POLITECNICO DI MILANO

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

IMPLEMENTAZIONE DI UN SISTEMA DI MULTICANALITA' BANCARIO VIA WEB-SERVICES

Relatore : Prof.ssa Chiara FRANCALANCI

Correlatore : Ing. Maurizio VERONESI

Tesi di laurea di :

Eric Michel KOUAYE NGAKOUE Matr. 681726

Anno Accademico 2010-2011

Alla memoria di mio padre

"Ti porterò sempre nel cuore"

INDICE GENERALE

CAPITOLO 1

Introduzione

Obiettivo del presente lavoro è quello di illustrare la soluzione implementata per la multicanalità del business retail della vendita di prodotti di consumer financing di un gruppo bancario. Per clientela retail si intende la clientela individuale – non aziende per intenderci – mentre per consumer financing si intendono quei prodotti finanziari quali carte di credito, prestiti e similari ideati per il finanziamento di acquisti di beni di consumo.
I sistemi di multicanalità della banca consentono al cliente di effettuare delle operazioni informative o dispositive relative ai propri prodotti acquistati utilizzando canali differenti dal classico sportello bancario, quali sito Internet, call center o IVR. Per IVR si intende l'Interactive Voice Recognition, un sistema automatico in grado di recepire i comandi vocali del cliente ed effettuare conseguenti operazioni sui propri servizi comunicando un risultato finale.

Nella situazione di partenza, Il cliente utilizza un sistema informatico che gestisce i contratti di consumer financing sia in termini di nuove richieste di apertura rapporti, sia in termini di gestione dei rapporti in essere.
Tale sistema è fruibile principalmente attraverso una serie di sportelli di filiale dislocati sul territorio. Gli operatori si connettono via terminale a caratteri sul sistema centrale e fruiscono delle informazioni sui clienti/prodotti e le modificano in base alle richieste del singolo cliente.
Oltre a questa modalità di fruizione, tuttavia, il cliente può comunicare le proprie esigenze, o informarsi sullo stato dei propri rapporti, utilizzando una serie di canali non convenzionali che elenchiamo di seguito:
Chiamare un numero verde per richiedere informazioni sui propri prestiti o carte di credito senza ausilio dell'operatore;
Chiamare un numero "nero" per bloccare la propria carta di credito in seguito a furto o smarrimento.
Consultare il sito internet della banca per visualizzare saldo residuo, elenco movimenti del mese, capitale residuo da finanziare.

Contattare telefonicamente un operatore per richieste non convenzionali o comunque non contemplate nell'albero automatico del risponditore IVR.
Oltre a queste funzionalità la banca ha inoltre bisogno di effettuare una rendicontazione di bilancio e una serie di contribuzioni a banche dati esterne che però esulano dallo scopo del presente lavoro.

Nell'architettura di partenza , ogni canale dedicato all'utente (sportelli della banca, sito internet, IVR) ha un set di servizi propri sviluppati con una determinata tecnologia e logica .
Il problema di quest' architettura è che la stessa funzionalità richiamata da canali diversi non da la garanzia di avere gli stessi risultati non essendoci un'unica fonte di servizi per i vari canali a disposizione dell'utente.

Sono stati richiesti lo sviluppo di nuovi servizi e una completa modifica della piattaforma esistente per poter offrire ai clienti /utenti un'unica fonte di servizi.
Le operazioni che il cliente interno (banca) ha bisogno di eseguire sono, tra le altre, le seguenti:

- Gestione dei dati anagrafici dei clienti con integrazione con l'anagrafe del gruppo per evitare duplicazioni;
- Gestione delle nuove richieste di contratto da parte dei clienti con possibilità di scelta tra differenti prodotti;
- Liquidazione dei prestiti
- Gestione delle richieste di post-vendita:
- cambio rata prestito
- cambio plafond carta
- variazione alert informativi e dispositivi

Soltanto alcune funzionalità devono essere invece accedibili tramite la multicanalità, in particolare quelle relative alla gestione delle carte di credito:

- elenco carte possedute
- dettaglio della carta di credito
- verifica attivazione della carta di credito
- elenco delle convenzioni presenti sulla carta
- cambio plafond della carta
- cambio rata di rimborso su carte revolving
- dettaglio dell'estratto conto

- disponibilità rimanente sulla carta per il mese corrente
- elenco movimenti del mese
- variazione dei dati anagrafici
- variazione della modalità di invio dell'estratto conto
- variazione del numero cellulare per alert sms
- variazione degli alert informativi

Dunque queste funzionalità devono essere fruibili direttamente dall'utente, senza il tramite di un operatore della banca che acceda ai sistemi interni in base alle necessità dell'utente stesso. Dunque l'utente della carta di credito si collega direttamente al sito internet della banca, per esempio, inserisce le sue credenziali, effettua il login e naviga sulle funzionalità a disposizione sulle sue carte. Alternativamente chiama un numero verde con risponditore automatico al quale impartisce delle disposizioni tramite i comandi della tastiera telefonica.

L'obbiettivo di questo lavoro è quindi quello di mettere in piedi un'architettura semplice nel rispetto degli standard aziendali che permetta di sviluppare e fornire servizi ai vari canali che la banca mette a disposizione dei propri clienti nella gestione dei prodotti offerti.

Questo documento è strutturato nel seguente modo :
Il secondo capitolo presenta il contesto tecnologico dell'azienda e la scelta tecnologica che è stata fata per raggiungere l'obbiettivo prefissato.
IL terzo capitolo presenta l'analisi funzionale e tecnica del progetto con particolare attenzione alle problematiche di sicurezza dovendo sviluppare il progetto in ambito bancario.
Il Quarto capito presenta l'implementazione della soluzione proposta con particolare riferimento allo sviluppo della parte server dei servizi e alle chiamate ai servizi di back-end messi a disposizione.
Il quinto capitolo presenta come vengono gestiti i sorgenti sviluppati e l'installazione dell'applicazione nei vari ambienti a disposizione.
Il sesto capitolo presenta i vari tipi di test che sono stati eseguiti e alla fine del capitolo faremo vedere qualche schermata dell'applicazione in produzione .
In fine nel capitolo sette riassume le conclusioni tratte al termine dell'intero lavoro di tesi e alcuni possibili sviluppi futuri.

CAPITOLO 2

CONTESTO TECNOLOGICO

2.1 Architettura di partenza

L'azienda opera in un mondo open source con il frame work di front-end proprietario basato su tecnologia JAVA(Standard ed Enterprise) /**XML** /ORACLE /DB2 per integrazione applicazioni legacy e progetti web-based. I servizi di back-end sono servizi host . COBOL/CICS/DB2 per il mondo mainframe e.
E' stata richiesta una profonda modifica dell'architettura di partenza usando la tecnologia dei web service per poter facilitare l'integrazione tra il fornitore di servizi e i vari clienti riuscendo in questo modo ad avere un'unica fonte di servizi.

Il sistema tecnologico di partenza è definito nel seguente modo:

- database DB2
- application server AS400
- terminal client 5250
- MQ Series con sistema a coda di messaggi per la comunicazione dei sistemi open (web per esempio) con l'AS400
- Applicazione web del sito banca

La base dati è dunque definita in un DB2 Ibm e la gestione è demandata a programmi Cobol dell'AS400. La fruizione e la gestione delle informazioni avviene tramite una interfaccia a caratteri (terminale 5250) alla quale possono accedere soltanto utenti banca abilitati. Dunque un operatore di sportello, oppure un operatore telefonico, compresa la necessità del cliente, accede direttamente sul terminale 5250, ricerca il rapporto del cliente e accede al menu delle funzioni di gestione.

Tutte le richieste che arrivano invece dal sito internet, dall'IVR o in genere dalla multicanalità vengono invece smistate in una coda MQ sulla quale è in ascolto un componente dell'AS400 che evade le richieste e manda un messaggio di "eseguito" al chiamante con la risposta alla richiesta.

2.2 La scelta Java

Si è decisa una rivisitazione molto spinta dell'architettura di partenza al fine di far rientrare tutta la gestione del consumer financing nel framework standard di sviluppo applicativo definito dalla banca.
Il gruppo bancario in questione ha definito un proprio framework di sviluppo applicativo che prevede le seguenti componenti:
database preferibilmente Oracle, se proprio necessario DB2
applicazioni Java J2EE sviluppate all'interno di un framework comune. Per framework si intende:
suddivisione delle applicazioni in layer logici: business, presentation, web services
definizione di componenti di accesso ai dati e layout grafici comuni
utilizzo di tool comuni per la defizione dei menu e dei profili abilitativi
versionatura del software
comunicazione tra applicazioni del framework e quelle esterne al framework tramite web services o EJB (Enterprise Java Beans)

Nel nostro caso la rivisitazione è consistita nei seguenti step:
si è mantenuto il DB2 come database, operando anche una rivisitazione dello schema dati
si è sviluppata una nuova applicazione Java non compliant al framework come sistema di back end (BE) che va a sostituire l'application server AS400
si è sviluppata una nuova applicazione del framework utilizzata come front-end (FE) delle filiali che comunica con BE tramite EJB (Entreprise java bean)
sono stati sviluppati dei web services su FE per l'accesso dalla multicanalità evitando dunque il passaggio attraverso le code MQ Series

Il presente lavoro si occuperà principalmente dell'ultimo punto, ovvero dell'illustrazione dell'architettura realizzata per la pubblicazione dei web services per la multicanalità

2.3 L'ambiente tecnologico

2.3.1 Back-end (BE)

E' costituito da una applicazione J2EE che consiste principalmente delle seguenti parti:
un database DB2 contenente le strutture dati per accogliere – tra gli altri – le informazioni del seguente tipo:

- clienti
- prodotti
- rapporti
- contabilità
- parametrizzazioni

una serie di progetti per la gestione delle operazioni di back end:

- per la gestione dei clienti
- per la gestione dei prodotti
- per la gestione delle richieste di nuovi rapporti
- per la gestione delle richieste di postvendita
- per la gestione della contabilità
- per le contribuzioni a sistemi esterni

un front-end per l'accesso alle funzionalità di cui al punto precedente.

BE è costruito non attenendosi agli standard di sviluppo imposti dalla banca, in quanto è stato realizzato da un fornitore esterno alla banca che ha realizzato il prodotto "a commessa".

2.3.2 Front-end (FE)

Il front end (FE), al contrario, è stato realizzato attenendosi agli standard di sviluppo dettati dalla banca. FE viene utilizzato dagli operatori di sportello o da partner commerciali della banca per gestire la relazione con il cliente, quindi:
accedere alle informazioni dei clienti e dei prodotti
fare dei preventivi
procedere all'allestimento della pratica di finanziamento
approvare la richiesta
liquidare la pratica

FE è stato sviluppato prevendendo la logica multi-tier del framework comune:
definizione della logica di business
definizione del presentation layer
definizione dei web services

Il grosso vantaggio dell'utilizzo del framework è quello di mettere a disposizione una serie di librerie e componenti prestabilite che possono essere riutilizzate per tutti i progetti della banca in modo da non dover sviluppare ogni volta parti cross per tutta la banca. In questo modo tutte le applicazioni del framework presenteranno un layout comune, comunicheranno in modo nativo con i legacy systems della banca,ecc.

L'obiettivo del presente lavoro è quello di illustrare il modo in cui FE è stato esteso con una serie di webservices per l'implementazione della multicanalità della banca. In questo modo i web services accedono alle strutture logiche di FE le quali accedono via Enterprise Java Beans alle strutture logiche di BE.

2.3.3 Sistemi esterni

I web services oggetto del presente lavoro verranno utilizzati da altre componenti web che realizzato il sito internet (WEB) della banca. I wsdl sviluppati in FE vengono forniti al gruppo di sviluppo del sito WEB per generare le classi client che vengono utilizzate nel sorgente del sito per effettuare le richieste di informazioni da visualizzare nelle pagine.

2.4 Scelta SOA

La ragione principale della scelta di utilizzare la tecnologia SOA basata sui Web Service è il "disaccoppiamento" che l'interfaccia standard esposta dal Web Service rende possibile fra il sistema utente ed il Web Service stesso: modifiche ad una o all'altra delle applicazioni possono essere attuate in maniera "trasparente" all'interfaccia tra i due sistemi; tale flessibilità consente la creazione di sistemi software complessi costituiti da componenti svincolati l'uno dall'altro e consente una forte riusabilità di codice ed applicazioni già sviluppate. Il secondo motivo è la possibilità di combinare facilmente i servizi sviluppati l'uno con l 'altro (indipendentemente da chi li fornisce e da dove vengono resi disponibili) per formare servizi "integrati" e complessi.

CAPITOLO 3

ANALISI FUNZIONALE E TECNICA

3.1 L'analisi funzionale

L'analisi funzionale si è basata sull'analisi funzionale esistente dei servizi forniti in precedenza ai vari clienti(internet banking, IVR). Dopo una serie di interviste con i vari utilizzatori del sistema usando i casi d'uso (Use Case Diagram) per chiarificare e concordare i requisiti del sistema abbiamo prodotto un documento che descrive l'interfaccia dei singoli servizi.
Nel seguito alcuni esempi di USE CASE DIAGRAM.

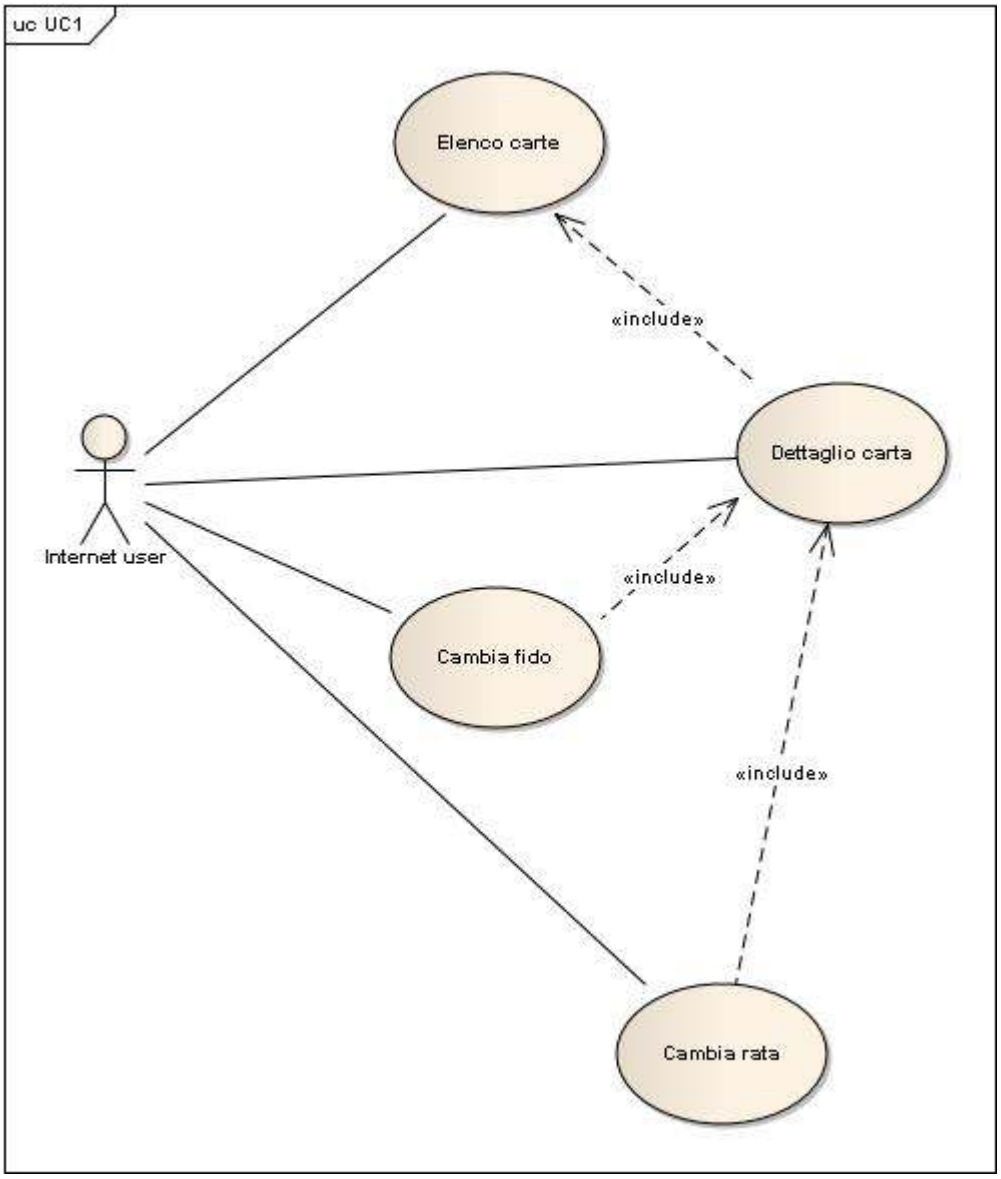

Figura 3.1 : use case relativo alla funzionalità di cambio fido

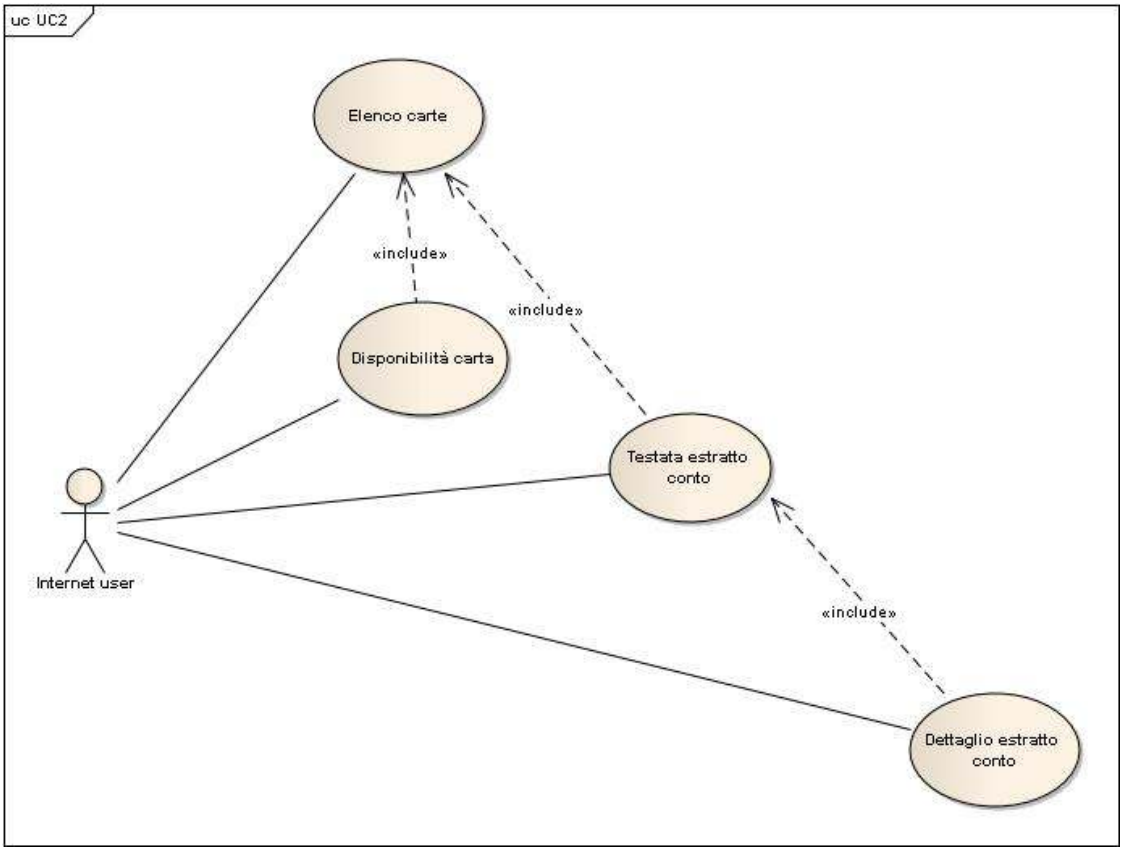

Figura 3.2 : use case relativo alla funzionalità di estratto conto

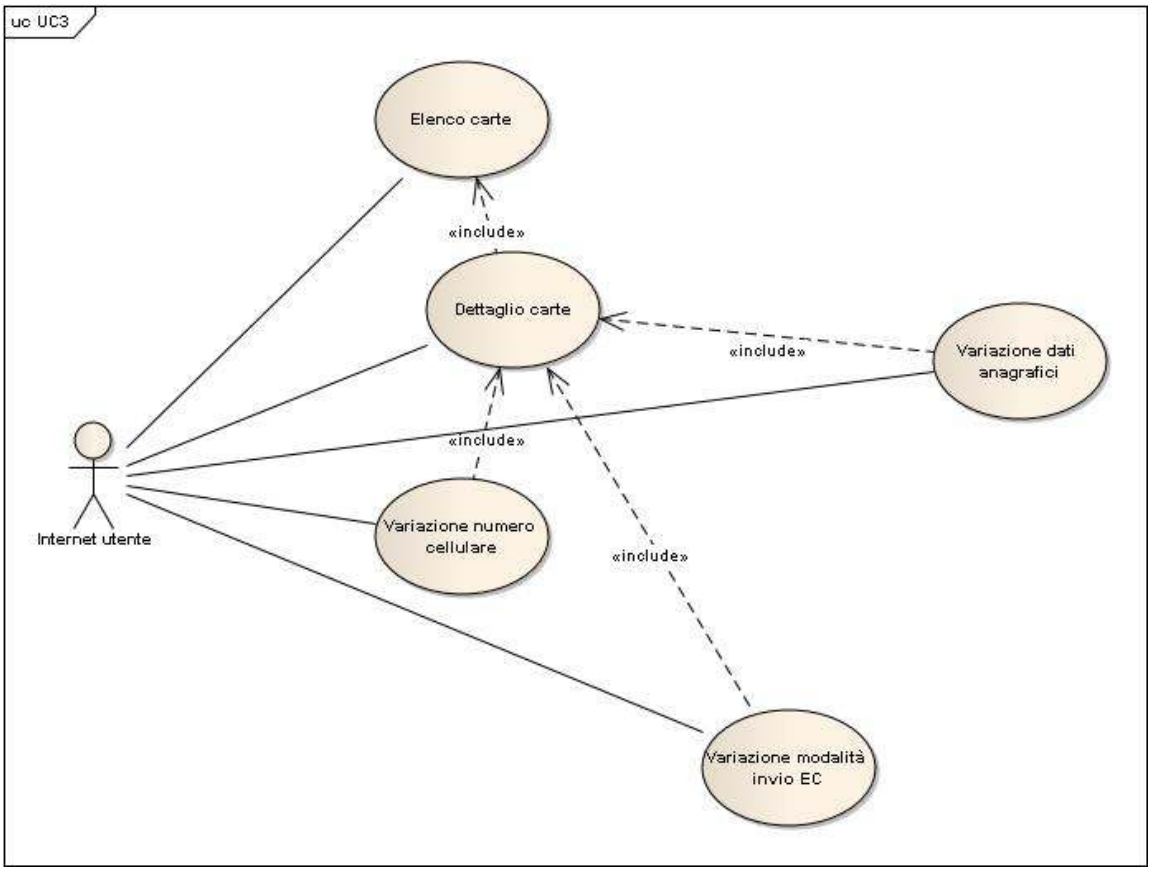

Figura 3.3 : use case relativo alle funzionalità che riguardano i dati anagrafici

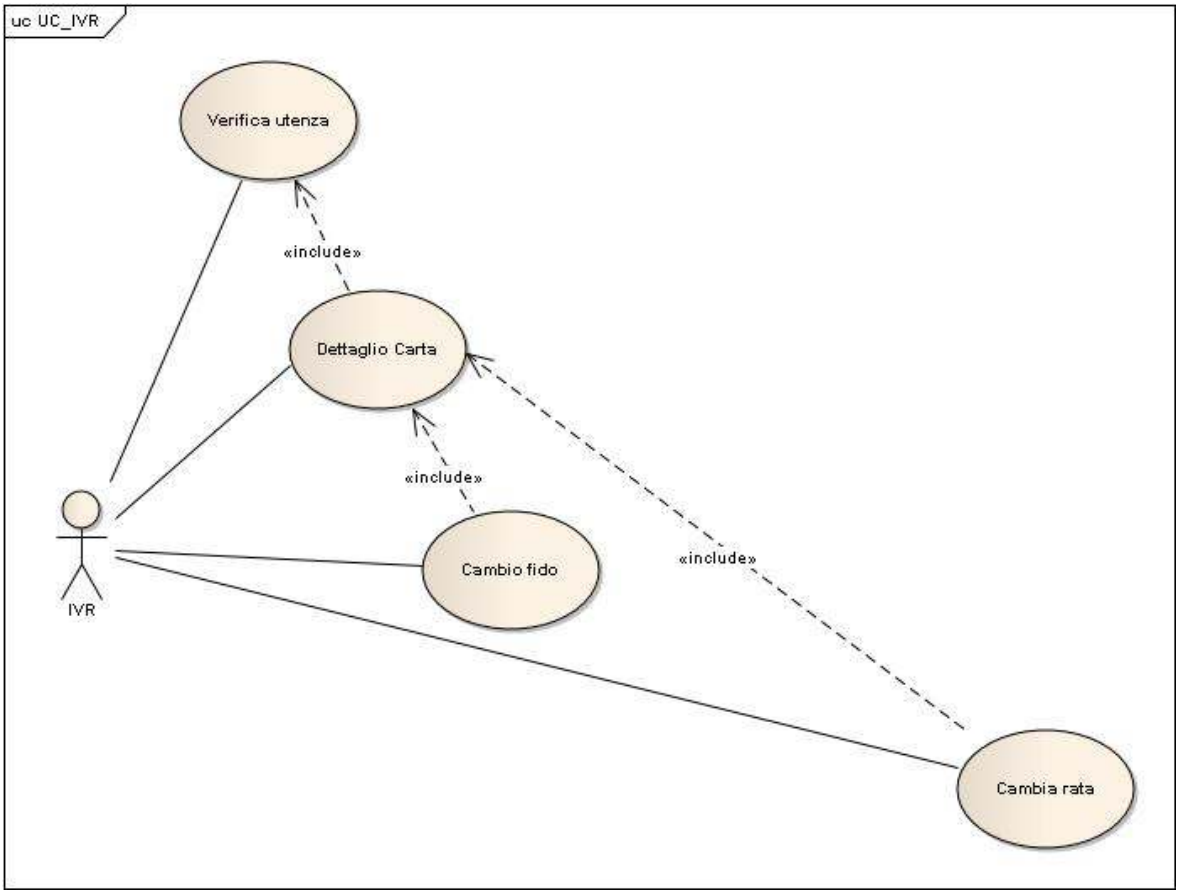

Figura 3.4 : use case relativo alla funzionalità di dettaglio carta

3.2 Analisi tecnica

3.2.1 Profili

Nonostante la banca abbia i propri profili definiti su HOST abbiamo provveduto a creare le nostre utenze tecniche per gestire le autorizzazioni ad eseguire o meno i servizi in base al profilo ricevuti in input dal client. Per cui i client mandano sempre in input i propri profili , se il profilo è abilitato ad eseguire l'operazione allora l'elaborazione continua in caso contrario viene mandato un messaggio di errore al chiamante.

