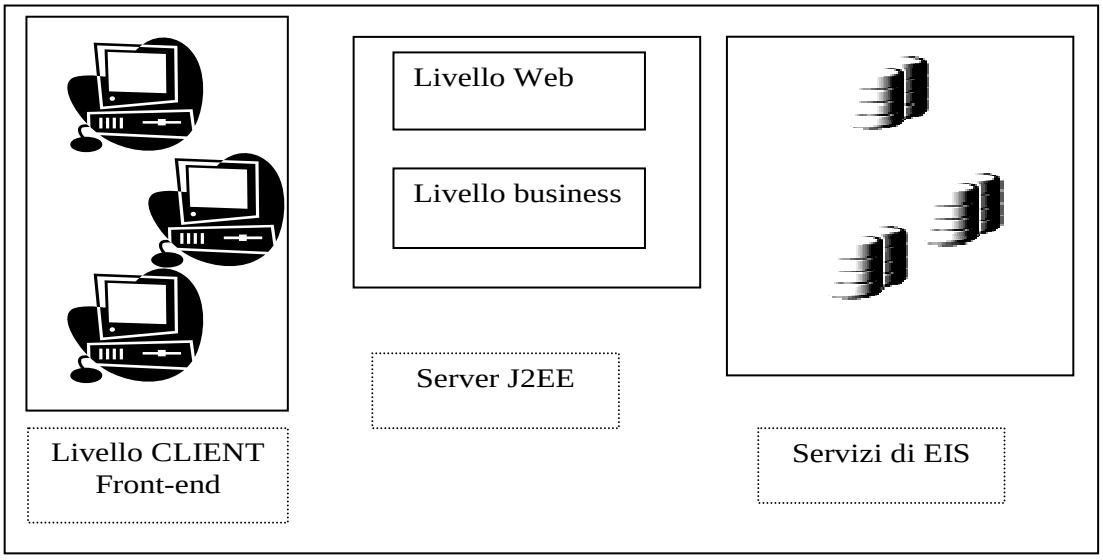


Figura 1 – Architettura d'insieme della piattaforma J2EE.

La piattaforma J2EE può essere utilizzata per sviluppare un client front-end, un middleware con connessioni via Web, connessioni di database back-end, la logica di business di un'organizzazione o ancora EIS (Enterprise Information System).

J2EE definisce quattro livelli distinti, dove i vari elementi sono eseguiti in un *Container* che fornisce uno standard di servizi.

I quattro livelli possono essere così suddivisi:

LIVELLO CLIENT: Contenente applicazioni ed applet orientate al terminale "cliente", ossia che vengono eseguite sul computer dell'utente finale (detto "client"). Queste componenti forniscono una interfaccia grafica utente (detta nel seguito *GUI*) attraverso pagine HTML, script lato client, file Macromedia Flash[3], musicali, video, documenti e altro ancora.

LIVELLO WEB: Contenente gli elementi che hanno lo scopo di dare maggiori funzionalità al sistema che fornisce ai terminali client le componenti da eseguire. Tali funzionalità sono realizzate tramite le Servlet Java (vedi paragrafo B.4.3.1) e le Pagine JSP (vedi paragrafo B.4.3.2). Queste componenti rispondono, generalmente, a richieste di tipo HTTP[4] o HTTPS[5] generate dai vari web client ,restituendo dati in

[3] Formato per animazioni di tipo vettoriale.

[4] Acronimo di HyperText Transfer Protocol (protocollo di trasferimento di un ipertesto), usato come principale sistema per la trasmissione di informazioni sul web.

formato HTML, XML (vedi paragrafo B.5.1.1) o di altro tipo. Questo livello supporta anche i livelli di autenticazione basati sulle tecnologie web.
LIVELLO BUSINESS: Contenente le componenti che forniscono la logica business nella forma di *JavaBean Enterprise* (nel seguito chiamati *EJB*) che vengono eseguiti sul server J2EE. Le componenti EJB operano in un ambiente gestito sul server J2EE che è in grado di supportare le transazioni; con transazione si indica un'unità di lavoro atomica o indivisibile che va a modificare i dati. Queste componenti generalmente elaborano dati, compiono operazioni seguendo le regole di business dell'organizzazione e possono supportare anche servizi Web.
LIVELLO ENTERPRISE INFORMATION SYSTEM (EIS): Contenente le componenti di gestione delle basi di dati e dei servizi di back-end[6] in genere che vengono eseguite sul server di database. In generale non tutti gli elementi J2EE devono accedere al DB, come ad esempio le applet del livello client, le quali appartengono al "front-end" dell'applicazione, ossia ai moduli che interagiscono direttamente con l'utente, e che quindi implementano la GUI.

Ognuno dei quattro livelli può essere allocato fisicamente su host diversi, anche se in generale i livelli business e web vengono istanziati sulla stessa macchina. Nella figura seguente sono mostrati diversi scenari di dislocazione dei livelli dell'architettura su un numero differente di host.

---

[5] È un URI (Uniform Resource Identifier) sintatticamente identico allo schema http:// ma con la differenza che tra il protocollo TCP e HTTP si interpone un livello di crittografia/autenticazione.

[6] Parte che elabora i dati generati tramite l'interazione con l'utente o con i sistemi esterni che producono dati di ingresso.

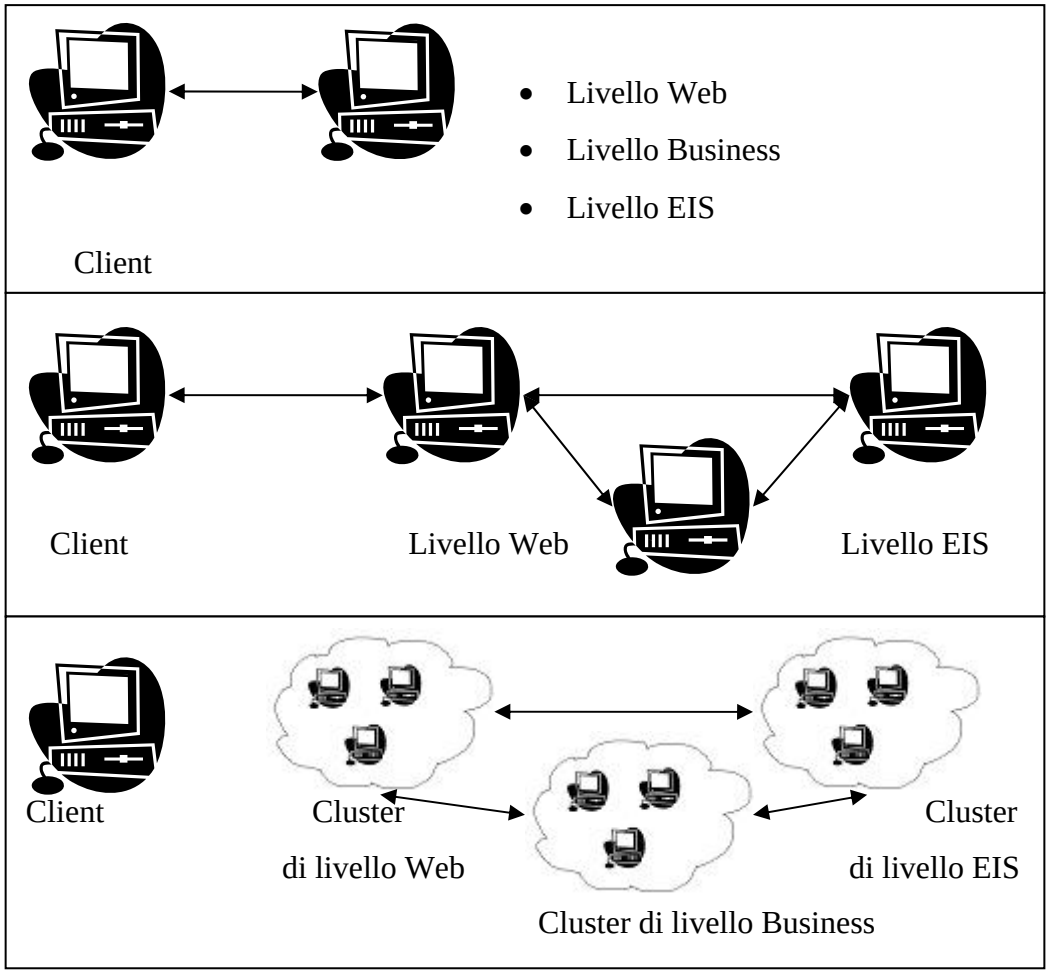


Figura 2 – I livelli Web, business e EIS di J2EE possono essere raggruppati per essere eseguiti sullo stesso computer o possono essere partizionati fisicamente su computer distinti.

4.1.1 Container

Le varie componenti vengono eseguite all'interno di un *Container* che fornisce una visione coerente delle API[7] sottostanti. Il container gestisce le varie componenti e fornisce loro una varietà di servizi di livello di sistema, come la gestione delle transazioni, delle eccezioni, del pooling[8] delle istanze, dei thread[9] e della sicurezza.

[7] Application Programming Interface, sono ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato compito

[8] Tecnica che permette il raggruppamento degli oggetti.

J2EE fornisce un tipo di container corrispondente per ogni classe principale di componente.

4.2 Livello Client

Il livello Client J2EE fornisce l'interfaccia utente all'applicazione mettendo a disposizione una "vista" sull'applicazione J2EE costituita da un insieme di componenti software eseguite nei relativi container.

I programmatori possono realizzare e supportare un'ampia gamma di Client che interagiscono con l'applicazione. La figura seguente ne mostra possibili realizzazioni di questo paradigma.

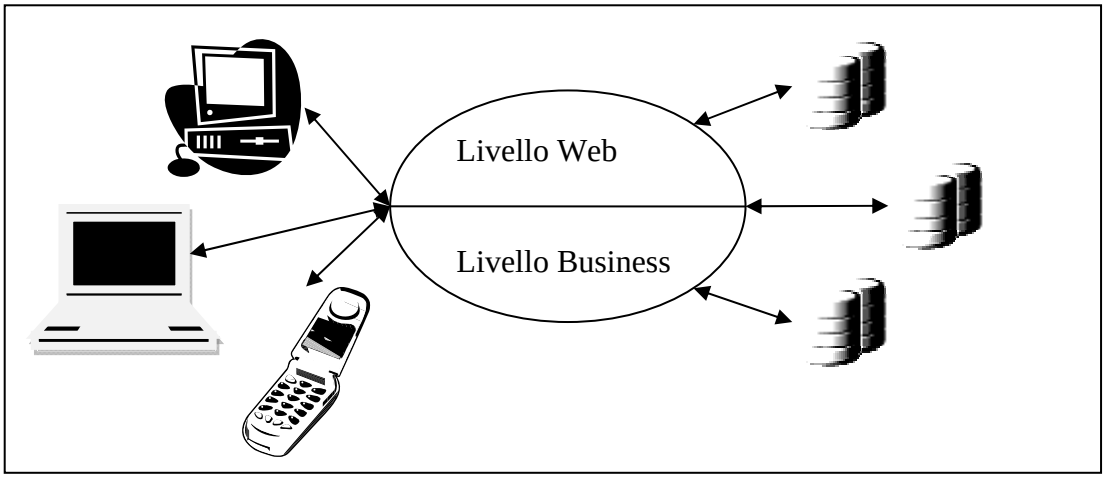


Figura 3 – J2EE supporta una grande varietà di client, attraverso i quali l'utente finale interagisce con le componenti lato server dell'applicazione.

Non è necessario che il client debba supportare una GUI, potrebbe essere anche un terminale di tipo testuale.

I principali tipi di client supportati dalla tecnologia J2EE sono:

- Client;
- Application;
- Client Web;
- Client Applet.

---

[9] Per thread si intende l'unità granulare in cui un processo (istanza di un programma in esecuzione) può essere suddiviso.

4.2.1 CLIENT APPLICATION

I client application sono applicazioni Java standard che vengono eseguite nelle *JVM*[10] ed hanno accesso alle API Java. Vengono eseguiti autonomamente all'esterno dei browser web in quanto sono applicazioni vere e proprie.

Possono accedere direttamente ai livelli business ed EIS, mentre i Client Web vi possono accedere solo utilizzando il livello Web.

É però generalmente preferibile passare attraverso il livello Web al fine di evitare una dipendenza diretta (accoppiamento) tra il client e il sistema di back-end. Un possibile esempio di utilizzo di accesso relativo ai livelli di business o EIS è dato in fase di test o di realizzazione di prototipi.

4.2.2 CLIENT WEB

I Client Web sono costituiti da due parti principali: un browser e l'insieme delle pagine web (pagine html, documenti xml e altri). Le varie pagine visualizzate vengono create dinamicamente dalle componenti Web, tipicamente, attraverso le Servlet JAVA e/o le Pagine JSP.

I Client Web sono preferibili quando gli utenti finali si trovano all'esterno dell'organizzazione, o nel caso in cui non siano facilmente gestibili, o ancora in tutti quei casi in cui l'applicazione debba essere utilizzata da un insieme disparato di utenti.

In generale non dovrebbero interagire con i livelli di Business, ma solo con il livello Web, in modo da poter rimanere indipendenti dalla logica.

4.2.3 CLIENT APPLET

I Client Applet "vivono" all'interno dei browser web, comunicano con il livello Web e non hanno accesso al livello Business.

Se il browser è equipaggiato di una JVM adeguata è possibile far accedere i Client Applet a tutte le funzionalità degli applet J2SE.

---

[10] Java Virtual Machine, è la macchina virtuale che esegue i programmi in linguaggio bytecode, ovvero i prodotti della compilazione dei sorgenti Java.

4.3 Livello Web

Le componenti Web sono entità software che elaborano le richieste fornite dai client per i quali creano e inviano risposte; per esempio i documenti HTML che corrispondono alle pagine web richieste dal client.

Oltre che ad interagire con i client, le componenti Web agiscono come intermediarie tra i Client Applet e i pezzi lato server, eseguite nei livelli di business e EIS.

Le specifiche J2EE definiscono quattro tipi di componenti:

- Servlet Java
- Pagine JSP
- Filtri Web
- Event Listener Web

che "girano" all'interno del container.

Sono frequentemente utilizzate per generare dinamicamente le pagine HTML che costituiscono l'interfaccia GUI dell'applicazione nel lato client, oppure per generare qualsiasi tipo di documento.

4.3.1 SERVLET JAVA

La Servlet Java è una classe java creata utilizzando l'API Java Servlet per estendere i servizi Web basati su HTTP. Il server Web associa una Servlet a una o più URL[11]. Le richieste dai client verso questi indirizzi invocano le Servlet associate.

Così facendo le Servlet intercettano effettivamente le richieste HTTP standard e possono elaborare risposte per i client.

Le Servlet possono anche utilizzare i filtri per "pre" o "post" processare le richieste e le risposte.

A differenza del protocollo HTTP che è di tipo *stateless* le Servlet possono mantenere informazioni sullo stato e sulla sessione per ogni client con il quale

[11] Universal Resource Locator, è una sequenza di caratteri che identifica univocamente l'indirizzo di una risorsa in Internet.

interagiscono. Questo è reso possibile grazie al container della Servlet che mantiene lo stato per la Servlet.

4.3.2 PAGINE JSP

La tecnologia JSP è complementare alle Servlet in quanto sono tecnologie correlate che hanno approcci differenti, ma interoperabili per creare contenuti da presentare sul client.

A differenza delle Servlet, che sono classi, le pagine JSP sono documenti testuali che possono combinare markup e codice Java, e sussistono per creare dinamicamente contenuti destinati ai client.

L'uso delle Pagine JSP favorisce ai non-programmatori un modo relativamente semplice per creare contenuti lato client. Esse sono tradotte in Servlet nel livello Web.

4.3.3 FILTRI WEB

I filtri Web sono oggetti che risiedono all'interno del livello Web, possono dinamicamente trasformare l'Header[12] e il contenuto di una richiesta client entrante o di una risposta uscente create da una Servlet o da una Pagina JSP. Di conseguenza anche i filtri sono eseguiti all'interno dello stesso container delle Servlet o delle JSP.

4.3.4 EVENT LISTENER WEB

Gli Event Listener Web sono classi Java che implementano una o più interfacce *EventListener* in modo da poter ricevere le notificazioni di particolari tipi di eventi orientati alle Servlet. Un esempio è quello del cambiamento nello stato delle sessioni di HTTP o delle richieste delle Servlet.

4.4 Livello business

Nel livello Business risiedono i JavaBean Enterprise e la logica di business è la combinazione di regole e codice mirate alla soluzione di problemi specifici.

---

[12] Con Header si ci riferisce a dati supplementari che contengono informazioni per il trattamento dei dati.

I JavaBean Enterprise vengono eseguiti lato server in un ambiente gestito, detto container EJB che risiede all'interno di un Application Server J2EE[13]. L'Application Server è una piattaforma in grado di fornire l'infrastruttura e le funzionalità di supporto, di sviluppo e di esecuzione di applicazioni e componenti server in un contesto distribuito. Si tratta di un complesso di servizi orientati alla realizzazione di applicazioni per il web, multilivello ed enterprise, con alto grado di complessità. A differenza del Web Server tradizionale gestisce l'intera piattaforma di J2EE. Tra i più diffusi abbiamo l'application Server JBoss[10], di tipo Open Source. È basato su Java ed è multipiattaforma, cioè utilizzabile su qualsiasi sistema operativo che supporti Java.

Le componenti EJB possono essere invocate direttamente da quelle del livello Web o da quelle del livello Client, anche se un uso corretto prevede le chiamate solo dalle prime.

Fondamentalmente gli enterprise bean supportano poche funzionalità chiave considerate essenziali. Possono ricevere dati da elaborare come previsto dalle specifiche dell'applicazione per poi interagire con le componenti del livello EIS in modo che i dati possano essere memorizzati permanentemente in un database.

Per visualizzare il ruolo che gli EJB giocano all'interno di un'applicazione J2EE possiamo fare un esempio basato sull'applicazione BDMC.

Un utente, per controllare i dati relativi ad un fascicolo, deve prima fornire i dati relativi alla sua identità, utilizzando l'interfaccia di front-end fornita dal client. Nel nostro caso abbiamo un browser web che ospita una complessa interfaccia grafica. I dati dell'utente sono trasmessi alla componente di business-logic sul server appropriato e un bean enterprise elabora i dati riconoscendo l'utente e i relativi ruoli che ha all'interno dell'applicazione. Nel caso in cui il riconoscimento dovesse andare a buon fine, l'utente potrà effettuare diverse operazioni che verranno gestite dai relativi EJB. In questo caso specifico non è evidenziato l'aspetto della connessione al data base.

---

[13] Tra gli Application Server più diffusi troviamo JBOSS di tipo open source, WebSphere della IBM e infine Java System Application Server della SUN.

Gli enterprise bean implementano una vasta gamma di logica di business, come da semplici calcoli statistici (che utilizzano dati persistenti) a complesse operazioni scientifiche (che possono non aver nulla a che far con dati).

Gli EJB possono essere suddivisi in tre tipi:

- Bean Session;
- Bean Entity;
- Bean Message-Driven.

4.4.1 SESSION BEAN

Rappresentano un gruppo di operazioni di cui un applicazione client o un altro bean può aver bisogno. Sono invocati tramite interfaccia e vengono creati per la durata di una sessione del client con l'applicazione J2EE. Quando il client termina la propria sessione (nel caso di un browser web, ciò accade ad esempio quando l'utente chiude il browser) il bean session e i dati da esso utilizzati vengono distrutti e non sono più disponibili.

Questi EJB possono essere di due tipi, *stateless* quando le informazioni sono perse tra due transazioni, o *stateful* quando le informazioni vengono mantenute durante un'intera sessione.

In caso di crash del sistema le informazioni sono perse. Se si volesse trovare una soluzione a tale problema, l'onere di implementare una soluzione spetterebbe al programmatore.

4.4.2 ENTITY BEAN

Sono componenti persistenti, rappresentano una vista orientata ai dati e sono invocati tramite interfaccia. Ad esempio un oggetto entity bean potrebbe rappresentare una riga di una tabella di un database.

A differenza dei session bean che sono eliminati alla fine di una sessione (o di una transazione) con il client, i bean entity esistono per tutta la durata dei dati che rappresentano.

Inoltre sono automaticamente salvati poiché la loro persistenza è gestita del container in modo automatico. Ciò non toglie tuttavia la possibilità agli sviluppatori di gestirli in modo diverso escogitando e implementando strategie diverse.

4.4.3 MESSAGE-DRIVEN BEAN

I Bean Message-Driven sono bean enterprise che reagiscono a messaggi asincroni. A differenza dei session e degli entity bean, i metodi dei message-driven non hanno un'interfaccia invocabile, ma partecipano nell'applicazione J2EE mettendosi in ascolto su una coda di messaggi ai quali reagiscono.

L'utilizzo più comune si ha per quelle applicazioni che prevedono un client che non disponga della possibilità di poter rimanere in ascolto per molto tempo e in definitiva questi bean possono essere paragonati a dei Bean Session stateless però di natura asincrona.

4.4.3.1 Differenze

I bean di tipo session e di tipo entity espongono delle interfacce attraverso le quali sono invocati. I tipi di interfacce supportate sono di due tipi. Abbiamo l'interfaccia *componente* che fornisce l'accesso alla logica business nel bean e l'interfaccia *home* che definisce i metodi utilizzati per creare ed eliminare i bean. Nel caso di entity bean quest'ultima interfaccia espone anche dei metodi di tipo *finder* utilizzati per localizzarli.

Le interfacce possono essere di tipo *local* o *remote* a seconda se debbano supportare o meno dei client indipendentemente dalla loro ubicazione. I client che utilizzano bean e che implementano interfacce di tipo remote possono essere eseguiti in una JVM diversa da quella del relativo bean.

4.5 Conclusioni

Il modello di applicazioni distribuite presentato fornisce due vantaggi principali agli sviluppatori.

Il primo riguarda la possibilità di creare codice riutilizzabile e applicazioni con un elevato grado di scalabilità grazie al disaccoppiamento della logica di programmazione imposta dalla piattaforma. Infatti, la divisione delle varie componenti, favorisce lo sviluppo di applicazioni modulari che consentono di ottenere del software riutilizzabile. Inoltre la distribuzione delle componenti su vari livelli introduce un ulteriore vantaggio per il mondo delle applicazioni che sono

destinate ad un unico sistema, cioè quello di raggiungere un elevato grado di scalabilità in quanto ogni livello viene eseguito nel proprio ambiente di elaborazione.

Il secondo vantaggio riguarda il processo di sviluppo dell'applicazione in quanto i gruppi di lavoro da formare saranno facilmente individuabili in funzione del livello che dovranno sviluppare.

Non tutte le componenti nella pratica hanno raggiunto un successo per quel che riguarda il loro utilizzo. Nel livello Web possiamo dire che i filtri e gli event listener non godono al momento di grande "popolarità".

5 Framework di supporto al Pattern MVC

5.1 Introduzione

I pattern descrivono delle soluzioni ampiamente collaudate per problemi di progetto ricorrenti. Inoltre sono soluzioni flessibili e riutilizzabili per problemi comuni, possono quindi far risparmiare tempo e fatica migliorando la qualità globale dell'applicazione.

Data la loro importanza, il team java BluePrints ha compilato un insieme di pattern J2EE più interessanti. Tra questi compare il pattern *MVC* (acronimo di Model-View-Controller) il quale fornisce una soluzione per disaccoppiare la rappresentazione dei dati, la loro presentazione ed il comportamento dell'applicazione.

L'intento del pattern MVC è quello di disaccoppiare il più possibile (come lascia intuire il nome stesso) le parti dell'applicazione adibite al controllo, all'accesso ai dati e alla presentazione.

Il vantaggio maggiore apportato da questo approccio, è quello di avere un'indipendenza tra la logica di business, quella di presentazione e quella di controllo.

Un modo per implementare il paradigma MVC nell'ottica J2EE può essere il seguente:

- *MODEL*: rappresenta i dati di business e la logica di business che operano sui dati. Il modello incapsula lo stato dell'applicazione e risponde alle interrogazioni da parte della view associata. Le componenti che realizzano questa parte solitamente sono rappresentate dagli EJB.
- *CONTROLLER*: agisce da intermediario tra la view e il model. Riceve dati dall'utente tramite la view e invoca i metodi corrispondenti nel modello. Infine definisce la view successiva da presentare all'utente.
- *VIEW*: si occupa della gestione della presentazione. Visualizza i dati interagendo con il model. Il tipo di view presentato cambia a seconda del client.

  La seguente figura racchiude i concetti appena espressi.

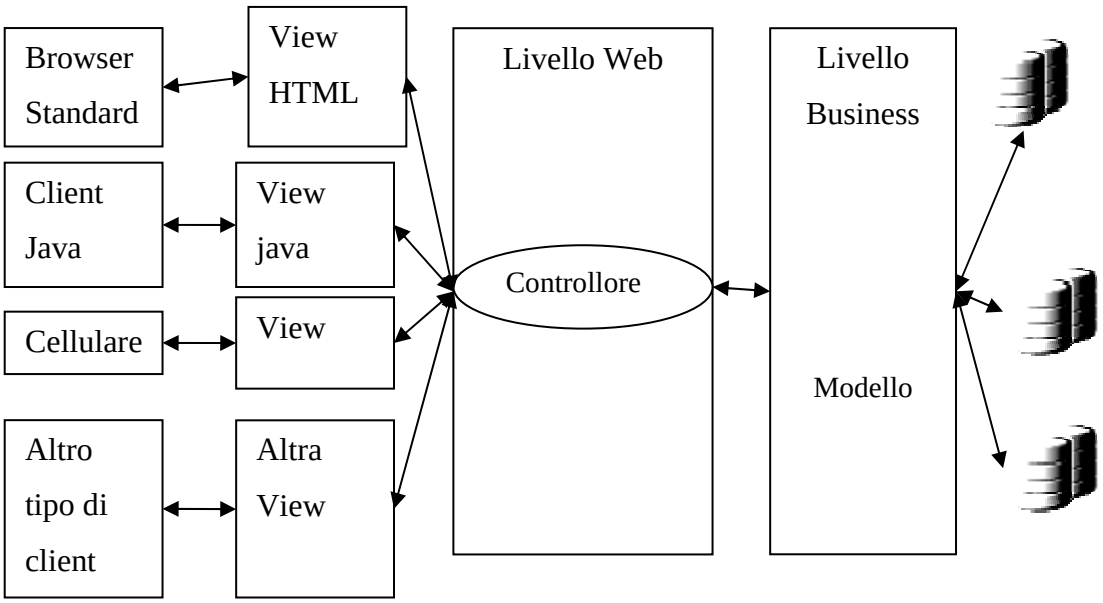


Figura 4 – Possibile implementazione del paradigma MVC.

I vantaggi legati allo schema MVC sono diversi, ad esempio quello di rendere l'applicazione più flessibile poiché si hanno interfacce grafiche multiple per la stessa applicazione, o di modificare l'applicazione indipendentemente dall'interfaccia utente cambiando solo i dati o la loro rappresentazione.

Il pattern MVC può essere applicato in diversi modi. Le applicazioni basate su Pagine JSP possono essere così rappresentate:

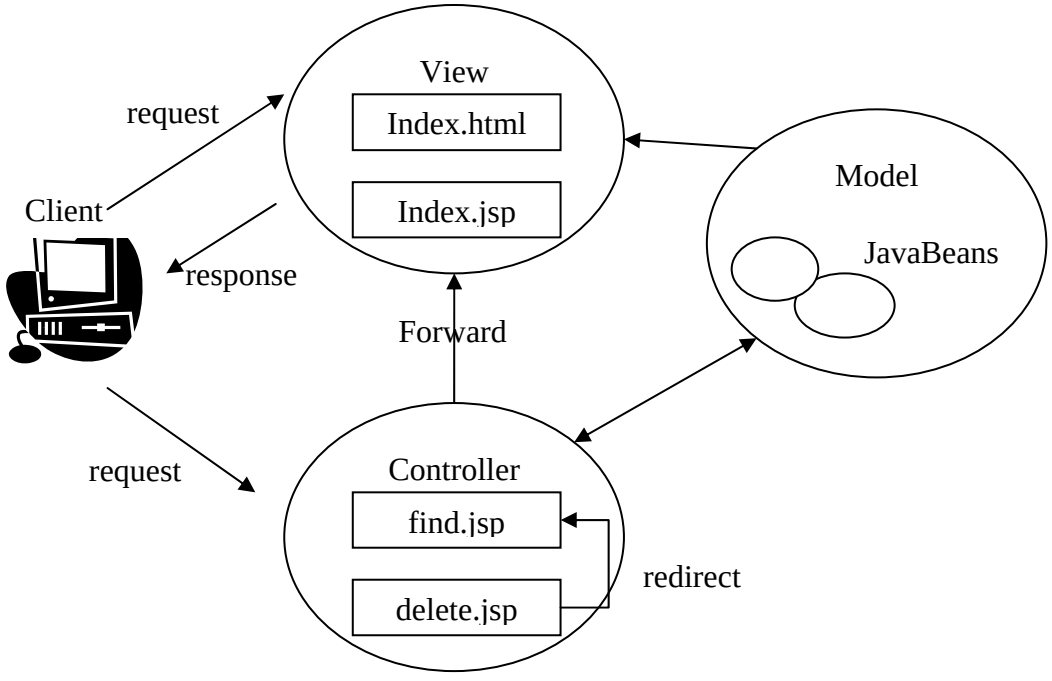


Figura 5 – Implementazione del pattern MVC per applicazioni basate su Pagine JSP.

Le pagine del controller possono inizializzare i bean, le pagine della vista generano le risposte per il client e i JavaBeans racchiudono il Model. Questo modello è utilizzato per la realizzazione di prototipi.

Per applicazioni basate su Pagine JSP e su Servlet lo schema MVC può essere rappresentato in modo differente.

Il Controller in questa configurazione è composto da una o più Servlet che ricevono tutte le richieste. La View e il Model sono rappresentati nello stesso modo dello schema precedente.

La seguente figura racchiude lo schema appena descritto.

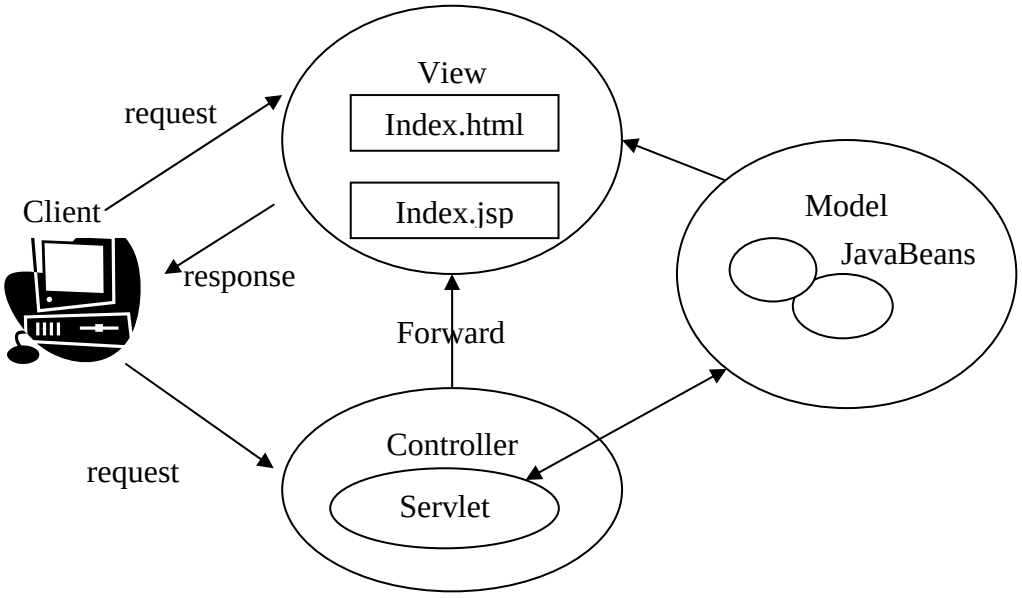


Figura 6 – Implementazione del pattern MVC per applicazioni basate su Pagine JSP e Servlet.

Un altro schema si ha nella realizzazione di applicazioni che racchiudono le Pagine JSP, le Servlet e gli EJB. Il Controller è costituito da Servlet e Session Bean, il Model da JavaBeans ed Entity Bean e la View dalle pagine JSP. La seguente figura mostra come questi elementi possono colloquiare.

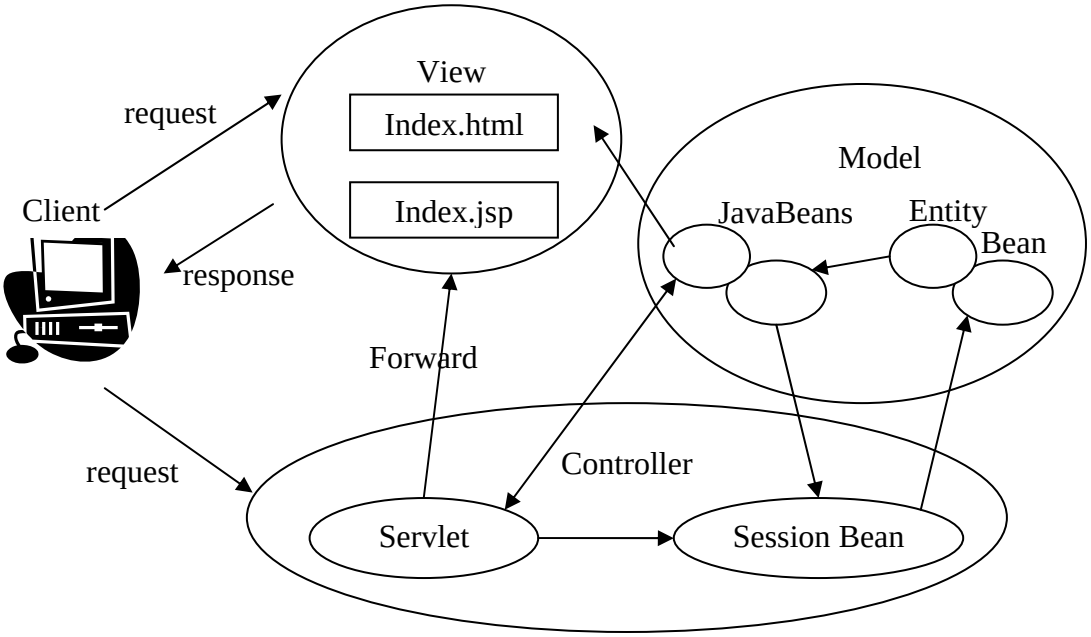


Figura 7 – Implementazione del pattern MVC per applicazioni basate su Pagine JSP, Servlet e EJB.

5.1.1 XML e XSD

XML è l'acronimo di eXtensible Markup Language, è un meta-linguaggio testuale con il quale si possono creare altri linguaggi. È il formato universale per strutturare i documenti e i dati sul Web.

Fondamentalmente XML è un insieme di regole volte a progettare dei formati di testo per strutturare i dati. Il termine "dati strutturati" si riferisce ad informazioni che sono organizzate in modo specifico, ad esempio fogli di calcolo, parametri di configurazione, descrizione dei servizi, messaggi RPC, transazioni finanziarie e così via.

XSD (XML Schema) è linguaggio di descrizione del contenuto di un file XML. Come tutti i linguaggi di descrizione del contenuto XML, il suo scopo è quello di delineare quali elementi sono permessi, quali tipi di dati sono ad essi associati e quale relazione gerarchica hanno fra loro gli elementi contenuti in un file XML. Ciò permette principalmente la validazione del file XML, ovvero la verifica che i suoi elementi siano in accordo con la descrizione in linguaggio XML Schema.

5.2 ORM: Tecniche di accesso ai dati tramite metodologie

Nel realizzare un'applicazione Object Oriented (OO) che utilizza database relazionali quello che si cerca di fare è di ottenere una relazione tra oggetti e dati. ORM è l'acronimo di Object-Relational Mapping e come si evince dal nome l'ORM è una tecnica che permette di ottenere tale relazione.

Dal punto di vista didattico esistono tre tecniche per effettuare il mapping e sono:

- la programmazione a basso livello;
- l'impiego di oggetti per accesso ai dati (Data Access Objects detto DAO);
- l'utilizzo di framework.

La strategia della programmazione a basso livello consiste nell'inserire all'interno delle classi dell'applicazione, un ventaglio di metodi che permettano l'interazione con il database. In questo modo il mapping è realizzato attraverso la definizione dei vari statement SQL. Questa tecnica è applicabile solo all'interno di applicazioni con piccola complessità.

Il secondo sistema prevede la realizzazione di uno strato dell'applicazione, detto DAO che deve gestire la comunicazione tra l'applicazione e il DBMS[14]. Anche in questo caso il mapping è realizzato a mano attraverso l'uso di SQL. L'accesso al database però è opportunamente incapsulato, migliorando la modularità e la scrittura del codice. Nonostante si ottengano maggiori vantaggi rispetto alla programmazione a basso livello anche questa tecnica è applicabile solo all'interno di applicazioni con piccoli livelli di complessità.

La terza strategia prevede l'utilizzo di un framework predefinito per la gestione della persistenza. L'obiettivo è di esonerare il programmatore dallo scrivere codice che possa riguardare l'SQL. Con l'utilizzo dei framework il codice SQL viene generato automaticamente sulla base di informazioni fornite dal programmatore.

[14] Data Base Management System, è un sistema software progettato per consentire la creazione e manipolazione efficiente di database (ovvero di collezioni di dati strutturati) solitamente da parte di più utenti.

Questa strategia è la più diffusa all'interno di applicazioni che hanno anche una complessità elevata.

5.2.1 Hibernate

Hibernate[4] è un framework che consente al programmatore di non scrivere codice in linguaggio SQL, ma di utilizzarne un altro detto Hibernate Query Language (*HQL*); l'oggetto *org.hibernate.Query* si occupa di tradurre la query dal linguaggio HQL al linguaggio SQL.

Hibernate utilizza il database ed i dati di configurazione per fornire servizi di persistenza e oggetti persistenti all'applicazione. Il seguente diagramma mostra una visione ad alto livello dell'architettura di Hibernate.

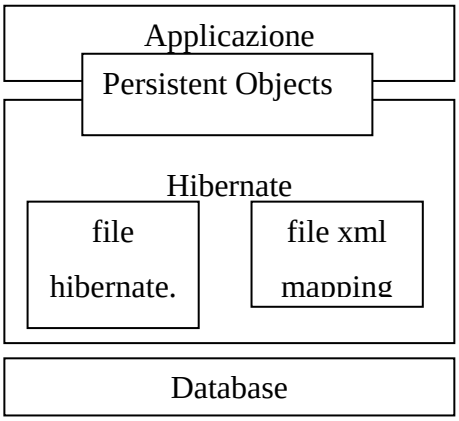


Figura 8 – Architettura ad altro livello di Hibernate.

Come si vede dalla figura precedente Hibernate maschera completamente all'applicazione, e pertanto allo sviluppatore, il database.

Hibernate permette all'applicazione di ottenere un mapping tra le tabelle di un database con semplici classi java. Queste classi sono dette *classi persistenti* ed hanno due tipi di istanze, le *transienti* e le *persistenti*. Le classi persistenti devono seguire le regole del modello dei POJO (vedi paragrafo B.5.4).

Inoltre è mostrato il file xml di mapping. Anche questo file è modificabile con qualsiasi editor di documenti testuali. È detto *java-centrico* poiché il mapping è orientato verso le classi persistenti e non alle tabelle del database. Il seguente Listato mostra un esempio di file di mapping.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 2.0//EN"
```

```
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping package="it">
  <class name="Prodotto" table="PRODOTTI">
    <property name="id" type="string"/>
    <property name="descrizione" type="string"/>
  </class>
<class name="Prezzi"  table="PREZZI" >
  <!-- mapping relativo ai prezzi -->
</class>
</hibernate-mapping>
```

Listato 1

Con il file di properties e con il file di mapping forniti, gli sviluppatori possono costruire un oggetto *SessionFactory* che permette di creare istanze di *Session*. L'oggetto SessionFactory è realizzato in modo da poter essere condiviso da tutti i flussi esecutivi dell'applicazione.

Hibernate fornisce diversi modi per effettuare le connessioni JDBC, ad esempio si possono inserire i parametri di configurazione o all'interno del file di properties o all'interno del file xml di mapping.

5.3 Apache Struts

Apache Struts[5] è un framework utilizzato per la realizzazione di Web Application secondo il pattern MVC. È di tipo Open Source ed è caratterizzato da una serie di classi e dalle relative API che permettono di:

- acquisire i dati dei form dall'utente durante la navigazione ed eseguire le elaborazioni su di essi;
- individuare i gestori detti *action* per processare le richieste;
- definire la navigazione fra le pagine all'interno della Web Application.

Il framework Include due plug-in:

- *Tiles*: utilizzato per la realizzazione del layout delle pagine con "mattonelle" componibili;
- *Validator*: fornisce uno strumento per la validazione dei dati immessi nei form dall'utente.

Inoltre, include molte librerie, tra le più utilizzate possiamo ricordare le seguenti:

- *Struts HTML*: utilizzata per la realizzazione dell'interfaccia utente con una serie di tag che permettono di introdurre le componenti della GUI all'interno delle pagine;
- *Struts Bean*: utilizzata per accedere da ciascuna pagina alle proprietà dei Java Beans;
- *Struts Logic*: utilizzata per definire dei costrutti di controllo che permettono la costruzione dinamica di una pagina.

Struts si basa sugli strumenti che costituiscono principalmente un'applicazione Web realizzata in Java, come le Pagine JSP e Servlet. Il vantaggio offerto è la semplicità di configurazione della Web Application in quanto è resa possibile tramite un file XML all'interno del quale vengono impostati tutti i parametri e gli elementi costitutivi dell'applicazione stessa.

Basandosi sul pattern MVC Struts propone una serie di elementi e di classi che compongono il pattern:

- *Controller*: composto dalle classi che gestiscono il flusso di esecuzione dell'applicazione e garantiscono la comunicazione tra il Model e la View. Ad esse si aggiungono una serie di classi di utilità
- *Model*: le classi che definiscono lo stato dell'applicazione e le funzionalità della logica di business.
- *View*: composto tipicamente da pagine JSP che utilizzano i tag delle librerie di Struts.

5.4 POJO

POJO è l'acronimo di Plain Old Java Object, in parole povere è una "semplice" classe Java che non ha un legame diretto con un container o con un Application Server (nel seguito chiamato *AS*). Può essere visto come un ordinario JavaBean.

La specifica EJB si basa su un principio base che vede il dualismo Container-Bean centrale. Un Session Bean o un Entity Bean vivono all'interno di un container il quale regola il ciclo di vita, gli attributi transazionali, il pooling e la sicurezza.

Con i POJO è stato riconsiderato il lavoro fatto dal punto di vista della semplificazione. La prima trasformazione è consistita nello spostare la complessità all'interno dell'Application Server, lasciando fuori solamente componenti che a prima vista sono semplici classi Java e non Enterprise Beans.

Questo passaggio prende spunto in parte dal lavoro fatto dal team di JBoss (il plugin JBoss IDE già da tempo utilizzava annotazione per definire il comportamento di un session e di un entity), in parte facendo propria la filosofia adottata dai moderni framework di mapping come Hibernate.

Un Entity Bean quindi diventa un semplice POJO (Plain Old Java Object) ovvero un bean che contiene metodi e attributi relativi al suo funzionamento non enterprise. Un POJO è quindi, in prima battuta, un componente che non ha nessun legame con il container.

Il seguente codice mostra un esempio di POJO:

```
public class Prodotto {
  String id;
  String descrizione;

  public String getId() {
    return id;
  }
  public void setId( String id ) {
    this.id = id;
  }
  public String getDescrizione() {
    return descrizione;
  }
  public void setDescrizione( String descrizione ) {
    this.descrizione = descrizione;
  }
}
```

Listato 2

Come si vede dal codice non è stato fatto nessun riferimento alle interfacce *javax.ejb.**.

Questo stile di programmazione è ormai divenuto uno standard, e per quel che riguarda la persistenza si ci appoggia al framework utilizzato.

Se, ad esempio, utilizzassimo Hibernate, dovremmo scrivere nel relativo file di configurazione il seguente codice:

```
<hibernate-mapping>
  <class name="Prodotto" table="PRODOTTI">
    <property name="id" type="string"/>
    <property name="descrizione" type="string"/>
  </class>
</hibernate-mapping>
```

Listato 3

Dove la tabella relativa al POJO Prodotto va sotto il nome di “PRODOTTI” ed i vari campi hanno i seguenti nomi: id e descrizione.

Inoltre, essendo indipendenti dal container i POJO sono normalmente utilizzati per trasferire dati tra le varie componenti e gli strati architetturali del sistema, come ad esempio tra lo strato di presentazione ed il livello web di un’applicazione J2EE.

5.5 SOA e Web Services

SOA è l'acronimo di *Service Oriented Architecture* ed è un paradigma architetturale in grado di supportare l'uso di servizi Web. Esso consente di conseguire un basso accoppiamento tra "agenti software" (unità autonome costituenti moduli dinamici di un'architettura distribuita). Un servizio è un'unità di lavoro svolta da un *service provider* per acquisire determinati risultati per un *service consumer*. Sia il provider che il consumer sono ruoli che agenti software svolgono per conto dei loro gestori. L'idea di SOA discende significativamente dal paradigma Object Oriented, il quale è caratterizzato dall'incapsulamento di dati ed elaborazioni, e nel quale un'applicazione è composta da tanti oggetti che interagiscono. Il paradigma SOA consente maggior dinamicità del paradigma OO, attraverso una serie di criteri quali:

- interfacce di comunicazione semplici, che codificano esclusivamente semantica di business generica, e disponibili per tutti gli agenti software partecipanti;
- messaggi descrittivi limitati nel vocabolario e nella struttura da uno schema che risulta però estensibile, ossia in grado di supportare servizi futuri senza interrompere la comunicazione con servizi esistenti;
- messaggi che non prescrivono (o lo fanno in minima parte) specifici comportamenti del sistema.

Nel nostro contesto, SOA consente di soddisfare le richieste degli utenti adoperando le singole applicazioni come *componenti* del processo di business.

Il concetto di Web Service (WS) può essere espresso in diversi modi. È generalmente accettato che un WS sia un'architettura SOA con almeno i seguenti vincoli addizionali:

- le interfacce devono essere basate sui protocolli di Internet quali HTTP, FTP, ed SMTP;
- eccetto per allegati di tipo binario, i messaggi devono essere in XML.

Vi sono diversi stili (modi di implementare) Web Service. Nel presente studio esaminiamo i SOAP WS, la forma di WS più diffusa e commercializzata nel mondo industriale.

Un SOAP WS introduce i seguenti vincoli:

- eccetto per allegati di tipo binario, i messaggi devono essere trasportati da SOAP[15];
- la descrizione dei servizi deve essere realizzata in WSDL[16].

Alcuni autori considerano un WS come l'unione di SOAP e WSDL. SOAP fornisce uno schema di messaggi che può essere usato da una varietà di sottostanti protocolli. SOAP è estendibile ed è progettato secondo il paradigma SOA. In altri termini, SOAP agisce come un "envelope" (involucro) che trasporta contenuti, e consente il trasporto di messaggi secondo diversi pattern, anche, per esempio, secondo il tradizionale schema richiesta-risposta, o uno schema broadcasting, o caratterizzato da sofisticare correlazioni. Esistono i SOA RPC Web Services ed il SOA Web Services di tipo "document centric". La prima implementazione non è aderente al paradigma SOA, perché codifica il comportamento del sistema e la semantica dell'applicazione, per cui le applicazioni create con SOAP RPC non sono interoperabili per natura. Questo è stato confermato da molte implementazioni commerciali.

Torniamo agli aspetti funzionali del WS. Un Web Service consente quindi di implementare un'applicazione distribuita sulla rete Internet. Un servizio web del tipo "richiesta-risposta", accetta una richiesta, esegue un servizio a seconda della richiesta e restituisce una riposta. La richiesta e la risposta possono far parte della medesima operazione o possono avvenire separatamente.

Come detto, sia la richiesta che la risposta sono in XML, formato per lo scambio dei dati, e sono trasmesse attraverso un protocollo di rete (solitamente HTTP). Le transazioni tra servizi web avvengono normalmente tra entità di business.

L'architettura J2EE supporta sia il modello dei servizi web orientato ai documenti, che quello orientato alle chiamate di procedure remota (*RPC*).

---

[15] Protocollo basato su XML, progettato per lo scambio di informazioni in ambiente distribuito.

[16] Web Services Description Language, standard per descrivere i servizi Web utilizzando XML.

Nell'ambito di un'architettura SOA è possibile modificare dinamicamente le modalità di interazione tra i servizi, la combinazione dei medesimi, ed il numero di servizi. Il processo di business non è più vincolato ad una specifica piattaforma, ma può essere considerato come una componente di un processo più ampio.

L'architettura orientata ai servizi si presenta particolarmente adatta per le aziende che presentano una discreta complessità di processi e applicazioni poiché agevola l'interazione tra le diverse realtà aziendali permettendo alle attività di business di sviluppare processi più efficienti e dinamici.

Nonostante SOA non sia legata ad una singola specifica tecnologia. tuttavia i Web Services rappresentano oggi un modo efficace, e abbastanza comune, di esporre servizi, infatti spesso si utilizzano entrambi i termini per rappresentare la stessa cosa.

Oltre agli standard basati su XML quali *SOAP/HTTP*, *WSDL* i Web Services impiegano servizi di directory come ad esempio *UDDI*[17].

I SOA nel contesto J2EE consentono alle applicazioni di esporre le proprie funzionalità, in termini di *endpoint*[18] di un servizio, descrivendo in modo astratto i propri metodi attraverso lo standard WSDL. Il documento WSDL può essere pubblicato in un registry utilizzando delle API (come ad esempio JAXR, Java API for XML Registry) per consentire alle applicazioni nella rete di trovare dinamicamente ed utilizzare il servizio così come è descritto. Partendo dal WSDL è possibile ottenere gli *stub*[19] delle classi contenute nel documento. In questo modo un client dopo aver trovato, o scelto, il servizio web da utilizzare si collega ad esso in modo da poterne invocare i metodi pubblicati.

Tale processo è raffigurato nella seguente figura.

---

[17] Universal Description Discovery and Integration, è un registry (ovvero una base dati ordinata ed indicizzata), basato su XML ed indipendente dalla piattaforma hardware, che permette la pubblicazione dei servizi offerti.

[18] Indirizzo del servizio e generalmente inteso come implementazione concreta del servizio

[19] È una visione remota dell'oggetto e contiene solo i metodi remoti.

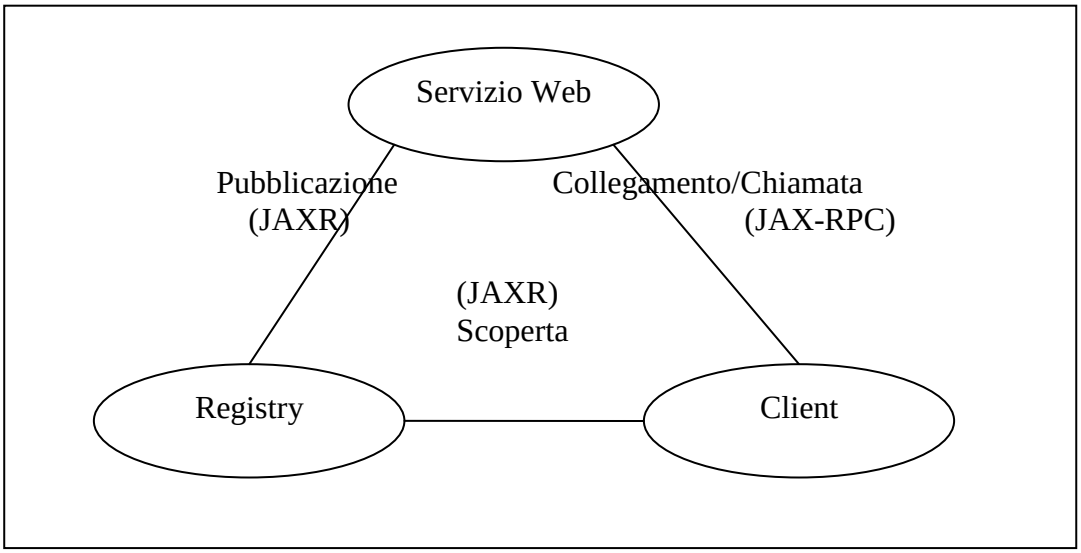


Figura 9 – I servizi Web generici seguono il modello pubblicazione-scoperta-collegamento

Le componenti J2EE che consumano servizi Web ottengono il collegamento utilizzando *JAX-RPC*[20] che fornisce le API fondamentali per accedervi e per effettuare le chiamate di procedura remota.

Se il client e il fornitore del servizio si "conoscono" i passaggi di pubblicazione e scoperta sono trascurabili.

Nel caso di servizi orientati all'RPC i client invocano i metodi, nel caso di servizi orientati allo scambio di documenti i client inviano i documenti al servizio. Un messaggio SOAP con allegati ("SOAP with attachment", una estensione di SOAP) può trasportare qualsiasi contenuto codificato con lo standard MIME[21] permettendo ai servizi Web di scambiarsi documenti XML, immagini, pagine Web o qualsiasi altro tipo di documento.

Le specifiche Web Services per J2EE definiscono l'architettura dei servizi i cui *endpoint* possono essere implementati con vari metodi; ad esempio è possibile utilizzare le Servlet o i Session Bean Stateless.

JAX-RPC fornisce le funzionalità fondamentali dei servizi Web offerte dalla piattaforma J2EE, permette il loro sviluppo, la loro messa in opera utilizzando

[20] Java Api per XML-based Remote Procedure Call

[21] È uno standard generico per il formato dei documenti scambiati sulla rete Internet.

tecnologie XML e supporta le chiamate di procedura remota utilizzando messaggi SOAP trasportati su protocollo HTTP.

JAX-RPC, inoltre, utilizza lo standard *WSDL* che consente ai Web Services di pubblicare la descrizione dei propri servizi.

In J2EE viene formalizzato il concetto di *Porta* la quale descrive le implementazioni degli endpoint e fornisce i servizi.

Le specifiche Web Services per J2EE esprimono tale concetto con le espressioni "*Interfaccia EndPoint del Servizio*". In altre parole una Porta consiste in una interfaccia EndPoint del servizio.

Un modo per realizzare Web Services consiste nell'utilizzare Apache Axis[22]. Si tratta di un'API di programmazione e deployment di WS che permette di lavorare ad un livello di astrazione elevato. È anche possibile sviluppare client di servizi di terzi.

[22] http://ws.apache.org/axis/

C. Banca Dati delle Misure Cautelari Personali

1 Analisi e Progettazione

1.1 Descrizione dell'architettura

Il sistema informatico BDMC utilizza l'architettura Web based e si appoggia sull'infrastruttura di networking costituita dalla Rete Unitaria della Giustizia (da ora in avanti denominata *RUG*).

Esistono due sottosistemi distinti dell'applicazione:

- BDMC locale, situato presso gli uffici giudiziari periferici che memorizza e gestisce i procedimenti relativi al territorio di competenza;
- BDMC nazionale, situato presso la sede centrale del Dipartimento dell'Amministrazione Penitenziaria (da ora in avanti chiamato *DAP*), il quale riceve dagli uffici giudiziari periferici i dati riguardanti tutti i procedimenti sul territorio italiano.

L'architettura del sistema è la classica 3-tier costituita dai livelli logici seguenti:

- LIVELLO DI PRESENTAZIONE: ha il compito di presentare le informazioni provenienti dai livelli inferiori dell'utente.
- LIVELLLO APPLICATIVO: gestisce le informazioni da e verso il livello della base dati, svincolando da quest'ultimo il livello di presentazione, e implementa la cosiddetta Business Logic
- LIVELLO DELLA BASE DATI: si occupa della gestione materiale delle informazioni memorizzate, con tutte le caratteristiche messe a disposizione dal DBMS utilizzato.

Dal lato utente è stata presa la scelta di utilizzare un client di tipo "leggero" dove l'unica componente effettivamente necessaria è il browser.

BDMC prevede l'interfacciamento con diversi sistemi informativi. Ciò è realizzato attraverso un'infrastruttura di tipo XML e realizzando una serie di Web Services che permettono all'applicazione di colloquiare con il mondo esterno con un

comportamento che risulta essere indipendente dalle piattaforme Hardware e Software.

1.1.1 Architettura a tre livelli

L'architettura adottata si basa sul paradigma 3-tier che si può riassumere nei seguenti punti:

- *RDBMS* centralizzato su un sistema robusto, specializzato nell'accesso e nel trattamento dei dati, interfacciato soltanto dall'Application Server, e quindi isolato dall'interazione con i sistemi client.
- *Logica Applicativa* (detta anche Business-Logic) processata da un middleware separato e integrato con le altre componenti, a garanzia di un ambiente "transazionale" di sostegno alle applicazioni sicuro e adeguato sotto il profilo delle prestazioni.
- Sistemi Client di tipo "leggero" (con solo il browser) o "pesante" (con parte della logica applicativa residente).

La cooperazione tra i diversi livelli avviene seguendo il seguente schema architetturale ben specifico:

- il livello di presentazione è gestito dalla coppia Client-Web Server;
- il livello applicativo è gestito interamente dall'Application Server;
- il livello della base dati è gestito sul DBMS.

Così facendo sono stati raggiunti i seguenti benefici:

- sicurezza implementabile in maniera autonoma su ciascuno dei 3 livelli, in modo da avere un risultato globale migliore di quello raggiungibile da una singola implementazione.
- minore traffico di rete, con conseguente miglioramento dei tempi di risposta.
- possibilità di utilizzare client "leggeri", senza necessità di installare software dedicato su di essi e semplificando quindi il deployment dell'applicazione.
- scalabilità, sia verticale (ad esempio incrementando il numero di CPU dei server ove questo si renda necessario per motivi prestazionali) che orizzontale (ad

esempio incrementando il numero di macchine fisiche ove installare server logici che offrono un medesimo servizio per gestire un aumento del carico di lavoro).
- affidabilità, garantita dalla possibilità di "distribuire" l'applicazione su più macchine in modo da minimizzare l'impatto di eventuali malfunzionamenti (fault-tolerance) e bilanciare opportunamente il carico di lavoro al mutare delle condizioni (load balancing).
- elevato grado di flessibilità e adattabilità a scenari differenti, che consente una configurazione che permette di raggiungere la soluzione migliore per le esigenze dell'amministrazione.

1.1.2 Prodotti Software

Nel presente paragrafo vengono presentati i prodotti software con cui è implementata e su cui è attiva l'applicazione del sistema BDMC:
- *Client*: S.O. MS Windows con Web Browser standard (Internet Explorer)
- *Web Server*: S.O. Windows NT o Windows2000, con Apache HTTP Server.
- *Application Server*: S.O. Windows NT o Windows2000, con JBoss.
- *Data Server*: S.O. Windows NT, oppure Windows2000, oppure UNIX o LINUX, con RDBMS Oracle.
- *Software Applicativo*: sviluppato in Java.

Si sottolinea che la scelta dei prodotti di Web Server e Application Server non comporta alcun aggravio economico per la Pubblica Amministrazione, in quanto si tratta di software Open Source.

Il Web Server Apache è diffuso su una grande percentuale di siti Internet per le sue caratteristiche prestazionali.

JBoss rappresenta uno strumento per la realizzazione e l'impiego efficiente di applicazioni Web a tre livelli in ambiente Intranet/Internet. È caratterizzato da un set di servizi e strumenti che consentono di mettere in esercizio le applicazioni garantendo elevati livelli di scalabilità ed affidabilità. In particolare i servizi di gestione della logica applicativa permettono di sviluppare applicazioni Java utilizzando J2EE, EJB, JSP, XML e Web Services.

Si tratta quindi di strumenti riconosciuti standard "*de facto*" che garantiscono modularità, apertura, flessibilità, integrazione e possibilità per le applicazioni sviluppate in questi ambienti di evolvere e crescere nel tempo, premesse per la realizzazione di progetti di alta qualità.

1.2 Manutenzione evolutiva

Il processo di attivazione e realizzazione della manutenzione evolutiva di intervento (da ora in avanti detta *MEV*) è illustrato nella seguente figura:

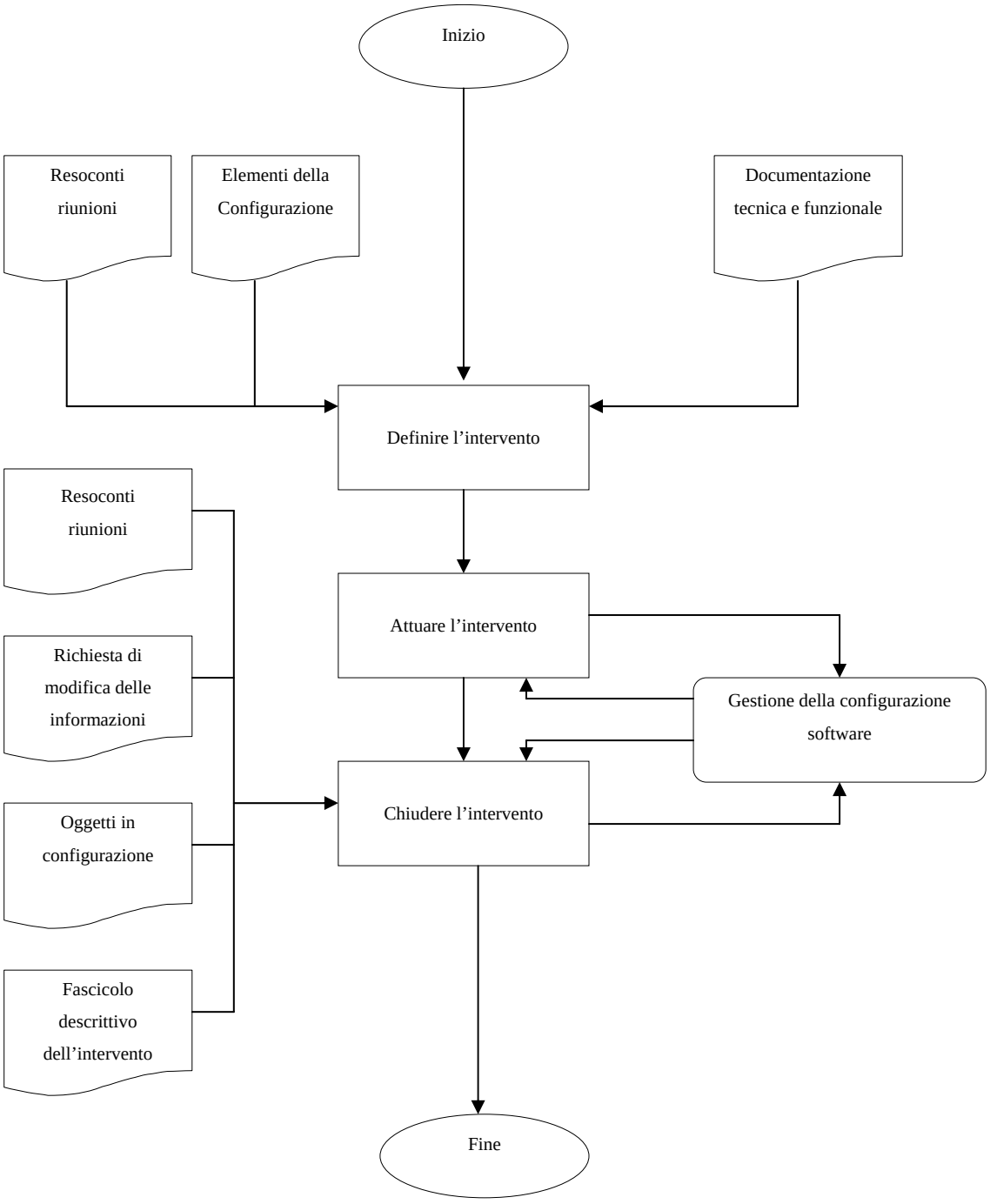


Figura 10 – Processo di attivazione e realizzazione della manutenzione evolutiva di intervento.

Nel seguito saranno mostrate nel dettaglio le fasi riguardanti la definizione, l'attuazione e la chiusura di un intervento di tipo MEV.

Definizione dell'intervento

Sono quelle attività volte a:

- dettagliare e formalizzare i requisiti dell'intervento ed ad individuare le azioni di modifica del sistema necessarie a soddisfarli;
- definire la progettazione e pianificazione delle attività;
- impostare il controllo dello stato di lavorazione.

1.2.1.1 Analisi dei requisiti

Il Responsabile dell'Intervento "attiva" la Data inizio dell'intervento, presente nel modulo "*PMD-SW/Registrazione Interventi di manutenzione*".

Il Responsabile dell'Intervento, inoltre, reperisce e consulta la documentazione esistente sull'attività da svolgere (resoconti riunione, documenti e corrispondenza scambiata con l'Utente ecc.) per individuare nel dettaglio i requisiti dell'intervento. Nel caso in cui le informazioni in suo possesso non siano considerate esaurienti, richiede ulteriori "interviste" con l'utente per approfondire aspetti gestionali e funzionali ed esigenze non sufficientemente definiti.

Dopo aver acquisito tutte le informazioni ritenute necessarie per iniziare le attività, compila la descrizione dell'intervento, con l'evidenziazione dei requisiti che lo caratterizzano, sul modulo delle modifiche al software applicativo indicando inoltre l'eventuale documentazione utilizzata o prodotta.

1.2.1.2 Delimitazione del contesto

Sulla base della documentazione tecnica e funzionale preesistente, il Responsabile dell'Intervento:

- individua tutte le implicazioni tecnologiche dell'intervento;
- opera le scelte sulle modalità di intervento;
- definisce le funzionalità da modificare e/o automatizzare;
- acquisisce lo stato attuale degli elementi del Sistema Informativo da sottoporre a manutenzione consultando le informazioni di configurazione;

- individua i moduli del software applicativo interessati dall'intervento riportandone l'elenco della configurazione;
- definisce gli interventi da effettuare su ogni modulo software riportando delle modifiche al software applicativo la descrizione, per ogni componente, della nuova funzionalità o della modifica a quella preesistente.

1.2.1.3 Controlli dello stato di lavorazione

In parallelo alle attività di Analisi dei Requisiti e Delimitazione del contesto il Responsabile dell'Intervento stabilisce la pianificazione delle fasi di lavorazione impostando sul modulo delle modifiche al software applicativo le date di previsione relative a ciascuna fase.

1.2.1.4 Riesame

Come ultima attività del passo "Definire l'intervento", il Responsabile dell'Intervento analizza e riesamina quanto riportato nelle diverse sezioni del modulo delle modiche al Software applicativo, ne verifica la rispondenza con i requisiti e formalizza il completamento del riesame.

Attuazione dell'intervento

Le attività volte all'attuazione all'intervento prevedono di:

- realizzare delle modifiche al software applicativo e/o di tutte le attività documentate nel modulo;
- controllare che quanto è stato prodotto sia conforme ai requisiti.

1.2.1.5 Progettazione dei Test

Il Responsabile dell'Intervento documenta i casi di prova ritenuti necessari e sufficienti a verificare l'intervento compilando il modulo di testing, sul quale, per ogni caso di test, dovranno essere specificati i valori di ingresso e la descrizione del caso.

Riesame della progettazione dei Test

Al termine della progettazione dei Test, il Responsabile del Servizio verifica che i casi di test individuati assicurino un livello di test adeguato per i moduli

software oggetto dell'intervento. Formalizza la fase di riesame approvandola oppure richiedendo al Responsabile dell'Intervento un'implementazione dei casi di test e ripetendo il riesame.

1.2.1.6 Realizzazione dell'Intervento

Il Responsabile dell'Intervento attiva la procedura "Gestione della configurazione del software applicativo" per verificare la disponibilità degli elementi in configurazione (oggetti del software applicativo) interessati dall'intervento.

Se non sussistono controindicazioni, conflitti, sulla base di quanto descritto nel modulo contenente gli "Elementi di Configurazione" movimentati dall'intervento trasferisce gli elementi software oggetto di modifica dall'ambiente di configurazione a quello di modifica secondo le modalità operative descritte nel piano di configurazione adottato.

Una volta avuti a disposizione i moduli software, il Responsabile dell'Intervento modifica il codice dei sorgenti e, se necessario, aggiorna la documentazione tecnica e utente.

Nel caso in cui le caratteristiche dell'intervento lo rendano necessario, il Responsabile si avvale della collaborazione della struttura tecnica competente per attuare specifiche verifiche.

Ad esempio, se l'intervento prevede variazioni alla struttura della Base Informativa, ciò potrebbe rendersi necessario per:

- la verifica dello schema logico rispetto ai requisiti prestazionali e di ottimizzazione dei dimensionamenti;
- la definizione di tutte le caratteristiche fisiche della base informativa (schema fisico).

Sulla base dello schema logico e delle caratteristiche hardware e software dell'ambiente viene inviata una comunicazione di richiesta di modifica dell'architettura alla competente struttura tecnica che provvederà alla realizzazione di quanto richiesto.

1.2.1.7 Esecuzione dei test

Il Responsabile dell'Intervento esegue i test previsti secondo quanto riportato sul modulo di testing (ripetendo i test fino al raggiungimento dell'esito positivo), registra gli esiti ottenuti (sia quelli negativi, che quelli positivi), documentandoli nell'apposita sezione del modulo sopra citato, e formalizza la corretta fine dell'attività.

1.2.1.8 Riesame

Il Responsabile del Servizio analizza e verifica la rispondenza di quanto attuato con quanto previsto nel passo "Definire l'intervento" e documentato nel modulo PMD-SW/Modifica del software applicativo. Se il riesame risulta positivo formalizza l'approvazione secondo le procedure previste.

1.2.1.9 Consuntivo

Come ultima attività del passo "Attuare l'intervento", il Responsabile dell'Intervento riporta sul modulo delle modifiche al software applicativo la chiusura della modifica ed eventuali note. Il Responsabile del Servizio comunica al Referente Informatico dell'Amministrazione la chiusura dell'intervento e la disponibilità al collaudo.

Chiusura dell'intervento

La chiusura dell'intervento è costituta dalle seguenti fasi volte:

- alla messa in esercizio degli oggetti di software applicativo modificati;
- al completamento e controllo qualità della documentazione;
- alla formalizzazione della chiusura dell'intervento.

1.2.1.10 Messa in esercizio del software

Il Responsabile dell'Intervento:

- trasferisce in ambiente di esercizio gli oggetti del software applicativo e verifica l'esito dell'operazione effettuata, secondo le modalità operative descritte nel piano di configurazione adottato; viene attivato il Servizio di Configurazione degli Ambienti e Dispiegamento degli Applicativi;

- registra sul modulo PMD-SW/Elementi di Configurazione movimentati dall'intervento la nuova versione assegnata agli elementi software movimentati;
- attiva, qualora sia necessario, il Servizio di Formazione in modo che venga pianificato e realizzato un corso di addestramento per gli utenti che utilizzeranno le nuove funzioni applicative.

1.2.1.11 Completamento della documentazione

Il Responsabile dell'Intervento verifica, cataloga ed eventualmente completa la documentazione prodotta o aggiornata nel corso dell'intervento (sia tecnica che utente).

Controllo qualità

Il Responsabile del Servizio controlla la completezza della documentazione utente prodotta e/o modificata e la rispondenza delle modifiche apportate rispetto ai requisiti dell'intervento.

Se l'esito del controllo è negativo fornisce indicazioni al Responsabile dell'Intervento per la rimozione delle anomalie qualitative riscontrate.

Se il controllo ha esito positivo convalida l'intervento, secondo quanto previsto dalle regole di revisione del documento stesso.

1.2.1.12 Chiusura intervento

Al termine delle attività il Responsabile dell'Intervento provvede alla compilazione delle informazioni di Chiusura Intervento, ed eventualmente delle Note, del modulo della registrazione di interventi di manutenzione convalidando le attività svolte.

Il Responsabile del Servizio:

- formalizza l'avvenuto completamento dell'intervento ed approva le registrazioni compilando le apposite informazioni sul modulo PM-SW Modifiche al software applicativo;
- attiva la procedura "Gestione della configurazione del software applicativo" per la chiusura dell'intervento di variazione della configurazione;
- provvede a comunicare al Referente Informatico dell'Amministrazione la conclusione dell'attività.

1.3 Use Case Diagram

Nel seguente Use Case Diagram vengono illustrati gli attori presenti nell'ambito applicativo di BDMC.

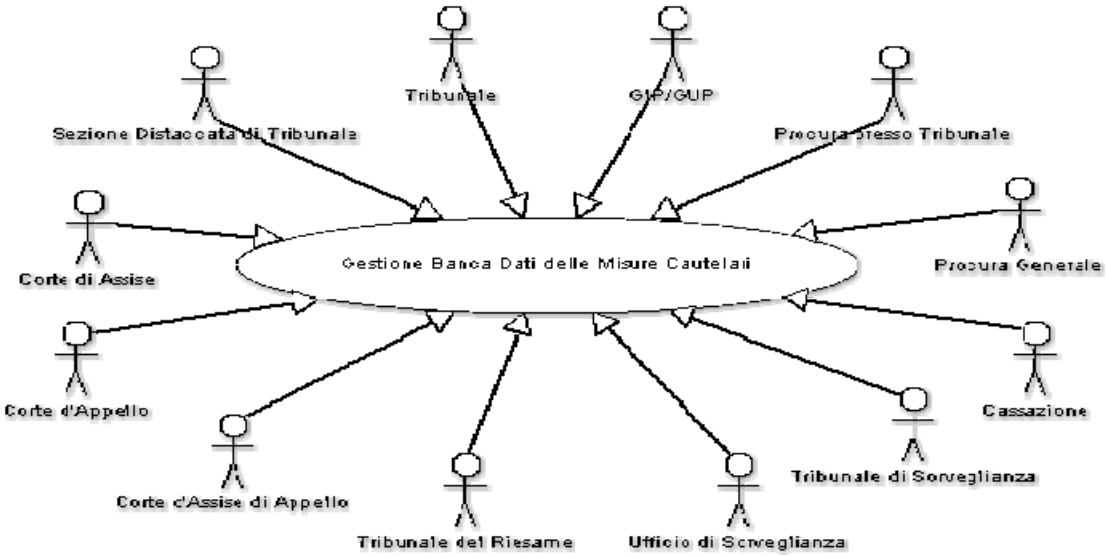


Figura 11 – Use Case Diagram

1.4 Class Diagram

Sulla base delle specifiche definite per l'applicazione il sistema prevede diverse entità e nel seguito verranno mostrate quelle principali.

Nel seguente Class Diagram vengono illustrati alcuni POJO.

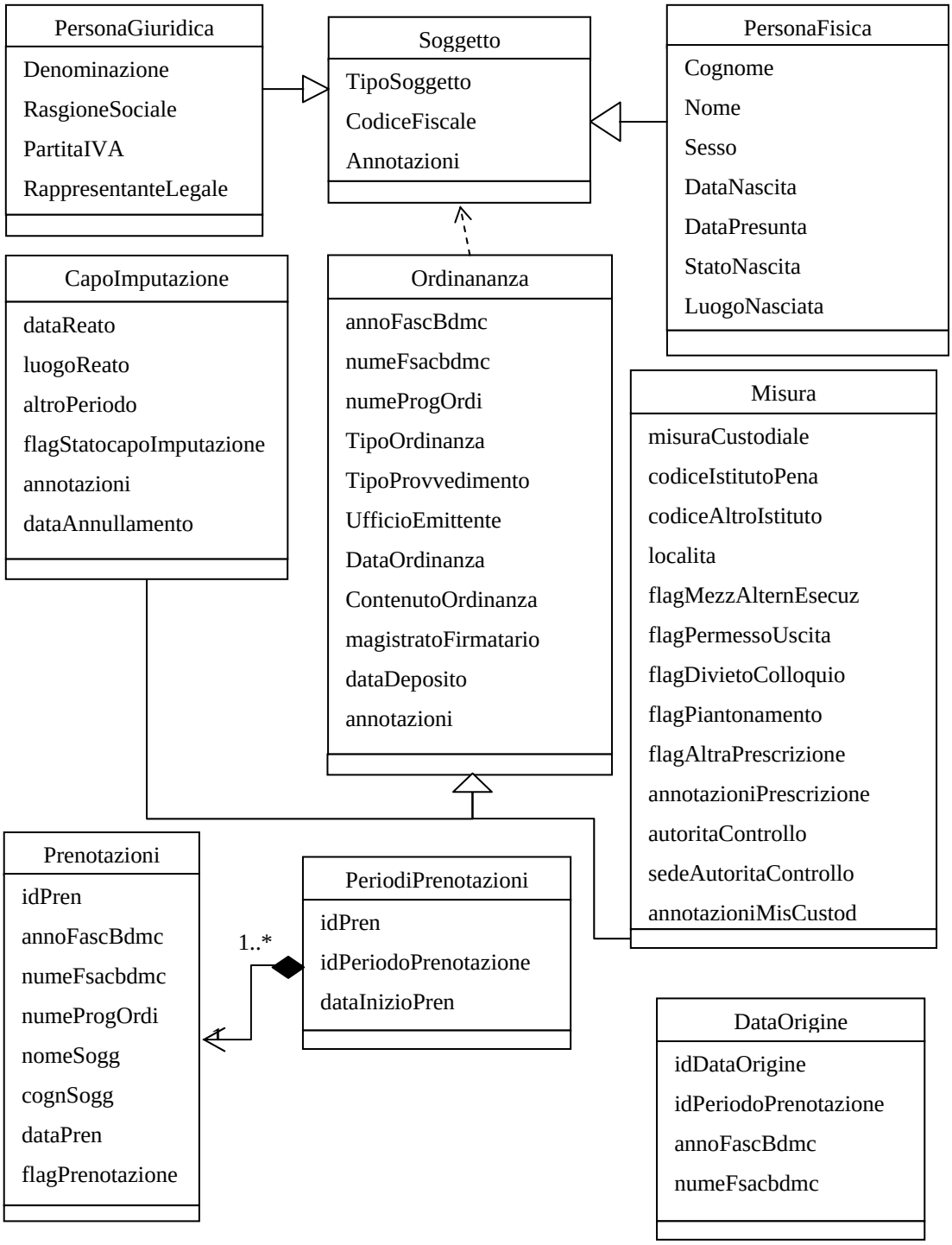


Figura 12 – Class Diagram

Nel successivo diagramma sono mostrate gli Enterprice JavaBeans implementati lato applicativo. Lo scopo di tale diagramma è quello di mostrare l'architettura degli EJB e non quello di esplicare tutti i metodi di business.

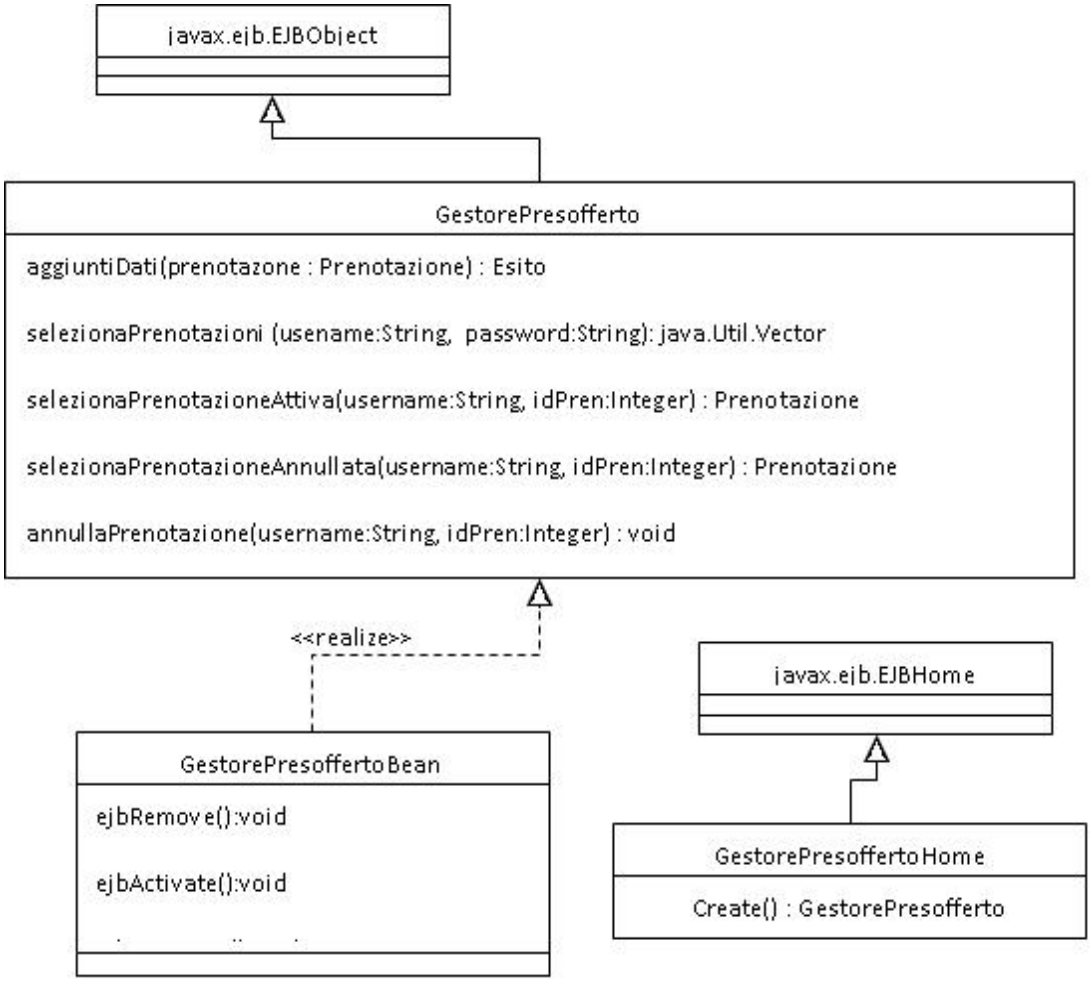


Figura 13 – Class Diagram

1.5 Activity Diagram

Nel seguente Activity Diagram è illustrata la prenotazione di un presofferto[23] che include le seguenti operazioni:

- inserimento criteri di ricerca di un soggetto;
- selezione di un fascicolo;
- selezione delle informazioni da prenotare;
- visualizzazione di quanto prenotato;
- eventuale nuova ricerca.

Le possibili informazioni da prenotare sono le seguenti:

[23] Un presofferto è il periodo di tempo di pena già scontata (ad esempio attraverso il carcere o gli arresti domiciliari) da parte di un pregiudicato prima di essere valutato colpevole.

- generalità del soggetto;
- periodi di presofferto con riferimento alle ordinanza da cui si stanno prenotando i periodi.

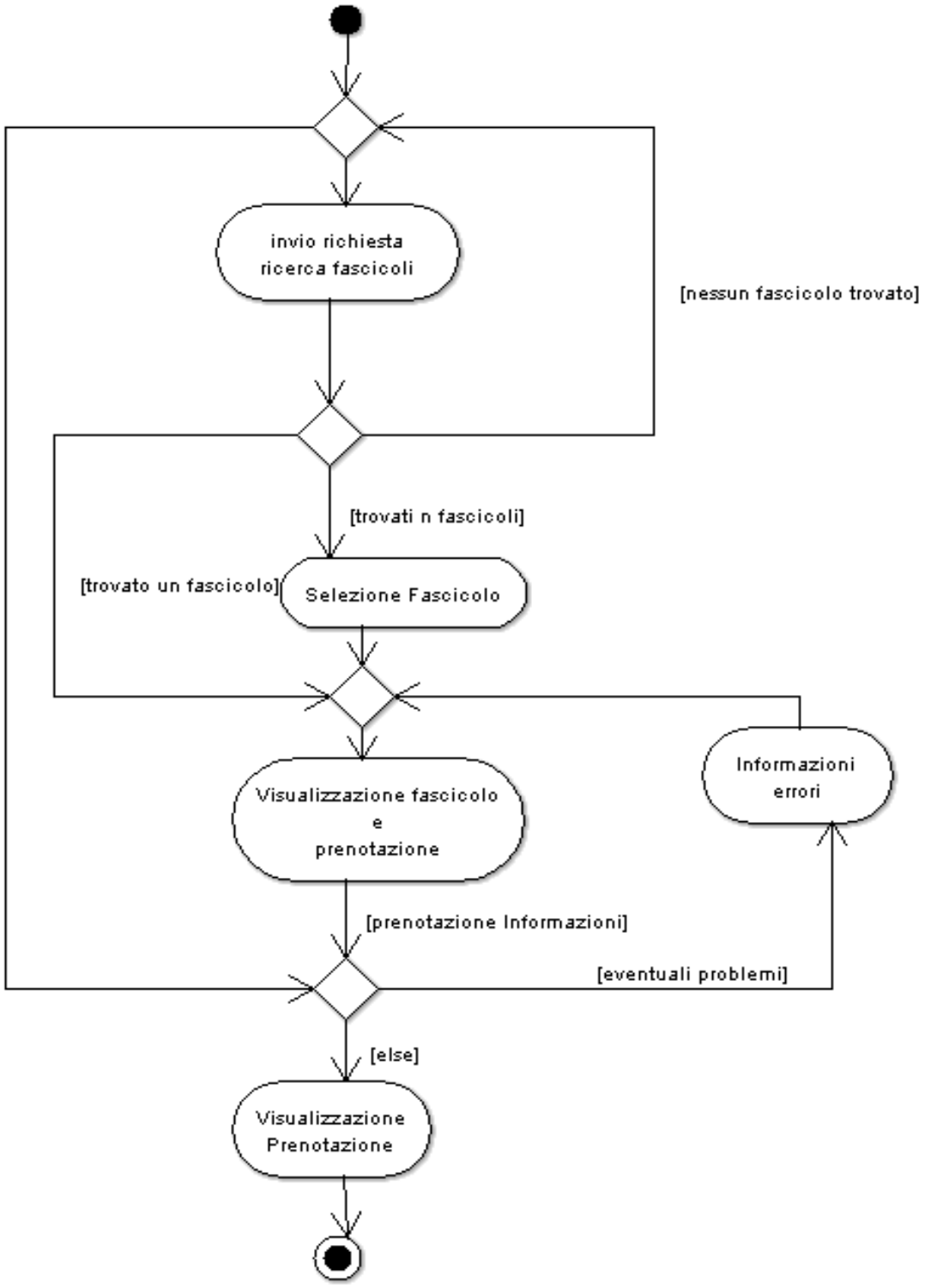


Figura 14 – Activity Diagram

1.6 La struttura

Introduzione

BDMC è stata implementata utilizzando il pattern MVC e si è avvalsa dei framework Apache Struts e Castor[7]; Struts è stato discusso abbondantemente nei capitoli precedenti, mentre Castor fornisce è strumento ORM, simile a Hibernate.

L'applicazione è stata organizzata in due moduli principali. Il primo risiede all'interno del Web Server Tomcat[11], mentre il secondo nell'Application Server di JBoss.

All'interno di Tomcat sono utilizzate le seguenti componenti:

- Le Pagine JSP;
- il framework Apache Struts;

mentre all'interno dell'Application Server risiedono:

- il framework Castor;
- gli EJB.

Pagine JSP

Il livello della presentazione è realizzato con l'impiego di Pagine JSP ed è completamente gestito all'interno del Web Server di Tomcat.

La struttura di ogni vista presentata all'utente è formata da varie sezioni. La seguente JSP:

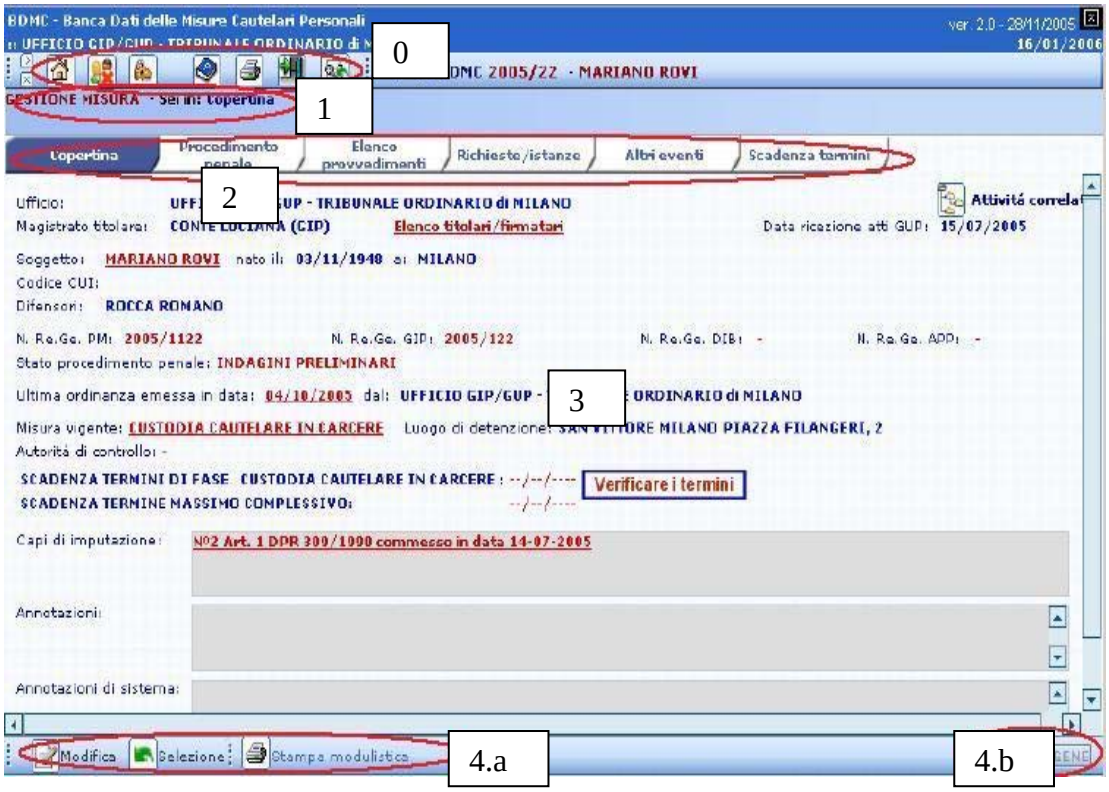


Figura 15 – Viene mostrata una Pagina JSP a titolo d'esempio per mostrare i vari elementi che compongono la vista per l'utente.

È costituita da diverse sezioni che possono essere così riassunte:

- (0) *Intestazione;*
- *(1) Percorso;*
- *(2) Menu;*
- *(3) Corpo;*
- *(4.a - 4.b) Footer.*

Ognuno di esse viene incluso all'interno di una unica Pagina JSP tramite le direttive di include.

Come emergerà dalle descrizioni dei singoli elementi che compongono il risultato finale mostrato all'utente, la divisione del corpo della JSP permette il conseguimento di diversi vantaggi, quali la modularità e la canalizzazione di controlli all'interno di singole pagine.

1.6.1.1 Intestazione

L'intestazione è comune a tutte le viste e permette all'utente di compiere varie scelte, quali il logout dall'applicazione, la modifica della password, la stampa

della schermata che si sta visualizzando, l'accesso alla pagina principale o la consultazione del manuale utente.

L'intestazione è comune a tutte le visualizzazioni e per tale motivo è unica ed inclusa in ogni Pagina JSP. La comprensione del vantaggio ottenuto è immediata, in quanto se per qualche motivo si volesse aggiungere (o modificare o eliminare) una funzionalità basterebbe modificare un solo punto dell'applicazione e non tutte le JSP dell'applicazione.

1.6.1.2 Percorso

Il percorso, detto anche cammino, è presente in tutte le viste e consiste nell'esplicazione della strada che si è percorsa per arrivare alla schermata che l'utente visualizza.

La figura seguente illustra un secondo esempio di percorso per il solo scopo di rendere più comprensibile la spiegazione.

Figura 16 – Viene mostrato il dettaglio del percorso di una Pagina JSP. Il percorso mostra la strada che l'utente ha percorso per arrivare alla schermata che è visualizzata.

Il percorso è realizzato con l'ausilio di documenti XML con la seguente struttura dati:

```
<nodo>
	<id>nomePagina.jsp</id>
	<idPadre>nomePaginaPadre.jsp</idPadre>
	<descrizione>Istanza</descrizione>
	<action>actionAssociata.do</action>
	<attivo>true</attivo>
</nodo>
```

Listato 4

Dove:

- *id* rappresenta l'identificativo del nodo;

- *idPadre* rappresenta l'identificativo del nodo padre, in questo modo si rende ricorsivo il cammino;
- *action* identifica l'azione da eseguire nel caso in cui l'utilizzatore dovesse selezionare il link;
- *attivo* indica se il link è attivo;
- *descrizione* è il testo riprodotto nella vista per l'utente.

Grazie all'impiego di questa soluzione è possibile aggiungere (modificare o eliminare) i percorsi in modo semplice.

Questa soluzione è realizzata tramite una JSP e una classe in grado di leggere il relativo file XML. In questo modo ogni volta che viene creato o modificato il cammino per una vista (nuova o esistente) il programmatore dovrà solo modificare il documento XML.

Inoltre grazie all'impiego di una unica Pagina JSP si rende la visualizzazione del cammino uniforme a tutte le viste e, come nel caso dell'intestazione, se si volesse modificare la presentazione basterebbe effettuare la modifica in un unico punto.

1.6.1.3 Menu

Il Menu mostra le possibili scelte che un utente può effettuare e la presentazione deve essere uniforme per tutte le viste.

Tutti i Menu sono stati "raccolti" all'interno di una Pagina JSP e per ogni schermata viene mostrato quello giusto.

Anche in questo caso la gestione delle opzioni è stata concentrata all'interno di una unica Pagina e come nei due casi precedenti si ottengono gli stessi vantaggi.

Inoltre grazie all'utilizzo delle librerie di Struts il contenuto del codice della Pagina JSP contenente il Menu è stato scritto senza una riga di codice Java. Il seguente Listato è un piccolo esempio.

```
<req:equalsAttribute name="iscrfasc" match="MISUCAUT">
  <td nowrap><bean:message key="fascmisucaut.schegene"/></td>
  <td><img src="/BDMC/web/graphics/menulink/BiancoBlu.gif"></td>
  <td nowrap id="selected"><bean:message
key="fascmisucaut.procedimento"/></td>
  <td><img src="/BDMC/web/graphics/menulink/BluBianco.gif"></td>
  <td nowrap><bean:message key="fascmisucaut.soggetto"/></td>
```

```
  <td><img src="/BDMC/web/graphics/menulink/BiancoBianco.gif"></td>
  <td nowrap><bean:message key="fascmisucaut.qualgiur"/></td>
  <td><img src="/BDMC/web/graphics/menulink/BiancoBianco.gif"></td>
  <td nowrap><bean:message key="fascmisucaut.ordinanza"/></td>
</req:equalsAttribute>
<req:equalsAttribute name="iscrfasc" match="FERMARRE">
  <td nowrap><bean:message key="fascmisucaut.schegene"/></td>
  <td><img src="/BDMC/web/graphics/menulink/BiancoBlu.gif"></td>
  <td nowrap id="selected">
    <bean:message key="fascmisucaut.procedimento"/>
  </td>
  <td>
    <img src="/BDMC/web/graphics/menulink/BluBianco.gif">
  </td>
  <td nowrap><bean:message key="fascmisucaut.soggetto"/></td>
  <td><img src="/BDMC/web/graphics/menulink/BiancoBianco.gif"></td>
  <td nowrap><bean:message key="fascmisucaut.qualgiur"/></td>
  <td><img src="/BDMC/web/graphics/menulink/BiancoBianco.gif"></td>
  <td nowrap><bean:message key="fascmisucaut.fermarre"/></td>
</req:equalsAttribute>
```

Listato 5

1.6.1.4 Corpo

Il Corpo è diverso per ogni vista e per ognuno ne esiste la rispettiva pagina JSP. È però possibile trovare al suo interno alcune JSP che gestiscono alcuni dettagli.

Ad esempio nelle specifiche è stato richiesto che tutte le date avessero la stessa forma e le stesse utility come ad esempio lo spostamento del focus da una casella all'altra o dell'auto-completamento. Per poter gestire tale situazione sono state create Pagine JSP in grado di contenere al loro interno solo questi pezzetti. In questo caso il vantaggio viene apprezzato nel caso in cui il cliente decidesse di cambiare il formato delle date, poiché basterebbe modificare un singolo file.

Le JSP che realizzano il corpo si devono preoccupare solo dei contenuti da mostrare e non della loro logica.

1.6.1.5 Footer

Il Footer contiene due informazioni; la prima (4.a) contiene le opzioni che l'utente può scegliere come ad esempio le classiche opzioni che permettono di salvare o di modificare le informazioni. La seconda informazione (4.b) contiene del semplice testo che permette di risalire in modo facile ed intuitivo alla Pagina JSP che si occupa della vista presenta all'utente.

Anche il Footer, come il Corpo è diverso per ogni schermata e per tale motivo ne esiste uno associato ad ogni corpo.

Apache Struts

La configurazione dei cammini, ovvero della determinazione della vista a cui passare il controllo, è gestita tramite l'unico file di configurazione *struts-config.xml*. suddiviso in tre frammenti logicamente distinti, cioè:

- *form-bean*: gestiscono i parametri delle form html;
- *global-forwards*: definisco i forward a livello globale;
- *action-mapping*: rappresenta la sezione in grado di gestire le Action da invocare.

Nella Figura 17 viene mostrato come sia stato implementato il modello MVC con l'utilizzo di Apache Struts.

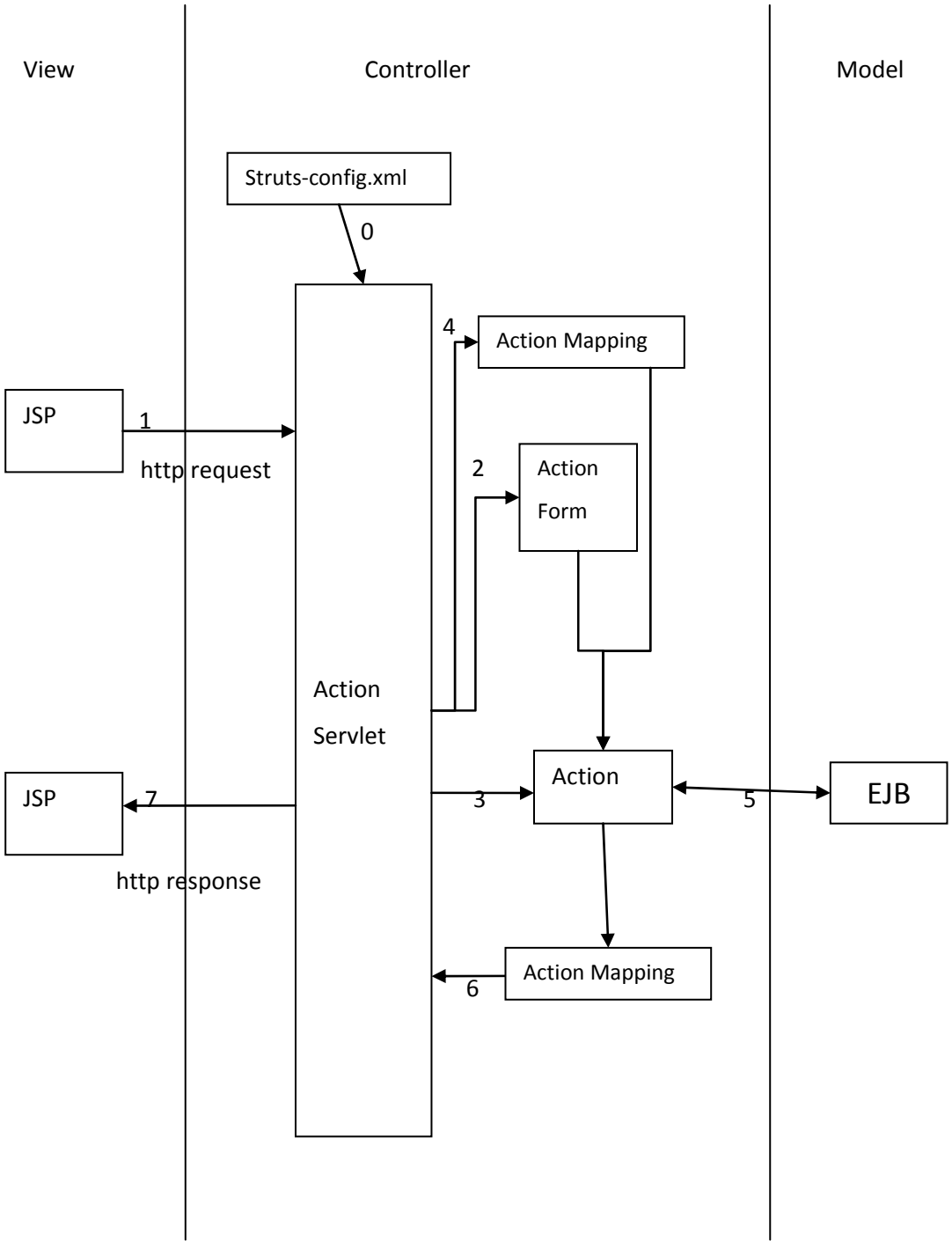


Figura 17 – viene mostrato il pattern MVC realizzato per lo sviluppo dell'applicativo BDMC.

Castor

Castor è il framework utilizzato all'interno dell'applicazione lato Web Server che si occupa della persistenza dei dati. È quindi una libreria che permette di rendere

persistenti oggetti Java su database relazionali, e di effettuare query relazionali ottenendo oggetti Java.

Appartiene alla categoria degli ORM che si occupano, come detto in precedenza, di mappare il modelli dei database relazionali con il modelli Object-Oriented e viceversa.

Il suo utilizzo ha permesso di gestire tutti i dati tramite codice Java, che è di più alto livello rispetto alle query SQL.

La relazione tra dati e oggetti è unita tramite un documento XML conosciuto con il nome *mapping.xml*.

La gestione dei dati tramite Castor permette notevoli vantaggi in fase di sviluppo e manutenzione in quanto è possibile non inserire nel file di mapping tabelle o campi di cui è stata gestita la sola creazione lato database. Sono inoltre "risolti" tutti quei problemi riguardanti le transazioni, la sicurezza in quanto è possibile lasciare tale gestione direttamente a Castor.

Gli oggetti Java che rappresentano la banca dati sono i POJO, che vengono utilizzati per tutte quelle azioni di modifica, di inserimento e di eliminazione dei dati.

Con l'ausilio di Castor è possibile inserire una nuova riga all'interno del db utilizzando il metodo *create()*, il quale accetta come unico parametro un oggetto che rappresenta la nuova riga. I vantaggi sono molti in quanto, come detto più volte, non viene scritto codice SQL e modificando, o aggiungendo, costruttori che inizializzano il POJO è anche possibile dargli una forma iniziale comoda allo sviluppatore. Ad esempio ogni qualvolta si inserisce

EJB

Gli EJB utilizzati all'interno di BDMC sono i Session Bean di tipo Stateless. Gestiscono le vere operazioni di business, quali ricerche, modifiche, inserimenti, calcoli e quant'altro. Il loro comportamento è descritto all'interno del file *ejb-.jar.xml*. Sono stati implementati diversi tipi di Session Bean, uno per ogni gruppo concettuale di business-logic. Il sistema è suddiviso in macroaree e per ogni macroarea è stato progettato un EJB.

2 Sviluppo e messa in opera

La manutenzione e lo sviluppo di alcune funzionalità di BDMC hanno previsto uno studio ben dettagliato dei casi d'uso e della struttura stessa dell'applicazione. Nel seguito verranno mostrate solo due attività svolte, per l'appunto la prima illustra due azioni di manutenzione, mentre la seconda ha previsto l'aggiunta di una funzionalità.

2.1 Manutenzione

La prima manutenzione presentata nel seguito ha previsto la revisione della modalità di scrittura del codice di un fascicolo. Il codice è rappresentato da due numeri, il primo rappresenta l'anno del fascicolo e il secondo il numero dello stesso.

In particolare è stata stilata la seguente nuova specifica:

- l'utente che scrive il codice del fascicolo se dovesse
    - scrivere le 4 cifre dell'anno del fascicolo il focus si deve spostare direttamente nella casella del numero del fascicolo
    - digitare 2 cifre dell'anno del fascicolo e nel seguito premere il tasto "tab" o spostare il focus l'anno del fascicolo deve auto completarsi in un anno di 4 cifre e il focus si deve spostare sul numero del fascicolo
        - se viene digitato un numero a due cifre il cui valore sommato a 2000 è minore dell'anno vigente più 30 allora l'anno diventa la somma di 2000 più il numero digitato
        - altrimenti l'anno diventa la somma di 1900 più il numero digitato.

Lo sviluppo di tale specifica ha previsto le seguenti operazioni:

- ricerca all'interno dell'applicazione di tutti i punti dove è possibile inserire il codice di un fascicolo;
- scrittura del nuovo codice.

La modifica ha previsto oltre che alla ricerca e alla scrittura di nuovo codice anche un modo unico di gestire tali campi. In questa ottica sono state effettuate le seguenti modifiche:

- scrittura del codice javascript all'interno dell'unico file contenente tali funzioni;
- creazione di una unica nuova Pagina JSP in grado di fornire la grafica e la gestione del codice del fascicolo.

Il codice javascript si occupa della trasformazione dell'anno e del focus dei campi; ne viene mostrato il codice nel seguenti Listati.