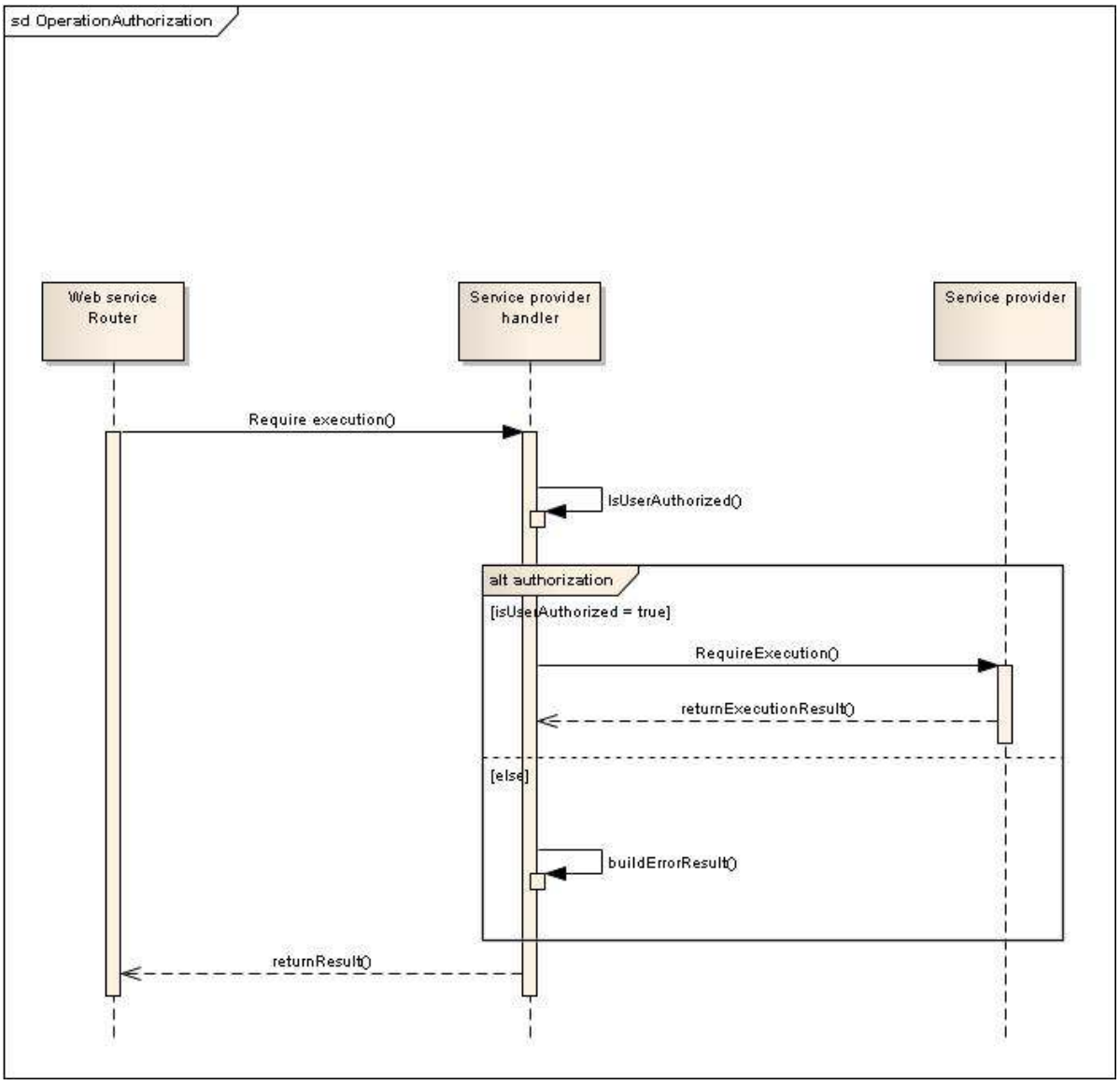

Figura 3.5 : sequence diagram relativo all'autorizzazione del profilo ad eseguire un'operazione

Il client manda una richiesta di esecuzione di un servizio con il proprio profilo. La richiesta di esecuzione è intercettata dal'handler del service provider che controlla se il client è autorizzato o meno ad eseguire l'operazione desiderata. In caso positivo la richiesta di esecuzione vienne soddisfatta in caso contrario un messaggio di errore vienne mandato al client.

3.2.2 Problematiche di sicurezza

L'applicazione è stata sviluppata in ambito bancario per cui è soggetta ai vincoli di sicurezza definiti dall'azienda. L'autenticazione si basa su token.
I client per essere autenticati aggiungono nell'header del messaggio SOAP le informazioni di autenticazione(Username e token). Se il messaggio ricevuto soddisfa i requisiti di sicurezza del server allora viene eseguita l'operazione richiesta in caso contrario viene mandato un messaggio di errore al chiamante.

La sicurezza dei web service si basa sul'utilizzo di SAML(**Security Assertion Markup Language** è uno standard informatico per lo scambio di dati di autenticazione e autorizzazione dette asserzioni tra domini di sicurezza distinti) per estendere e trasformare le informazioni presenti nei messaggi SOAP scambiati tra client e server. Il WAS fornisce tre tecniche di sicurezza applicabili ai web service a livello di messaggio,singolarmente o combinati tra di loro.

Autorizzazione

Si basa sul controllo delle concessioni fatte al client(client's grant), usate per filtrare e gestire gli accessi ai servizi pubblicati sul server.

Il client fornisce le informazioni necessarie al server per poter essere identificato, se questo ultimo è autorizzato a richiedere il servizio specificato, il servizio viene eseguito ed la risposta mandata al chiamante. In caso contrario (cliente sconosciuto o non autorizzato) , il server torna un messaggio di errore al client.

Il WAS 5.0 fornisce diversi meccanismi di autenticazione:

basic authentication: il client fornisce un nome utente ed una password. Il sever deve essere configurato per esguire la look up nel registro corretto.

id assertion: l'autenticazione è basara su un nome utente ed u token utente inclusi nel'header del messaggio di richiesta.

Ci sono diverse tecnologie per eseguire l'autenticazione usando il server di applicazione Websphere : O.S. based, LDAP repository, LTPA, Kerberos etc

Integrità

Meccanismo introdotto per permettere al ricevente del messaggio di poter controllare se quest'ultimo è stato modificato.

Il controllo di integrità si basa sul'inserimento di una signature nel header del messaggio di richiesta calcolato sulla del contenuto di una o più parti del messaggio(specificato dal'utente) usando in combinazione un algoritmo ed una chiave. Quando il server riceve il messaggio clacola la signature sulle stessi parti del messaggio usando lo stesso algoritmo e chiave e la confronta cpoon quella inserita nel'header del messaggio.Se le signature sono diverse il messaggio è scartato dal server che manda un messaggio di errore.

La signature può essere considerate sicura soltanto se il client e server hanno ottenuto la chiave usata per calcolare la signature e le hanno scambiato attraverso un canale sicuro.

Confidenzialità

Tecnica usata per assicurare che solo le persone autorizzate abbiano accesso alle risorse scambiate (messaggi).

I dati sono cifrati dal cliente usando una chiave specifica ed un algoritmo specifico per rendere illeggibili una o più parti del messaggio come specificato dallo sviluppatore. Lato server è usata una combinazione chiave –algoritmo per decifrare i dati ricevuti. Il messaggio di risposta mandato al cliente può essere cifrato.

Il principale vantaggio nel'usare questo metodo di cifratura in sostituzione della soluzione che fa uso del protocollo SSL è la mancanza di scambio di chiave tra client e server prima della trasmissione della richiesta poi la tecnica di cifratura è basata su parti di messaggio specificate dal'utente. Abbiamo un miglioramento delle prestazioni dovuto al fatto di non necessariamente cifrare tutte le informazioni.

Indipendentemente dal ruolo nell'architettura, il client o il server,può scartare messaggi che non soddisfino le regole di sicurezza definite. Un ricevente che non richiede nessuna informazione di integrità, token di autentificazione oppure altre informazioni di sicurezza ignora le informazioni aggiuntive senza segnalare errori. Ovviamente se un client manda un messaggio cifrato, il ricevente lo scarta non essendo in grado di decrittarlo senza le informazioni di confidenzialità appropriate.

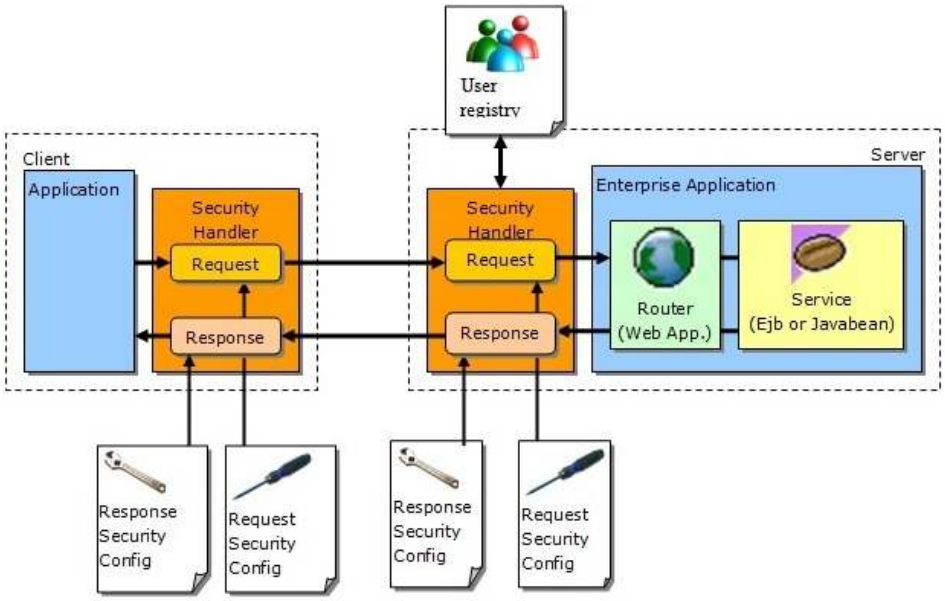

Figura 3.6 : Client SOAP usando le regole e i tag SAML

Client: La figura descrive un'applicazione client S.O.A.P che supporta l'uso delle regole e dei tag S.A.M.L nei message di richiesta. La presenza del security handler è una caratteristica particolare dei client create con I wizar del WSAD

Security Handler: è il component che legge le configurazioni di sicurezza specificate dal'utente e applica le estensioni S.A.M.L nel messaggio di richiesta prima di mandarlo al server. Quando arriva il messaggio l handler controlla le informazioni di sicurezza come specificate nel file di configuration When arrive a message the handler checks the security information as specified (*webservices.xml* per il sever and *webservicesclient.xml* per il client).

Web Service Router: è la servlet usata per ricevere i messaggi S.O.A.P attraverso il canale HTTP e chiama l'implementazione vera del servizio-

Service: può essere un enterprise java bean o un bean java generico.

Transport channel: può essere HTTP, HTTPS o JMS. HTTPS può essere un'alternativa o usata insieme alle tecniche di sicurezza studiata prima intrucendo però un overhead nello scambio di messaggi client-server dovuto alla neccessità di applicare le tecniche di sicurezza a tutte le informazioni scambiate tra i due attori.

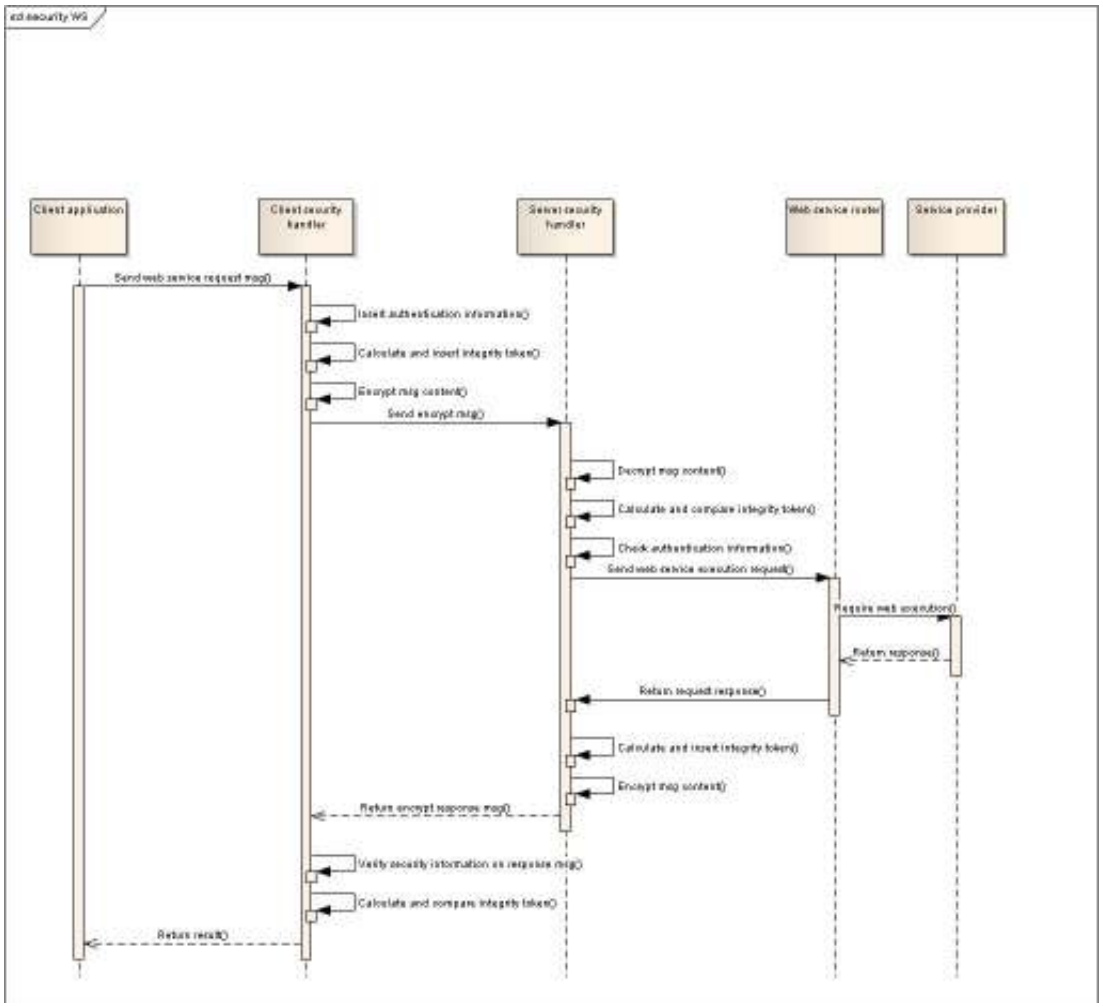

Figura 3.7 : Sequence diagram di richiesta di esecuzione di un'operazione

Il diagramma descrive il flusso di una richiesta di esecuzione di un'operazione.

Il client manda una richiesta di esecuzione.

In base ai requisiti del server e delle configurazioni di sicurezza ,il messaggio di richiesta verrà modificato come descritto in nella figura precedente:

Vengono aggiunte le informazioni di authenticazione.Se il server non si aspetta nessuna informazione di authenticazione non scarta la richiesta mandando al client un messaggio di errore.

Vengono aggiunte la signature di integrità calcolata usando le informazioni di authenticazione, il timestamp di invio e il contenuto del body del messaggio.

Nel caso in cui il server non si aspetta nessuna signature di integrity,il server non scarta larichiesta mandando un messaggio di errore al client.

Il Security Handler cifra parti specifiche del messaggio(informazioni di authenticazione,contenuto del messaggio). Se il server non si aspetta messaggi cifrati scarta il messaggio mandando un messaggio di errore al client dovuto all'impossibiltà di decifrare il messaggio e capirne il contenuto.

Il messaggio vienne mandato al server.

Se il messaggio non soddisfa I requisiti di sicurezza del server,questo ultimo lo scarta e manda un messaggio di errore al client.

N.B. Il messaggio di errore mandato al client deve soddisfare le sue regole di sicurezza in caso contrario il client genererà un altro messaggio di errore.

Il server verifica se il messaggio soddisfa I requisiti di sicurezza:

Il messagio vienne prima decifrato, poi il server controlla la signature di integrità,

infine controlla le informazioni di authenticazione.

Una volta che I controlli sono andati tutti a buon fine, il Router inoltra la richiesta in basa al protocollo al Service provider.

La risposta ottenuta è trasformata in un messaggio di risposta SOAP e vengono applicate le regole di sicurezza richieste dal client(Integrità e cifratura).

Il client quando riceve il messaggio di risposta, come appena visto per il server controlla se sono soddisfatte le regole di sicurezza.

3.2.3 Integrazione con le altre applicazioni

In Azienda ci sono varie applicazioni sviluppate ,ognuno delle quali ha un referente che stabilisce le interfacce verso l'esterno e le modalità di con le quali le operazioni vengono invocate.
Il primo passo è stato quello di identificare le applicazioni del gruppo con le quali bisogna interfacciarsi per realizzare le funzionalità desiderate.
Dopo aver identificato queste applicazioni sono state fissati degli incontri con i vari referenti per capire come vanno richiamate le funzionalità desiderate circa parametri di input,output, protocollo e gestione delle eccezioni.

Anagrafica
Le informazioni sull'anagrafica cliente vengono richiamate chiamando i metodi esposti via ejb.
L'applicazione che si occupa di gestire l'anagrafica è pure essa un'applicazione XFRAME.

Servizi Host
Integrazione con i servizi host avviene attraverso l 'integration framework che si occupa della comunicazione con il mondo mainframe. L'azienda ha messo a disposizione il prodotto XA-IFG che viene richiamato via ejb.

Back-end java
I servizi di back-end sviluppati ad hoc per l'applicazione vengono richiamati via ejb.

3.2.4 Localizzazione dei servizi interni ed esterni

La logica di business dell'applicazione vienne sviluppata nella parte di business framework.
Nell'elaborazione delle le richieste dei client i web services devono accedere alla logica di business per cui devono chiamare via RMI dei servizi(operazioni) sviluppati internamente o richiamare servizi forniti dall'esterno(Altri fornitori Ad esempio AFA,IFG, BACK-END SOXXX).
Le complessità dovute alla localizzazione e le relativa creazione dei servizi che possono richiedere accessi a risorse di rete e creare problemi di performance ci hanno spinto ad usare il pattern SERVICE LOCATOR per risolvere il problema.

Il pattern Service Locator gestisce le operazioni di localizzazione e creazione di Business Objects occupandosi di :

"nascondere" la complessità delle operazioni di ricerca (lookup) dei servizi,
fornire una modalità uniforme di lookup e creazione dei servizi,
isolare le eventuali operazioni di lookup "vendor dependent",
nascondere eventuali problematiche di networking,
gestire eventuali necessità di caching,
concentrando così la gestione delle problematiche in un singolo punto.

Motivazione

Il pattern garantisce una modalità uniforme e semplificata di localizzazione di risorse.

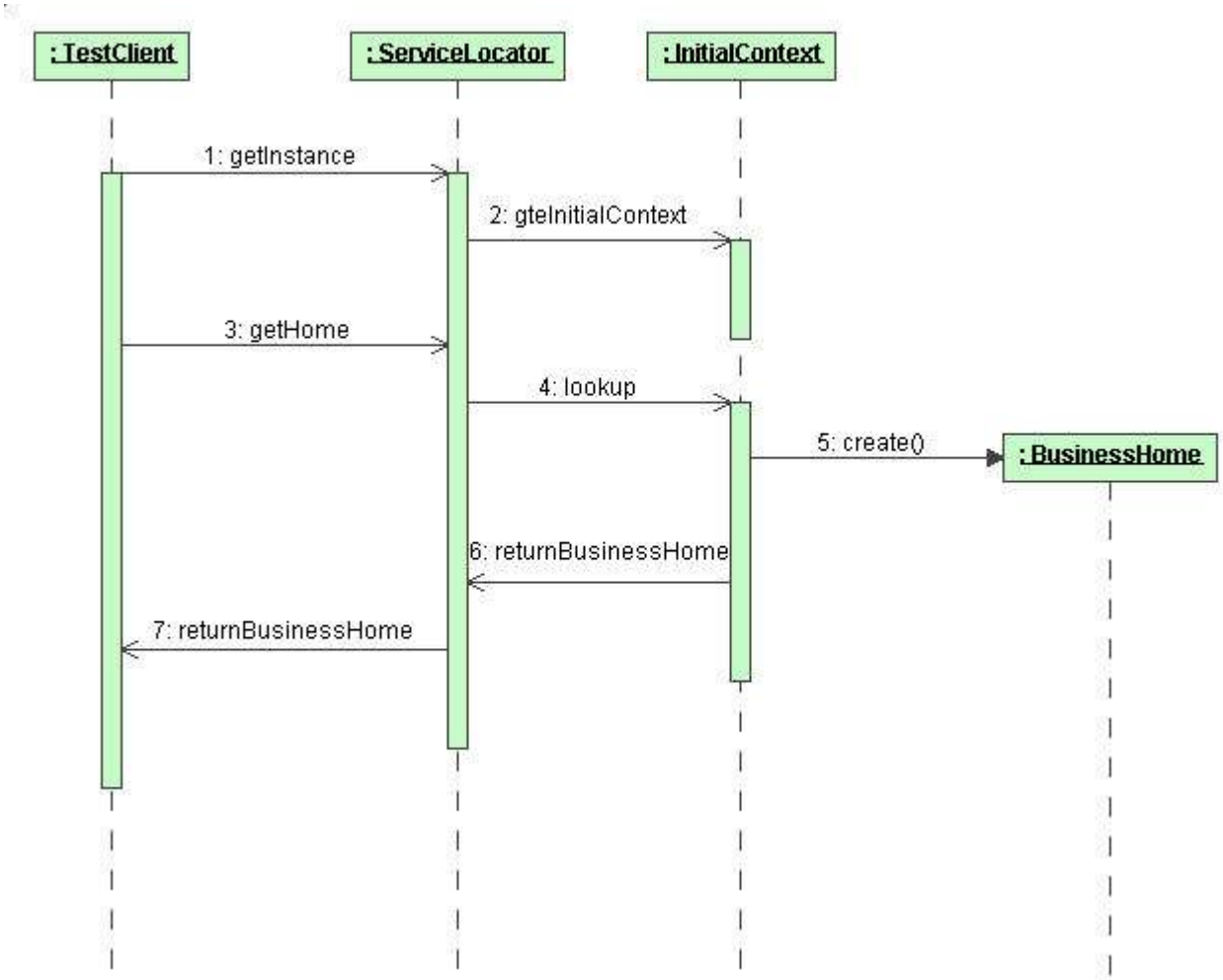

Figura 3.8: Sequence diagram del pattern LOCATOR

3.2.5 Log dei messaggi SOAP

Per poter risolvere eventuali problemi e motivi statistici è stato deciso di loggare sul database i messaggi S.O.A.P. scambiati tra client e server. Questi log possono essere abilitati o meno. Per cui quando arriva un messaggio se il log è attivo il messaggio viene salvato sul database e nel caso contrario no.