```
function transfAnno(obj) {
  var mYear = obj.value;
  if (mYear.length == 2) {
    var mToday = new Date();
    var checkYear = mToday.getFullYear() + 30;
    var mCheckYear = '20' + mYear;
    if (mCheckYear >= checkYear) {
      mYear = '19' + mYear;
    } else {
      mYear = '20' + mYear;
    }
    obj.value = mYear;
  }
}
```

Listato 6

```
function spostaFocusSuNumero(obj, nomeNumero) {
  var mYear = obj.value;
  if((mYear.length == 4) && (window.event.keyCode!=16 &&
window.event.keyCode!=9 && window.event.keyCode!=39 &&
window.event.keyCode!=37)) {
    document.getElementById(nomeNumero).focus();
  }
}
```

Listato 7

La Pagina JSP creata, con il nome *annonume.jsp*, contiene al suo interno il seguente codice:

```
<%
if(valoreAnno==null) {
  valoreAnno="";
}
if(valoreNumero==null) {
```

```
  valoreNumero="";
}
%>
<input name="<%=nomeAnno%>" id="<%=nomeAnno%>" type="text"
value="<%=valoreAnno%>"
onkeyup="javascript:spostaFocusSuNumero(this,'<%=nomeNumero%>','<%=n
omeNumero%>');"  onblur="javascript:transfAnno(this);"
onfocus="keyPressOnFocusAnnoNumero(this);" />
/
<input name="<%=nomeNumero%>" id="<%=nomeNumero%>" type="text"
value="<%=valoreNumero%>"
onfocus="keyPressOnFocusAnnoNumero(this);" />
```

Listato 8

Tutte le JSP dell'applicazione che devono utilizzare la pagina annonume.jsp la possono include utilizzando le direttive. Prima di includerla devono valorizzare il nome dei campi dell'anno e del numero ed eventualmente anche i relativi valori. Un esempio di come possa essere inserita la Pagina JSP è il seguente:

```
<%
{
    String  nomeAnno        = "annoRegiPmpm";
    String  valoreAnno      = annoRegiPmpm;
    String  nomeNumero      = "numeRegiPmpm";
    String  valoreNumero    = numeRegiPmpm;
    int     maxlengthAnno   = 4;
    int     maxlengthNumero = 6;
%><%@include  file="/web/annonume.jsp"%><%
}
%>
```

Listato 9

Così facendo se nel seguito dovessero ricambiare le specifiche riguardanti il codice del fascicolo sarà necessario cambiare la sola Pagina JSP.

La seconda modifica ha previsto l'aggiunta di una informazione per quel che riguarda le ordinanze. L'informazione aggiunta riguarda una data.

Tale variazione ha previsto il ritocco di diversi livelli dell'applicazione. La prima rettifica ha previsto la modifica del database (in quanto deve contenere la nuova data) e del relativo POJO (aggiungendo il nuovo campo con i relativi metodi di get e set); infine POJO e DB sono stati allineati modificando il file di mapping

In seguito sono state corrette le Pagine JSP relative alla modifica, all'inserimento ed alla visualizzazione dell'ordinanza inserendo semplicemente la nuova informazione. Il nuovo campo è stato inserito nel formbean di Struts per permettere la gestione del nuovo parametro.

Infine è stata modificata la Action relativa alla modifica ed all'inserimento in quanto si è reso necessario il reperimento della nuova data inserita e la valorizzazione del nuovo campo del POJO.

Gli EJB non hanno subito alcuna modifica in quanto, tramite l'ausilio del framework di Castor, gli inserimenti e le modifiche si effettuano passando l'intero oggetto, che incapsula tutte le informazioni inserite all'interno della relativa Action.

2.2 Aggiunta di una funzionalità

La funzionalità aggiunta è quella relativa alla prenotazione di un periodo di presofferto da parte di utente. È stato seguito lo schema di Figura 17 e precisamente sono state effettuate le seguenti azioni:

- aggiunta di:
    - Action;
    - Pagine JSP;
    - un Enterprice JavaBean;
- modifica:
    - del file di configurazione di Apache Struts;
    - degli EJB.

Poiché alcune funzionalità, quali la ricerca del fascicolo, la visualizzazione dei reati ed altre, erano state già implementate si è scelto di non aggiungere del codice che sarebbe risultato una copia quasi esatta, ma di modificare il file di configurazione di Struts in modo da aggiungere solo dei nuovi cammini che utilizzino le Action già scritte.

In pratica nel file struts-config.xml è stato eseguito il seguente lavoro:

cammino esistente:

```
<action path="/fascmisucaut/soggetto/visusogg"
type="org.apache.struts.webapp.BDMC.VisuSoggAction"
name="visusoggForm" scope="request"
input="/web/gestfasc/iscrfasc/inseschegene.jsp">
  <forward name="insert"
path="/web/gestfasc/soggetto/tiposoggetto.jsp" />
  <forward name="visupersfisi"
path="/web/gestfasc/soggetto/visupersfisi.jsp" />
  <forward name="visupersgiur"
path="/web/gestfasc/soggetto/visupersgiur.jsp" />
  <forward name="cancel"
path="/web/gestfasc/iscrfasc/inseschegene.jsp" />
</action>
```

Listato 10

nuovo cammino:

```
<action path="/isies/ricepresoffe/visupersfisi"
type="org.apache.struts.webapp.BDMC.VisuSoggAction"
name="visusoggForm" scope="request"
input="/isies/ricepresoffe/show.do">
  <forward name="visupersfisi"
path="/web/ISIES/ricePresOffe/visupersfisi.jsp"/>
  <forward name="errore" path="/web/ISIES/esito.jsp"/>
</action>
```

Listato 11

Una soluzione alternativa all'inserimento di un nuovo cammino potrebbe essere quella dell'aggiunta di un semplice elemento di tipo forward nel cammino già esistente, ma questo comporterebbe la modifica della Action con la conseguenza di dover rieffettuare il test su elementi già in uso e perfettamente funzionanti. Con il secondo cammino, al contrario, non sono state effettuate modifiche alla Action esistente, si è utilizzato la stessa parola chiave per il forward e ovviamente la destinazione finale è diversa.

La ricerca dei periodi di prenotabili per il presofferto già implementata in un EJB non soddisfaceva a pieno i requisiti per un corretto inserimento, per questo motivo è stata aggiunta della logica. In particolare per ogni periodo di prenotazione è

necessario tener traccia di varie informazioni, quali il fascicolo di provenienza della data selezionabile per il presofferto. Poiché il metodo di business è utilizzato solo in questa sezione è stato possibile effettuare una modifica e non un'aggiunta di codice.

Per tutta la nuova macroarea è stato aggiunto un EJB, di tipo Session Stateless, in grado di gestirla. L'EJB è stato aggiunto nel file di configurazione *ejb-jar.xml* nel seguente modo:

```
<session>
  <display-name>GestorePresofferto</display-name>
  <ejb-name>GestorePresofferto</ejb-name>
  <home>GestorePresoffertoHome</home>
  <remote>GestorePresofferto</remote>
  <ejb-class>GestorePresoffertoBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
</session>
```

Listato 12

Il seguente listato mostra alcuni metodi di business implementati.

```
1  public Prenotazione selezionaPrenotazioneAttiva(String
2  username, Integer idPren) throws Exception {
3    return selezionaPrenotazioni(username, "001", idPren);
4  }
5  public Prenotazione selezionaPrenotazioneAnnullata(String
6  username, Integer idPren) throws Exception {
7    return selezionaPrenotazioni(username, "002", idPren);
8  }
9  private Prenotazione selezionaPrenotazioni(String username,
10 String flagPren, Integer idPren)
11                                        throws Exception {
12   InitialContext ic = new InitialContext();
13
14   DataObjects      _jdo = new JDO();
15   _jdo = (DataObjects) ic.lookup("java:/jdo/BdmcJDO");
16
17   Database      db;
18   OQLQuery      userOql;
```

```
19    QueryResults  results;
20
21    Prenotazione prenotazione = null;
22
23    db = _jdo.getDatabase();
24
25    javax.transaction.UserTransaction ut =
26  sessionContext.getUserTransaction();
27    ut.begin();
28    db.begin();
29
30    try {
31      java.util.Vector prenotazioni = new java.util.Vector();
32      String query = "SELECT a FROM Prenotazione a WHERE
33  a.flagPren = $1 AND a.utente = $2 "
34        +((idPren!=null)? "AND a.idPren = $3 ":"")+
35        "ORDER BY a.idPren";
36      userOql = db.getOQLQuery( query );
37      userOql.bind(flagPren);
38      userOql.bind(username);
39      if(idPren!=null) {
40        userOql.bind(idPren);
41      }
42      results = userOql.execute( Database.ReadOnly );
43
44      if ( results.hasMore() ) {
45        prenotazione = ( (Prenotazione) results.next() );
46      }
47      results.close();
48      userOql.close();
49
50    } catch (Exception ex) {
51      db.rollback();
52      db.close();
53      ut.commit();
54    }
```