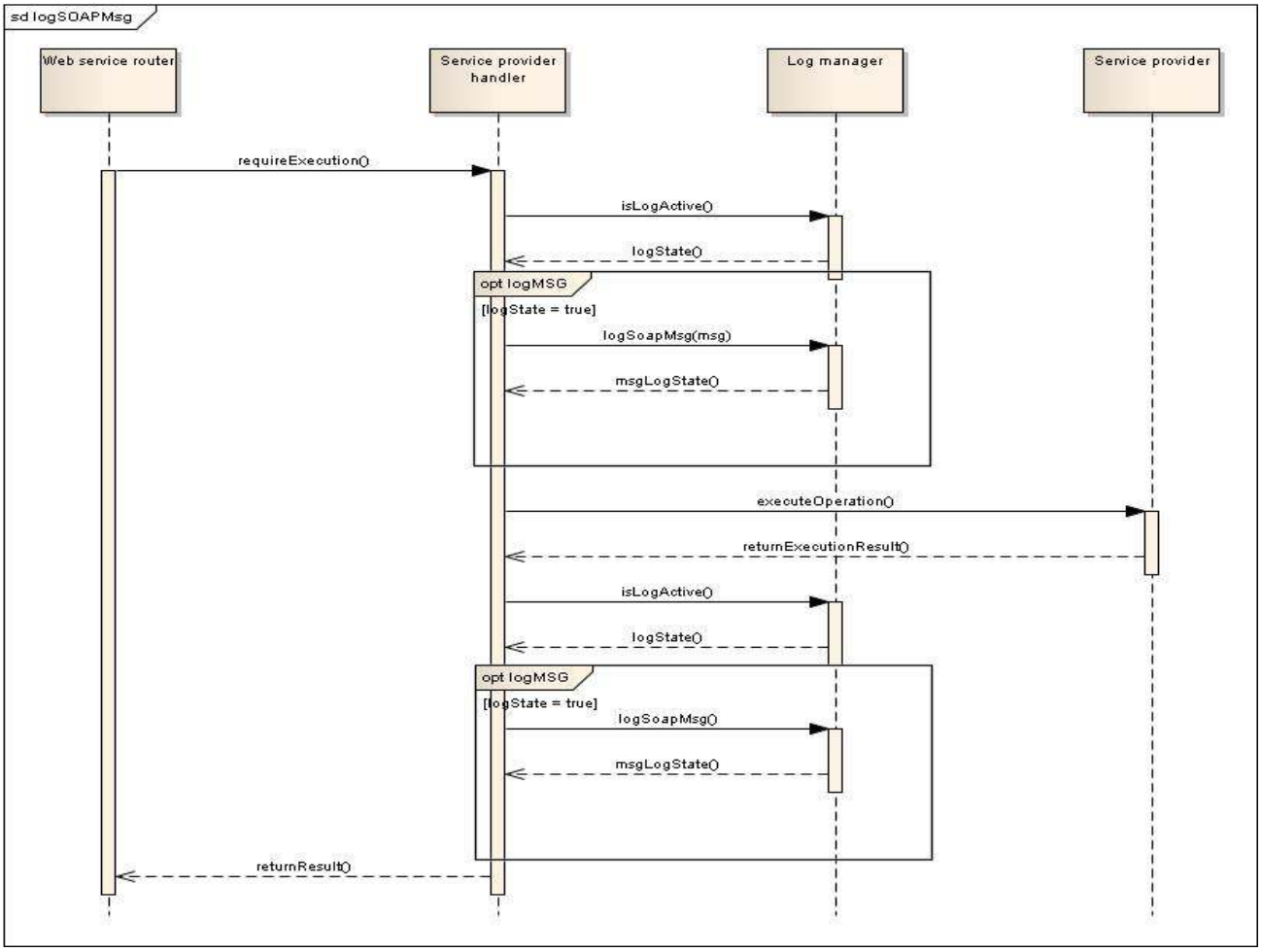

Figura 3.9: Sequence diagram del log dei messaggi SOAP

Il service provider handler intercetta i messaggi in entrata ed uscita.
Quando viene intercettato un messaggio , l'handler controlla se il log dei messaggi SOAP è attivo , in caso positivo il messaggio viene salvato sul database in caso contrario il messaggio viene inoltrato al service provider se il messaggio era in entrata oppure al client se il messaggio era in uscita.

3.2.6 Studio del framework

E stata sviluppata un'applicazione XFRAME che è lo standard aziendale.

3.2.6.1 Cos'è Xframe

Un framework è un insieme di componenti software, di processi di progettazione e di scrittura del software predefiniti e standardizzati (Design Pattern, best practices and guidelines).
XFrame è un framework per lo sviluppo di applicazioni orientate al web.
Utilizzando un framework comune per tutte le applicazioni è possibile ricomporre e riutilizzare componenti o servizi creati da altri sviluppatori per altre applicazioni
Fornisce un aiuto e una guida per chi progetta e sviluppa applicazioni web, sia per la possibilità di utilizzare componenti prefabbricate che per la possibilità di utilizzare metodologie già sperimentate con risultati determinati.
Esempio :Dovendo creare un'applicazione di conti correnti che permetta ad un operatore di aprire un conto corrente, il progettista e/o lo sviluppatore può riutilizzare i servizi dell'applicazione anagrafe per effettuare la ricerca anagrafica creati dallo sviluppatore dell'anagrafe

3.2.6.2 Perché usare XFRAME

Internazionalizzazione:
Creare un sistema standard di sviluppo di applicazioni web che avessero funzionalità multilingua e personalizzabili su diverse realtà nazionali.
Esempio: Creare l'applicazione di anagrafe sia per le banche italiane che per quelle estere.

SOA (Service Oriented Architecture):
Possibilità di riutilizzo delle funzionalità (servizi) create da UGIS da parte di clienti esterni ad UGIS (sistemi informativi di altre banche o di clienti) sempre con modalità standard.
Esempio: rendere disponibile le informazioni dei clienti tra banche e/o società del gruppo con sistemi informativi diversi.

Disaccoppiamento:
Rendere quanto più indipendenti le logiche di business (applicazioni di back-end su mainframe) dalle logiche di interazione dell'utente (applicazioni di front-end su web).
Esempio: rende facile la modifica del software web per cambiare la veste grafica delle informazioni senza impattare sui programmi mainframe.

Industrializzazione:
Rendere standard i processi comuni e ripetitivi durante le fasi di progettazione, sviluppo, test e distribuzione del software UGIS con la conseguente automazione, diminuzione dei tempi, tracciabilità e aumento della qualità dei processi e dei servizi

Scalabilità:
Creare applicazioni che vengono eseguite su sistemi facilmente adattabili alle esigenze di elaborazione. Esempio: con l'aumento del numeri di utenti è possibile aggiungere hardware al sistema senza dover necessariamente modificare il software.

Indipendenza da piattaforme hardware e da fornitori:
Utilizzare sistemi che siano quanto più indipendenti da piattaforme hardware (PC, AIX, System 370) e da soluzioni dipendenti dai fornitori (IBM, Microsoft, Sun, Oracle).
Esempio: spostare le applicazioni da un sistema con server Windows a un sistema con server Unix con un intervento minimo o addirittura nullo sulle applicazioni

3.2.6.3 Contesto tecnologico di XFRAME

XFrame è basato sul linguaggio di programmazione Java che è indipendente dalla piattaforma dove poi l'applicazione viene eseguita (può essere eseguito su computer Windows, Linux, Unix o IBM 370).

Più in specifico usa lo standard J2EE (Java 2 Enterprise Edition) che è la parte di linguaggio e di architettura software che permette di creare applicazioni il cui carico elaborativo può essere distribuito su più server e quindi essere molto elevato (molti utenti contemporanei).

XFrame è costituito da componenti scritte da UGIS orientate a facilitare la progettazione e scrittura di applicazioni con le caratteristiche già menzionate in precedenza (internazionalizzazione, disaccoppiamento, SOA, scalabilità)

A corredo del progettista e programmatore XFrame fornisce anche standard e direttive su come progettare e scrivere le applicazione fornendo un know-how comune tra gli sviluppatori.

3.2.6.4 Strati

Logicamente è diviso in tre strati (layer) dove sono confinate e risolte le problematiche tipiche di applicazioni Enterprise:

Presentation

Business

Integration

Data

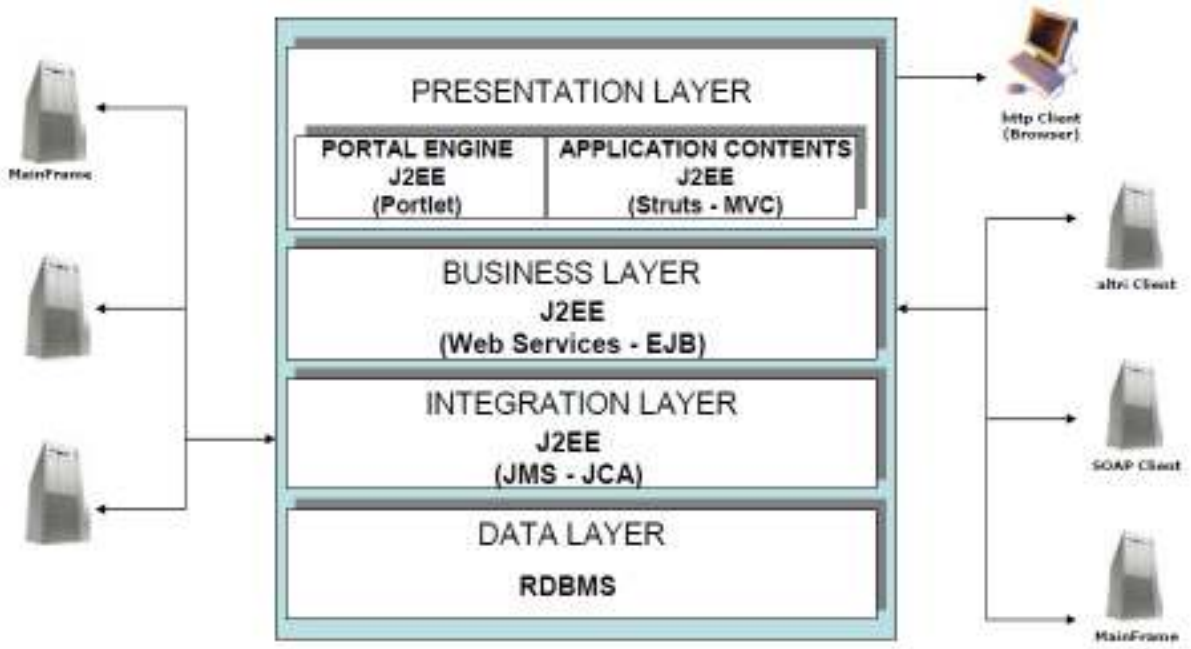

Figura 3.10: Layer XFRAME

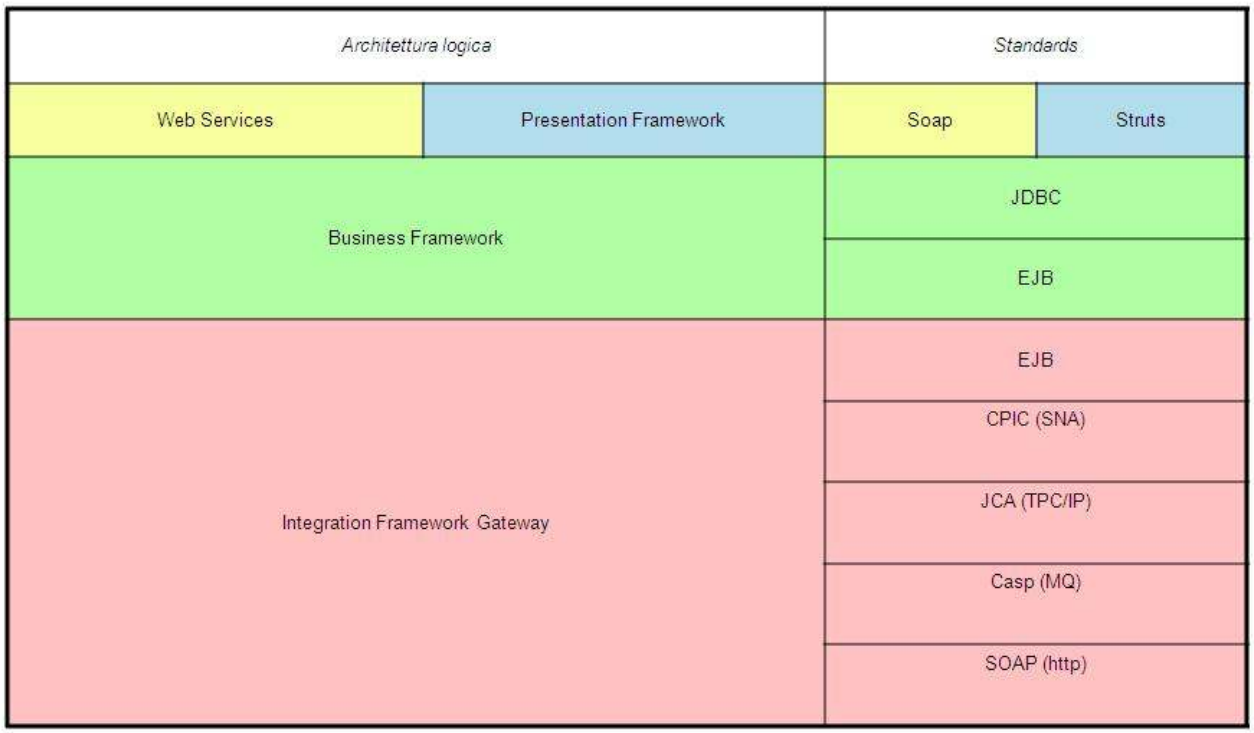

Figura 3.11: Layer XFRAME e tecnologie

PF (Presentation Framework)

Questo è lo strato dove vengono confinate e risolte le problematiche relative all'interfaccia con l'utente. In questo strato lo sviluppatore crea le logiche di interazione e le logiche di visualizzazione delle informazioni del sistema.

Gestisce quindi l'interazione con l'utente, permettendo all'utente di fruire delle funzionalità a lui necessarie.
Esempio:
permette di crea le pagine web di ricerca anagrafica, visualizzazione elenco clienti, selezione dell'cliente con il dettaglio del cliente (elenco conti), ecc.

BF (Business Framework)
In questo strato vengono risolte le problematiche di business dell'applicazione. Lo sviluppatore scrive il codice che interroga gli archivi dati e le logiche di elaborazione che modificano tali dati. Lo stato di business consente il disaccoppiamento tra la logica di presentazione e il modello business. Controlla la logica di servizi di business complessi, composti da uno o più servizi atomici. L'implementazione tecnica di questo strato è basato su EJB di tipo session o stateless.

I metodi del bean sono esposti da una facade comune come entry point alla logica di business. Possono essere esposti in interfacce WSDL completamente disaccoppiate e sono accedibili come web services attraverso il protocollo SOAP su una Web Application dedicata.
I WSDL permettono ai client esterni di chiamare Web Services attraverso il protocollo SOAP senza interagire con lo stato di presentazione.
Esempio:
crea il codice che aggiorna i dati anagrafici di un utente o registra un movimento di denaro o semplicemente riutilizza le transazioni su mainframe già esistenti per avere o modificare le informazioni.
La logica di business dovrebbe essere indipendente da come le informazioni vengono rappresentate all'utente finale in quanto le problematiche di visualizzazione (formato dati, impaginazioni, lingua) sono risolte dallo strato di Presentation.

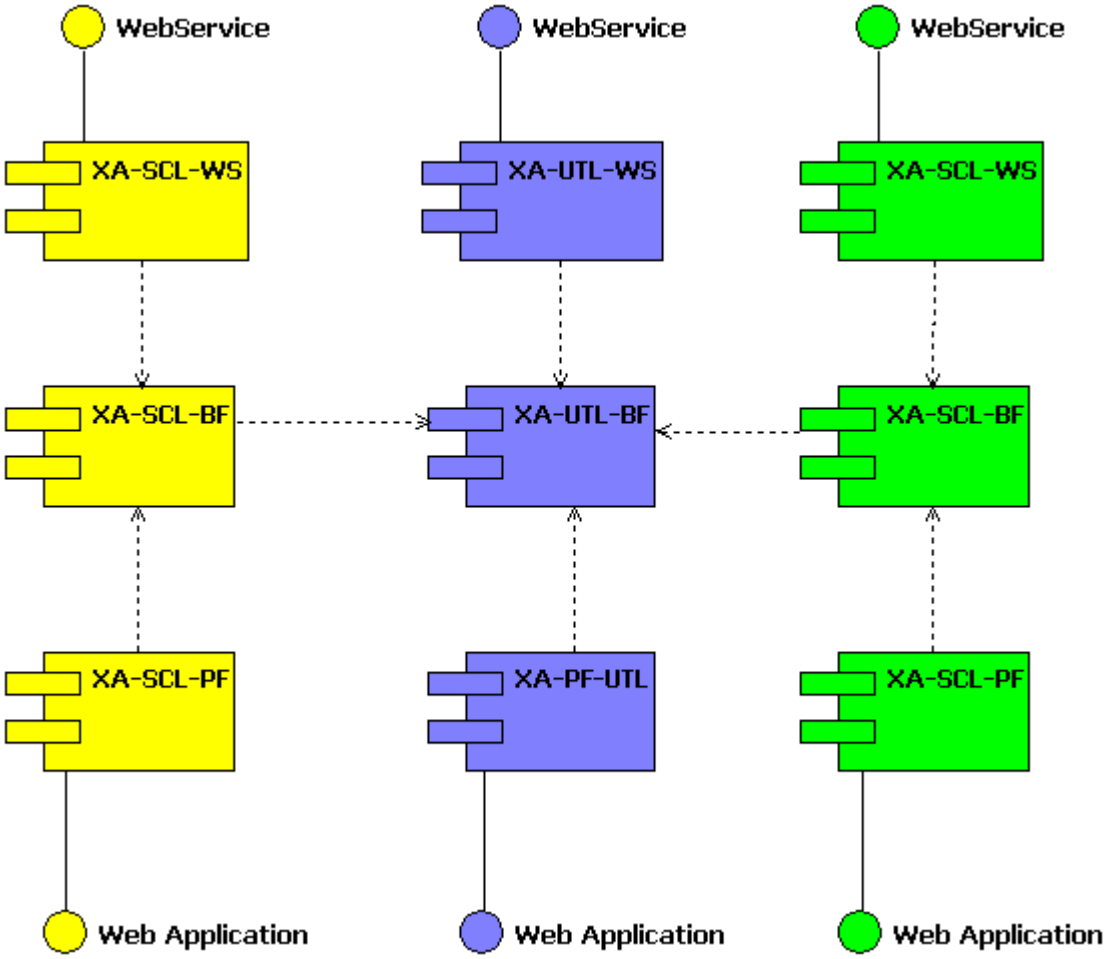

Figura 3.12: Business Framework

Questo strato permette di isolare le applicazioni e condividere i servizi di business.

IF - IFG

IF Integration Framework

In questo strato vengono risolte le problematiche di comunicazione con i sistemi esterni di back-end quali applicazioni, mainframe, sistemi informativi di fornitori esterni.
Lo strato di integration si prende carico di elaborazioni comuni a tutte le richieste verso i sistemi esterni:
Riceve le richieste da parte dello strato di business (BF),
Rende comprensibili le informazioni al particolare sistema esterno,comunica le informazioni al sistema esterno,

Riceve le informazioni prodotte dal sistema esterno.

Le rende comprensibili allo strato di business.

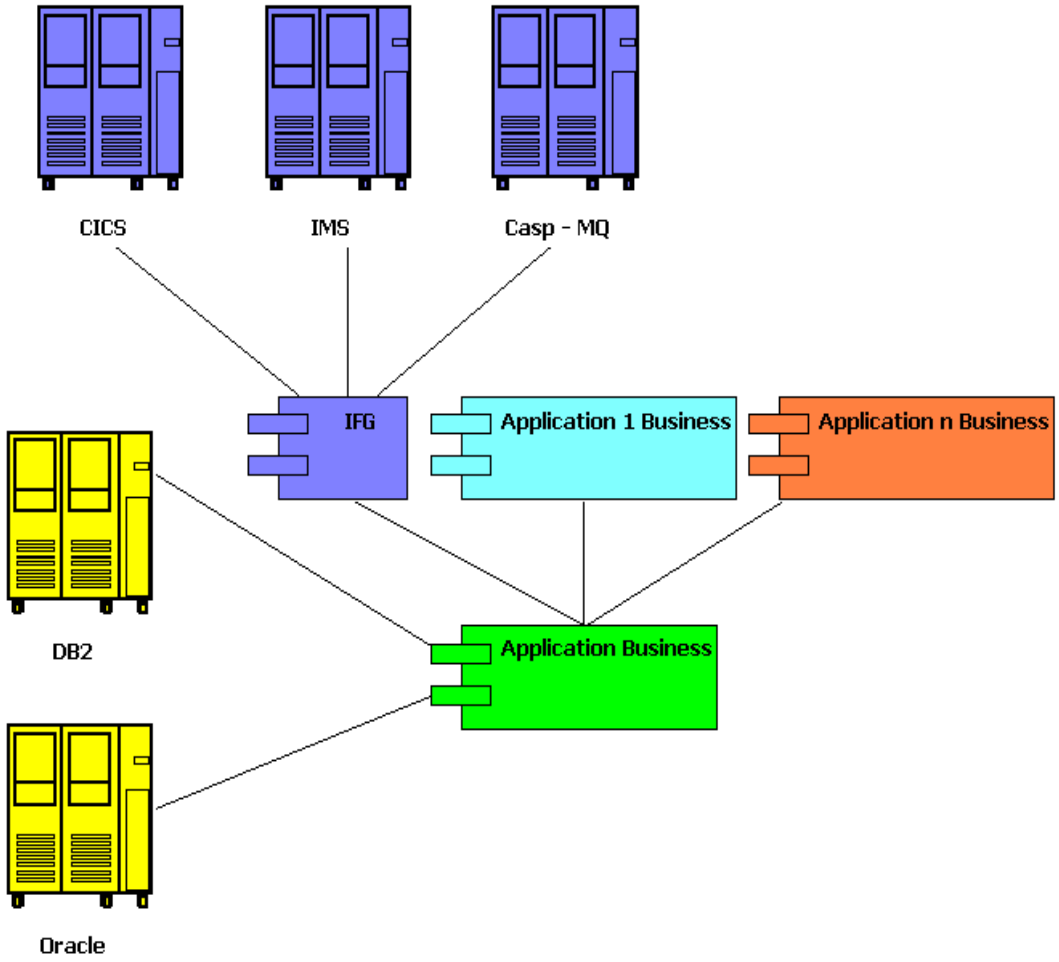

Figura 3.13: Integration Framework

Ricapitolando L'itegration framework permette:

L'accesso ai database (DB2, Oracle) via Interfacce JDBC.

L'accesso ai servizi di back end (IMS, CICS, Casp MQ) .

L'accesso ai servizi di business delle altre applicazioni via EJB.

3.2.6.5 Elementi correlati

Oltre alle componenti prettamente tecnologiche con XFRAME si è cercato di standardizzare le aree e i processi correlati allo sviluppo di applicazioni web

Fornire ai progettisti, sviluppatori e sistemisti strumenti standard utili ad automatizzare e facilitare i lavoro, selezionando i prodotti di mercato o creando prodotti interni.

Favorire un linguaggio di comunicazione e modelli di riferimento comuni.

Fornire documentazione specializzata all'uso di XFRAME.

Fornire una base formativa comune a tutti gli sviluppatori.

3.2.6.6 Prodotti

IBM - WAS (Websphere Application Server): Prodotto che permette di eseguire e gestire le applicazioni J2EE.

IBM - WSAD (Websphere Studio Application Developer): Prodotto orientato allo sviluppatore per scrivere codice J2EE.

Serena - PVCS: Prodotto per il versioning e change management. Permette gestire e versionare i cambiamenti del codice applicativo.

Rational - XDE: Prodotto per la modellazione UML. Permette di progettare applicazioni Java.

UGIS - Gandalf: Prodotto per il rilascio e le installazione delle applicazioni nelle macchine server.

3.2.7 Moduli sviluppati

E' stata sviluppata un'applicazione secondo lo standard aziendale con i seguenti moduli.

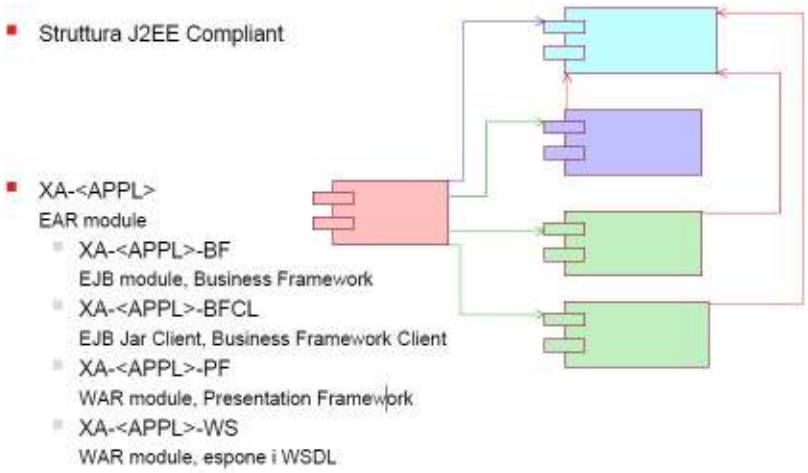

Figura 3.14: Moduli XFRAME

XA-XXX-BF (modulo EJB)

In questo modulo viene implementata la parte di logica applicativa.

XA-XXX-BFCL(Utility)

Questo modulo contiene tutte le classi di cui ha bisogno il client(in questo caso il modulo XA-XXX-WS) per poter usa la propria vista dei entreprise beans contenuti nel modulo XA-XXX-BF.

XA-XXX-WS (modulo war)

In questo modulo vengono sviluppati I web service.

3.2.8 Servizi Esistenti

E’ stato necessario incapsulare i seguenti servizi esistenti che stavano su macchine HOST :

Dettaglio Anagrafica, Elenco Carte.

Per accedere ai servizi su HOST abbiamo dovuto usare l’Integration frame work nel modo descritto in precedenza.

CAPITOLO 4

IMPLEMENTAZIONE

In base all'analisi funzionale e quella tecnica , lo sviluppo si è mosso nelle seguenti linee :

- Sviluppi di servizi web
- Sviluppo di web service handler
- Logica di integrazione (Entreprise java beans)
- Sviluppo database

4.1 Servizi web

Sono stati svilupatti dei servizi web con l'IDE (WSAD) che mette a disposizione dei plugin che permettono di genere tutte le classi , i file di mapping e di configurazione richiesti per il deploy di questi ultimi .
I servizi sviluppati , basati su JAX-RPC sono delle normali classi java che verrano poi esposte come servlet e dunque ospitate all'interno del web container.
Le classi esposte come servlet devono rispettare le seguenti regole :

- La classe deve essere pubblica non statica e non *final.*
- La classe deve dichiarare un metodo pubblico senza argomenti per la sua creazione, mentre gli altri metodi non possono essere *final.*
- Devono essere implementi tutti i metodi dell'interfaccio del servizio.

4.1.1 Web servizi con JAX-RPC

JAX-RPC sta per Java API for XML-based RPC ed implementa un generico sistema di chiamata a procedura remota (RPC) su XML.
JAX-RPC è una tecnologia usata per sviluppare servizi e client che usano chiamate a procedure remote e XML

I meccanismi di chiamate a procedure remote sono spesso usate nei modelli distribuiti Client-Server per permettere ai client di eseguire procedure che stanno su altri sistemi.

Con JAX-RPC , le chiamate a procedure remote sono rappresentate da un protocollo basato su XML tipo SOAP.