```
55    return prenotazione;
56  }
```

Listato 13

Nel Listato 13 viene mostrato come si cerca di eliminare il codice ripetuto e di separare la logica di business dai livelli sottostanti. Ad esempio è possibile voler cercare una prenotazione annullata o valida. Per evitare che le Action debbano specificare, e quindi conoscere, il valore di alcuni campi, sono stati esposti metodi che se ne occupano direttamente. I metodi di riga 1 e di riga 5 non chiedono il valore del flag che sancisce se una prenotazione è valida o meno, ma lo gestiscono all'interno del proprio corpo e invocano un metodo privato a cui passare il valore giusto. Così facendo si ottengono vantaggi anche in fase di manutenzione in quanto basterà modificare pochi valori in pochi metodi.

2.3 Messa in opera

Per fare funzionare l'applicazione BDMC è necessario installare i seguenti moduli:

- JBoss
- Web Server Tomcat
- DBMS.

Nel seguito sarà illustrata l'installazione dei primi due moduli. Per installare l'Application Server JBoss è sufficiente decomprimere il file compresso *jboss.zip* in una cartella. Dopodiché va inserita la variabile di ambiente *JBOSS_HOME* (uguale alla cartella contenente JBoss).

L'installazione dell'applicazione BDMC sull'application server si effettua con i seguenti passi:

- installazione del plugin JDO di Castor copiando il file *CastorJDOPlugin.jar* nella cartella *%JBOSS_HOME%\server\default\lib;*
- collegamento alla base di dati copiando il file *database.xml* contenente i parametri per la connessione al DB nella cartella *%JBOSS_HOME%\server\default\conf*;
- installazione del container copiando nella cartella *%JBOSS_HOME%\server\default\conf* il file *BDMCEjbContainer.jar*.

Per installare la componente Web Server di Tomcat è sufficiente mandare in esecuzione il programma di installazione. È necessario però che il nome della cartella contenente Tomcat sia senza spazi o caratteri accentati. Al termine dell'installazione vanno inserite le seguenti variabili d'ambiente:

- *CATALINA_HOME* uguale alla cartella di installazione di Tomcat;
- *TOMCAT_HOME* uguale alla cartella di installazione di Tomcat.

Per installare l'applicazione BDMC si deve copiare il file *BDMC.war* nella cartella *%TOMCAT_HOME%/webapps*. Per default l'applicazione BDMC sul Web Server si connette all'Application Server sulla stessa macchina. Se l'AS non si dovesse trovare sulla stessa macchina sarà necessario impostare il giusto indirizzo IP.

Per modificare l'IP è sufficiente utilizzare un qualsiasi programma in grado di leggere i formati ZIP per aprire il file BDMC.war ed inserire l'indirizzo nel file *ApplicationResources.properties*.

3 Conclusioni

BDMC è un progetto di dimensioni notevoli: il database è composto da 163 tabelle, da 66 viste e da diversi trigger, per ogni tabella e per ogni vista è stato realizzato il relativo POJO, sono stati implementati 10 SB di oltre 20000 righe di codice ciascuno, sono state scritte 564 Action, circa 20 classi di utilità, il file di configurazione struts-config.xml è di oltre 11500 righe e infine tutti i controlli javascript sono stati inseriti all'interno di un unico file di circa 4000 righe.

Per ogni intervento il primo lavoro da svolgere è stato quello di verificare che il codice che si stava inserendo non fosse stato già scritto, e date le sopra citate dimensioni, questo compito non sempre è stato semplice.

Il vantaggio principale della politica adottata per lo sviluppo dell'applicativo BDMC sta nel fatto che una volta studiata l'applicazione e la sua struttura, la manutenzione diviene un po' più semplice; aiuta lo sviluppatore nel momento in cui si debbano prendere delle decisioni riguardanti, ad esempio, l'aggiunta di features, gli permette di intuire, e quindi conoscere, la posizione del codice dove compiere la manutenzione.

Tuttavia, se raggruppare il codice da un lato può essere utile, dall'altro può recare diverse difficoltà. Ad esempio spostarsi all'interno di file lunghi migliaia di

righe è spesso faticoso sia per lo sviluppatore che per gli strumenti di lavoro utilizzati causando, a volte, anche il blocco del computer.

Una seconda critica può essere mossa nei confronti di Castor, il quale presenta diverse carenze, a volte anche elementari. Non è in grado di invocare chiamate di procedura gestite dal DBMS (come le PL/SQL), non permette la realizzazione di query con un minimo di complessità, ad esempio non si possono scrivere join o semplici ricerche che prevedano il soddisfacimento dei criteri di ricerca tra più tabelle.

D. TurismoPublisher

1 Analisi e Progettazione

1.1 Introduzione

TurismoPublisher è un Web Service che permette ai propri utilizzatori la ricerca delle varie strutture turistiche della Regione Toscana.

Il Web Service è stato realizzato utilizzando il framework Axis mentre per la persistenza dei dati è stato utilizzato il framework di Hibernate, entrambi descritti in precedenza.

Il linguaggio scelto per lo scambio dei dati è quello dell'XML, in quanto sin dall'inizio si è proposto come linguaggio di modellazione dei dati grazie al fatto che consente di esprimere dati e metadati (ossia l'informazione sui dati, rappresentata dagli elementi) Pertanto possiamo stabilire uno schema XML che modelli i nostri dati.

Le fasi di sviluppo hanno seguito il tipico ciclo di vita del software secondo il modello a cascata e possono essere riassunte nel seguente modo:

- studio di fattibilità;
- analisi e specifica dei requisiti;
- progettazione;
- codifica ed implementazione;
- testing;
- messa in esercizio;
- manutenzione.

Lo studio di fattibilità in questo caso è stato immediatamente superato in quanto non sono servite considerazioni riguardanti le possibili soluzioni da adottare, o le risorse finanziarie ed umane a disposizione, tenendo conto che sono stati fissati a priori gli strumenti di base da utilizzare.

L'analisi e la specifica dei requisiti ha previsto la realizzazione del Documento di Specifica dei Requisiti Software (*SRS*) elencando le funzionalità fornite dal sistema software all'utente.

Nella fase di progettazione sono stati realizzati i principali diagrammi *UML*[24] previsti per lo sviluppo:

- Class Diagram;
- Use Case Diagram;
- Sequence Diagram;
- Activity Diagram.

Inoltre sono stati realizzati i seguenti modelli:

- Data Model;
- Physical Data Model;

per la descrizione concettuale e fisica della base dati utilizzata dal Web Service.

Lo sviluppo ha previsto due fasi distinte realizzando l'applicazione con l'ausilio dei framework. La fase di testing ha permesso la correzione di errori e problemi rilevati durante il funzionamento del software.

1.2 Requisiti

Il caso di studio preso in esame prevede lo sviluppo di un Web Service che permette ad un utente di poter richiedere informazioni su una o più strutture turistiche. Le strutture per le quali è prevista la gestione sono state raggruppate nelle seguenti categorie:

- Affittacamere
- Affittacamere non professionali
- Albergo
- Residenza Turistico Alberghiera
- Area di sosta
- Stabilimento balneare
- Campeggio - Parco vacanza
- Villaggio Turistico
- Casa per vacanze
- Ostello

[24] Unified Modelling Language

- Casa per ferie
- Residenza d'epoca
- Residence
- Rifugio alpino – escursionistico
- Agriturismo

Gli utilizzatori possono specificare i criteri di ricerca mediante un documento di tipo XML; la struttura di tale file è descritta tramite un file di tipo XSD.

L'utilizzatore può specificare i seguenti criteri, obbligatori, di ricerca:

- la provincia di appartenenza della/e struttura/e da ricercare;
- se si vogliono avere solo le seguenti informazioni:
    - nome della struttura;
    - indirizzo;
    - recapito;
    - posti letto;
    - camere;
    - la descrizione delle unità abitative;
    - la descrizione dei servizi;
    - i prezzi;
- se si tratta di dati storici o meno.

Inoltre ha a disposizione criteri opzionali di ricerca, come:

- la data di inizio intervallo sul quale effettuare le ricerche; tale data è riferita a quella del censimento della struttura;
- la data di fine intervallo sul quale effettuare le ricerche; tale data è riferita a quella del censimento della struttura;
- il genere della struttura da ricercare che può essere dei seguenti tipi:
    - Affittacamere (nel seguito *AFR*);
    - Affittacamere non professionali (nel seguito *ALL*);
    - Albergo (nel seguito *ALB*);
    - Residenza Turistico Alberghiera (nel seguito *RTA*);
    - Area di sosta (nel seguito *AST*);
    - Stabilimento balneare (nel seguito *STB*);

  - Campeggio - Parco vacanza (nel seguito *CAM*);
  - Villaggio Turistico (nel seguito *VIT*);
  - Casa per vacanze (nel seguito *CAV*);
  - Ostello (nel seguito *OST*);
  - Casa per ferie (nel seguito *CAF*);
  - Residenza d'epoca (nel seguito *REP*);
  - Residence (nel seguito *RES*);
  - Rifugio alpino – escursionistico (nel seguito *RAL*);
  - Agriturismo (nel seguito *AAT*);
- il comune di appartenenza della struttura;
- il codice *APT* (acronimo di Aziende Provinciali Turismo) indicante l'APT di competenza la struttura da cercare;
- la classificazione della struttura, ad esempio se si trattasse di un albergo questa informazione rappresenterebbe il numero di stelle;
- il codice di esercizio della struttura, il quale rappresenta in modo univoco la struttura.

La risposta da fornire al richiedente è contenuta all'interno di un documento di tipo XML con le informazioni sulla/e struttura/e. Come nel caso della domanda, anche la risposta deve essere valida secondo lo schema di un documento XSD. I documenti XSD sono di dominio pubblico, in questo modo Regione e Provincia possono lavorare in modo coordinato.

Ogni risposta deve contenere le seguenti informazioni:

- le varie strutture dove ciascuna struttura contiene le seguenti informazioni[25]:
  - se il documento restituito è un messaggio di test o meno;
  - il tipo della struttura contenente due gruppi di informazioni:
    - i dati anagrafici contenenti le seguenti informazioni:
      - il codice dell'esercizio;
      - la provincia;

[25] Data l'enorme mole di dati che devono essere forniti saranno mostrati solo gli aspetti più salienti. Per un maggior dettagli si possono consultare documenti XSD (Codice 2 e Codice 3)

- il genere della struttura;
- il tipo della struttura (definisce le sottotipologie della struttura);
- l'anno relativo alla presentazione delle attrezzature e dei prezzi;
- la tipologia di comunicazione che può essere di vari tipi:
  - Principale, di validità immediata,
  - Annuale (indica che è valida a partire dall'inizio di un dato periodo di riferimento),
  - Suppletiva (indica che è valida a partire dall'inizio di un dato periodo di riferimento),
  - Variazioni (indica che la validità è immediata),
  - Chiusura di una struttura,
  - Correzioni di comunicazioni già inviate;
- la denominazione dell'esercizio;
- l'indirizzo;
- l'indirizzo da utilizzare per i periodi di chiusura;
- il recapito (eventuale informazioni quali sito web, indirizzo di posta elettronica, numero di telefono, di cellulare e o di fax);
- i dati del titolare e del gestore;
- informazioni sull'apertura;
- la percentuale di accessibilità per i portatori di handicap;
- il nominativo della persona che ha inviato i dati che sono stati censiti
- la data della presentazione della struttura

- i dati relativi alla struttura.

1.3 Use Case Diagram

I diagrammi dei casi d'uso (Use Case Diagram) costituiscono uno strumento utile per catturare il comportamento esterno del sistema da sviluppare senza dover

specificare come tale comportamento debba essere realizzato; il sistema è visto come una scatola nera.

Gli Use Case Diagram forniscono una descrizione dei modi in cui il sistema potrà essere utilizzato, ossia ciò che l'utilizzatore può fare su di esso. La realizzazione dei diagrammi ha previsto un approccio top-down, partendo dallo scenario di carattere generale fino ad una decomposizione in cui sono descritti i casi di utilizzo.

Use Case Diagrams del TurismoPublisher

L'Use Case Diagram del Web Service TurismoPublisher prevede il semplice diagramma mostrato di seguito:

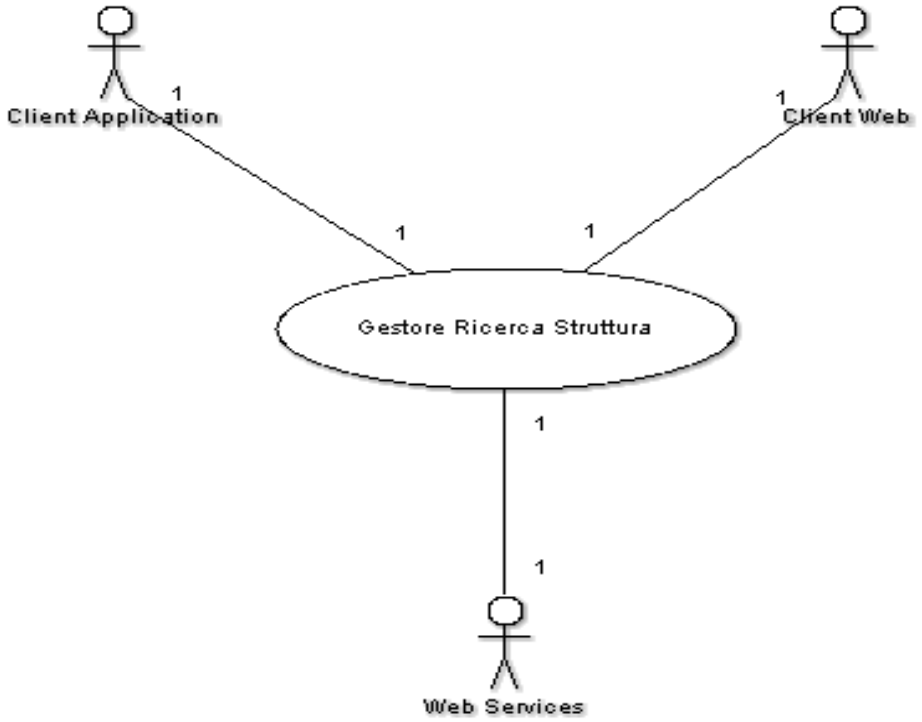


Figura 18 – Use Case Diagrams TurismoPublisher.

Esistono diversi attori che hanno lo stesso ruolo e non esistono "poteri". A partire da questo diagramma è possibile effettuare la decomposizione, se pur semplice ed intuitiva, scendendo in dettaglio ad un livello di astrazione minore.

Gestore Ricerca Struttura

La modalità "*Gestore Ricerca Struttura*" è relativa all'azione che l'utilizzatore può eseguire, ossia la ricerca della struttura e può essere considerata atomica.

1.4 Class Diagram

Mediante il Class Diagram è possibile definire tutte quelle che sono le entità caratteristiche del sistema software e le relazioni che ci sono tra di esse. Ciascuna entità è modellabile attraverso il concetto di classe che ne definisce le caratteristiche ed il comportamento (campi e metodi).

Anche il Class Diagram può essere realizzato a diversi livelli di astrazione, passando da un livello elevato ad un livello più basso e strettamente legato alla fase di implementazione e quindi relazionato al framework ed al linguaggio utilizzato. È possibile modellare la vista statica del sistema tenendo anche conto di quelle che sono le funzionalità offerte da quest'ultimo e rese disponibile attraverso le entità che lo compongono.

Sulla base delle specifiche definite per l'applicazione, il sistema prevede le seguenti entità e le relative classi che le modellano:

- *WebService*: Riceve il messaggio contenente i criteri di ricerca e ritorna il messaggio contenente le strutture turistiche che soddisfano tali criteri.
- *CreaMsgXml*: Crea e valida il messaggio di risposta a partire dalle strutture trovate.
- *Publisher*: Interroga la base dati e restituisce gli oggetti rappresentanti le strutture.
- *QueryString*: Classe di utilità per la creazione delle query per interrogare il database.

Di seguito viene mostrato il relativo Class Diagram.

Publisher
elencoStrutture(provincia_ricerca : String,dati_pubblici : String,dati_storici : String,data_inizio_intervallo : Date,data_fine_intervallo : Date,generestruttura_ricerca : String,comune_ricerca : String,apt : String,classificazione_ricerca : String,codicestruttura : String) : List createQueryString(session : org.hibernate.Session,provincia_ricerca : String,dati_pubblici : String,dati_storici : String,data_inizio_intervallo : Date,data_fine_intervallo : Date,generestruttura_ricerca : String,comune_ricerca : String,apt : String,classificazione_ricerca : String,codicestruttura : String) : org.hibernate.Query getStrutture(generestruttura_ricerca : String) : QueryString

Figura 19 – Class Diagram TurismoPublisher.

1.5 Activity Diagram

Riferendoci alla funzionalità offerta dal sistema e alla modalità di utilizzo, attraverso gli Activity Diagram è possibile modellare le azioni necessarie per compiere un'attività ed il flusso di controllo tra di esse. Ciascuna "activity" rappresenta un'azione che viene eseguita dal sistema oppure dall'utilizzatore e può essere atomica o scomposta in più "action" elementari. Nella gestione del flusso di controllo, è possibile definire sia esecuzioni parallele di operazioni che sincronizzazione, nonché esecuzioni condizionali. Inoltre, attraverso lo strumento dell'object flow è possibile specificare quali sono gli oggetti del sistema che entrano in gioco per ciascuna azione eseguita e quale sia lo stato associato.

È stato definito un Activity Diagram di carattere generale, che si pone ad un elevato livello di astrazione e descrive tutte le possibili azioni ed il relativo flusso che possono essere eseguite da un utilizzatore del sistema.

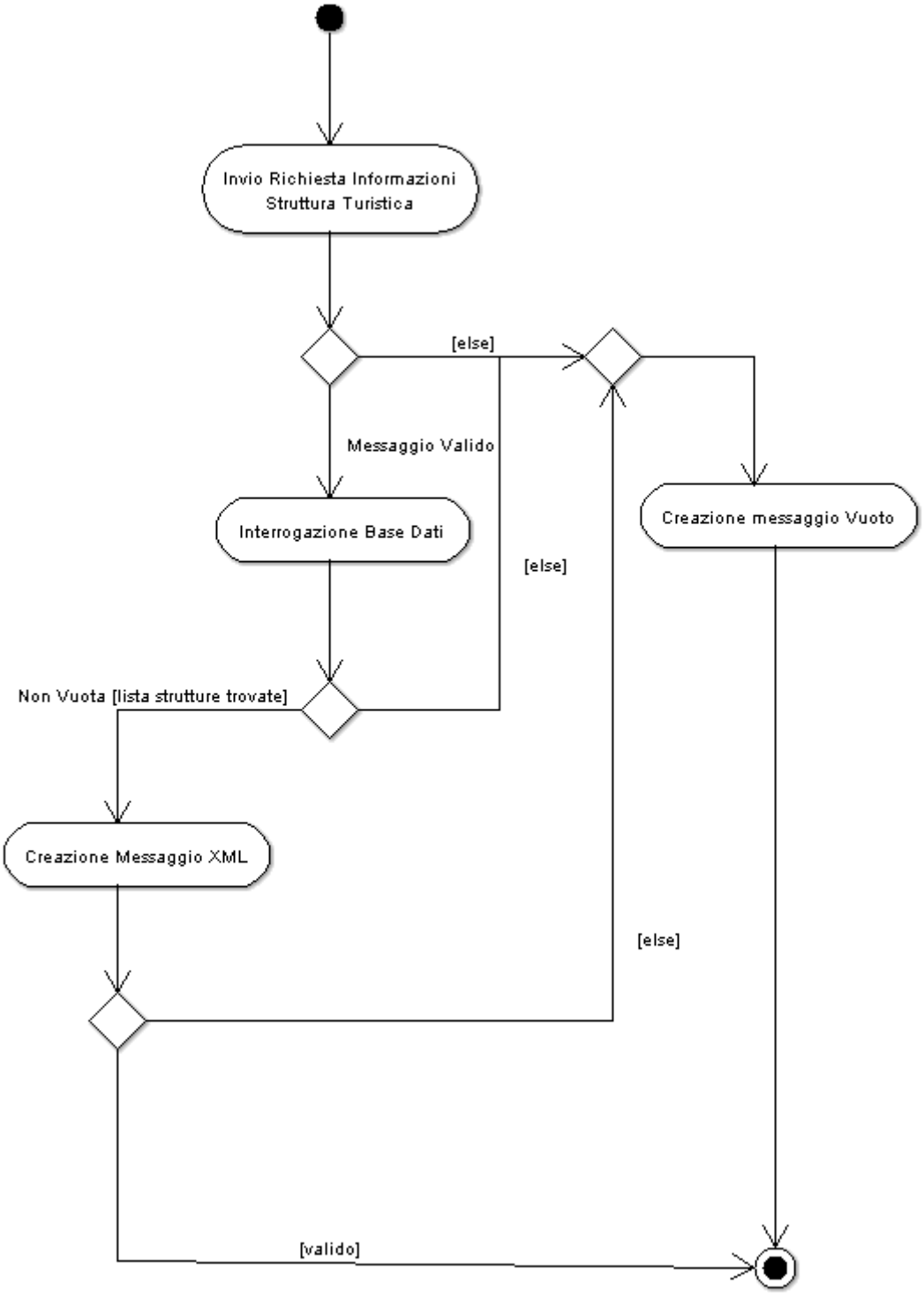


Figura 20 – Activity Diagram.

Vediamo nel dettaglio l'azione che riguarda la creazione del messaggio XML abbassando il livello di astrazione.

Creazione Messaggio XML

Le azioni svolte, durante la creazione del messaggio di risposta contenente le informazioni delle strutture turistiche che soddisfano i criteri di ricerca selezionati dall'utilizzatore del servizio, sono le seguenti:

- Creazione messaggio xml;
- Validazione del messaggio.

Per ogni struttura che soddisfa i parametri di ricerca, viene costruita la relativa sezione XML e immediatamente viene eseguita la validazione. Il seguente Activity Diagram mostra i passaggi di quanto appena descritto.

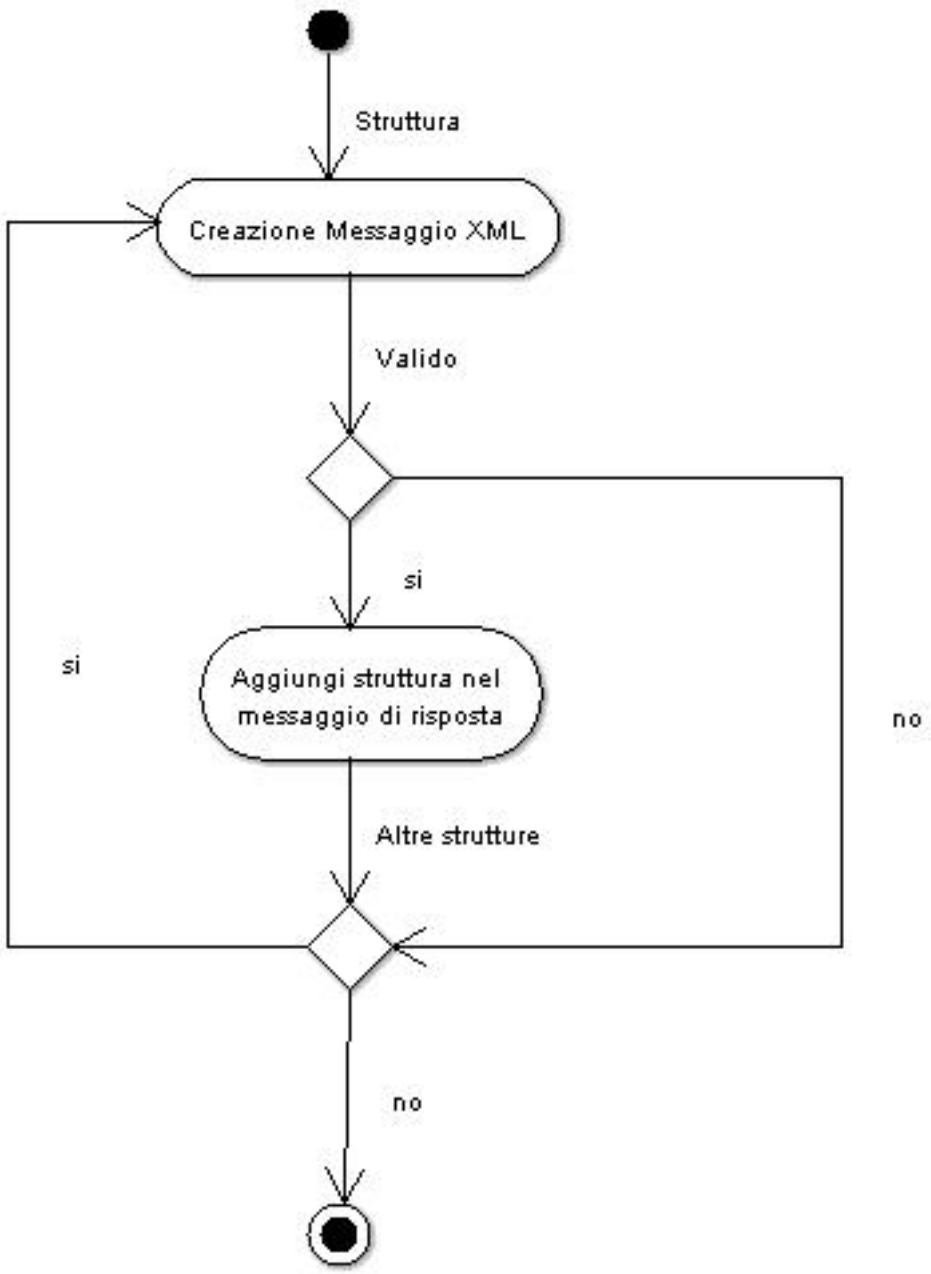


Figura 21 – Activity Diagram Creazione Messaggio XML.

1.6 Conceptual Data Model

La definizione del modello concettuale rappresenta il primo passo verso la progettazione della base di dati. Esso permette di descrivere tutte quelle che sono le entità del mondo reale e le relazioni che ci sono fra esse. Inoltre, per ciascuna entità o relazione che sia, è possibile definirne gli attributi caratteristici. In un certo senso, lo stesso Class Diagram può essere considerato in moltissimi casi il modello concettuale di un database, poiché le entità e le relazioni rappresentate sono praticamente le stesse, a meno di un formalismo differente.

Facendo riferimento al Web Service in esame, le entità del modello sono le seguenti:

- Affittacamere;
- Affittacamere non professionali;
- Albergo;
- Residenza Turistico Alberghiera;
- Area di sosta;
- Stabilimento balneare;
- Campeggio;
- Parco vacanza;
- Villaggio Turistico;
- Casa per vacanze;
- Ostello;
- Casa per ferie;
- Residenza d'epoca;
- Residence;
- Rifugio alpino – escursionistico;
- Agriturismo.

Per ogni entità ne esistono altre che ne forniscono il completo dettaglio. Nella figura seguente è mostrata una sola sezione relativa a due sole strutture, in particolare quella degli alberghi e quella dei residence.

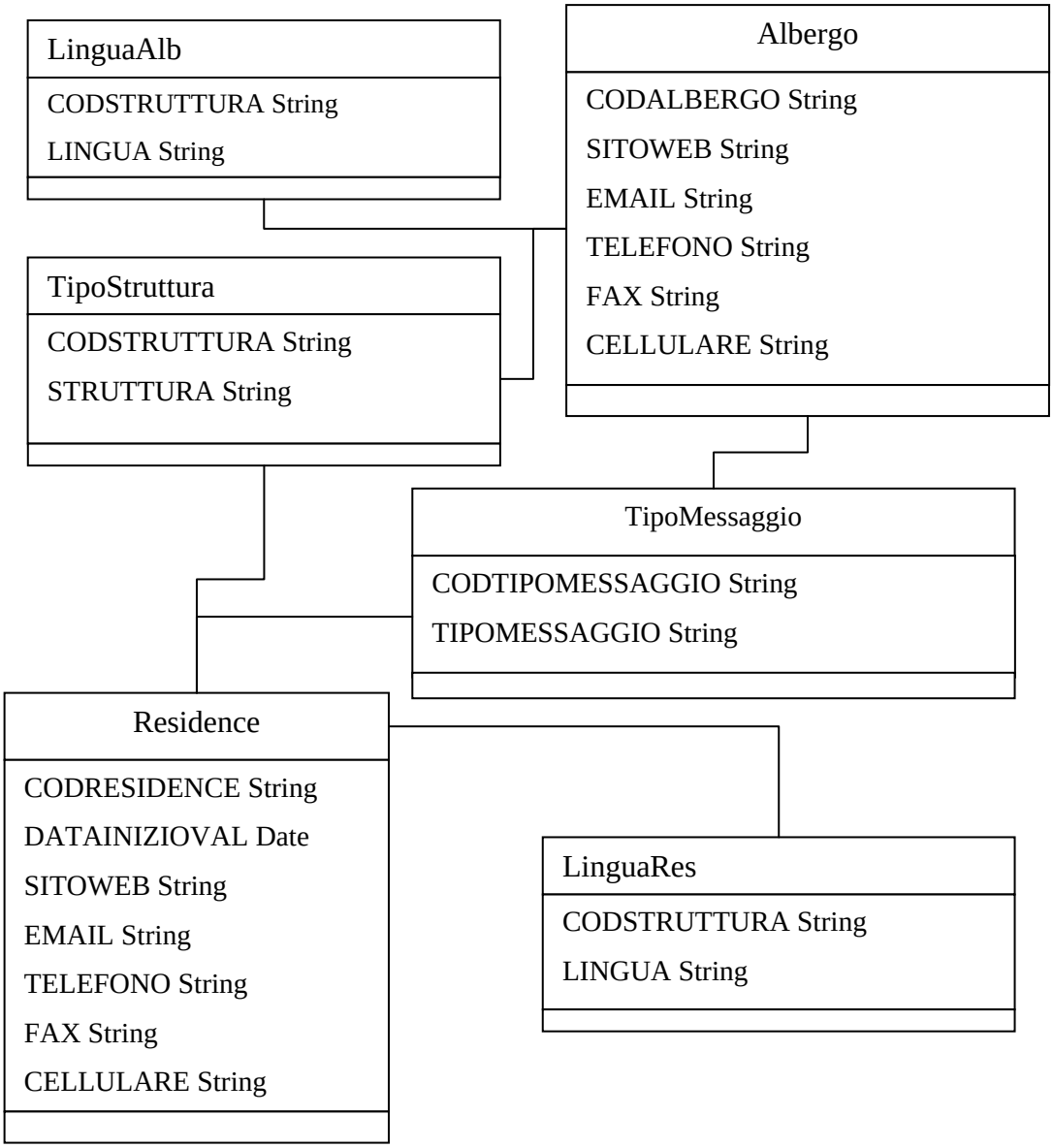


Figura 22 – Conceptual Data Model

1.7 Physical Data Model

Per completare la progettazione della base di dati, è necessario effettuare la trasformazione del modello concettuale nel modello logico, detto anche modello fisico. Quest'ultimo è costituito da una serie di tabelle e le relative colonne che prendono il posto delle entità e delle relazioni con i relativi attributi, presenti nel modello concettuale. La trasformazione viene eseguita sulla base di una serie di regole, che permettono di adattare il mondo reale, rappresentato dal modello concettuale, all'implementazione fisica che verrà successivamente adottata per uno

specifico *DBMS*[26]. La seguente figura mostra come sia stato disegnato il modello fisico.

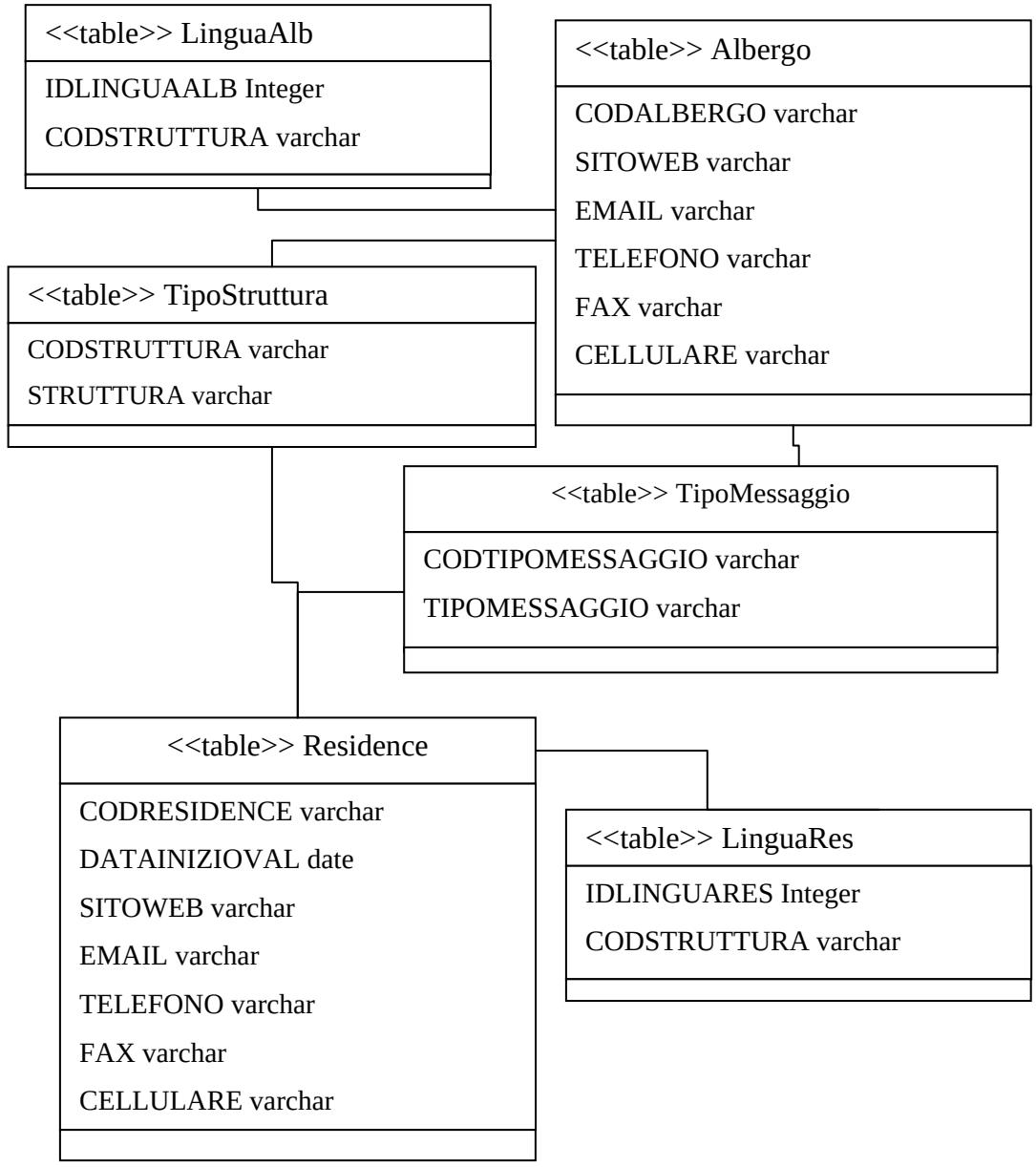


Figura 23 – Physical Data Model.

[26] Data Base Management System

1.8 Struttura dell'applicazione

Il Web Service TurismoPublisher è stato implementato utilizzando la piattaforma J2EE e i seguenti strumenti:

- Hibernate;
- Server Apache Axis;
- *XMLBeans*;
- *Log4j*;

Questi strumenti sono riconosciuti standard "*de facto*" che garantiscono modularità, apertura, flessibilità, integrazione e possibilità per le applicazioni sviluppate in questi ambienti di evolvere e crescere nel tempo, premesse per la realizzazione di progetti di alta qualità.

Inoltre tale scelta ha portato diversi vantaggi, quali la semplice gestione della base dati, la messa in opera del servizio e la creazione del messaggio XML di risposta.

Il primo aspetto considerato riguarda la struttura dell'applicazione, in termini di moduli che la compongono.

Il Web Service è implementato come una Servlet grazie all'utilizzo del framework Axis, che permette di ottenere il documento WSDL il quale è in grado di fornire la descrizione del servizio. La configurazione è realizzata per mezzo del file *web.xml* dell'Applicazione Web che ospita il servizio. La Web Application intercetta le richieste HTTP e le passa al WS che elabora le risposte da fornire. Tutto ciò avviene in modo trasparente al programmatore che potrà dedicarsi direttamente alla programmazione.

Per quel che concerne la connessione alla base dati è stato utilizzato il framework Hibernate, che ha permesso la gestione della persistenza in modo trasparente al programmatore. Il mapping tra database e classi java è stato realizzato mediante file XML (uno per ogni tabella con un unico file di raccordo) e le classi generate seguono le regole dei POJO.

Per la generazione della risposta, da fornire all'utente tramite documento di tipo XML, è stata utilizzata una libreria che permette di:

- generare la struttura ad oggetti a partire da un file XML;
- generare un file XML a partire da una struttura ad oggetti.

Con l'utilizzo di questa libreria è possibile creare un file XML a partire da una struttura ad oggetti, creata sempre per mezzo della stessa libreria. Anche, in questo caso quindi, il tutto avviene in modo completamente trasparente al programmatore.

L'ultimo strumento utilizzato è Log4j, libreria della Apache che permette di gestire il log. Anche questo aspetto viene reso trasparente al programmatore.

Abbiamo visto come l'impiego di tutte queste componenti permetta al programmatore di dedicarsi solo alla *semplice* implementazione una volta che i requisiti siano ben definiti.

XMLBeans

XMLBeans[14] è una libreria di Apache Group, di tipo Open Source che permette l'accesso alle piene funzionalità di XML utilizzando Java come linguaggio di programmazione.

L'idea di base è quella di utilizzare le caratteristiche di XML e di XSD e di ottenere una relazione tra XML e Java.

XMLBeans utilizza lo schema XML per compilare delle interfacce e classi Java che possono essere utilizzate per accedere al file XML mediante i classici metodi get() e set().

Con l'ausilio di XMLBeans si ottiene un supporto completo verso lo schema XSD ed una corrispondenza con le classi Java. Il programmatore non utilizza il file XML, ma classi Java molto simili ai POJO. Ciò permette di manipolare il contenuto di un file XML e di gestire in modo automatico le operazioni di lettura, scrittura e validazione. Rispetto ad altre tecnologie come *jDOM*[27] e *Xerces*[28], XMLBeans non fornisce una libreria generica, ma crea una vera e propria libreria dedicata alla gestione di un certo tipo di file XML definito tramite un XML Schema.

---

[27] API open source che fornisce uno strumento per la gestione di documenti XML.

[28] libreria, sviluppata in seno all' Apache Xml Project, che permette di creare e analizzare documenti scritti in liguaggio XML. Xerces è una implementazione delle API JAXP (Java Api for Xml Processing) che troviamo in ogni JDK dalla versione 1.3 in poi.

Facciamo alcuni esempi per capire meglio quali siano i vantaggi apportati da questa libreria. Il primo beneficio si ottiene già in fase di sviluppo. Se dovessimo creare il file XML manualmente dovremmo specificare tutti i TAG XML sempre in modo manuale nel codice Java e poi una volta generato il file XML dovremmo validarlo per controllare se gli elementi siano stati scritti correttamente; mentre con l'utilizzo della libreria questi errori verrebbero scoperti in fase di compilazione, e con l'utilizzo di un qualsiasi Ide di programmazione questi errori (o probabili "sviste") sarebbero riscontrabili immediatamente (e con il "completamento automatico" si eliminerebbero sul nascere). Oltre che ad essere un vantaggio è anche un sostegno al programmatore.

Un secondo vantaggio si ottiene nella fase di manutenzione, infatti, è altamente probabile che il file XSD subisca delle modifiche nel corso del tempo. Se, pertanto, dovesse subire dei cambiamenti di qualsiasi genere (ad esempio il nome o il tipo di un elemento) il programmatore dovrebbe:

- ricontrollare tutto il codice;
- riprodurre, almeno, un esempio di file XML;
- validarlo.

Al contrario, utilizzando XMLBeans si avrebbero errori di compilazione, per esempio se dovesse cambiare il tipo o semplicemente il nome di un elemento si avrebbero errori nei metodi di set e get in quanto la nuova libreria creata per il nuovo schema avrebbe oggetti e/o metodi diversi.

Un ulteriore vantaggio è quello legato alla programmazione ad oggetti, in quanto se non si utilizzasse la libreria il programmatore si troverebbe a scrivere del codice quasi procedurale.

1.9 Sequence Diagram

Attraverso i Sequence Diagram è possibile descrivere la vista dinamica del sistema, enfatizzando le interazioni dovute allo scambio di messaggi tra gli oggetti e il loro ordinamento temporale.

Ciascun diagramma evidenzia il modo in cui uno scenario, ossia uno specifico percorso in un caso d'uso, viene risolto dalla collaborazione tra un insieme di oggetti.

È da osservare che nell'ambito dei Sequence Diagram sono previsti degli oggetti di tipo "Interfaccia", che definiscono le parti del sistema che interagiscono con l'utente, ossia la cosiddetta UI (User Interface).

In questa fase, non viene specificata la tipologia di interfaccia utente, in modo da poter sfruttare i diagrammi realizzati anche con implementazioni diverse. È ovvio che nel caso del nostro Web Service non sia possibile definire un'interfaccia utente.

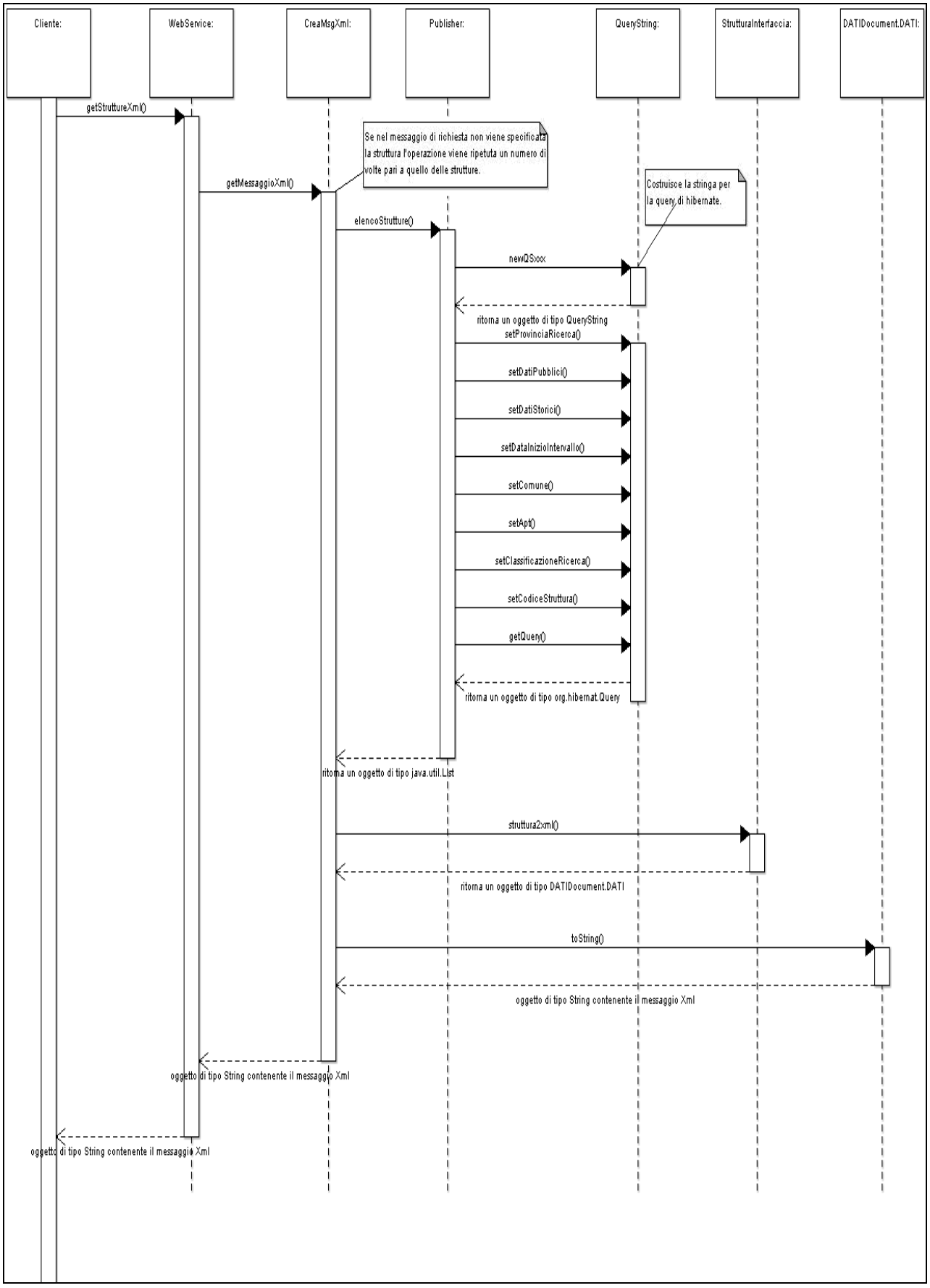


Figura 24 – Sequence Diagram.

1.10 Scelte adottate

Nel seguito verranno mostrate alcune scelte adottate in fase di sviluppo con le relative motivazioni.

Si è scelto di far implementare ad ogni POJO l'interfaccia *StrutturaInterfaccia*, la quale espone il un unico metodo *struttura2Xml()* che ha come parametro di ritorno un oggetto di tipo *DATIDocument.DATI*.

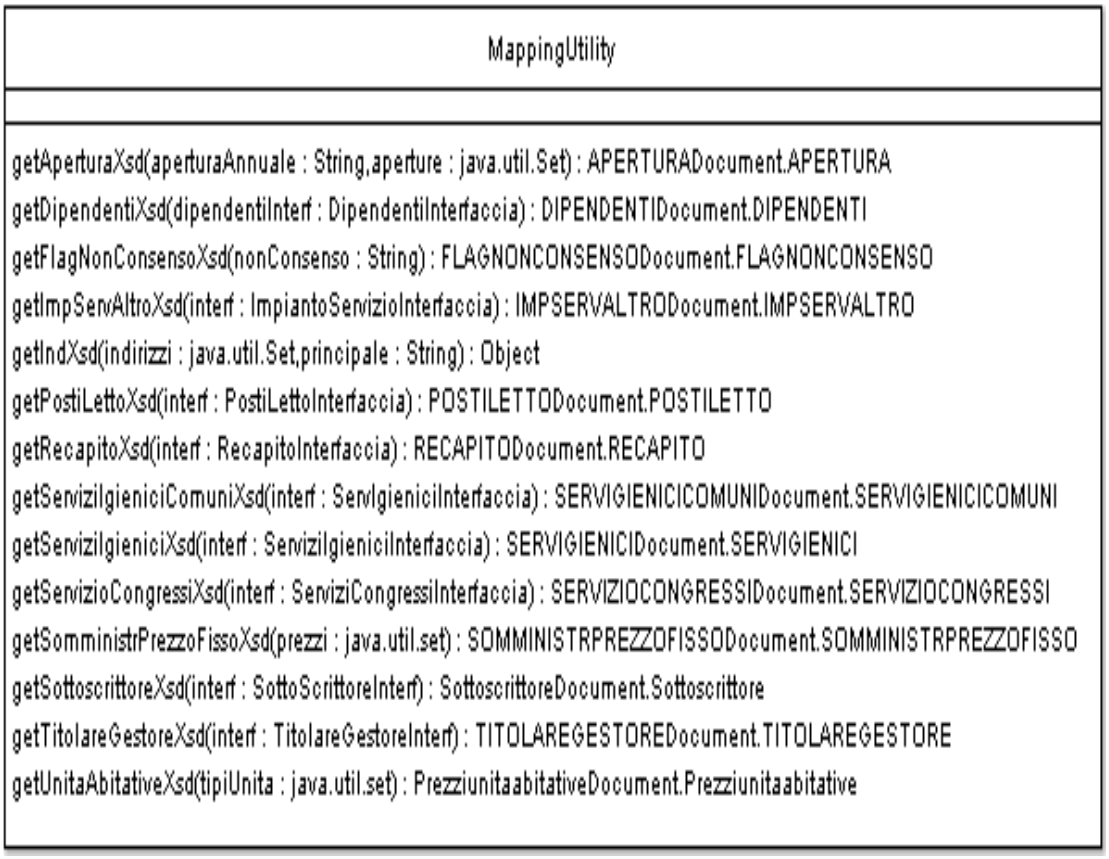
DATIDocument.DATI è un oggetto ottenuto tramite la libreria XMLBeans e rappresenta l'elemento principale della descrizione della struttura che dovrà essere contenuta all'interno del file XML. Così facendo si potrà generare il file XML con la semplice chiamata di metodo toString() descritto in precedenza.

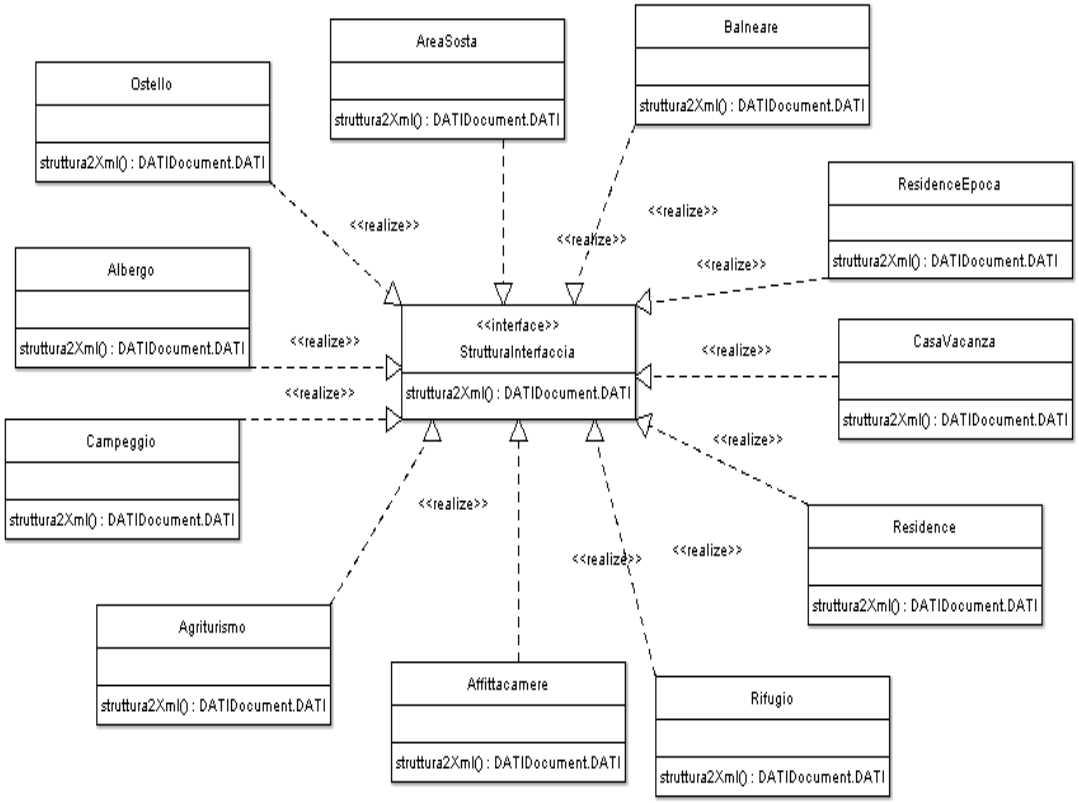
Per ogni struttura è stata creata una classe con un unico metodo pubblico il cui nome ha la seguente sintassi: *<<nomeStruttura>>2xml()*. Tale metodo si occupa di "riempire" la struttura ad oggetti ottenuta tramite XMLBeans.

Nello sviluppo hanno avuto un ruolo molto importante le interfacce che hanno consentito di definire un comportamento per alcune informazioni delle strutture turistiche.

Poiché ogni struttura possiede una parte comune in quel che riguarda la descrizione dell'anagrafe (e a volte anche nella descrizione dettagliata) è stata presa la decisione di far implementare ai POJO delle interfacce. Così facendo è stato possibile creare una classe di utilità, in grado di gestire il mapping tra i due tipi di JavaBeans. La classe di utilità ha diversi metodi pubblici che accettano come parametri di ingresso delle interfacce. In questo modo le classi con il nome nomeStruttura2Xml possono accedere ai metodi senza dover riscrivere codice che risulterebbe uguale per ogni struttura.

Nel seguito verrà mostrata una parte del Class Diagram ottenuto con le modifiche appena descritte.


MappingUtility
getAperturaXsd(aperturaAnnuale : String,aperture : java.util.Set) : APERTURADocument.APERTURA
getDipendentiXsd(dipendentiInterf : DipendentiInterfaccia) : DIPENDENTIDocument.DIPENDENTI
getFlagNonConsensoXsd(nonConsenso : String) : FLAGNONCONSENSODocument.FLAGNONCONSENSO
getImpServAltroXsd(interf : ImpiantoServizioInterfaccia) : IMPSERVALTRODocument.IMPSERVALTRO
getIndXsd(indirizzi : java.util.Set,principale : String) : Object
getPostiLettoXsd(interf : PostiLettoInterfaccia) : POSTILETTODocument.POSTILETTO
getRecapitoXsd(interf : RecapitoInterfaccia) : RECAPITODocument.RECAPITO
getServiziIgieniciComuniXsd(interf : ServIgieniciInterfaccia) : SERVIGIENICICOMUNIDocument.SERVIGIENICICOMUNI
getServiziIgieniciXsd(interf : ServiziIgieniciInterfaccia) : SERVIGIENICIDocument.SERVIGIENICI
getServizioCongressiXsd(interf : ServiziCongressiInterfaccia) : SERVIZIOCONGRESSIDocument.SERVIZIOCONGRESSI
getSomministrPrezzoFissoXsd(prezzi : java.util.set) : SOMMINISTRPREZZOFISSODocument.SOMMINISTRPREZZOFISSO
getSottoscrittoreXsd(interf : SottoScrittoreInterf) : SottoscrittoreDocument.Sottoscrittore
getTitolareGestoreXsd(interf : TitolareGestoreInterf) : TITOLAREGESTOREDocument.TITOLAREGESTORE
getUnitaAbitativeXsd(tipiUnita : java.util.set) : PrezziunitaabitativeDocument.Prezziunitaabitative


AreaSosta
struttura2Xml() : DATIDocument.DATI
Balneare
struttura2Xml() : DATIDocument.DATI
Ostello
struttura2Xml() : DATIDocument.DATI
ResidenceEpoca
struttura2Xml() : DATIDocument.DATI
Albergo
struttura2Xml() : DATIDocument.DATI
<<interface>>
StrutturaInterfaccia
struttura2Xml() : DATIDocument.DATI
CasaVacanza
struttura2Xml() : DATIDocument.DATI
Campeggio
struttura2Xml() : DATIDocument.DATI
Residence
struttura2Xml() : DATIDocument.DATI
Agriturismo
struttura2Xml() : DATIDocument.DATI
Affittacamere
struttura2Xml() : DATIDocument.DATI
Rifugio
struttura2Xml() : DATIDocument.DATI
<<realize>>


AlbergoBeanToXml
albergo2Xml(dati : DATIDocument.DATI,alb : Albergo) : DATIDocument.DATI
AgriturismoBeanToXml
agriturismo2Xml(dati : DATIDocument.DATI,agr : Agriturismo) : DATIDocument.DATI
CampeggioBeanToXml
campeggio2Xml(dati : DATIDocument.DATI,camp : Campeggio) : DATIDocument.DATI
AffittacamereBeanToXml
albergo2Xml(dati : DATIDocument.DATI,aff : Affittacamere) : DATIDocument.DATI
RifugioBeanToXml
rifugio2Xml(dati : DATIDocument.DATI,rif : Rifugio) : DATIDocument.DATI
OstelloBeanToXml
ostello2Xml(dati : DATIDocument.DATI,ost : Ostello) : DATIDocument.DATI
ResidenceBeanToXml
residence2Xml(dati : DATIDocument.DATI,res : Residence) : DATIDocument.DATI
AreaSostaToXml
areaSosta2Xml(dati : DATIDocument.DATI,area : AreaSosta) : DATIDocument.DATI
BalneareBeanToXml
balneare2Xml(dati : DATIDocument.DATI,baln : Balneare) : DATIDocument.DATI
ResidenzaEpocaBeanToXml
resEpocaXml(dati : DATIDocument.DATI,rep : ResidenceEpoca) : DATIDocument.DATI
CasaVacanzaBeanToXml
casaVacanza2Xml(dati : DATIDocument.DATI,cas : CasaVacanza) : DATIDocument.DATI

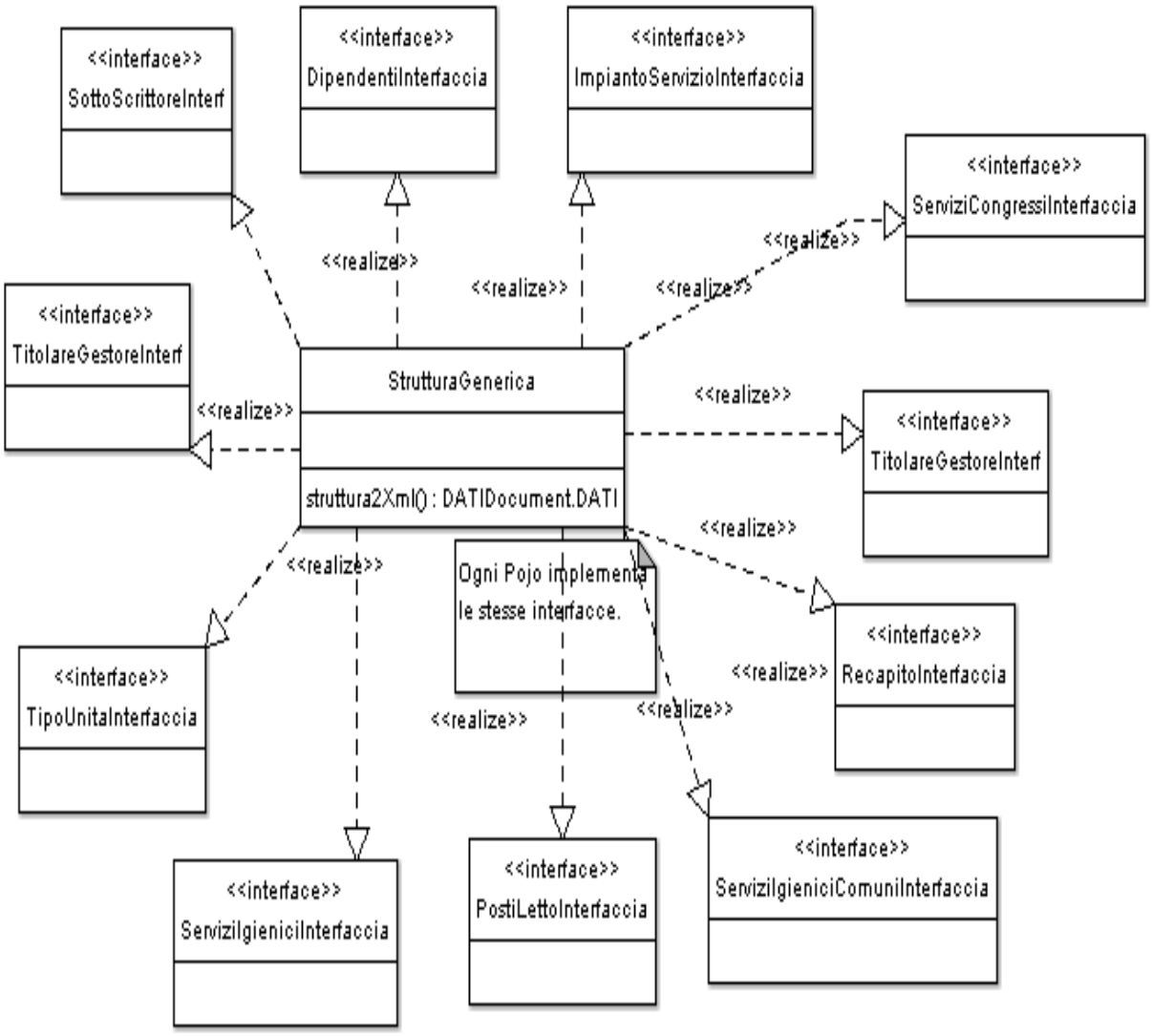


Figura 25 – Class Diagram.

2 Sviluppo e messa in opera

2.1 Sviluppo

Lo sviluppo di TurismoPublisher ha previsto la creazione di un unico Java Web Services (nel seguito detto *JWS*) Endpoint. JWS è un meccanismo che permette di generare WS Server automaticamente da codice Java.

La “classe” JWS creata è la seguente:

```
public class webService extends WebService {
}
```

La classe *WebService* estesa che si occupa di fornire il servizio ai vari utilizzatori è la seguente:

```
public class WebService {
  private static Log log=LogFactory.getLog(WebService.class);
  public static byte[] getStruttureXml(byte[] xml) {
        try {
            // corpo
    } catch (turismo.publisher.InterneException ex) {
      log.error(ex.getMessage());
      xml = new
StringBuffer(XMLConstants.INTESTAZIONE).toString().getBytes();
    }catch (IllegalArgumentException ex) {
      log.error(ex.getMessage());
      xml = new
StringBuffer(XMLConstants.INTESTAZIONE).toString().getBytes();
    }  catch (Exception ex) {
      log.error(ex.getMessage(), ex);
      xml = new
StringBuffer(XMLConstants.INTESTAZIONE).toString().getBytes();
    }
    return xml;
  }
}
```

Listato 14

Il Web Service, come si vede dal precedente Listato, risponde sempre con un messaggio XML contenuto in un array di byte anche in caso di errori; tutti gli errori sono gestiti tramite l'oggetto *org.apache.commons.logging.Log*. Così facendo il log del server di Tomcat, che ospita il servizio, non viene "sporcato" anche in virtù del fatto che lo stesso server ospita applicazioni diverse. In caso di errore la risposta contiene solo un'intestazione contenuta all'interno di una variabile; pertanto se dovesse cambiare il messaggio di risposta in caso di errore si andrebbe a modificare un solo campo.

Con il metodo *getStruttureXml()* viene dato inizio alla creazione della descrizione delle varie strutture istanziando l'oggetto *CreaMsgXml*. Tale oggetto a sua volta utilizza un'istanza dell'oggetto *Publisher* che si occupa di restituire gli

XMLBeans relativi alle strutture trovate. Una semplificazione del codice della classe CreaMsgXml è la seguente:

```
1  public class CreaMsgXml {
2    private static Log log=LogFactory.getLog(CreaMsgXml.class);
3
4    public String getMessaggioXml(/*criteri di ricerca*/) throws
5  Exception {
6      Iterator iterator = null;
7      java.util.List list = null;
8      StringBuffer result = new StringBuffer();
9      RispostaRichiestaServizioSincronaType risposta = null;
10     DATIDocument.DATI dati = null;
11
12     // caso in cui viene richiesta una struttura specifica
13     if(generestruttura_ricerca!=null &&
14 generestruttura_ricerca.length()>0) {
15       getStrutturaDati(result, /*criteri di ricerca*/);
16     } else {
17     // caso in cui NON viene richiesta una struttura specifica
18       for(int i=0; i<XMLConstants.DESCR_STRUTTURE.length; i++)
19 {
20         getStrutturaDati(result, /*criteri di ricerca*/);
21       }
22     }
23
24     return result.toString();
25   }
26   private void getStrutturaDati(StringBuffer
27 stringBuffer,/*criteri di ricerca**/) throws Exception {
28     StringBuffer testStrutturaTuristica = new StringBuffer();
29     java.util.List list = null;
30
31     list = new Publisher().elencoStrutture(/*criteri di
32 ricerca*/);
33
```

```
34      noNamespace.DATIDocument.DATI dati = null;
35      if(list!=null && !list.isEmpty()) {
36        StrutturaInterfaccia struttura = null;
37        for (Iterator it = list.iterator(); it.hasNext(); ) {
38          struttura = (StrutturaInterfaccia)it.next();
39          dati = struttura.struttura2Xml();
40          if(dati!=null) {
41            testStrutturaBalneare = new
42  StringBuffer(XMLConstants.INTESTAZIONE+
43              XMLConstants.risposta_RichiestaServizioSincrona +
44              +dati.toString()+
45
46  XMLConstants.chiusoTag_risposta_RichiestaServizioSincrona
47              );
48
49  if(Validazione.validate(testStrutturaTuristica.toString().getB
50  ytes(), XMLConstants.FILEXSD)) {
51              stringBuffer.append(dati.toString());
52            } else {
53
54  StampaErrore.stampaCodiceStrutturaErrore(generestruttura_ricer
55  ca, dati);
56              log.error(testStrutturaTuristica.toString());
57            }
58          }
59        }
60      }
61    }
62  }
```

Listato 15

Nel caso in cui si volessero ottenere informazioni relative ad una unica struttura turistica viene effettuata la ricerca per l'unica tipologia desiderata, mentre se tale criterio non venisse scelto si effettuerebbero tante ricerche quante sono le tipologie. Il codice delle tipologie è cablato all'interno di una classe di utilità e tramite un'iterazione vengono effettuate le varie ricerche. In questo modo se

venissero aggiunti nuovi complessi turistici basterebbe aggiungere il codice all'interno della classe di utilità.

Ogni volta che viene "pescata" una nuova struttura, la relativa descrizione viene "appesa" alle precedenti. Se la singola descrizione non dovesse essere valida verrebbe registrata la motivazione all'interno del file di log. Per una lettura più facile del file è stata realizzata la classe *StampaErrore* in grado di fornire la descrizione della struttura. L'oggetto *Validazione* contribuisce alla composizione del file di Log inserendo il motivo per il quale la descrizione non è valida. Infine il messaggio XML si ottiene semplicemente invocando il metodo toString() dell'oggetto XMLBeans.

Il metodo *elencoStrutture()*, riga 31 del precedente Listato, restituisce una lista delle strutture che soddisfano i criteri di ricerca e con il metodo *struttura2xml()*, riga 39 del precedente Listato, viene effettuato il mapping tra i due bean (POJO e XMLBeans).

L'oggetto QueryString si occupa della creazione dell'oggetto org.hibernate.Query contenente la query in linguaggio HQL. Il seguente Listato mostra il codice della suddetta classe.