Le specifiche di SOAP definiscono la strutture, le regole di codifica, le convenzioni per rappresentare le chiamate a procedure remote e le risposte. Le richieste e le risposte sono trasmesse come messaggi SOAP su http.

JAX-RPC nasconde la complessità dei messaggi ai sviluppatori. Nella parte server , lo sviluppatore specifica le chiamate a procedure remote definendo metodi in un'interfaccia scritta in java e scrive una o più classi per implementare questi metodi. Il client è semplice da sviluppare dato che deve solo creare un proxy (un oggetto locale che rappresenta il servizio da invocare) ed invoca il metodo del proxy. Lo sviluppatore non ha bisogno di generare o fare il parsing dei messaggi SOAP. Se ne occupa invece il sistema di runtime di JAX-RPC .

JAX-RPC implementa invece un generico sistema di RPC su XML; la tecnologia non è vincolata a SOAP, ma aperta ad eventuali evoluzioni future. E' invece basata fortemente su WSDL, che utilizza per implementare la mappatura XML->Java e viceversa. Con JAX-RPC non è necessario codificare a basso livello, se si dispone di un WSDL, si possono generare le classi Java che implementano il client ed il server.

La stessa cosa può avvenire a partire da una interfaccia che estende *java.rmi.Remote*.
JAX-RPC ha il vantaggio di non essere restrittivo dato che un client JAX-RPC può accedere ad un servizio che non è in esecuzione su una piattaforma java e viceversa. Questa flessibilità è dovuta al fatto che si basa su tecnologie definite dal World Wide Web Consortium (W3C): HTTP, SOAP e Web Service Description Language (WSDL).

JAX-RPC è dunque più indicato per sviluppare applicazioni complesse, con molti servizi, e per integrare Web Service esistenti e completamente standard, quando non sia necessario entrare nel dettaglio tecnico della struttura dei messaggi SOAP di richiesta e risposta. Con JAX-RPC un servizio Web viene acceduto come se fosse un oggetto Java locale.

4.1.2 Architettura di JAX-RPC

JAX-RPC si basa su una architettura composta da codice intermedio che implementa la comunicazione a basso livello, lasciando a livello del programmatore solo le interfacce del servizio. (FIG) è presente un esempio, dove sia il client che il server utilizzano JAX-RPC per la comunicazione.

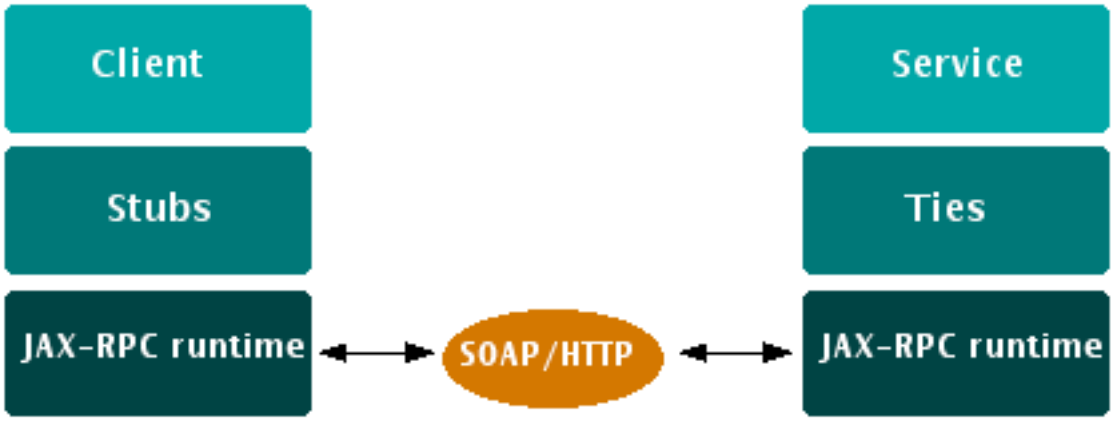

Figura 4.1 : Architettura di JAX-RPC

Il client dialoga con una interfaccia, generata da strumenti di sviluppo (in JWSDP è presente lo strumento xrpcc), prodotta a partire da un file WSDL. L'interfaccia è implementata da una classe che implementa i dettagli della comunicazione SOAP, lo *Stub*. Al di sotto di questo, il runtime di JAX-RPC si occupa della comunicazione tramite SOAP/HTTP con il *Tie*, il corrispondente dello *Stub* sul lato server. E' poi compito del *Tie* di comunicare con la reale implementazione del servizio.

Gli *Stub* sono oggetti locali che rappresentano le procedure remote. I *Tie* sono classi lato server che si occupano della comunicazione con i client.

La sequenza di esecuzione è la seguente :

1. Il client chiama un metodo dello *Stub* che rappresenta la procedura remota .
2. Lo *Stub* esegue le routine necessarie usando le rutine JAX-RPC di sistema per convertire la chiamata al metodo in un messaggio SOAP e lo trasmette al server come richiesta HTTP.
3. Il server quando riceve il messaggio SOAP invoca metodi del runtime JAX-RPC per convertire il messaggio SOAP in una chiamata a procedura.
4. Il runtime di JAX-RPC richiama il metodo sul *Tie* dell'oggetto.
5. Il *Tie* chiama il metodo che implementa il servizio web.

6. La risposta è mandata in messaggio SOAP come risposta http

E' però possibile utilizzare xrpcc (un tool che legendo un file di configurazione chiamto config.xml riesce a generare vari file necessari per il deploy di un servizio web) per generare solo uno degli strati: quello client o quello server. Per creare uno stub è sufficiente avere a disposizione un file WSDL: dato in input allo strumento xrpcc, produce tutto il codice necessario per comunicare con il servizio. Il client non deve far altro che importare il package relativo al codice generato e deciso prima dallo sviluppatore ed instanziare la classe che rappresenta il servizio.
Per generare il lato server, è possibile partire, oltre che dal documento WSDL, anche da una interfaccia Java che estenda java.rmi.Remote e dove ciascun metodo sollevi l'eccezione java.rmi.RemoteException. A partire da questa xrpcc è in grado di generare i tie ed il documento WSDL di descrizione, oltre agli altri file necessari per installare il servizio in un Web Container.

4.1.3 Mappatura dei tipi

Nell'implementare la corrispondenza biunivoca tra WSDL e Java, JAX-RPC deve basarsi su un preciso standard che indichi in che modo un determinato tipo, presente in uno dei due mondi, debba essere rappresentato nell'altro. JAX-RPC dispone di una mappatura dei dati primitivi (FIG), ma anche degli array, delle strutture dati complesse e delle enumerazioni.

xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.rpc.namespace.QName
xsd:dateTime	java.util.Calendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]

Figura 4.2 : Mappatura dati primitivi tra WSDL e Java

Se da una parte i dati primitivi trovano spesso diretta corrispondenza, e gli array WSDL possono essere mappati direttamente in array Java, le strutture dati complesse e le enumerazioni WSDL richiedono più lavoro. In questi casi vengono generate dagli strumenti di sviluppo classi apposite. Nel primo caso vengono generati componenti Javabeans strutturati secondo il modello presente nel documento WSDL.

4.1.4 Modelli di interazione

Nonostante WSDL definisca quattro diversi modelli di interazione tra il client ed il servizio, JAX-RPC supporta solo l'oneway e la request-response, più uno diverso, detto invocazione non-bloccante. Quest'ultimo è una variante della modalità request-response e prevede l'invio di un messaggio di richiesta e la ricezione di un messaggio di risposta, ma nel tempo che intercorre tra la ricezione della richiesta e la produzione della risposta, il client è svincolato dall'attesa. Questa modalità è disponibile però solo all'interno di container J2EE, dove è il server a poter rimanere in attesa della risposta al posto del client.

4.1.5 Sviluppo servizi web

Come è stato detto in precedenza, i servizi implementati sono stati esposti come servlet. I passi per implementare i singoli servizi sono i seguenti :

1. Scrivere una classe che implementa i metodi di business del servizio web. La classe è categorizzata come servlet ma non ha bisogno di estendere Servlet o HttpServlet.
2. Scrivere un’interfaccia remota per tutti i metodi che il servizio dovrà esporre.
3. Fare il mapping della servlet creata nel web.xml .
4. Scrivere o generare il file WSDL(web service description language). Questo file fa parte dello standard dei web service e descrivere il servizio da pubblicare assieme ai tipi dei parametri i valori di ritorno. Esso è indipendente dalla piattaforma.
5. Scrivere o generare il file di mapping JAX-RPC. L’application server usa questo file per fare il mapping tra una richiesta di servizio web ed una servlet.
6. Scrivere i dati relativi al servizio web nel file web service.xml .

Per realizzare i servizi con l‘ambiente di sviluppo usando i wizard ed i plugin messi a disposizione , l’unico passo da compiere è il primo.

Si parte quindi dall'implementazione del servizio e tutti gli altri file vengono generati (oppure aggiornati) automaticamente. Vengono anche generate le classi del *Tie.*
Nel seguito Faremo vedere un esempio completo di sviluppo di servizio web .
Useremo il servizio che torna il saldo disponibile sulle varie linee della carta di credito di un determinato utente.

4.1.6 Implementazione del servizio

Le classi esposte come servlet devono rispettare le seguenti regole :

- La classe deve essere pubblica non statica e non *final.*
- La classe deve dichiarare un metodo pubblico senza argomenti per la sua creazione, mentre gli altri metodi non possono essere *final.*
- Devono essere implementi tutti i metodi dell'interfaccio del servizio.
- Le classi passate in input ed in output devono implementare l interfaccia *java.io.Serializable* .

Codice di una classe che implementa un servizio

```
package it.usi.xframe.anw.services;

import it.usi.xframe.an2.xpe.bfcl.util.QXBetwixUtil;
import it.usi.xframe.anw.beans.AnwGeneralResponse;
import it.usi.xframe.anw.beans.LoginInput;
import it.usi.xframe.anw.beans.LoginOutput;
import it.usi.xframe.anw.bfutil.ANWConstants;
import it.usi.xframe.anw.externalinput.ANWDisponibilitaCartaInput;
import it.usi.xframe.anw.internalinput.ANWDisponibilitaCartaInputWS;
import it.usi.xframe.anw.externaloutput.ANWDisponibilitaCartaOutput;
import it.usi.xframe.anw.internaloutput.ANWDisponibilitaCartaOutputWS;
import it.usi.xframe.anw.util.LoginUtil;
import it.usi.xframe.anw.util.WebServiceUtil;
import it.usi.xframe.anw.ws.general.ANWBeanWS;

import org.apache.log4j.Logger;

public class DisponibilitaCarta {
	private static final Logger m_Log = Logger.getLogger(DisponibilitaCarta.class);
	private static DisponibilitaCarta instance = null;
	protected static synchronized DisponibilitaCarta getServiceInstance() {
		if (instance == null)
			instance = new DisponibilitaCarta();
		return instance;
	}

	public ANWDisponibilitaCartaOutput disponibilitaCarta(
```

```
ANWDisponibilitaCartaInput input){
        ANWDisponibilitaCartaOutput output = new ANWDisponibilitaCartaOutput();
        ANWDisponibilitaCartaOutputWS outputWS
        = new ANWDisponibilitaCartaOutputWS();
        try{
                LoginInput loginInput = new LoginInput();
                loginInput.setUserID(input.getAnwGeneralHeader().getOperatorID());
                //Chiamata al servizio di login
                loginInput.getUserInfo().setUserId(
                        input.getAnwGeneralHeader().getOperatorID()
                );
                LoginOutput loginOutput = LoginUtil.eseguiLogin(loginInput);
                //Controllo se è censito l'utente
                if(loginOutput.getEsito().equalsIgnoreCase(
            ANWConstants.ANW_UNSUCCESS_ESITO)
    ){
            AnwGeneralResponse anwGeneralResponse = WebServiceUtil.setWebServiceOutputWithErrors(
            input.getAnwGeneralHeader().getOperatorID(),
            input.getAnwGeneralHeader().getPartnerCode(),
            loginOutput.getDescrizioneErrore(), input.getAnwGeneralHeader().getSequenceNumber()
            );
                        output.setAnwGeneralResponse(anwGeneralResponse);
                        return output;
                }
                //Chiamata Al servizio disponibilita Carta
                outputWS = callService( input, loginOutput );
                //Trasformo l'output interno in quello esterno
                output = transFormOutput( output, outputWS,  input );
                m_Log.info("disponibilitaCarta.output-->" + QXBetwixUtil.Bean2XML(output));
        }catch (Exception e) {
    AnwGeneralResponse anwGeneralResponse =
    WebServiceUtil.setWebServiceOutputWithErrors(
             input.getAnwGeneralHeader().getOperatorID(),
        input.getAnwGeneralHeader().getPartnerCode(),                    e.getCause().getMessage(),
                                                    input.getAnwGeneralHeader().getSequenceNumber()

    );
                output.setAnwGeneralResponse(anwGeneralResponse);
                m_Log.error(e, e.getCause());
        }
        return output;
}

private ANWDisponibilitaCartaOutputWS callService(ANWDisponibilitaCartaInput input,LoginOutput loginOutput ) throws Exception{
        ANWDisponibilitaCartaOutputWS outputWS = new ANWDisponibilitaCartaOutputWS();
        ANWDisponibilitaCartaInputWS inputWS = new ANWDisponibilitaCartaInputWS();
        inputWS.setUserInfo(loginOutput.getUserInfo());
        inputWS.setUserIntermediationChain(loginOutput.getUserIntermediationChain());
        inputWS.setDisponibilitaCartaInput(input);
        outputWS = ANWBeanWS.getInstance().disponibilitaCarta(inputWS);
```

```
            return outputWS;
        }
}
```

Dopo aver scritto il codice precedente viene usato il wizard del WSAD che genera in modo automatico le classi necessarie per il deploy del servizio web.

4.1.7 Service endpoint interface (SEI) generato

Il SEI è una semplice interfaccia java che dichiara dei metodi che il client del web service può invocare.
Questa interfaccia deve rispettare le seguenti regole:

- Deve estender l interfaccia *java.rmi.Remote*
- Non deve avere dichiarazioni di costanti del tipo *public final static*
- I metodi devono lanciare l'eccezione *java.rmi.RemoteException* oppure una delle sue sottoclassi .(I metodi possono anche lanciare eccezioni specifiche all'applicazione)
- I parametri dei metodi e i tipi di ritorno devono essere quelli sopportati dai tipi di JAX-RPC.

Codice di un service endpoint interface

```
package it.usi.xframe.anw.services;

public interface DisponibilitaCarta_SEI extends java.rmi.Remote
{
 public it.usi.xframe.anw.externaloutput.ANWDisponibilitaCartaOutput
disponibilitaCarta(it.usi.xframe.anw.externalinput.ANWDisponibilitaCartaInput input);
}
```

4.1.8 Web service language description (WSDL) generato

Il Web Services Description Language (**WSDL**) è un linguaggio formale in formato XML utilizzato per la creazione di "documenti" per la descrizione di Web Service.

L'utilizzo di WSDL ha due vantaggi: per prima cosa, alcuni strumenti di sviluppo sono in grado di prendere in input un file WSDL e produrre del codice che implementa l'infrastruttura per realizzare il servizio, oppure per crearne un runtime per la semplice implementazione di un client; in secondo luogo, è un modo per disaccoppiare il servizio Web dal protocollo di trasporto e dai percorsi fisici, come gli endpoint.

Mediante WSDL può essere, infatti, descritta l'interfaccia pubblica di un Web Service ovvero creata una descrizione, basata su XML, di come interagire con un determinato servizio: un "documento" WSDL contiene infatti, relativamente al Web Service descritto, informazioni su:

- *cosa* può essere utilizzato (le "operazioni" messe a disposizione dal servizio).
- *come* utilizzarlo (il protocollo di comunicazione da utilizzare per accedere al servizio, il formato dei messaggi accettati in input e restituiti in output dal servizio ed i dati correlati) ovvero i "vincoli" (*bindings* in inglese) del servizio.
- *dove* utilizzare il servizio (cosiddetto *endpoint* del servizio che solitamente corrisponde all'indirizzo - in formato URI - che rende disponibile il Web Service).

Le operazioni supportate dal Web Service ed i messaggi che è possibile scambiare con lo stesso sono descritti in maniera astratta e quindi collegati ad uno specifico protocollo di rete e ad uno specifico formato.

Il WSDL è solitamente utilizzato in combinazione con SOAP e XML Schema per rendere disponibili Web Services su reti aziendali o su internet: un programma client può, infatti, "leggere" il documento WSDL relativo ad un Web Service per determinare quali siano le funzioni messe a disposizione sul server e quindi utilizzare il protocollo SOAP per utilizzare una o più delle funzioni elencate dal WSDL.

Un documento WSDL è un file XML contentente un insieme di definizioni. Nello standard WSDL è possibile trovare quattro diversi tipi di informazioni. Sicuramente l'aspetto principale sono l'insieme di regole che consentono di definire in modo astratto un servizio Web; oltre a ciò, sono presenti le specifiche per collegare il servizio a SOAP, HTTP e MIME.
La definizione astratta dei servizi Web di WSDL è composta da cinque diversi elementi:

- i tipi (types);
- i messaggi (messages);
- le operazioni (portType);
- i collegamenti (bindings);
- la definizione del servizio (service);

Esempio di WSDL

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions targetNamespace="http://services.anw.xframe.usi.it" xmlns:impl="http://services.anw.xframe.usi.it"
xmlns:intf="http://services.anw.xframe.usi.it" xmlns:tns2="http://externalinput.anw.xframe.usi.it" xmlns:tns3="http://beans.anw.xframe.usi.it"
xmlns:tns4="http://externaloutput.anw.xframe.usi.it" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <wsdl:types>

  <schema elementFormDefault="qualified" targetNamespace="http://externalinput.anw.xframe.usi.it"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:impl="http://services.anw.xframe.usi.it" xmlns:intf="http://services.anw.xframe.usi.it"
xmlns:tns3="http://beans.anw.xframe.usi.it" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
  <import namespace="http://beans.anw.xframe.usi.it"/>
  <complexType name="ANWDisponibilitaCartaInput">
   <sequence>
    <element name="anwGeneralHeader" nillable="true" type="tns3:AnwGeneralHeader"/>
    <element name="cardId" nillable="true" type="xsd:decimal"/>
   </sequence>
  </complexType>
 </schema>
  <schema elementFormDefault="qualified" targetNamespace="http://beans.anw.xframe.usi.it" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:impl="http://services.anw.xframe.usi.it" xmlns:intf="http://services.anw.xframe.usi.it" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <complexType name="AnwGeneralHeader">
   <sequence>
    <element name="operatorID" nillable="true" type="xsd:string"/>
    <element name="partnerCode" nillable="true" type="xsd:string"/>
    <element name="sequenceNumber" type="xsd:long"/>
   </sequence>
  </complexType>
  <complexType name="AnwGeneralResponse">
   <sequence>
    <element name="descrizioneErrore" nillable="true" type="xsd:string"/>
    <element name="sequenceNumber" type="xsd:long"/>
    <element name="operatorID" nillable="true" type="xsd:string"/>
    <element name="partnerCode" nillable="true" type="xsd:string"/>
    <element maxOccurs="unbounded" name="errorArray" nillable="true" type="tns3:AnwGeneralError"/>
    <element name="esitoServizio" nillable="true" type="xsd:string"/>
   </sequence>
  </complexType>
  <complexType name="AnwGeneralError">
   <sequence>
    <element name="codErr" nillable="true" type="xsd:string"/>
    <element name="desErr" nillable="true" type="xsd:string"/>
   </sequence>
  </complexType>
 </schema>
  <schema elementFormDefault="qualified" targetNamespace="http://services.anw.xframe.usi.it" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:impl="http://services.anw.xframe.usi.it" xmlns:intf="http://services.anw.xframe.usi.it" xmlns:tns2="http://externalinput.anw.xframe.usi.it"
xmlns:tns4="http://externaloutput.anw.xframe.usi.it" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://externaloutput.anw.xframe.usi.it"/>
  <import namespace="http://externalinput.anw.xframe.usi.it"/>
  <element name="disponibilitaCarta">
   <complexType>
    <sequence>
     <element name="input" nillable="true" type="tns2:ANWDisponibilitaCartaInput"/>
    </sequence>
   </complexType>
  </element>
```

```
  <element name="disponibilitaCartaResponse">
   <complexType>
    <sequence>
     <element name="disponibilitaCartaReturn" nillable="true" type="tns4:ANWDisponibilitaCartaOutput"/>
    </sequence>
   </complexType>
  </element>
 </schema>
  <schema elementFormDefault="qualified" targetNamespace="http://externaloutput.anw.xframe.usi.it"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:impl="http://services.anw.xframe.usi.it" xmlns:intf="http://services.anw.xframe.usi.it"
xmlns:tns3="http://beans.anw.xframe.usi.it" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://beans.anw.xframe.usi.it"/>
  <complexType name="ANWDisponibilitaCartaOutput">
   <sequence>
    <element name="totalAvailability" nillable="true" type="xsd:decimal"/>
    <element name="totalPlafond" nillable="true" type="xsd:decimal"/>
    <element name="anwGeneralResponse" nillable="true" type="tns3:AnwGeneralResponse"/>
   </sequence>
  </complexType>
 </schema>
</wsdl:types>

  <wsdl:message name="disponibilitaCartaResponse">
     <wsdl:part element="intf:disponibilitaCartaResponse" name="parameters"/>
  </wsdl:message>

  <wsdl:message name="disponibilitaCartaRequest">

     <wsdl:part element="intf:disponibilitaCarta" name="parameters"/>

  </wsdl:message>

  <wsdl:portType name="DisponibilitaCarta">

     <wsdl:operation name="disponibilitaCarta">

        <wsdl:input message="intf:disponibilitaCartaRequest" name="disponibilitaCartaRequest"/>

        <wsdl:output message="intf:disponibilitaCartaResponse" name="disponibilitaCartaResponse"/>

     </wsdl:operation>

  </wsdl:portType>

  <wsdl:binding name="DisponibilitaCartaSoapBinding" type="intf:DisponibilitaCarta">

     <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
```

```
    <wsdl:operation name="disponibilitaCarta">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="disponibilitaCartaRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="disponibilitaCartaResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="DisponibilitaCartaService">
    <wsdl:port binding="intf:DisponibilitaCartaSoapBinding" name="DisponibilitaCarta">
      <wsdlsoap:address location="http://localhost:9080/XA-ANW-WS/services/DisponibilitaCarta"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Tipi

I tipi definiti da WSDL sono l'equivalente delle strutture (struct) del linguaggio C: strutture dati anche complesse che sono utilizzate come elementi base per costruire i messaggi di input ed output definiti nella sezione "messaggi". Ad esempio, la porzione di file XML precedente definisce l'elemento : *AnwGeneralResponse*, composto tre stringe (*descrizione Errore, operatorID, partnerCode*) ed un array di oggetti di tipo *AnwGeneralError* ed un long (*sequenceNumber*) .

Messaggi

I messaggi sono gli elementi che costituiscono gli input e gli output dei servizi. I singoli messaggi

possono contenere i tipi di dati complessi definiti nella sezione types (tipi), oppure semplici dati primitivi.
Relativamente al WSDL precedente abbiamo due messaggi , uno relativo alla richiesta (*disponibilitaCartaRequest*) ed un altro relativo alla risposta (*disponibilitaCartaResponse*).

Operazioni

Questa sezione definisce le operazioni fornite dal servizio Web. Se si ponessero in relazione i servizi Web con la programmazione distribuita, ad esempio RMI, le operazioni WSDL equivarrebbero ai singoli metodi dell'oggetto remoto . Per ciascuna operazione, WSDL definisce i messaggi di input e di output.

L'elemento <operation> definisce una singola operazione e può contenere opzionalmente almeno uno dei due sottoelementi <input> ed <output>. La presenza e l'ordine di questi elementi determina la tipologia di servizio, che può rientrare all'interno di quattro diverse nature:

- one-way. Anche detto fire&forget, è una configurazione dove l'endpoint si limita a ricevere il messaggio inviato dal client. In questo caso è presente un solo elemento di input;
- request-response. In questo caso l'endpoint riceve un messaggio di richiesta, esegue l'elaborazione necessaria e restituisce al client un messaggio di risposta correlato alla richiesta ricevuta. Sono presenti gli elementi input ed output;
- solicit-response. E' l'opposto del precedente. In questo caso è l'endpoint che inizia la comunicazione inviando un messaggio al client che a sua volta dovrà rispondere di conseguenza. Nel file WSDL è presente prima l'elemento output e poi quello input;
- notification. E' l'opposto della tipologia one-way. In questo caso è l'endpoint ad inviare un messaggio al client, senza che questo debba inviare una risposta. E' presente il solo elemento output.

Il modello di interazione da utilizzare dipende dalla natura del servizio. Se l'interazione request-response potrebbe essere quella usata per la maggior parte dei servizi, in quanto costituisce il tipico modello di comunicazione RPC, altre modalità possono essere utili in altri ambiti. Ad esempio, una interazione one-way può essere utile in sistemi dove il client invia una serie di informazioni al servizio, e questo si limita a raccoglierle. In questo caso il client non è interessato al risultato dell'elaborazione dei suoi dati, come nell'esempio relativo alla raccolta dati ambientali descritto all'inzio del capitolo.

Il WSDL precedente presenta un'unica operazione chiamata *disponibilitaCarta* che corrisponde all'unico metodo che il client può invocare.

Collegamenti

In questa sezione avviene la mappatura del servizio astratto, definito nelle sezioni precedenti, al protocollo concreto di comunicazione (ad esempio SOAP). Il contenuto di questa sezione è fortemente dipendente dal protocollo utilizzato, e quindi delle estensioni WSDL relative.

Definizione del servizio

L'ultimo elemento di un file WSDL è la definizione del servizio: questa sezione consente di raccogliere tutte le operazioni sotto un unico nome . Il servizio è identificato da un nome (l'attributo name dell'elemento service) e può avere una descrizione, contenuta nel sotto-elemento opzionale documentation. All'interno dell'elemento service vengono elencate tutte le operazioni esposte dal servizio, sottoforma di elementi port. Per ciascuno di questi viene indicato il collegamento utilizzato. Il contenuto dell'elemento port cambia in funzione del tipo di collegamento utilizzato. Nel esempio precedente è presente un esempio di definizione di servizio, in particolare il servio DisponibilitaCartaService , dove il collegamento utilizzato è quello con SOAP. In questo caso nell'elemento port è presente un elemento address appartenente al namespace soap: (utilizzato in tutto il WSDL per riferirsi agli elementi strettamente relativi al collegamento di WSDL con SOAP). Questo elemento indica l'URL fisico dell'endpoint del servizio.

4.1.9 File di mapping JAX-RPC generato

Questo file aiuta il compilatore JAX-RPC a mappare gli oggetti java in oggetti WSDL. E' un file XML standard che descrive il legame tra il WSDL ed il service endpoint interface (SEI).

Esempio di file di mapping JAX-RPC

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE java-wsdl-mapping PUBLIC "-//IBM Corporation, Inc.//DTD J2EE JAX-RPC mapping 1.0//EN"
"http://www.ibm.com/webservices/dtd/j2ee_jaxrpc_mapping_1_0.dtd">

  <java-wsdl-mapping id="JavaWSDLMapping_1273501137373">
    <package-mapping id="PackageMapping_1273501137373">
      <package-type>it.usi.xframe.anw.externalinput</package-type>
```

```
    <namespaceURI>http://externalinput.anw.xframe.usi.it</namespaceURI>
  </package-mapping>
  <package-mapping id="PackageMapping_1273501137374">
    <package-type>it.usi.xframe.anw.externaloutput</package-type>
    <namespaceURI>http://externaloutput.anw.xframe.usi.it</namespaceURI>
  </package-mapping>
  <package-mapping id="PackageMapping_1273501137375">
    <package-type>it.usi.xframe.anw.beans</package-type>
    <namespaceURI>http://beans.anw.xframe.usi.it</namespaceURI>
  </package-mapping>
  <package-mapping id="PackageMapping_1273501137376">
    <package-type>it.usi.xframe.anw.services</package-type>
    <namespaceURI>http://services.anw.xframe.usi.it</namespaceURI>
  </package-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137373">
    <class-type>it.usi.xframe.anw.externaloutput.ANWDisponibilitaCartaOutput</class-type>
    <root-type-qname id="RootTypeQname_1273501137373">
      <namespaceURI>http://externaloutput.anw.xframe.usi.it</namespaceURI>
      <localpart>ANWDisponibilitaCartaOutput</localpart>
    </root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping id="VariableMapping_1273501137373">
      <java-variable-name>totalAvailability</java-variable-name>
      <xml-element-name>totalAvailability</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137374">
      <java-variable-name>totalPlafond</java-variable-name>
      <xml-element-name>totalPlafond</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137375">
      <java-variable-name>anwGeneralResponse</java-variable-name>
      <xml-element-name>anwGeneralResponse</xml-element-name>
    </variable-mapping>
  </java-xml-type-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137374">
    <class-type>it.usi.xframe.anw.beans.AnwGeneralError[]</class-type>
    <root-type-qname id="RootTypeQname_1273501137374">
      <namespaceURI>http://beans.anw.xframe.usi.it</namespaceURI>
      <localpart>AnwGeneralError[unbounded]</localpart>
    </root-type-qname>
    <qname-scope>complexType</qname-scope>
  </java-xml-type-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137375">
    <class-type>it.usi.xframe.anw.beans.AnwGeneralHeader</class-type>
    <root-type-qname id="RootTypeQname_1273501137375">
      <namespaceURI>http://beans.anw.xframe.usi.it</namespaceURI>
      <localpart>AnwGeneralHeader</localpart>
    </root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping id="VariableMapping_1273501137376">
      <java-variable-name>operatorID</java-variable-name>
```

```
      <xml-element-name>operatorID</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137377">
      <java-variable-name>partnerCode</java-variable-name>
      <xml-element-name>partnerCode</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137378">
      <java-variable-name>sequenceNumber</java-variable-name>
      <xml-element-name>sequenceNumber</xml-element-name>
    </variable-mapping>
  </java-xml-type-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137376">
    <class-type>it.usi.xframe.anw.externalinput.ANWDisponibilitaCartaInput</class-type>
    <root-type-qname id="RootTypeQname_1273501137376">
      <namespaceURI>http://externalinput.anw.xframe.usi.it</namespaceURI>
      <localpart>ANWDisponibilitaCartaInput</localpart>
    </root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping id="VariableMapping_1273501137379">
      <java-variable-name>anwGeneralHeader</java-variable-name>
      <xml-element-name>anwGeneralHeader</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137380">
      <java-variable-name>cardId</java-variable-name>
      <xml-element-name>cardId</xml-element-name>
    </variable-mapping>
  </java-xml-type-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137377">
    <class-type>it.usi.xframe.anw.beans.AnwGeneralResponse</class-type>
    <root-type-qname id="RootTypeQname_1273501137377">
      <namespaceURI>http://beans.anw.xframe.usi.it</namespaceURI>
      <localpart>AnwGeneralResponse</localpart>
    </root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping id="VariableMapping_1273501137381">
      <java-variable-name>descrizioneErrore</java-variable-name>
      <xml-element-name>descrizioneErrore</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137382">
      <java-variable-name>sequenceNumber</java-variable-name>
      <xml-element-name>sequenceNumber</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137383">
      <java-variable-name>operatorID</java-variable-name>
      <xml-element-name>operatorID</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137384">
      <java-variable-name>partnerCode</java-variable-name>
      <xml-element-name>partnerCode</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137385">
```

```
      <java-variable-name>errorArray</java-variable-name>
      <xml-element-name>errorArray</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137386">
      <java-variable-name>esitoServizio</java-variable-name>
      <xml-element-name>esitoServizio</xml-element-name>
    </variable-mapping>
  </java-xml-type-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137378">
    <class-type>java.math.BigDecimal</class-type>
    <root-type-qname id="RootTypeQname_1273501137378">
      <namespaceURI>http://www.w3.org/2001/XMLSchema</namespaceURI>
      <localpart>decimal</localpart>
    </root-type-qname>
    <qname-scope>simpleType</qname-scope>
  </java-xml-type-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137379">
    <class-type>java.lang.String</class-type>
    <root-type-qname id="RootTypeQname_1273501137379">
      <namespaceURI>http://www.w3.org/2001/XMLSchema</namespaceURI>
      <localpart>string</localpart>
    </root-type-qname>
    <qname-scope>simpleType</qname-scope>
  </java-xml-type-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137380">
    <class-type>it.usi.xframe.anw.beans.AnwGeneralError</class-type>
    <root-type-qname id="RootTypeQname_1273501137380">
      <namespaceURI>http://beans.anw.xframe.usi.it</namespaceURI>
      <localpart>AnwGeneralError</localpart>
    </root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping id="VariableMapping_1273501137387">
      <java-variable-name>codErr</java-variable-name>
      <xml-element-name>codErr</xml-element-name>
    </variable-mapping>
    <variable-mapping id="VariableMapping_1273501137388">
      <java-variable-name>desErr</java-variable-name>
      <xml-element-name>desErr</xml-element-name>
    </variable-mapping>
  </java-xml-type-mapping>
  <java-xml-type-mapping id="JavaXMLTypeMapping_1273501137381">
    <class-type>long</class-type>
    <root-type-qname id="RootTypeQname_1273501137381">
      <namespaceURI>http://www.w3.org/2001/XMLSchema</namespaceURI>
      <localpart>long</localpart>
    </root-type-qname>
    <qname-scope>simpleType</qname-scope>
  </java-xml-type-mapping>
  <service-interface-mapping id="ServiceInterfaceMapping_1273501137373">
    <service-interface>it.usi.xframe.anw.services.DisponibilitaCartaService</service-interface>
    <wsdl-service-name id="WSDLServiceName_1273501137373">
```

```
      <namespaceURI>http://services.anw.xframe.usi.it</namespaceURI>
      <localpart>DisponibilitaCartaService</localpart>
    </wsdl-service-name>
    <port-mapping id="PortMapping_1273501137373">
      <port-name>DisponibilitaCarta</port-name>
      <java-port-name>DisponibilitaCarta</java-port-name>
    </port-mapping>
  </service-interface-mapping>
  <service-endpoint-interface-mapping id="ServiceEndpointInterfaceMapping_1273501137373">
    <service-endpoint-interface>it.usi.xframe.anw.services.DisponibilitaCarta</service-endpoint-interface>
    <wsdl-port-type id="WSDLPortType_1273501137373">
      <namespaceURI>http://services.anw.xframe.usi.it</namespaceURI>
      <localpart>DisponibilitaCarta</localpart>
    </wsdl-port-type>
    <wsdl-binding id="WSDLBinding_1273501137373">
      <namespaceURI>http://services.anw.xframe.usi.it</namespaceURI>
      <localpart>DisponibilitaCartaSoapBinding</localpart>
    </wsdl-binding>
    <service-endpoint-method-mapping id="ServiceEndpointMethodMapping_1273501137373">
      <java-method-name>disponibilitaCarta</java-method-name>
      <wsdl-operation>disponibilitaCarta</wsdl-operation>
      <wrapped-element></wrapped-element>
      <method-param-parts-mapping id="MethodParamPartsMapping_1273501137373">
        <param-position>0</param-position>
        <param-type>it.usi.xframe.anw.externalinput.ANWDisponibilitaCartaInput</param-type>
        <wsdl-message-mapping id="WSDLMessageMapping_1273501137373">
          <wsdl-message id="WSDLMessage_1273501137373">
            <namespaceURI>http://services.anw.xframe.usi.it</namespaceURI>
            <localpart>disponibilitaCartaRequest</localpart>
          </wsdl-message>
          <wsdl-message-part-name>input</wsdl-message-part-name>
          <parameter-mode>IN</parameter-mode>
        </wsdl-message-mapping>
      </method-param-parts-mapping>
      <wsdl-return-value-mapping id="WSDLReturnValueMapping_1273501137373">
        <method-return-value>it.usi.xframe.anw.externaloutput.ANWDisponibilitaCartaOutput</method-return-value>
        <wsdl-message id="WSDLMessage_1273501137374">
          <namespaceURI>http://services.anw.xframe.usi.it</namespaceURI>
          <localpart>disponibilitaCartaResponse</localpart>
        </wsdl-message>
        <wsdl-message-part-name>disponibilitaCartaReturn</wsdl-message-part-name>
      </wsdl-return-value-mapping>
    </service-endpoint-method-mapping>
  </service-endpoint-interface-mapping>
</java-wsdl-mapping>
```

4.1.10 Dichiarazione e definizione della servlet nel web.xml

Il web.xml è il deployement descriptor dell'applicazione web
In questo file viene dichiarato e definito il servizio web dato che è esposto come servlet che sarà caricata all'avvio dell'applicazione.

Configurazione della servlet esposta nel web.xml

```
<servlet>
        <servlet-name>it_usi_xframe_anw_services_DisponibilitaCarta</servlet-name>
        <servlet-class>it.usi.xframe.anw.services.DisponibilitaCarta</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
```

4.1.11 Deployement descriptor del servizio web (Webservices.xml)

Questo è il file di configurazione del servizio web, devono essere specificati per il singolo servizio il WSDL:

- il file mapping JAX-RPC.
- un port-component che mappi un elemento port nel relativo WSDL (definisce il servizio web, le operazioni che possono essere eseguite e i messaggi scambiati).
- il service endpoint interface
- il link alla servlet che implementa il servizio
- può anche essere definito un handler (che vedremo più avanti).

Configurazione del servizio nel webservice.xml

```
<webservice-description id="WebServiceDescription_1270567896574">
    <webservice-description-name>DisponibilitaCartaService</webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/DisponibilitaCarta.wsdl</wsdl-file>
    <jaxrpc-mapping-file>WEB-INF/DisponibilitaCarta_mapping.xml</jaxrpc-mapping-file>
    <port-component id="PortComponent_1270567896574">
      <port-component-name>DisponibilitaCarta</port-component-name>
      <wsdl-port id="WSDLPort_1270567896574">
        <namespaceURI>http://services.anw.xframe.usi.it</namespaceURI>
        <localpart>DisponibilitaCarta</localpart>
      </wsdl-port>
      <service-endpoint-interface>it.usi.xframe.anw.services.DisponibilitaCarta</service-endpoint-interface>
      <service-impl-bean id="ServiceImplBean_1270567896574">
        <servlet-link>it_usi_xframe_anw_services_DisponibilitaCarta</servlet-link>
      </service-impl-bean>
      <handler id="Handler_1270568840471">
        <handler-name>DisponibilitaCarta.SoapRequestResponseHandler</handler-name>
        <handler-class>it.usi.xframe.handler.SoapRequestResponseHandler</handler-class>
      </handler>
    </port-component>
```

```
</webservice-description>
```

4.1.12 Ulteriori classi generate dal server

Durante il deploy del sevizio web vengono generate ulteriori classi necessari . I messaggi SOAP sono trasmessi usando il protocollo HTTP. Per agire da interprete il runtime JAX-RPC crea dei ties a partire dal service endpoint interface e dall'implementazione. Il tie fa da proxy per l'implementazione della classe e chiama l 'helper corretto per deserializzare i tipi SOAP in tipi JAVA o viceversa.

In pratica c'è una servlet in ascolto che si occupa di fare le conversioni necessarie usando gli ulteriori file generati durante il deploy .

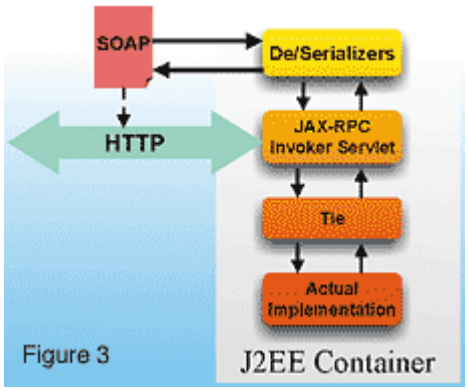

Figura 4.3 : Runtime JAX-RPC

Codice della classe helper generata

```
public class ANWDisponibilitaCartaInput_Helper {
   // Type metadata
   private static com.ibm.ws.webservices.engine.description.TypeDesc typeDesc =
      new com.ibm.ws.webservices.engine.description.TypeDesc(ANWDisponibilitaCartaInput.class);

   static {
      com.ibm.ws.webservices.engine.description.FieldDesc field = new com.ibm.ws.webservices.engine.description.ElementDesc();
      field.setFieldName("anwGeneralHeader");
      field.setXmlName(com.ibm.ws.webservices.engine.utils.QNameTable.createQName("http://externalinput.anw.xframe.usi.it",
"anwGeneralHeader"));
      field.setXmlType(com.ibm.ws.webservices.engine.utils.QNameTable.createQName("http://beans.anw.xframe.usi.it", "AnwGeneralHeader"));
      typeDesc.addFieldDesc(field);
      field = new com.ibm.ws.webservices.engine.description.ElementDesc();
      field.setFieldName("cardId");
      field.setXmlName(com.ibm.ws.webservices.engine.utils.QNameTable.createQName("http://externalinput.anw.xframe.usi.it", "cardId"));
      field.setXmlType(com.ibm.ws.webservices.engine.utils.QNameTable.createQName("http://www.w3.org/2001/XMLSchema", "decimal"));
      typeDesc.addFieldDesc(field);
   };
```

```
    /**
     * Return type metadata object
     */
    public static com.ibm.ws.webservices.engine.description.TypeDesc getTypeDesc() {
        return typeDesc;
    }

    /**
     * Get Custom Serializer
     */
    public static com.ibm.ws.webservices.engine.encoding.Serializer getSerializer(
           java.lang.String mechType,
           java.lang.Class javaType,
           javax.xml.namespace.QName xmlType) {
        return
          new ANWDisponibilitaCartaInput_Ser(
            javaType, xmlType, typeDesc);
    };

    /**
     * Get Custom Deserializer
     */
    public static com.ibm.ws.webservices.engine.encoding.Deserializer getDeserializer(
           java.lang.String mechType,
           java.lang.Class javaType,
           javax.xml.namespace.QName xmlType) {
        return
          new ANWDisponibilitaCartaInput_Deser(
            javaType, xmlType, typeDesc);
    };

}
```

Codice della classe di de serializzazione generata

```
package it.usi.xframe.anw.externalinput;

public class ANWDisponibilitaCartaInput_Ser extends com.ibm.ws.webservices.engine.encoding.ser.BeanSerializer {
    public ANWDisponibilitaCartaInput_Ser(
           java.lang.Class _javaType,
           javax.xml.namespace.QName _xmlType,
           com.ibm.ws.webservices.engine.description.TypeDesc _typeDesc) {
        super(_javaType, _xmlType, _typeDesc);
    }
    public void serialize(
        javax.xml.namespace.QName name,
        org.xml.sax.Attributes attributes,
        java.lang.Object value,
        com.ibm.ws.webservices.engine.encoding.SerializationContext context)
        throws java.io.IOException
```

```
{
   context.startElement(name, addAttributes(attributes,value,context));
   addElements(value,context);
   context.endElement();
}
protected org.xml.sax.Attributes addAttributes(
   org.xml.sax.Attributes attributes,
   java.lang.Object value,
   com.ibm.ws.webservices.engine.encoding.SerializationContext context)
   throws java.io.IOException
{
   return attributes;
}
protected void addElements(
   java.lang.Object value,
   com.ibm.ws.webservices.engine.encoding.SerializationContext context)
   throws java.io.IOException
{
   ANWDisponibilitaCartaInput bean = (ANWDisponibilitaCartaInput) value;
   java.lang.Object propValue;
   javax.xml.namespace.QName propQName;
   {
    propQName = QName_0_0;
    propValue = bean.getAnwGeneralHeader();
    context.serialize(propQName, null,
       propValue,
       QName_1_4,
       true,null);
    propQName = QName_0_65;
    propValue = bean.getCardId();
    context.serialize(propQName, null,
       propValue,
       QName_2_5,
       true,null);
   }
}
   public final static javax.xml.namespace.QName QName_2_5 =
        com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
            "http://www.w3.org/2001/XMLSchema",
            "decimal");
   public final static javax.xml.namespace.QName QName_0_0 =
        com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
            "http://externalinput.anw.xframe.usi.it",
            "anwGeneralHeader");
   public final static javax.xml.namespace.QName QName_1_4 =
        com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
            "http://beans.anw.xframe.usi.it",
            "AnwGeneralHeader");
   public final static javax.xml.namespace.QName QName_0_65 =
        com.ibm.ws.webservices.engine.utils.QNameTable.createQName(
            "http://externalinput.anw.xframe.usi.it",
```

```
                "cardId");
}
```

Codice della classe di deserializzazione generata

```
package it.usi.xframe.anw.externalinput;

public class ANWDisponibilitaCartaInput_Deser extends com.ibm.ws.webservices.engine.encoding.ser.BeanDeserializer {
   public ANWDisponibilitaCartaInput_Deser(
         java.lang.Class _javaType,
         javax.xml.namespace.QName _xmlType,
         com.ibm.ws.webservices.engine.description.TypeDesc _typeDesc) {
      super(_javaType, _xmlType, _typeDesc);
   }
}
```

4.2 Webservice Handler

Un Handler per i Web Services è un componente software in grado di eseguire operazioni a ogni richiesta, risposta o fault. Anche se non esiste un vero limite alle operazioni svolte dagli Handler, essi in genere, visto il punto in cui intervengono, eseguono i seguenti tipi di attività:

- tracciamento di richieste e risposte
- modifica della richiesta, della risposta o del fault
- modifica dei dati contenuti nel contesto dei Web Services
- compensazione dei fault

E' stato deciso di usare un web service handler per poter intercettare i messaggi scambiati tra client e server e loggarli per motivi statistici e per poter usare questi messaggi nella risoluzione di segnalazioni .

4.2.1 Contesto di Applicazione

Un handler può essere visto come una servlet Filter (Praticamente i filtri fanno da "man in the middle" nelle applicazioni web e riescono a collezionare e/o manipolare i dati in uscita e in entrata) dato che è una unità di logica di business che può modificare una richiesta prima questa venga processata dal servizio web oppure modificare la risposta dopo aver processato la richiesta.

Gli handler JAX-RPC sono specifici alle richieste SOAP e sono indipendenti dal protocollo di trasporto per cui un handler definito per SOAP/HTTP dovrebbe essere utilizzabile per SOAP/JMS.

Più handler possono essere combinati per creare una catena di handler (handler chain). Ogni handler processa il messaggio SOAP poi lo passa all'handler successivo. La sequenza è definita nel file di deploy del servizio web Webservice.xml lato server oppure nel webservicesclient.xml , web.xml , ejb-jar.xml file (nel tag the service-ref) lato client.

Le attività indicate per gli Handler sono riconducibili a due scenari molto importanti: l'integrazione e la compensazione di eventuali errori.
In effetti l'uso massiccio dei Web Services è strettamente legato all'evoluzione delle applicazioni verso modelli architetturali di tipo SOA.In questi modelli l'applicazione o il sistema informativo è composto da un insieme di servizi completamente separati tra loro (e a volte gestiti da organizzazioni diverse) e quindi può essere necessario implementare attività di monitoring per riuscire a seguire i flussi delle singole richieste. In questo caso l'uso di un Handler è una opzione importante poiche' consente di tracciare le richieste e le risposte dei singoli servizi e quindi verificare, in caso di errore, quale di questi non ha funzionato correttamente.
Anche la possibilità di modificare la richiesta, la risposta o il fault si sposa molto bene con gli scenari di integrazione. Questa funzionalità può essere usata, per esempio, per arricchire il messaggio di tutte le informazioni opzionali con dei dati di default.
Inoltre è possibile ricavare delle informazioni infrastrutturali che non si vogliono esporre direttamente nella firma del servizio, ma che possono essere dedotte in qualche modo e passate al servizio, per esempio, tramite il MessageContext.
Infine possiamo invocare altri servizi o compensare eventuali condizioni di errore.
Questa possibilità consente di creare dei sistemi basati sul paradigma ad eventi in modo tale da iniziare determinate attività solo in corrispondenza di richieste specifiche.

Questo tipo di responsabilità vengono facilmente collocate all'interno di un Handler perché esso può lavorare in modo trasversale a tutti i servizi e lo può fare sia durante la richiesta che durante la risposta.

4.2.2 Implementazione del web service handler

Per sviluppare un handler JAX-RPC, bisogna creare una classe che implementa l'interfaccia *javax.xml.rpc.handler.Handler* . Questa interfaccia ha tre metodi per getire le richieste, risposte e fault.

Classe handler generico

```
package javax.xml.rpc.handler;

import javax.xml.namespace.QName;

public interface Handler {

   public boolean handleRequest(MessageContext context); // ‘false’ blocca la catena di handler

   public boolean handleResponse(MessageContext context); // false’ blocca la catena di handler

   public boolean handleFault(MessageContext context); // 'false' blocca la catena di handler

   public abstract void init(HandlerInfo config);

   public abstract void destroy();

   public QName[] getHeaders();

}
```

Configurazione dell’handler nel web service.xml

Gli handler nel nostro caso sono stati configurati nel file deploy dei servizi web (webservices.xml)