```
1  import org.hibernate.Session;
2  import org.hibernate.Query;
3  import java.util.Date;
4
5  class QueryString {
6    private String tabelle;
7    private String where;
8    private String provincia_ricerca = null;
9    private String dati_pubblici = null;
10   private String dati_storici = null;
11   private String comune = null;
12   private Date data_inizio_intervallo = null;
13   private Date data_fine_intervallo = null;
14   private String classificazione_ricerca = null;
15   private String codicestruttura = null;
16   private String nomeCodStruttura = null;
17   private String genereStruttura = null;
18
```

```
19    public QueryString(String tiboBean, String
20  indirizzoStruttura, String codStruttura) {
21      this.nomeCodStruttura = codStruttura;
22      tabelle = "SELECT DISTINCT struttura FROM " +
23        "it.tdgroup.turismo.beans."+tiboBean+" as struttura, " +
24        "it.tdgroup.turismo.beans."+indirizzoStruttura+" as
25  indStr ";
26
27      where   = " WHERE " +
28        " struttura.id.dataInizioVal = (SELECT MAX
29  (strutturaMaxDataInizioVal.id.dataInizioVal) FROM
30  it.tdgroup.turismo.beans."+tiboBean+"
31  strutturaMaxDataInizioVal WHERE
32  struttura.id."+this.nomeCodStruttura+" =
33  strutturaMaxDataInizioVal.id."+this.nomeCodStruttura+ ") "+
34        "AND struttura.id."+this.nomeCodStruttura+" =
35  indStr."+tiboBean.toLowerCase()+".id."+this.nomeCodStruttura+"
36  AND struttura.id.dataInizioVal =
37  indStr."+tiboBean.toLowerCase()+".id.dataInizioVal " +
38        "AND indStr.provincia.sigla = :provincia_ricerca " +
39        "AND struttura.nonConsenso = :dati_pubblici ";
40    }
41
42    public static QueryString newQSAlbergo() {
43      return new QueryString("Albergo", "IndirizzoAlb",
44  "codAlbergo");
45    }
46    public static QueryString newQSCampeggio() {
47      return new QueryString("Campeggio", "IndirizzoVit",
48  "codCampeggio");
49    }
50    public static QueryString newQSResidenzaEpoca() {
51      return new QueryString("ResidenzaEpoca", "IndirizzoRep",
52  "codResidenzaEpoca");
53    }
54    public static QueryString newQSResidence() {
```

```
55        return new QueryString("Residence", "IndirizzoRes",
56    "codResidence");
57      }
58      public static QueryString newQSCasaVacanza() {
59        return new QueryString("CasaVacanza", "IndirizzoCav",
60    "codCasaVacanza");
61      }
62      public static QueryString newQSBalneare() {
63        return new QueryString("Balneare", "IndirizzoBaln",
64    "codBalneare");
65      }
66      public static QueryString newQSAffittacamere() {
67        return new QueryString("Affittacamera", "IndirizzoAll",
68    "codAffittacamera");
69      }
70      public static QueryString newQSAgriturismo() {
71        return new QueryString("Agriturismo", "IndirizzoAat",
72    "codAgriturismo");
73      }
74      public static QueryString newQSOstello() {
75        return new QueryString("Ostello", "IndirizzoCaf",
76    "codOstello");
77      }
78      public static QueryString newQSRifugio() {
79        return new QueryString("Rifugio", "IndirizzoRif",
80    "codRifugio");
81      }
82      public static QueryString newQSAreaDiSosta() {
83        return new QueryString("AreaSosta", "IndirizzoAst",
84    "codAreaSosta");
85      }
86
87      public void setProvinciaRicerca(String provincia_ricerca) {
88        if((provincia_ricerca != null) &&
89    (provincia_ricerca.trim().length()>0)) {
90          this.provincia_ricerca = provincia_ricerca;
```

```
91        }
92      }
93      public void setDatiPubblici(String dati_pubblici) {
94        this.dati_pubblici = dati_pubblici;
95      }
96      public void setDatiStorici(String dati_storici) {
97        if((dati_storici != null) &&
98    (dati_storici.trim().length()>0)) {
99          this.dati_storici = dati_storici;
100         where += "AND (" + (Boolean.parseBoolean(dati_storici) ?
101   "" : "NOT ") + "((struttura.dataFineVal IS NOT NULL) AND
102   (struttura.dataFineVal < :dataOggi))) ";
103       }
104     }
105     public void setDataInizioIntervallo(Date
106   data_inizio_intervallo) {
107       if (data_inizio_intervallo != null) {
108         this.data_inizio_intervallo = data_inizio_intervallo;
109         where += "AND ((struttura.dataFineVal IS NOT NULL) and
110   (struttura.dataFineVal >= :data_inizio_intervallo)) ";
111       }
112     }
113     public void setDataFineIntervallo(Date data_fine_intervallo)
114   {
115       if (data_fine_intervallo != null) {
116         this.data_fine_intervallo = data_fine_intervallo;
117         where += "AND ((struttura.dataFineVal IS NOT NULL) and
118   (struttura.dataFineVal <= :data_fine_intervallo)) ";
119       }
120     }
121     public void setComune(String comune) {
122       if ((comune != null) && (comune.trim().length()>0)) {
123         this.comune = comune;
124         where += "AND UPPER(indStr.comune.comune) = :comune ";
125       }
126     }
```

```
127     public void setClassificazioneRicerca(String
128   classificazione_ricerca) {
129       if((classificazione_ricerca != null) &&
130   (classificazione_ricerca.trim().length()>0)) {
131         this.classificazione_ricerca = classificazione_ricerca;
132         where += "AND
133   struttura.classificazione.classificaZoneNum =
134   :classificazione_ricerca ";
135       }
136     }
137     public void setCodicestruttura(String codicestruttura) {
138       if((codicestruttura != null) &&
139   (codicestruttura.trim().length()>0)) {
140         this.codicestruttura = codicestruttura.trim();
141         where += "AND struttura.id."+this.nomeCodStruttura+" =
142   :codicestruttura ";
143       }
144     }
145
146     public Query getQuery(Session session) throws Exception {
147       Query query = null;
148       if(provincia_ricerca!=null && dati_pubblici!=null &&
149   dati_storici!=null) {
150
151         query =
152   session.createQuery(tabelle+where).setReadOnly(true);
153         query.setString("provincia_ricerca",provincia_ricerca);
154         query.setString("dati_pubblici",dati_pubblici);
155         query.setDate("dataOggi", new Date());
156
157         if(data_inizio_intervallo!=null) {
158
159   query.setDate("data_inizio_intervallo",data_inizio_intervallo)
160   ;
161         }
162         if(data_fine_intervallo!=null) {
```

```
163
164   query.setDate("data_fine_intervallo",data_fine_intervallo);
165         }
166          if(comune != null) {
167           query.setString("comune",comune.trim().toUpperCase());
168         }
169         if(classificazione_ricerca!=null) {
170
171   query.setString("classificazione_ricerca",classificazione_rice
172   rca.trim());
173         }
174         if(codicestruttura!=null) {
175           query.setString("codicestruttura",codicestruttura);
176         }
177         if(genereStruttura!=null) {
178           query.setString("genereStruttura",genereStruttura);
179         }
180       }
181       return query;
182     }
183   }
```

Listato 16

La classe QueryString mette a disposizione un metodo per ogni struttura il quale invoca il costruttore della classe che si occupa di istanziare i campi che danno una forma iniziale alla query in linguaggio HQL. In questo modo se fosse aggiunta una nuova tipologia basterebbe inserire un nuovo metodo capace di istanziare l'oggetto.

Una volta istanziato, attraverso dei semplici metodi di set vengono valorizzati i criteri ricerca che andranno a formare la query finale. A questo punto attraverso il metodo *getQuery()* si ottiene l'oggetto org.hibernate.Query che permette di eseguire la ricerca.

Come si vede dal Listato 16 la classe Publisher invoca semplici metodi, senza parametri, in grado istanziare l'oggetto QueryString. Così facendo si evita del codice

ripetuto e in fase di manutenzione sarà necessario modificare i questi metodi della classe QueryString.

Ogni POJO implementando l'interfaccia StrutturaInterfaccia implementa il metodo *struttura2Xml()*. In generale ha la seguente forma:

```
public DATI struttura2Xml() {
    return AlbergoBeanToXml.albergo2Xml
(DATIDocument.DATI.Factory.newInstance(),this);
}
```

Listato 17

Gli oggetti con il nome *xxxBean2Xml* si occupano del mapping tra i POJO e gli XMLBeans e hanno tutti una forma molto simile alla seguente:

```
1  public class AlbergoBeanToXml {
2    private static Log
3  log=LogFactory.getLog(AlbergoBeanToXml.class);
4
5    public static noNamespace.DATIDocument.DATI albergo2Xml(DATI
6  dati, Albergo alb) {
7        AlbergoDocument.Albergo  albergo = null;
8        try {
9          albergo = dati.addNewAlbergo();
10         albergo.addNewANAGRAFE();
11         albergo.setANAGRAFE(getAnagrafeXsd(alb));
12         albergo.addNewDESCRALB();
13         albergo.setDESCRALB(getDescrAlbXsd(alb));
14         dati.setAlbergo(albergo);
15       } catch (it.tdgroup.turismo.publisher.InterneException
16 ex) {
17         String codiceEserc = "";
18         if(alb !=null && alb.getId()!=null &&
19 alb.getId().getCodAlbergo()!=null &&
20 alb.getId().getCodAlbergo().length()>0) {
21           codiceEserc = "codice esercizio: " +
22 alb.getId().getCodAlbergo();
```

```
23          }
24          log.error(codiceEserc + " " + ex.getMessage());
25          dati = null;
26        } catch (Exception ex) {;
27          String codiceEserc = "";
28          if(alb !=null && alb.getId()!=null &&
29  alb.getId().getCodAlbergo()!=null &&
30  alb.getId().getCodAlbergo().length()>0) {
31            codiceEserc = "codice esercizio: " +
32  alb.getId().getCodAlbergo();
33          }
34          log.error(codiceEserc + " " + ex.getMessage(), ex);
35          dati = null;
36        }
37        return dati;
38      }
39      // INIZIO SEZIONE RELATIVA ANAGRAFE
40      private static ANAGRAFEDocument.ANAGRAFE
41  getAnagrafeXsd(Albergo alb) throws Exception {
42        // corpo
43      }
44      // INIZIO SEZIONE RELATIVA ANAGRAFE
45      private static DESCRALBDocument.DESCRALB
46  getDescrAlbXsd(Albergo alb) throws Exception {
47        // corpo
48      }
49  }
```

Listato 18

Il metodo albergo2Xml() invoca due metodi privati della classe, righe 11 e 13 del Listato precedente, che si occupano del mapping tra i due bean. Il mapping non è altro che una serie di metodi set e get esposti dai due oggetti e hanno una forma molto simile alla seguente:

```
oggXmlbean.getPropertyName(oggPojo.setPropertyName());
```

Alcune descrizioni delle strutture sono risultate uguali e per evitare ridondanza di codice, il relativo mapping è stato gestito all'interno di una unica classe, denominata *MappingUtility*.

MappingUtility offre una serie di metodi in grado di gestire il mapping di zone specifiche dei bean, accettano come parametri di ingresso delle interfacce e ritornano oggetti XMLBeans rappresentanti la porzione della descrizione mappata.

Nel seguente Listato viene mostrato un singolo metodo della suddetta classe in grado di mostrare il concetto appena espresso.

```
1  public static DIPENDENTIDocument.DIPENDENTI
2  getDipendentiXml(DipendentiInterfaccia dipendentiInterf)
3  throws Exception {
4    DIPENDENTIDocument.DIPENDENTI dipendenti =
5  DIPENDENTIDocument.DIPENDENTI.Factory.newInstance();
6
7  dipendenti.setDipstagionale(Utility.integer2bigIngeger(dipende
8  ntiInterf.getDipendentiStag()));
9
10 dipendenti.setDiptempoparziale(Utility.integer2bigIngeger(dipe
11 ndentiInterf.getDipendentiParz()));
12
13 dipendenti.setTotale(Utility.integer2bigIngeger(dipendentiInte
14 rf.getDipendentiTot()));
15   return dipendenti;
16 }
```

Listato 19

Per quel che concerne l'invocazione del Web Service un possibile esempio è illustrato dal seguente codice:

```
1  try {
2        Call call = (Call) new Service().createCall();
3        call.setTargetEndpointAddress(new
4  URL("http://localhost:8081/TurismoPublisher/webService.jws"));
```

```
5         call.setOperationName(new QName("uri:webService",
6   "getStruttereXml"));
7         String criteriRicerca = /*messaggio contenente i criteri
8   di ricerca */
9         String descrizioneStrutture = new
10  String((byte[])(call.invoke(new Object[] {
11  criteriRicerca.getBytes() })));
12      } catch (Exception ex) {
13        ex.printStackTrace();
14  }
```

Listato 20

Gli oggetti Service e Call, righe 2 e 3 del Listato 20, fanno parte della libreria di Axis.

Un altro aspetto curato nello sviluppo di TurismoPublisher è quello legato alle eccezioni. È stata realizzata una classe che gestisce le eccezioni detta "*InterneException*" la quale estende la classe "*Exception*". È fornita di un costruttore che non fa altro che richiamare il costruttore della classe che estende passando il messaggio di errore. Questo tipo di eccezione viene sollevato ogni qualvolta non viene trovata una informazione obbligatoria per lo schema XSD.

Questa scelta di gestione delle eccezioni in modo così "*flat*" nasce dal fine che si prefigge il servizio, cioè quello di descrivere una struttura turistica tramite un file XML il quale deve seguire un dato schema XSD.
Se si dovesse trovare una struttura nella base dati priva di quelle informazioni obbligatorie sarebbe inutile continuare il "riempimento" dei JavaBeans della libreria XMLBeans in quanto il file generato non sarebbe valido.

Le eccezioni generate durante la creazione della struttura sono sempre propagate e l'unica classe che si cura della loro gestione è quella della relativa struttura. Così facendo se ci dovessero essere problemi durante la stesura si può passare alla prossima senza bloccare il servizio.

2.2 Messa in opera

Per utilizzare il Web Service è prevista l'installazione del Web Server di Tomcat. Per installarlo è sufficiente mandare in esecuzione il programma di

installazione; è necessario però che il nome della cartella contenente Tomcat sia senza spazi o caratteri accentati. Al termine dell'installazione vanno inserite le seguenti variabili d'ambiente:

- *CATALINA_HOME* uguale alla cartella di installazione di Tomcat;
- *TOMCAT_HOME* uguale alla cartella di installazione di Tomcat.

Per installare il servizio si deve copiare il file *TurismoPublisher.war* nella cartella *%TOMCAT_HOME%/webapps*.

2.3 Esempio di impiego dei dati ottenuti

Nella seguente figura è mostrato come un generico utente potrebbe impiegare i dati ottenuti dal Web Service-.

Codice Esercizio	052014AAT0007	**Provincia**	LU
Genere	AAT	**Tipo**	agriturismo
Comunicazione dei prezzi e delle caratteristiche della struttura per l'anno		2007	
Comunicazione		ANNUALE	
Denominazione Esercizio		AGRITURISMO I FICUPALI	

Classificazione D.P.G.R. 3 Agosto 2004, n. 46/R	☐ 1 spiga	☐ 2 spighe	☑ **3 spighe**
Menzioni aggiuntive	animali	degustazioni	equitazione

Indirizzo dell'esercizio			**N.**	100
	Località	LUCCA	**Frazione**	frazione
	Comune	Lucca	**CAP**	55100

Indirizzo per le comunicazione nel periodo di chiusura			**N.**	100
	Località	LUCCA	**Frazione**	frazione
	Comune	Lucca	**CAP**	55100

Addetto all'attività agrituristica	**Cognome:** Cognome addetto	**Nome:** Nome addetto

DIA (denuncia iniz. attività) o Comunicazione o Autorizzazione	**Del**	2007-01-01
	Comune/Suap di	LUCCA
	Numero autorizzazione	1234567898

- AZIENDA AGRITURISTICA **NON** AUTORIZZATA PER PERNOTTAMENTO

Apertura	◉ **Annuale** ○ Stagionale

Camere e posti letto

CAMERE CON 4 LETTI	**CAMERE DOPPIE**	**CAMERE SINGOLE**	**CAMERE CON 6 LETTI**
con bagno	con bagno	con bagno	con bagno
n. 10	n. 20	n. 10	n. 10

TOTALE POSTI LETTO AUTORIZZATI IN CAMERE n. 140	**TOTALE CAMERE DA LETTO** n. 50	Camere con riscaldamento n.50

Unità abitative, camere in unità abitative e posti letto in unità abitative

TOTALE UNITÀ ABITATIVE	TOTALE CAMERE DA LETTO IN UNITÀ ABITATIVE	TOTALE POSTI LETTO AUTORIZZATI IN UNITÀ ABITATIVE
N. 10	N. 30	N. 60
UNITÀ ABITATIVE CON RISCALDAMENTO n. 30		

Spazi aperti - Agricampeggio

Totale piazzole n. 3	Capacità ricettiva totale persone n. 90	Superficie totale delle piazzole in mq. 200

Servizi igienici della struttura agrituristica

BAGNI IN UNITÀ ABITATIVE	n. 10	**di cui per portatori di handicap**	n. 5
BAGNI PRIVATI IN CAMERE	n. 5	**di cui per portatori di handicap**	n. 5
BAGNI COMUNI PER LE CAMERE	n. 5	**di cui per portatori di handicap**	n. 5
BAGNI COMUNI A DISPOSIZIONE PER OSPITI GIORNALIERI (ATTIVITÀ DIDATTICHE RICREATIVE CULTURALI - RISTORAZIONE)	n. 5	**di cui per portatori di handicap**	n. 5
TOTALE BAGNI	n. 25	**di cui per portatori di handicap**	n. 20

AGRICAMPEGGIO

W.C.	n. 10	**di cui per portatori di handicap**	n. 10

VUOTATOI PER W.C. CHIMICI	n. 10	**LAVELLI PER STOVIGLIE**	n. 5
DOCCE CHIUSE	n. 5	**LAVATOI PER PANNI**	n. 5
LAVABI	n. 10		

Attività di ristorazione agrituristica

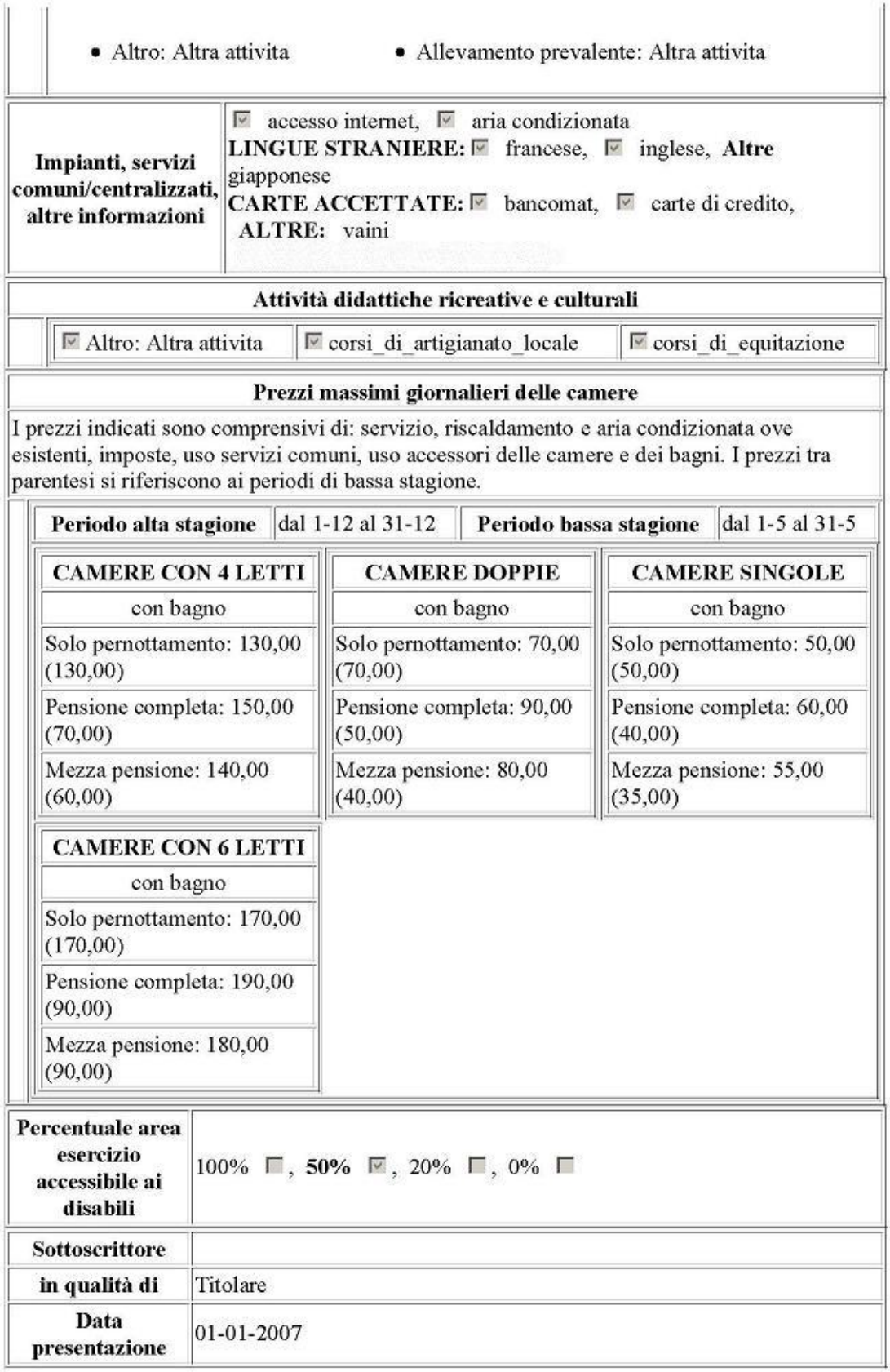
L'azienda è autorizzata, secondo la legislazione vigente, a:

- somministrazione pasti, alimenti e bevande agli ospiti per n. posti tavola = 200
- somministrazione pasti, alimenti e bevande indipendentemente dall'esercizio di altre attività agrituristiche per n. posti tavola = 200
- degustazione, assaggio dei prodotti aziendali: si

Dimensione azienda agricola Superficie totale espressa in ettari (ha) 10

Ordinamento colturale (colture/attività prevalenti o che comunque caratterizzano l'attività dell'azienda agricola)

- bosco
- viticolo
- cerealitico
- foraggero
- ortoflorovivaistico
- olivicolo



- Altro: Altra attivita
- Allevamento prevalente: Altra attivita

Impianti, servizi comuni/centralizzati, altre informazioni	☑ accesso internet, ☑ aria condizionata **LINGUE STRANIERE:** ☑ francese, ☑ inglese, **Altre** giapponese **CARTE ACCETTATE:** ☑ bancomat, ☑ carte di credito, **ALTRE:** vaini

Attività didattiche ricreative e culturali

☑ Altro: Altra attivita	☑ corsi_di_artigianato_locale	☑ corsi_di_equitazione

Prezzi massimi giornalieri delle camere

I prezzi indicati sono comprensivi di: servizio, riscaldamento e aria condizionata ove esistenti, imposte, uso servizi comuni, uso accessori delle camere e dei bagni. I prezzi tra parentesi si riferiscono ai periodi di bassa stagione.

Periodo alta stagione	dal 1-12 al 31-12	**Periodo bassa stagione**	dal 1-5 al 31-5

CAMERE CON 4 LETTI	CAMERE DOPPIE	CAMERE SINGOLE
con bagno	con bagno	con bagno
Solo pernottamento: 130,00 (130,00)	Solo pernottamento: 70,00 (70,00)	Solo pernottamento: 50,00 (50,00)
Pensione completa: 150,00 (70,00)	Pensione completa: 90,00 (50,00)	Pensione completa: 60,00 (40,00)
Mezza pensione: 140,00 (60,00)	Mezza pensione: 80,00 (40,00)	Mezza pensione: 55,00 (35,00)

CAMERE CON 6 LETTI
con bagno
Solo pernottamento: 170,00 (170,00)
Pensione completa: 190,00 (90,00)
Mezza pensione: 180,00 (90,00)

Percentuale area esercizio accessibile ai disabili	100% ☐, **50%** ☑, 20% ☐, 0% ☐
Sottoscrittore	
in qualità di	Titolare
Data presentazione	01-01-2007

Figura 26 – Le informazioni ottenute possono essere utilizzate in vari modi.

3 Conclusioni

Come abbiamo visto il lavoro che il programmatore dovrà svolgere si riduce a "riempire" la struttura ad oggetti creata tramite la libreria XML in quanto la:

- gestione del database è affidata ad Hibernate;
- gestione del Web Service è affidata alla Web Application basata sulla piattaforma J2EE;
- creazione del messaggio XML è affidata alla libreria XMLBeans;
- gestione del log è affidata alla libreria Log4j.

Infine tutti i software utilizzati sono di tipo Open Source come richiesto dalla Regione Toscana.

E. Conclusioni

Nella realizzazione di applicativi web per la Pubblica Amministrazione è di fondamentale importanza adottare dei framework che garantiscano scalabilità, integrazione con altre piattaforme, possibilità di interoperabilità a livello di servizi. Questo è conseguito anche grazie all'adozione di paradigmi cardine quali SOA e MVC, e tecnologie standard quali Web Services. Un altro aspetto particolarmente importante è quello relativo all'impiego di prodotti di pubblico dominio.

Nel contesto sperimentale del presente lavoro, la piattaforma Java 2 Enterprise Edition è risultata idonea ad offrire tutti questi aspetti, con un adeguato supporto nei due casi di studio industriali che sono stati trattati.

Il primo, quello della Banca Dati delle Misure Cautelari Personali, ha consentito di sperimentare l'impiego dei Session Bean per le macro aree dell'applicazione, delle Action del framework Apache Struts per la logica di business, della piattaforma Castor per ottenere la persistenza dello stato e dei POJO per il trasporto dei dati a livello web. BDMC rappresenta un progetto di dimensioni notevoli, e come tale ha permesso anche di sperimentare la scalabilità.

Il secondo caso di studio, TurismoPublisher, un'applicazione della regione Toscana per servizi concernenti il turismo, è stato anch'esso realizzato utilizzando la piattaforma J2EE. Il servizio è ospitato in una Web Application e configurato come una Servlet, contiene la business-logic in classi Java Standard Edition, la persistenza realizzata tramite Hibernate e adopera i POJO per la rappresentazione object-oriented dei dati relazionali. Infine, sono stati adoperati gli XMLBeans per la rappresentazione XML dei dati.

Pertanto, il lavoro svolto ha consentito di operare a tutti i livelli dell'applicazione e di compiere interventi sia di manutenzione che di realizzazione in modo rigorosamente aderente ai pattern di riferimento.

Bibliografia

[1] M. Calvo, “Frontiere di rete”. Casa Editrice Laterza.

[2] http://www.governo.it/Presidenza/web/circ13mar2001_FP.html

[3] http://java.sun.com/javaee/

[4] http://www.hibernate.org/

[5] http://struts.apache.org/

[6] http://www.oasis-open.org/home/index.php

[7] http://www.castor.org/

[8] http://www.regione.toscana.it/

[9] http://www.giustizia.it/

[10] http://labs.jboss.com/

[11] http://tomcat.apache.org/

[12] http://www2.ing.unipi.it/~d9395/corsi/sari/

[13] http://www2.ing.unipi.it/~o1553499/

[14] http://xmlbeans.apache.org/