```
<webservice-description id="WebServiceDescription_1270567896574">
        <webservice-description-name>DisponibilitaCartaService</webservice-description-name>
        <wsdl-file>WEB-INF/wsdl/DisponibilitaCarta.wsdl</wsdl-file>
        <jaxrpc-mapping-file>WEB-INF/DisponibilitaCarta_mapping.xml</jaxrpc-mapping-file>
        <port-component id="PortComponent_1270567896574">
          <port-component-name>DisponibilitaCarta</port-component-name>
          <wsdl-port id="WSDLPort_1270567896574">
            <namespaceURI>http://services.anw.xframe.usi.it</namespaceURI>
            <localpart>DisponibilitaCarta</localpart>
          </wsdl-port>
          <service-endpoint-interface>it.usi.xframe.anw.services.DisponibilitaCarta</service-endpoint-interface>
          <service-impl-bean id="ServiceImplBean_1270567896574">
            <servlet-link>it_usi_xframe_anw_services_DisponibilitaCarta</servlet-link>
          </service-impl-bean>
          <handler id="Handler_1270568840471">
            <handler-name>DisponibilitaCarta.SoapRequestResponseHandler</handler-name>
            <handler-class>it.usi.xframe.handler.SoapRequestResponseHandler</handler-class>
          </handler>
        </port-component>
      </webservice-description>
```

4.3 Entreprise java beans (EJB)

L'architettura XFRAME è stata spiegata nel capitolo 3 . Abbiamo visto che la logica di business è sviluppata nel modulo XXX-BF dell'applicazione XFRAME. Questa logica di business è implementata accendendo ad esempio al database, richiamando servizi host via IFG oppure facendo chiamate ad funzionalità già sviluppate da altre applicazioni fruibili via EJB (Entreprise java bean).

I servizi web sviluppati per eseguire le richieste dei vari client devono quindi accedere alla logica di business che è sviluppata usando come tecnologia: gli entreprise java bean (EJB) .
La nostra applicazione si deve interfacciare con altre applicazioni del gruppo come detto in precedenza che mettono a disposizione delle interfacce che possono essere usate per richiamare diverse funzionalità .

Abbiamo inoltre un fornitore esterno che ci mette a disposizione delle funzionalità sviluppate ad hoc sempre fruibili attraverso chiamate ad EJB.

Gli EJB offrono vari vantaggi allo sviluppatore di una applicazione

- Un ambiente di esecuzione che gestisce :
    - naming di oggetti, sicurezza, concorrenza, transazioni,
    - persistenza, distribuzione oggetti (location transparency),messaggi asincroni
    - ciclo di vita oggetti (modello a pool), caching, sharing.
- Il modello a componenti che permette di comprare/riusare parti e rende lo sviluppo più veloce.
- La chiara separazione del lavoro di sviluppo, deployment ed amministrazione di una applicazione EJB.

4.3.1 Definizione

Un EJB (o bean) è un componente lato server con una business logic che è conforme a specifiche EJB ed è eseguito in un container EJB.
Componente: classe Java con un ben definito ruolo nel contesto dell'architettura EJB e conforme ad un set di interfacce. E' una unità che può essere installata da sola.

Container: offre funzionalità legate al ciclo di vita dei componenti e alla gestione di transazioni e sicurezza. Il modello a container scherma i componenti dalla conoscenza dell'architettura sottostante.
Non vi è una connessione diretta tra il chiamante (client) e l'EJB

- Il server genera il codice che viene inserito tra client ed EJB
- Questo codice permette di intervenire con i servizi richiesti

Le specifiche EJB definiscono come creare una implementazione di un framework per servizi Java lato server (ovviamente, gli sviluppatori di un'applicazione non dovranno preoccuparsi dell'implementazione del framework).
L'architettura EJB consiste di:

- server EJB
- container EJB
- home interface
- remote interface

4.3.2 Architettura EJB

Server

Il server EJB è il processo che gestisce i container EJB e che fornisce accesso ai servizi di sistema. Il server EJB può anche fornire interfacce per l'accesso ad un database, a servizi CORBA, etc.

Container

Il container EJB è un'astrazione che gestisce una o più classi e/o istanze EJB. Attualmente, il container è sempre fornito dal server EJB, poiché l'interfaccia tra server e container non è ancora stata standardizzata.

Home interface

La home interface definisce i metodi per gestire il ciclo di vita di un oggetto EJB. Questi sono invocati dal container per trovare, creare e rimuovere istanze. Lo sviluppatore deve definire la home interface estendendo l'interfaccia *javax.ejb.EJBHome*.
Al momento del deployment, a partire dall'home interface il server crea un oggetto detto HomeObject.
Permette di

- Creare una istanza di un bean
- Trovare un bean
- Rimuovere una istanza di un bean

Restituisce una remote interface

Remote interface

La remote interface definisce i metodi business di un EJB (quelli che il client usa). Lo sviluppatore crea la remote interface estendendo *javax.ejb.EJBObject*.

Al momento del deployment, un oggetto che implementa la remote interface, detto EJBObject, è creato dal server.

In un'applicazione con EJB, un client contatta un server per richiedere di eseguire delle elaborazioni.

1. Il client trova il bean tramite Java Naming and DirectoryInterface (JNDI).
2. Il client usa l'EJBHome per creare o distruggere istanze di un EJB. Il server crea l'oggetto (EJB instance) e ritorna un proxy object (EJBObject) che ha la stessa interfaccia dell'EJB.
3. Il client usa l'EJBObject per invocare i metodi di una istanza.

Il client conosce solo riferimenti ad istanze di EJBObject e quando invoca un metodo, l'EJBObject delega la richiesta ad un'istanza EJB.

Ruoli

Lo sviluppatore EJB scrive le classi usando le specifiche EJB. Le classi conterranno sia metodi per creare e rimuovere una istanza EJB, sia metodi (business) che implementano le funzionalità della classe.

L'installatore (deployer) EJB ha il compito di installare una classe EJB sul server EJB e di configurare le proprietà dell'ambiente del server come richiesto dalla classe EJB. L'installatore conosce le caratteristiche dell'ambiente server, come il tipo di database e la sua posizione.

Il fornitore del container EJB fornisce il software per installare una classe EJB sul server.

Abbiamo svolto sia il ruolo di sviluppatore che di installatore.

Tipi di EJB

Ci sono tre tipi di EJB:

- Un session bean è un EJB che implementa una business logic per i client. Ad esempio fornire l'elenco delle carte di credito, lista movimenti.
    - Ha vita pari a quella della sessione con il client che lo usa. E'creato quando il client richiede un servizio e "distrutto" quando ilclient si disconnette (il container gestisce un pool di bean che fornisce ai client rapidamente quando ne fanno richiesta).

  - Uno stateful session bean è associato con un singolo client e mantiene i dati conversazionali relativi a quel client.
  - Uno stateless session bean non mantiene nessun dato.L'invocazione di un metodo da parte di un client può essere delegata dal container a qualunque session bean del pool.
- Un entity bean rappresenta una informazione immagazzinata persistentemente in un database. Fornisce a vari utenti l'accesso condiviso ai dati del database.
  - Un entity bean ha un identificatore unico (primary key) che permette di ricercare il bean (e quindi il dato del database). Ha una vita lunga (come quella del database) e sopravvive a crash del server (viene aggiornato all'ultimo stato committed).
- Un message-driven bean esegue al ricevimento di un messaggio del client. E' come un session bean ma invocato in modo asincrono. Ha vita breve e non rappresenta direttamente dati del database.

Nel nostro caso abbiamo sviluppato un session bean con vari metodi che implementano la logica di business.

Per sviluppare un session bean è necessario :

1. Definire la home e la localhome che sono delle interfacce che estendono *javax.ejb.EJBHome* e *javax.ejb.EJBLocalHome* rispettivamente
2. Definire l'interfaccia remota e locale che estendono *javax.ejb.EJBObject* ed *javax.ejb.EJBLocalObject.*
3. Implementare la classe EJB con i metodi busines (Questa classe deve che deve implementare javax.ejb.SessionBean)
4. creare il deployment descriptor: un set di file XML che contiene informazioni su come assemblare parti dell'applicazione (è usato in fase di deployment).

4.3.3 Sviluppo di entreprise java beans

L'ambiente di sviluppo WSAD ha dei plugin che permettono di creare l'ossatura di un session bean che permette quindi allo sviluppatore di concentrarsi sulla parte di sviluppo dei metodi che implementano la logica di business.

Il plugin ci permette quindi di creare la home (XXXGatewayHome.java), la localhome (XXXGatewayLocalHome.java), l'interfaccia remota (XXXGateway.java) e locale (

XXXGatewayLocal.java), che estendono entrambe un'interfaccia che conterrà poi la firma di tutti i metodi che implementano la logica di business (IXXXGatewaySerficeFacade.java). Questi file vengono generati nella parte di BFCL.

Nella parte BF dell'applicazione vengono generate le classi del session bean che conterranno poi l'implementazione dei metodi esposti nell'interfaccia IXXXGatewaySerficeFacade.java generata nella parte di BFCL.

- XXXGatewaySerficeFacade.java classe che conterrà l'implementazione dei metodi esposti nell'interfaccia.
- XXXGatewayBean che implementa *javax.ejb.SessionBean* ed estende XXXGatewaySerficeFacade.java.

XXXGatewayHome

```
package it.usi.xframe.anw.bfintf;
public interface XXXGatewayHome extends javax.ejb.EJBHome {
	/**
	 * Creates a default instance of Session Bean: AnwGateway
	 */
	public it.usi.xframe.anw.bfintf.XXXGateway create()
		throws javax.ejb.CreateException, java.rmi.RemoteException;
}
```

XXXGatewayLocalHome

```
package it.usi.xframe.anw.bfintf;
public interface XXXGatewayLocalHome  extends javax.ejb.EJBLocalHome {
	/**
	 * Creates a default instance of Session Bean: AnwGateway
	 */
	public it.usi.xframe.anw.bfintf. XXXGatewayLocal create()
		throws javax.ejb.CreateException;
}
```

XXXGateway

```
package it.usi.xframe.anw.bfintf;
public interface XXXGateway
	extends it.usi.xframe.anw.bfintf.IXXXGatewayServiceFacade, javax.ejb.EJBObject {
}
```

XXXGatewayLocal

```
package it.usi.xframe.anw.bfintf;
public interface XXXGatewayLocal
	extends
```

```
                it.usi.xframe.anw.bfintf.IXXXGatewayFacade,
                javax.ejb.EJBLocalObject {
}
```

IXXXGatewayServiceFacade

```
package it.usi.xframe.anw.bfintf;
import java.rmi.RemoteException;
import it.usi.xframe.anw.internalinput.ANWDisponibilitaCartaInputWS;
...
import it.usi.xframe.anw.internaloutput.ANWDisponibilitaCartaOutputWS;
public interface IXXXGatewayServiceFacade extends IServiceFacade {
        ...
        public ANWDisponibilitaCartaOutputWS disponibilitaCarta (ANWDisponibilitaCartaInputWS input) throws RemoteException;
                ...
}
```

XXXGatewayServiceFacade

```
package it.usi.xframe.anw.bfimpl;

import java.rmi.RemoteException;
import it.usi.xframe.anw.an2bean.DisponibilitaCartaService;
...
import it.usi.xframe.anw.internalinput.ANWDisponibilitaCartaInputWS;
...
import it.usi.xframe.anw.internaloutput.ANWDisponibilitaCartaOutputWS;
import it.usi.xframe.system.eservice.AbstractSupportServiceFacade;

public class XXXGatewayServiceFacade
        extends AbstractSupportServiceFacade
        implements IXXXGatewayServiceFacade {


        ...

        public ANWDisponibilitaCartaOutputWS disponibilitaCarta (ANWDisponibilitaCartaInputWS input) throws RemoteException {
                return DisponibilitaCartaService.getServiceInstance().disponibilitaCarta(input);
        }
        ...
}
```

XXXGatewayBean

```
package it.usi.xframe.anw.bfimpl;
/**
 * Bean implementation class for Enterprise Bean: XXXGateway
```

```
 */
public class XXXGatewayBean
	extends it.usi.xframe.anw.bfimpl.XXXGatewayServiceFacade
	implements javax.ejb.SessionBean {
	private javax.ejb.SessionContext mySessionCtx;

	public javax.ejb.SessionContext getSessionContext() {
		return mySessionCtx;
	}

	public void setSessionContext(javax.ejb.SessionContext ctx) {
		mySessionCtx = ctx;
	}
	public void ejbCreate() throws javax.ejb.CreateException {
	}
	public void ejbActivate() {
	}
	public void ejbPassivate() {
	}
	public void ejbRemove() {
	}
}
```

Implementazione concreta del metodo

```
package it.usi.xframe.anw.xx2bean;

import java.rmi.RemoteException;

import it.usi.xframe.xx2.beans.cards.multichannel.InquiryDisponibilitaOutput;
import it.usi.xframe.xx2.beans.cross.BaseBean;
import it.usi.xframe.xx2.beans.exception.AN2SevereException;
import it.usi.xframe.xx2.exceptions.ManagedException;
import it.usi.xframe.xx2.xpe.bfcl.util.QXBetwixUtil;
import it.usi.xframe.xxw.bfutil.ANWConstants;
import it.usi.xframe.xxw.bfutil.TransformUserChainUserInfo;
import it.usi.xframe.xxw.internalinput.ANWDisponibilitaCartaInputWS;
import it.usi.xframe.xxw.internaloutput.ANWDisponibilitaCartaOutputWS;

import org.apache.log4j.Logger;


public class DisponibilitaCartaService extends XX2CardBeanForANW{
	private static final Logger m_Log = Logger.getLogger(DisponibilitaCartaService.class);
	private static DisponibilitaCartaService instance = null;

	public static DisponibilitaCartaService getServiceInstance() {
		if (instance == null)
			instance = new DisponibilitaCartaService();
		return instance;
	}
```

```
	public ANWDisponibilitaCartaOutputWS disponibilitaCarta(ANWDisponibilitaCartaInputWS input)throws RemoteException{

		ANWDisponibilitaCartaOutputWS output = new ANWDisponibilitaCartaOutputWS();
		InquiryDisponibilitaOutput partnerOutput = eseguiDisponibilitaCarta(input);
		output = getAnwOutput(partnerOutput);
		m_Log.info("ANWoutput-->" + QXBetwixUtil.Bean2XML(output));
		return output;
	}

	public InquiryDisponibilitaOutput eseguiDisponibilitaCarta(ANWDisponibilitaCartaInputWS input)throws RemoteException{
		m_Log.info("Executing DisponibilitaCartaService.eseguiDisponibilitaCarta...");
		XX2CardBeanForANW xx2Card = XX2CardBeanForANW.getInstance();
		try {
			//Recupera l interfaccia remota
			xx2Card.createFacade();
			BaseBean partnerInput = getpartnerInput(input);
			//Chiama il metodo remoto
			InquiryDisponibilitaOutput partnerOutput = xx2Card.facade.inquiryDisponibilita(partnerInput);
			m_Log.info("partnerOutput-->" + QXBetwixUtil.Bean2XML(partnerOutput));
			return partnerOutput;
		}catch (Exception e){
			m_Log.error(e.getMessage(), e);
			throw new RemoteException(ANWConstants.ANW_BACKEND_UNAVAILABLE);
		} finally {
			xx2Card.disposeFacade();
		}
	}

	private BaseBean getpartnerInput(ANWDisponibilitaCartaInputWS input){
		//trasforma il nostro inpunt nell'input del partner
	}

	private ANWDisponibilitaCartaOutputWS getAnwOutput(InquiryDisponibilitaOutput output){
		//trasforma l'output del partner
	}
}
```

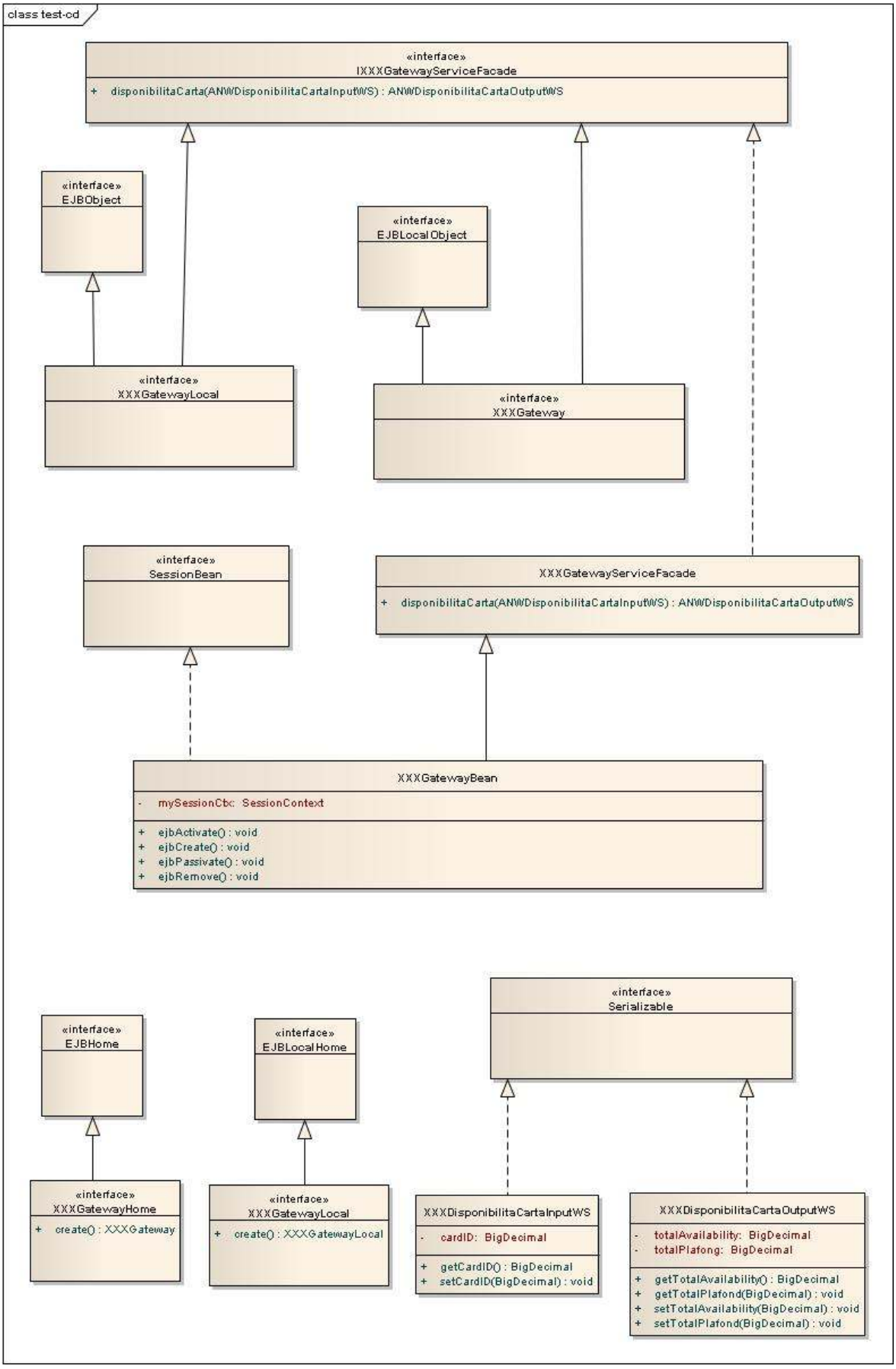

Figura 4.4 : Class diagram delle classi conivolte nel servizio di back-end

Deployement descriptor dell 'EJB

In questo file vengono specificate :

- le interfacce remota e locale, la home e la localhome,
- il tipo di entreprise java bean (stateless nel nostro caso),
- la modalità di gestione delle transazioni (gestita dal container nel nostro caso),
- le risorse alle quali il nostro sessionbean dovrà accedere (un database oracle),
- le informazione sugli EJB che il nostro session bean potrà chiamare (questo entreprise bean potrà richiamare funzionalità messe a disposizione da altre applicazioni del gruppo), vengono specificate
    - la home,
    - l'interfaccia remota
    - il tipo di entreprise java bean
- come gestire le transazioni quando vengono chiamati metodi di business.

Configurazione EJB nel file ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar id="ejb-jar_ID">
	<display-name>XA-XXX-BF</display-name>
	<enterprise-beans>
		<session id="XXXGateway">
			<ejb-name>XXXGateway</ejb-name>
			<home>it.usi.xframe.anw.bfintf.XXXGatewayHome</home>
			<remote>it.usi.xframe.anw.bfintf. XXXGateway</remote>
			<local-home>it.usi.xframe.anw.bfintf. XXXGatewayLocalHome</local-home>
			<local>it.usi.xframe.anw.bfintf. XXXGatewayLocal</local>
			<ejb-class>it.usi.xframe.anw.bfimpl.XXXGatewayBean</ejb-class>
			<session-type>Stateless</session-type>
			<transaction-type>Container</transaction-type>
			<ejb-ref id="EjbRef_1266503563480">
				<ejb-ref-name>ejb/An2Gateway</ejb-ref-name>
				<ejb-ref-type>Session</ejb-ref-type>
				<home>it.usi.xframe.an2.bfintf.An2GatewayHome</home>
				<remote>it.usi.xframe.an2.bfintf.An2Gateway</remote>
			</ejb-ref>
			<ejb-ref id="EjbRef_1268058830749">
				<ejb-ref-name>ejb/An2Card</ejb-ref-name>
				<ejb-ref-type>Session</ejb-ref-type>
				<home>it.usi.xframe.an2.bfintf.An2CardHome</home>
				<remote>it.usi.xframe.an2.bfintf.An2Card</remote>
			</ejb-ref>
			<resource-ref id="ResourceRef_1266594791701">
```

```
                    <description></description>
                    <res-ref-name>jdbc/XA-ANW-DS</res-ref-name>
                    <res-type>javax.sql.DataSource</res-type>
                    <res-auth>Container</res-auth>
                    <res-sharing-scope>Shareable</res-sharing-scope>
                </resource-ref>
            </session>
        </enterprise-beans>
        <assembly-descriptor>
            <container-transaction>
                <method>
                    <ejb-name>AnwGateway</ejb-name>
                    <method-intf>Remote</method-intf>
                    <method-name>*</method-name>
                </method>
                <trans-attribute>Never</trans-attribute>
            </container-transaction>
        </assembly-descriptor>
        <ejb-client-jar>XA-ANW-BFCL.jar</ejb-client-jar>
</ejb-jar>
```

4.4 Database

E' stato necessario uno sviluppo lato database.
La nostra applicazione appoggiandosi su un database Oracle ha la necessità di loggare tutti i messaggi SOAP in entrata ed uscita. Questo log può essere disabilitato oppure abilitato a tempo di esecuzione.

Abbiamo bisogno anche di effettuare dei controlli sui profili dei chiamanti per sapere se sono abilitati ad eseguire le operazioni richieste prima di passare queste ultime all'implementazione del servizio.
Alcune proprietà di configurazione dinamica o statica sono definite su tabella.
E' stato fatto quindi un lavoro di design della base di dati e di sviluppo di stored procedure Oracle per l'accesso in lettura e scrittura alle tabelle.

CAPITOLO 5

VERSIONAMENTO ED RILASCIO APPLICAZIONE

5.1 Versionamento del progetto

In informatica, il **controllo versione** è la gestione di versioni multiple di un insieme di informazioni.

Viene usato prevalentemente nello sviluppo di progetti ingegneristici o informatici per gestire la continua evoluzione dei documenti digitali come il codice sorgente del software, i disegni tecnici, la documentazione testuale e altre informazioni importanti su cui può lavorare una squadra di persone. Le modifiche a questi documenti sono identificate incrementando un numero o un codice associato ad essi, denominato "numero di versione", "etichetta di versione", o semplicemente "versione", e sono etichettate con il nome della persona che ha apportato la modifica.

Una semplice forma di controllo versione, per esempio, assegna il numero 1 alla prima versione di un progetto. Quando viene apportata la prima modifica, il numero identificativo di versione passa a 2 e così via.

Gli strumenti software per il controllo versione sono sempre più riconosciuti essere necessari per la maggior parte dei progetti di sviluppo software.

PANORAMICA

Il controllo versione ingegneristico si è sviluppato dai processi formali basati sui disegni cartacei.

In questo controllo era implicita la possibilità di tornare ad uno stato precedente del progetto, nei casi in cui si raggiungeva un vicolo cieco ingegneristico. Parimenti, nell'ingegneria del software, il controllo versione è qualunque pratica che tiene traccia e permette di controllare i cambiamenti al codice sorgente. Gli sviluppatori software talvolta usano il controllo versione software anche per i file di documentazione e di configurazione, oltre che per il codice sorgente. In teoria, il controllo versione può essere applicato a qualunque tipo di registrazione di informazioni. Tuttavia, in pratica le tecniche e gli strumenti più sofisticate per il controllo versione sono stati usati raramente al di fuori degli ambienti di sviluppo software (sebbene potrebbero effettivamente essere utili in molte

altre aree). Comunque, si sta cominciando ad usarli per tener traccia delle modifiche a file di CAD, soppiantando la gestione "manuale" delle versioni.

Man mano che il software viene sviluppato e dispiegato, è sempre più probabile che versioni distinte dello stesso software siano dispiegate in posti diversi, e che gli sviluppatori del software lavorino privatamente allo sviluppo di aggiornamenti. I *bug* e altre questioni riguardanti il software sono spesso presenti solamente in certe versioni (a causa del fatto che man mano che il software evolve alcuni problemi vengono corretti e altri ne vengono rilevati). Pertanto, allo scopo di individuare e correggere i *bug*, è di importanza vitale per il programmatore poter recuperare e mandare in esecuzione diverse versioni del software per determinare in quali versioni il problema si verifica. Può anche essere necessario sviluppare parallelamente due versioni del software (quando, per esempio, in una versione sono stati corretti dei bug, ma non ha nuove caratteristiche, mentre nell'altra versione vengono sviluppate le nuove caratteristiche).

Al livello più semplice, gli sviluppatori possono conservare una copia per ogni diversa versione del software, e identificarle appropriatamente. Questo approccio è stato usato in molti grandi progetti software. Sebbene questo metodo possa funzionare, è inefficiente (dato che verranno conservate molte copie quasi identiche del software), richiede molta disciplina da parte degli sviluppatori, e conduce spesso ad errori. Conseguentemente, sono stati sviluppati dei sistemi per automatizzare (in parte o in toto) il procedimento di controllo versione.

Tradizionalmente, i sistemi di controllo versione hanno usato un modello centralizzato, in cui tutte le funzioni di controllo versione sono eseguite da un server condiviso. Alcuni anni fa, certi sistemi come TeamWare, BitKeeper e GNU arch hanno cominciato a usare un modello distribuito, in cui ogni sviluppatore lavora direttamente con il suo repository locale, e le modifiche sono condivise tra i repository in un passo separato. Questa modalità di operare permette di lavorare senza una connessione di rete, e consente anche agli sviluppatori di accedere alle funzioni di controllo versione senza aver bisogno di permessi concessi da un'autorità centrale.

Nella maggior parte dei progetti di sviluppo software, più sviluppatori lavorano in parallelo sullo stesso software. Se due sviluppatori tentano di modificare lo stesso file contemporaneamente, in assenza di un metodo di gestione degli accessi, essi possono facilmente sovrascrivere o perdere le modifiche effettuate contestualmente. La maggior parte dei sistemi di controllo versione possono risolvere questo problema in due diversi modi. Questo problema riguarda solamente i sistemi di controllo versione centralizzati, poiché i sistemi distribuiti permettono intrinsecamente più modifiche simultanee.

Alcuni sistemi prevengono i problemi dovuti ad accessi simultanei, semplicemente bloccando (*lock*) i file, cosicché solamente uno sviluppatore per volta ha diritto di accesso in scrittura alla copia di quel file contenuta nel repository centrale. Altri, come CVS, permettono a più sviluppatori di modificare lo stesso file nello stesso tempo, e forniscono degli strumenti per combinare le modifiche in seguito (merge). In quest'ultimo tipo, può esistere una operazione di blocco facoltativa.

I pregi e i difetti del blocco dei file sono in discussione. Tale operazione può fornire una certa protezione da difficili conflitti di *merge* quando un utente sta facendo delle modifiche radicali a molte sezioni di un grande file (o di un gruppo di file). Ma se i file sono lasciati bloccati troppo a lungo, gli altri sviluppatori possono essere tentati di scavalcare il software di controllo versione e di modificare comunque i file localmente, e ciò può condurre a problemi più seri.

Alcuni sistemi tentano di gestire i profili di chi ha il permesso di apportare modifiche; per esempio, richiedendo che le modifiche a un file siano approvate da un recensore designato prima di essere aggiunte.

La maggior parte dei sistemi di controllo versione usano la *compressione delta*, che conserva solamente le differenze fra le versioni successive dei file. Questo consente un immagazzinamento efficiente di più versioni di un file, purché, come solitamente succede, le modifiche tra una versione e la successiva riguardino solamente una piccola parte del testo.

Alcuni degli strumenti di controllo versione più avanzati offrono molte altre funzionalità, consentendo una più profonda integrazione con altri strumenti e procedimenti di ingegneria del software. Spesso sono disponibili dei *plug-in* per IDE come Eclipse e Visual Studio.

La cronologia di Wikipidia è un esempio di sistema di controllo versione.

GLOSSARIO

Repository

> Il **repository** è dove i file sono memorizzati, spesso su un server. Talvolta è chiamato anche **depot**

Commit

> Un **commit** (o, più raramente, **install**, **submit**, **check-in** o **ci**) si effettua quando si copiano le modifiche fatte su file locali nella directory (il software di controllo versione controlla quali file sono stati modificati dall'ultima sincronizzazione).

Modifica

Una **modifica** (**change**) rappresenta una specifica modifica ad un documento sottoposto al controllo di versione. La granularità delle modifiche considerate come cambiamenti varia tra i sistemi di controllo versione.

Change List

Su molti sistemi di controllo versione con commit di modifiche multiple atomiche, una **changelist** identifica un insieme di **changes** fatti in un singolo commit.

Check-Out

Un **check-out** (o **checkout** o **co**) effettua una copia di lavoro dal repository (può essere visto come l'operazione inversa dell'importazione).

Update

Un **update** (o **sync**) copia le modifiche fatte sul repository nella propria directory di lavoro (può essere visto come l'operazione inversa del commit).

Merge / Integrazione

Un **merge** o **integrazione** unisce modifiche concorrenti in una revisione unificata.

Revisione

Una **revisione** o **versione** è una versione in una catena di modifiche.

Import

Il termine **import** è usato per descrivere la copiatura dell'intero albero di directory locale sul repository.

Export

Un **export** è simile ad un **check-out** eccetto il fatto che crea un albero di directory vuoto senza metadati di controllo versione (spesso è usato precedentemente alla pubblicazione dei contenuti).

Conflitto

Un conflitto si presenta quando diversi soggetti fanno modifiche allo stesso documento. Non essendo il software abbastanza intelligente da decidere quale tra le modifiche è quella 'corretta', si richiede ad un utente di **risolvere** il conflitto.

Risolvere

L'intervento di un utente per la risoluzione di un conflitto tra modifiche differenti di uno stesso documento.

Come detto nell'analisi funzionale, in azienda si usa Serena - PVCS: Prodotto per il versioning e change management. Permette gestire e versionare i cambiamenti del codice applicativo.

Serena-PVCS usa un repository centralizzato per gestire i diversi file dei progetti.

Sulla macchina locale viene installato il client di Serena.

Sono anche disponibili dei plug-in per l'IDE di sviluppo WSAD.

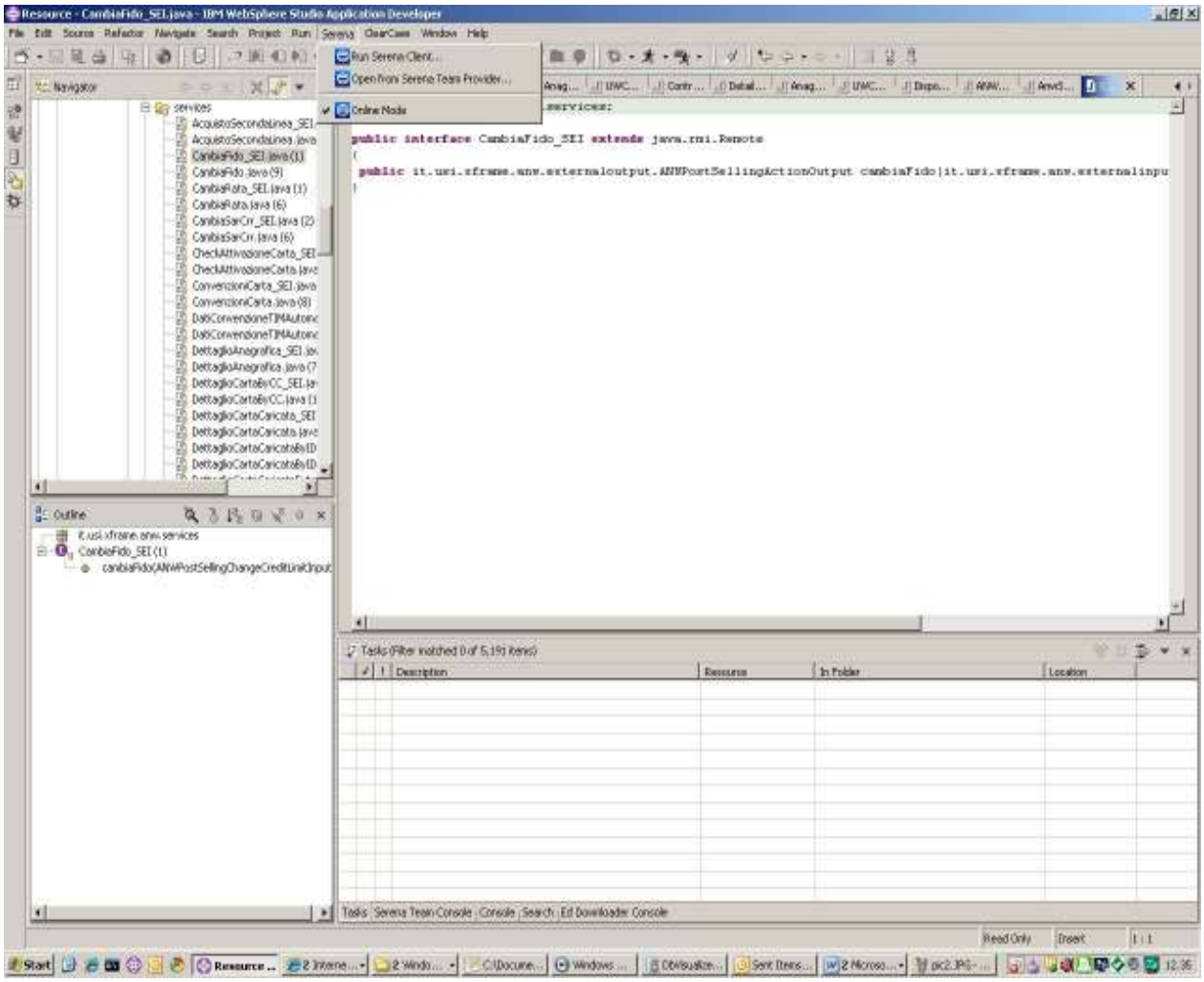

Figura 5.1 : Accesso a Serena dall'ambiente di lavoro

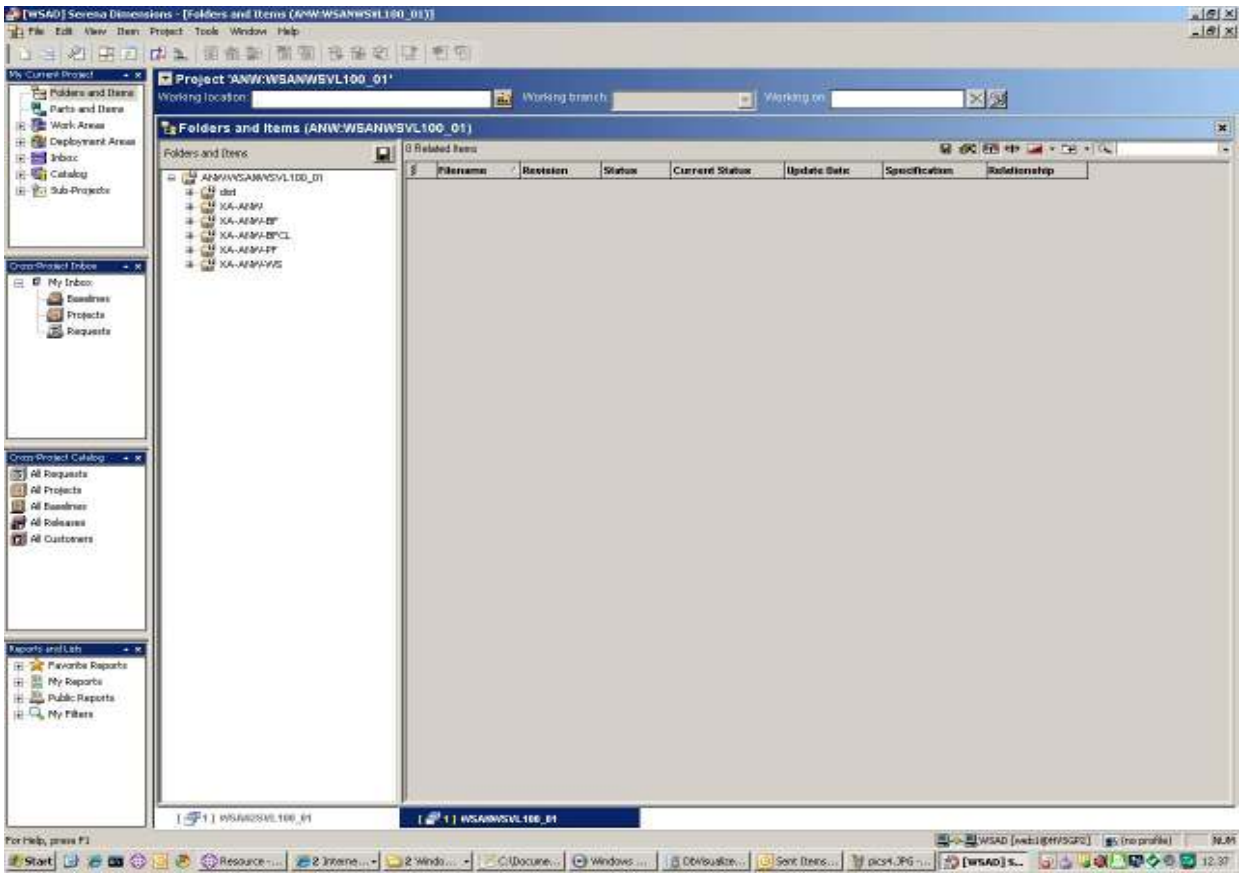

Figura 4.5 : Accesso a Serena e menu

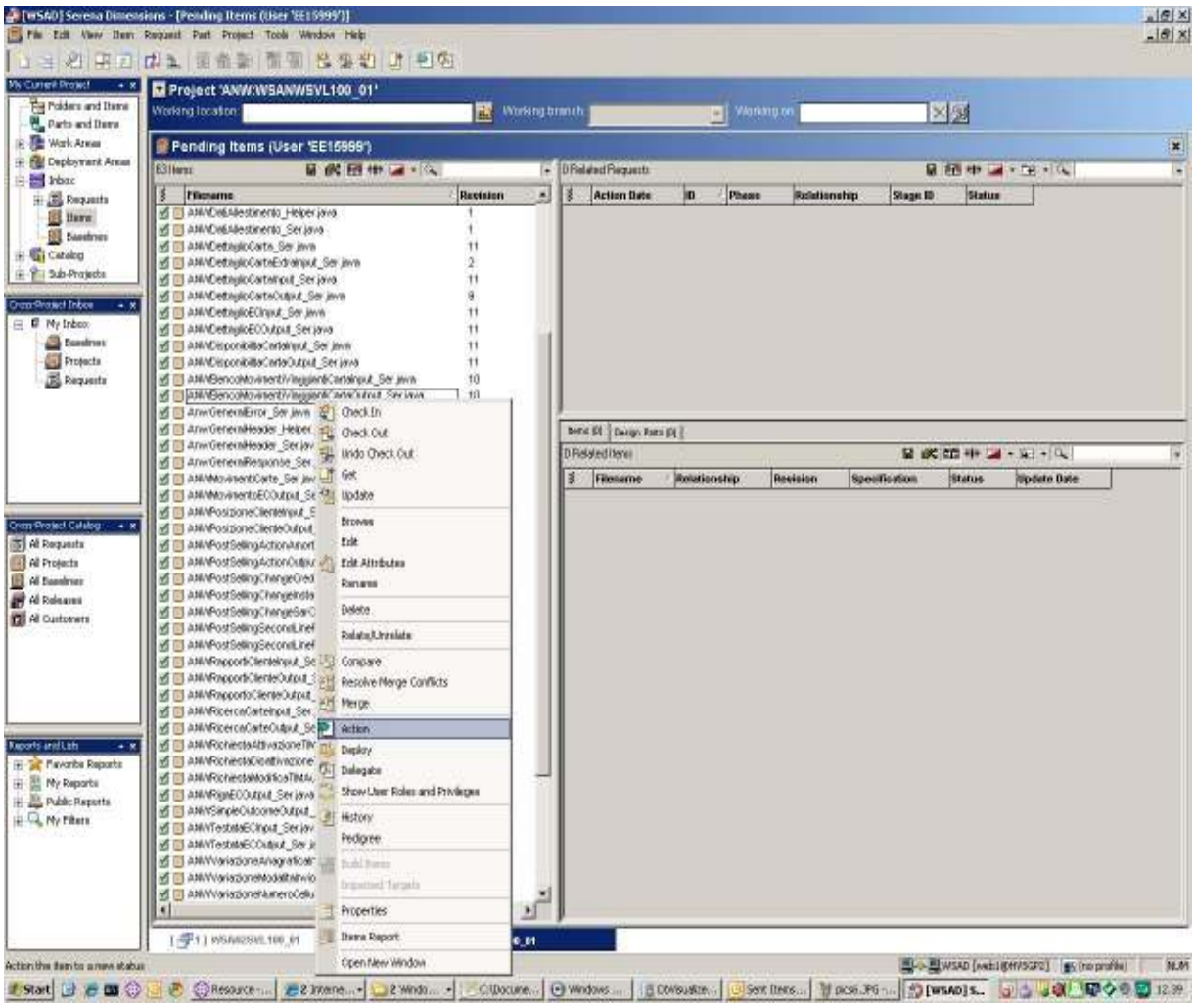

Figura 5.2 : Selezione file da mettere in pronto per il deploy

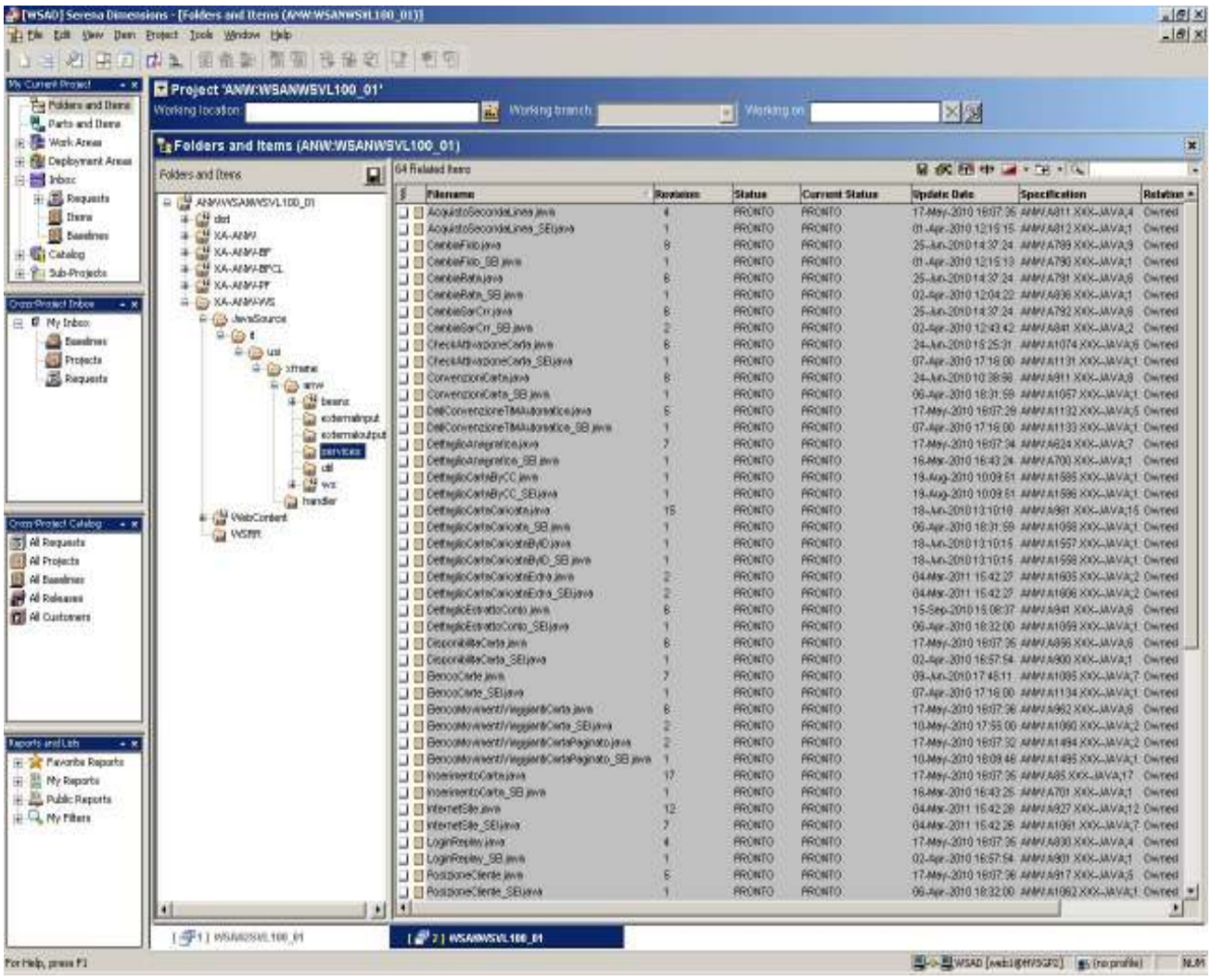

Figura 5.3 : Files nello stato pronto

5.2 Installazione applicazione

Una volta che si decide di installare l'applicazione nelle varie ambienti si parte dal repository di Serena-PVCS dove si cambia lo stato dei vari file che hanno subito una modifica e che devono essere portati nella nuova installazione.

In ambiente di sviluppo dopo aver fatto check-in (commit) di una modifica su file , esso è in stato "definito" il che significa che il file non è pronto per una installazione per cui bisogna cambiare il suo stato da "definito" a "pronto".

Per il rilascio e le installazione delle applicazioni si usa il prodotto Gandalf che è un'applicazione XFRAME sviluppata in casa.

In quanto esterno ho le credenziali per eseguire il rilascio dell'applicazione solo in ambiente di collaudo, per gli ambienti di validazione e produzione c'è la necessità di una richiesta formale di installazione che verrà eseguita di preferenza di notte e durante il fine settimana.

Accendendo a GANDALF , il primo passo da compiere è generare una baseline che consiste nel recuperare i sorgenti dell'applicazione su serena(tutti i file che sono nello stato "pronto") , compilarli e generare il file EAR (entreprise archive) che verrà usata nella fase successiva.
Dopo l'esito positivo della fase precedente viene creata una versione nuova dell'applicazione e l'installazione può quindi essere schedulata nell'ambiente desiderato.
si mettono i vari file in pronto e si genera una versione, prodotto sviluppato in casa gandalf si genera una baseline(spiegazione) e si schedula l'installazione.
Se qualcosa è andata male senza aver bisogno di ripartire dai sorgenti sempre con questo strumento si può richiedere al middleware l'installazione di una versione precedente (in ambiente di validazione o produzione) in ambiente di collaudo sono in grado di tornare ad una versione precendente senza l'aiuto del middleware.
La mia applicazione si deve interfacciare con altre applicazioni come spiegato in fase di analisi tecnica per cui queste applicazioni devono essere installate nei vari ambienti prima di richiedere l'installazione della nostra.
In tutti gli ambienti sono già installate le applicazione da cui dipende la mia (Anagrafica, e l'integration frame work che mi permette di chiamare i servizi HOST).
Per i servizi di back-end sviluppati ad hoc dal fornitore esterno, devo accedere al suo di repository per scaricare i file generare il file EAR che porto su Serena-PVCS.
In una fase successiva vado su Gandalf a schedulare l'installazione.

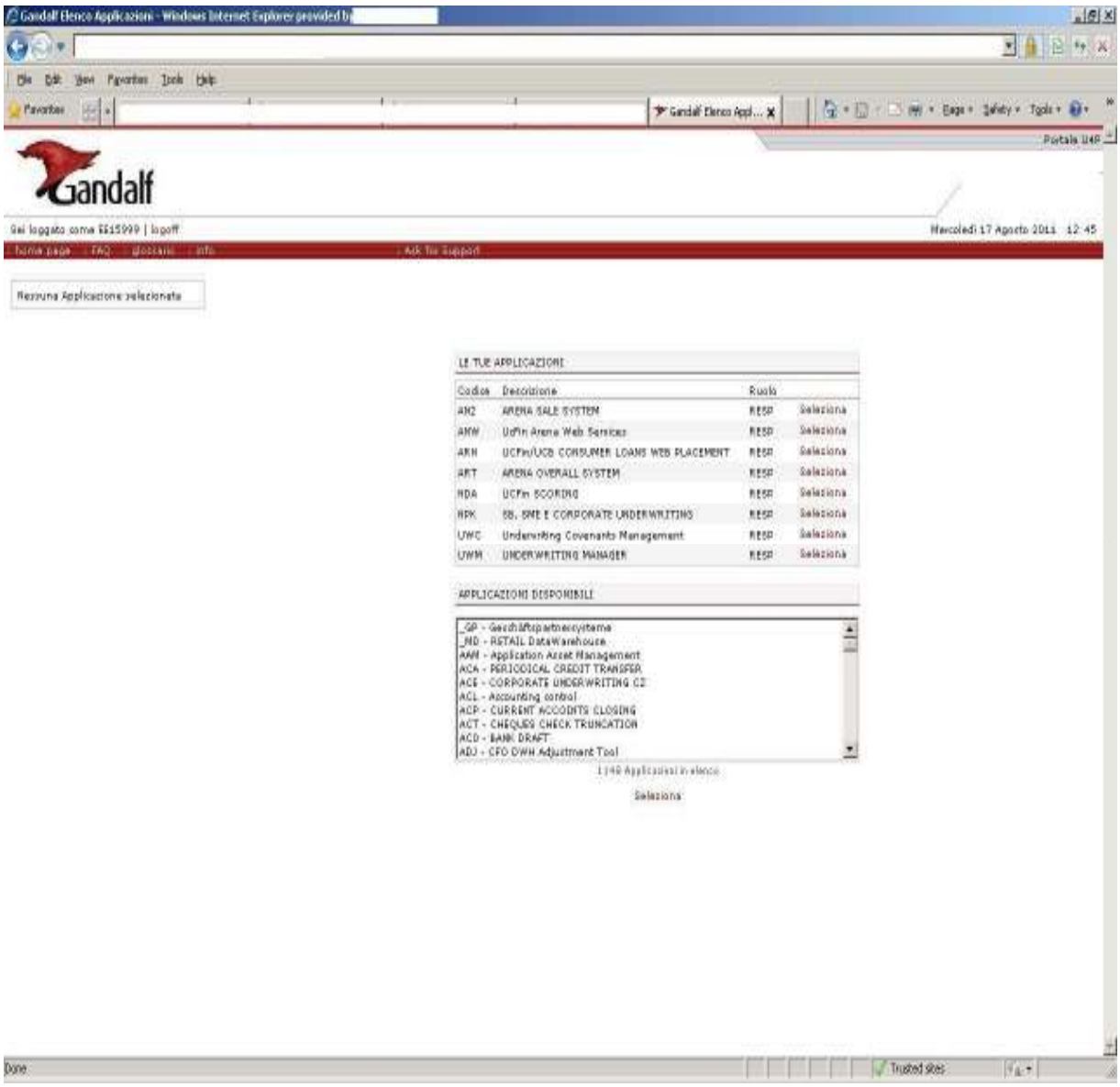

Figura 5.4 : Gandalf - Scelta del prodotto per la generazione e l'installazione

Figura 5.5 : Gandalf - Menu principale

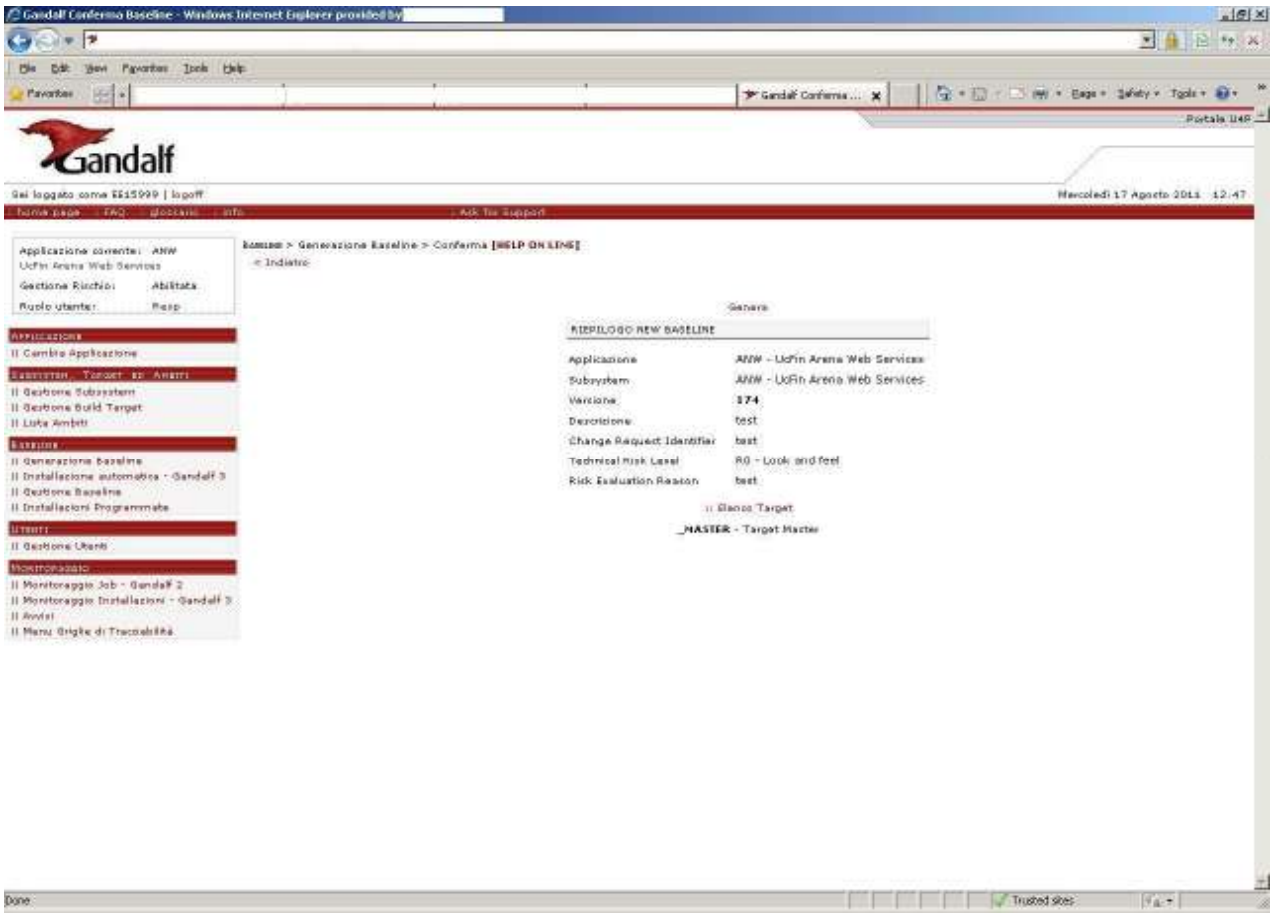

Figura 5.6 : Gandalf - Generazione baseline

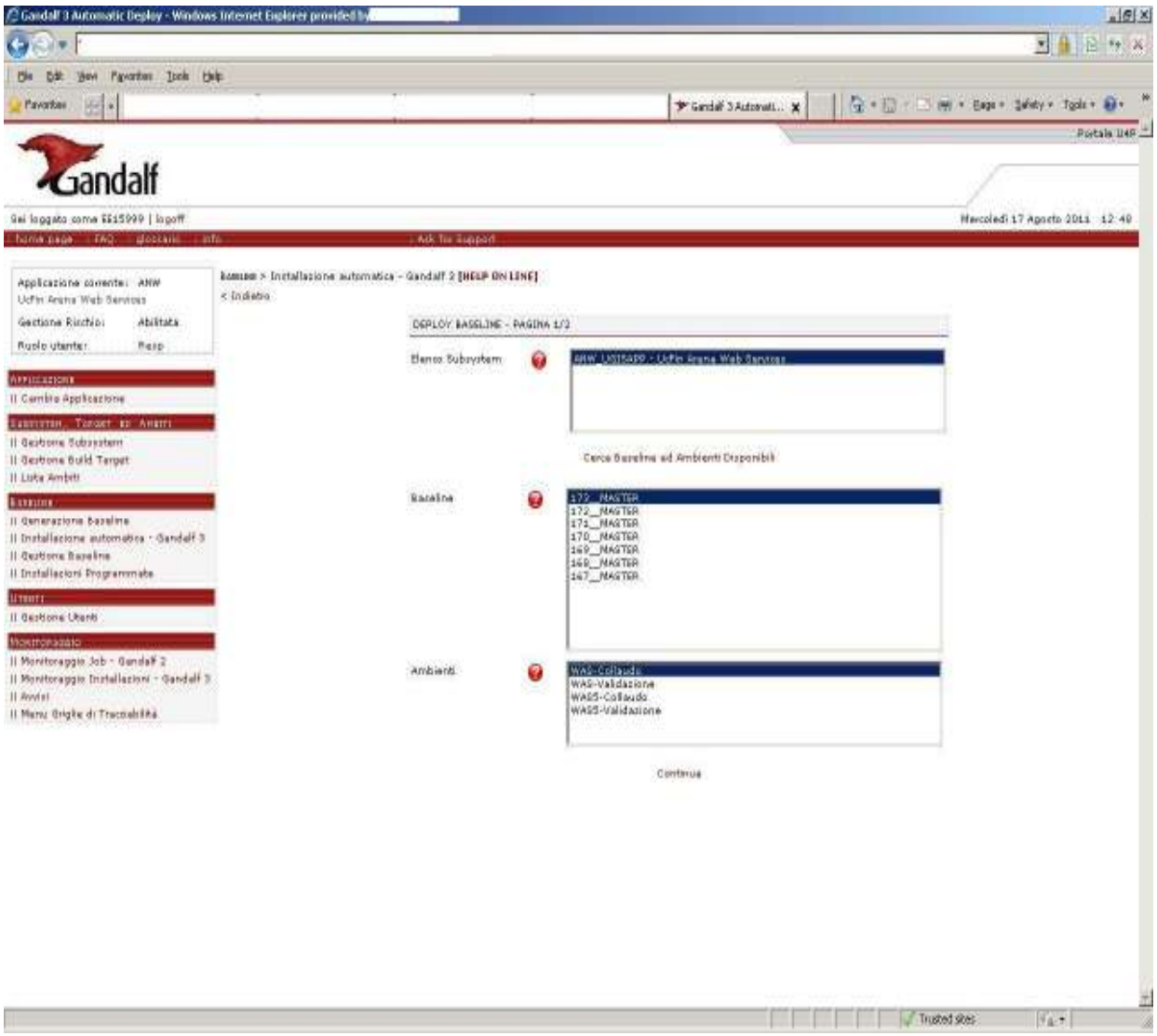

Figura 5.7 : Gandalf - Scelta ambiente per l'installazione del prodotto

CAPITOLO 6

TESTING E PASSAGGIO IN PRODUZIONE

6.1 Unit testing

Nella programmazione informatica, lo **unit testing** è una procedura usata per verificare singole parti di un codice sorgente . Per unità si intende genericamente la minima parte testabile di un codice sorgente: nella programmazione procedurale un'unità può rappresentare un singolo programma, funzione, procedura, etc.; nella Programmazione orientata agli oggetti, la più piccola unità può essere il metodo.

Lo Unit Testing si articola in test case ciascuno dei quali dovrebbe essere indipendente dagli altri ed viene normalmente eseguito dagli sviluppatori, non da utenti finali.

BENEFICI

Lo scopo dell'Unit testing è quello di verificare il corretto funzionamento di parti di programma permettendo così una precoce individuazione dei bug. Uno unit testing accurato può dare una prova certa se un pezzo di codice funziona correttamente, con importanti vantaggi:

Semplifica le modifiche

Lo unit testing facilita la modifica del codice del modulo in momenti successivi con la sicurezza che il modulo continuerà a funzionare correttamente. Il procedimento consiste nello scrivere en:test case per tutte le funzioni e i metodi, in modo che se una modifica produce un fallimento del test, si possa facilmente individuare la modifica responsabile.

Unit test già predisposti semplificano la vita al programmatore nel controllare che una porzione di codice stia ancora funzionando correttamente. Un buon unit testing produce en:test case che coprano tutti i percorsi del codice dell'unità, con particolare attenzione alle condizioni nei cicli (test sugli if, while, for).

In sistemi con unit testing continuo, tali test sono in grado di garantire l'automatica integrità del codice ad ogni modifica.

Semplifica l'integrazione

Lo unit testing semplifica l'integrazione di moduli diversi perché limita i malfunzionamenti dovuti a problemi nella interazione tra i moduli e non nei moduli stessi, rendendo i test di integrazione più semplici.

Un argomento molto dibattuto è quello della non necessità di test di integrazione manuali, in caso si sia organizzata una procedura di unit testing sufficientemente completa. In realtà spesso un elaborato sistema di unit testing fornisce una falsa sicurezza e un test di integrazione gestito da esseri umani è in genere ugualmente necessario. Probabilmente la reale necessità del fattore umano nella procedura di test dipende dalle caratteristiche del sistema nel quale si sviluppa e soprattutto dalla disponibilità di risorse.

Supporta la documentazione

Lo unit testing fornisce una documentazione "viva" del codice, perché è intrinsecamente un esempio di utilizzo dell'API del modulo.

I test case incorporano le caratteristiche critiche per il successo di un'unità di codice. Tali caratteristiche indicano l'uso appropriato dell'unità e i comportamenti errati che devono essere identificati nel suo funzionamento. Pertanto lo unit testing documenta tali caratteristiche, sebbene in molti ambienti questi non possono costituire la sola documentazione necessaria. In compenso, la tradizionale documentazione diventa spesso obsoleta a cause di successive modifiche del codice non documentate.

Separazione dell'interfaccia dall'implementazione

Poiché alcune classi possono far riferimento ad altre, il test di una classe spesso si propaga alle altre. Un esempio è una classe che interagisce con un database: testare la classe spesso implica la scrittura del codice che interagisce con il database. Questo è un problema perché lo unit test non dovrebbe mai varcare i confini della classe. La conseguenza è che il programmatore, nel progettare lo unit testing, impara ad isolare la classe da analizzare, individuando l'interfaccia con il database ed implementandola con un mock object, una simulazione dell'oggetto reale che può essere effettuata in condizioni controllate. L'effetto è un test più approfondito e quindi uno unit testing di qualità più elevata.

Limitazioni dello unit testing

In generale il testing non riesce ad identificare tutti gli errori in un programma e lo stesso vale per lo Unit Testing che, analizzando per definizione le singole unità, non può identificare gli errori di

integrazione, problemi legati alla performance e altri problemi legati al sistema in generale. Lo unit testing è più efficace se utilizzato in congiunzione con altre tecniche di testing del software.

Come ogni forma di testing, anche lo Unit Testing non può individuare l'assenza di errori ma può solo evidenziarne la presenza.

Il testing del software è un problema di matematica combinatoria. Per esempio, ogni test booleano richiede almeno due test, uno per la condizione di "vero" e uno per quella di "falso". Si può dimostrare che, per ogni linea di codice funzionale, siano necessarie dalla 3 alle 5 linee di codice per il test. È quindi irrealistico testare tutte le possibili combinazioni di input di qualsiasi codice non banale senza un tool apposito di generazione di casi di test.

Per ottenere gli sperati benefici dallo unit test, è richiesto un rigoroso senso di disciplina durante tutto il processo di sviluppo. È essenziale mantenere traccia non solo dei test che non stati sviluppati ed eseguiti, ma anche di tutte le modifiche effettuate al codice funzionale dell'unità in esame e di tutte le altre. L'uso di un sistema di controllo versione è essenziale. Se una versione successiva di una unità fallisce un test che aveva passato in precedenza, il sistema di controllo versione permette di evidenziare le modifiche al codice intervenute nel frattempo.

Mock-object

Nella programmazione orientata agli oggetti, i mock sono degli oggetti simulati che riproducono il comportamento di oggetti reali in maniera controllata.

Un programmatore crea un mock con lo scopo di testare il comportamento di oggetti reali non ancora implementati o non disponibili.

Sviluppo dei test

I test in ambiente di sviluppo sono stati effettuati usando dei mock object per simulare ad esempio la chiamata al servizio dell’anagrafica utenti, ai servizi di back-end sviluppati dal fornitore esterno dato che questi ultimi non sono disponibili in ambiente di sviluppato.

Lo scopo del test a questo punto è verificare il corretto funzionamento del mio codice.

Devo quindi testare i singoli servizi sviluppati.

Per eseguire il singolo test uso un tool del IDE WSAD che partendo dal WSDL del servizio sviluppato mi permette di chiamare tutte le operazioni disponibili .

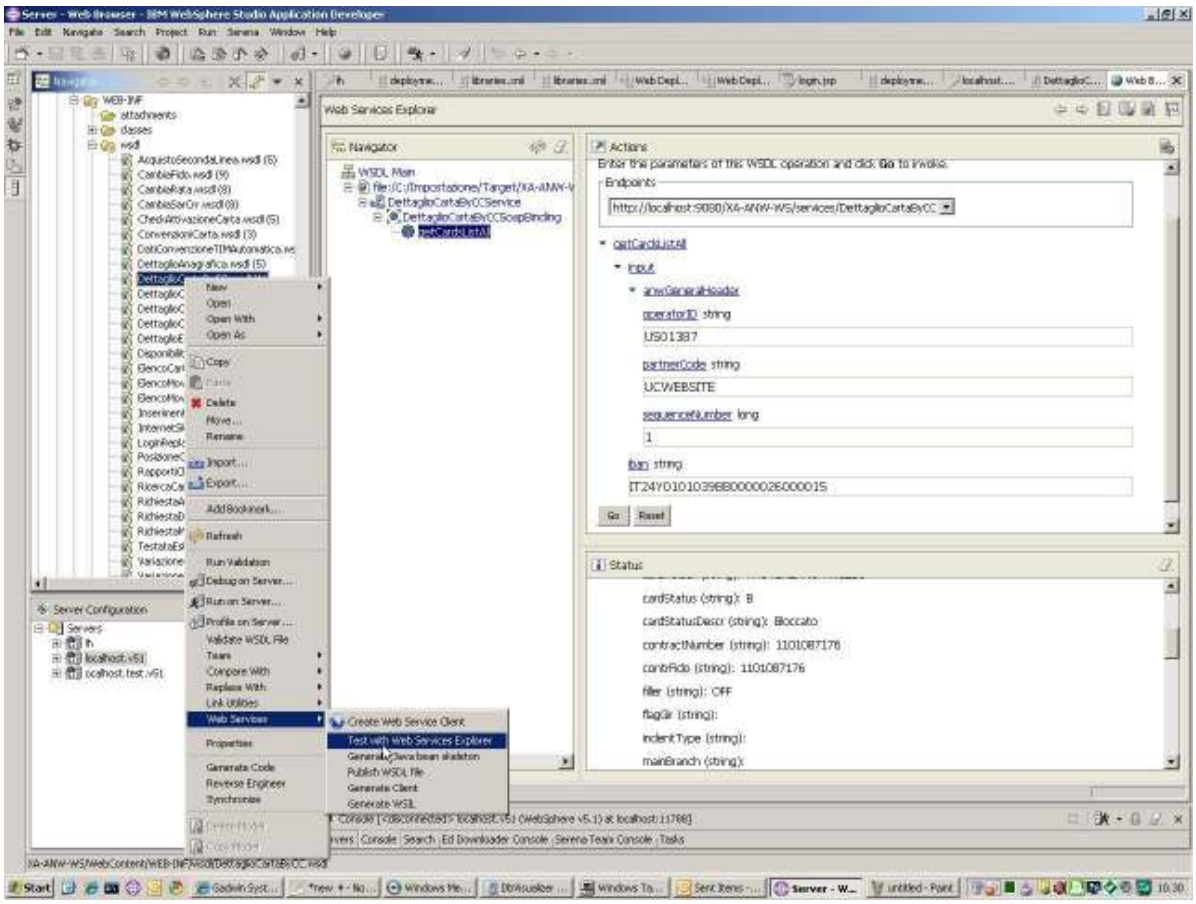

Figura 6.1 : WSAD - Selezione del WSDL relativo al servizio da testare

Dal WSDL del servizio che si desidera testare usando il tool dell' IDE si riesce a testare le varie operazioni.

C'è la possibilità di settare l'endpoint del servizio (lo si può fare puntare quindi in base alle esigenze in sviluppo,collaudo o validazione).

Una volta settato l 'endpoint e selezionato il servizio da testare, compare la mascherina (FIG XXX) che permette di compilare il form in alto a destra con i dati di input se richiesti dopo di che cliccando sul tasto "go" viene eseguita la chiamata e il risultato compare in basso a destra.

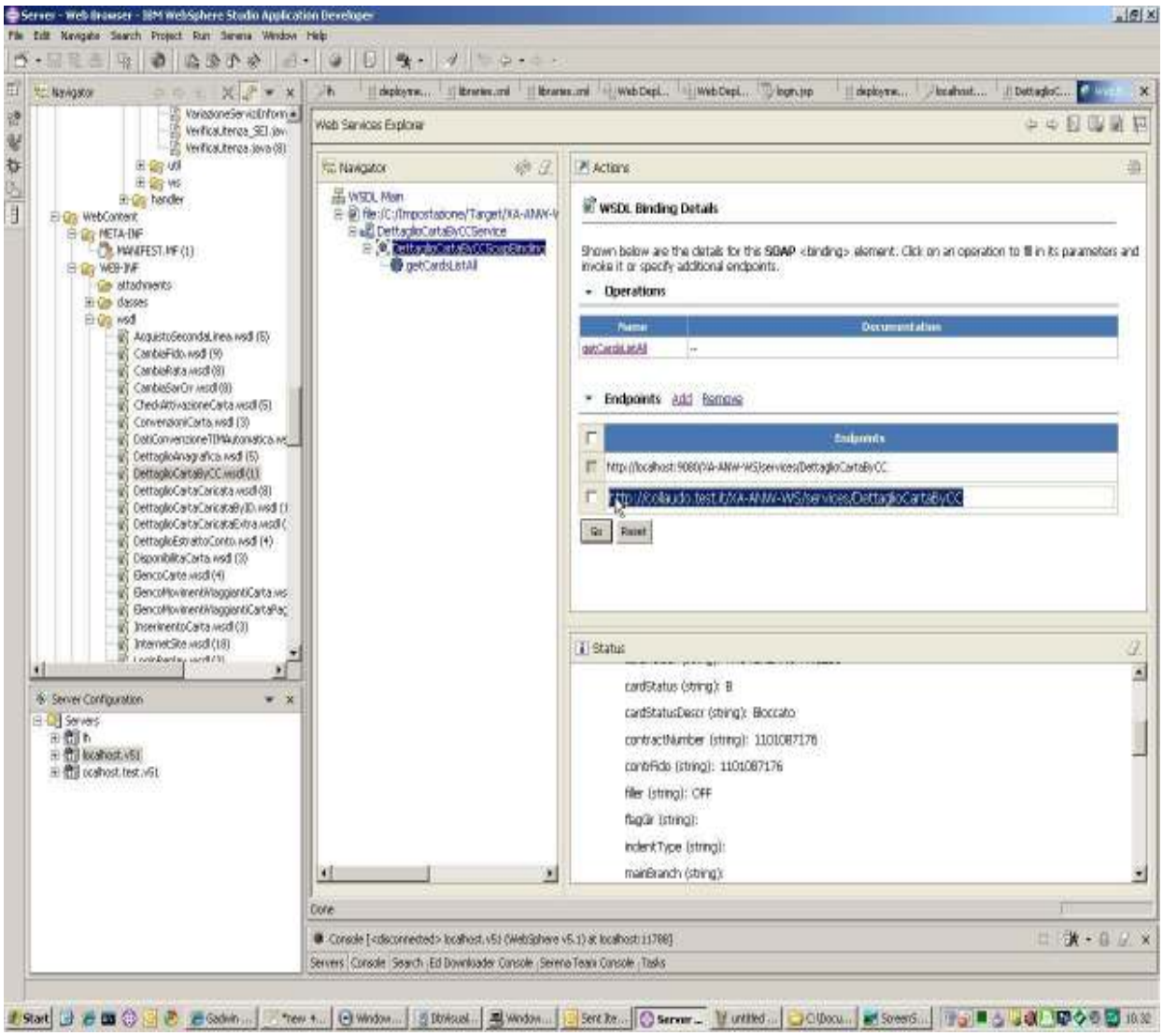

Figura 6.2 : WSAD - Modifica dell'endpoint relativo al servizio da testare

Una volta che i test sono andati a buon fine, si effettua il deploy dell'applicazione in collaudo per i test d'integrazione.

Quando i bug nelle componenti sono stati rilevati e corretti, le componenti sono pronte per essere integrate in sottosistemi più grandi

6.2 Test d'integrazione

Il test di integrazione è il processo di verifica dell'interazione tra componenti software. Le strategie classiche di test di integrazione, top-down o bottom-up, vengono utilizzate con il software tradizionale, strutturato in modo gerarchico. Le strategie moderne di integrazione sistematica sono invece guidate dall'architettura, cioè i componenti e sottosistemi software vengono integrati sulla

base delle funzionalità identificate. Il test a livello di integrazione è un'attività continuativa. Tranne i casi di software molto piccoli e semplici, le strategie di test di integrazione sistematiche ed incrementali sono da preferire rispetto alla strategia di mettere tutti i componenti insieme nello stesso momento, in quello che viene chiamato "big bang" testing.

Partendo da una architettura organizzata gerarchicamente, le integrazioni possono essere realizzate con approccio top-down o bottom-up o misto.

Il testing di integrazione rileva bug che non sono stati individuati durante il testing di unità, focalizzandosi su un insieme di componenti che integrate.

Due o più componenti sono integrate e analizzate, e quando dei bug sono rilevati nuove componenti possono essere aggiunte per riparare i bug.

Sviluppare test stub e test driver per un test di integrazione sistematico è oneroso in termini di tempo.

I test di integrazione verificano come due (o più) unità lavorano insieme.

I test d'integrazione richiede :

- la costruzione del sistema a partire dai suoi componenti ed il testing del sistema risultante per scoprire problemi che nascono dall'interazione fra I vari componenti.
- l'identificazione di gruppi di componenti che realizzano le varie funzionalità del sistema e la loro integrazione mediante componenti che li fanno lavorare insieme.

STRATEGIE DI TESTING

L'ordine in cui le componenti sono integrate può influenzare lo sforzo richiesto per l'integrazione.

L'ordine in cui i sottosistemi sono selezionati per il testing e l'integrazione determina la strategia di testing :

• Integrazione Big bang (Non incrementale)

• Integrazione Bottom up

• Integrazione Top down

• Sandwich testing

• Varianti

Ognuna di queste strategie è stata originariamente concepita per una decomposizione gerarchica del sistema

• Ciascuna componente appartiene ad una gerarchia di layer, ordinati in base all'associazione "Call".

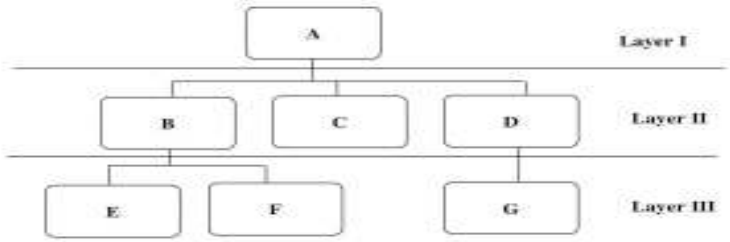

Figura 6.3 : Gerarchia a tre livelli

Approccio BIG-BANG

Le componenti sono prima testate individualmente e poi testate insieme come un singolo sistema.

Sebbene sia semplice da realizzare è costoso: se un test scopre un fallimento è impossibile distinguere i fallimenti nelle interfacce dai fallimenti all'interno delle componenti.

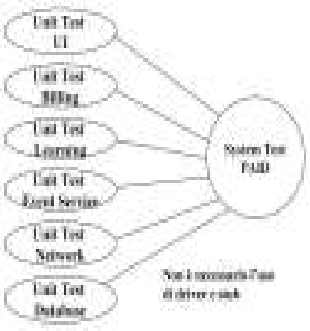

Figura 6.4 : Approccio BIG-BANG

Approccio BOTTOM-UP

I sottosistemi nel layer più in basso della gerarchia sono testati individualmente.

I successivi sottosistemi da testare sono quelli che "chiamano" i sottosistemi testati in precedenza.

Si ripete questo processo finché non sono testati tutti i sottosistemi.

I test driver sono usati per simulare le componenti dei layer più "in alto" che non sono stati ancora integrati.

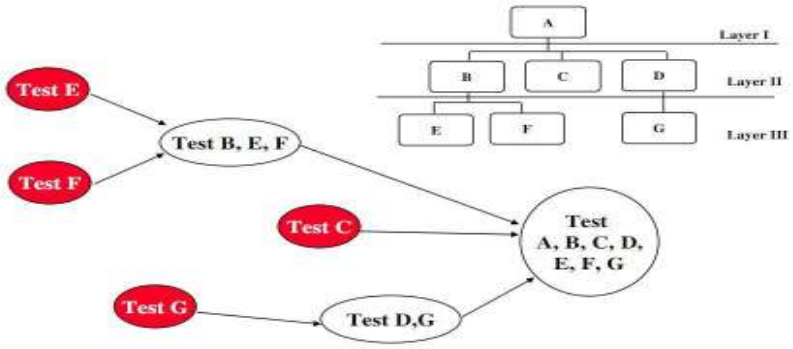

Figura 6.5 : Approccio BOTTOM-UP

Pro e contro dell'approccio BOTTOM-UP

Non è ideale per sistemi funzionalmente decomposti:

- Testa i sottosistemi più importanti alla fine
- Utile per integrare i seguenti sistemi
  - » Sistemi Object-oriented
  - » Sistemi real-time

» Sistemi con requisiti stringenti sulle performance

Non sono necessari test stub.

Gli errori nelle interfacce possono essere individuati piùfacilmente:

• Se la componente ad alto livello viola le assunzioni fatte dallacomponente a basso livello, questo può essere individuato più facilmente dagli sviluppatori.

Apprioccio TOP-DOWN

Testa prima i layer al top o i sottosistemi di controllo.

Successivamente combina tutti i sottosistemi che sono chiamati dai sottosistemi testati e quindi testa la collezione di sottosistemi risultante.

Itera questa attività fino a quando tutti i sistemi non sono integrati e testati.

I test stub sono usati per simulare le componentidei layer al bottom che non sono state ancora integrate.

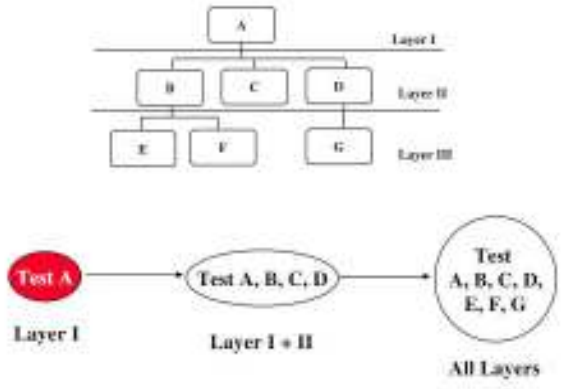

Figura 6.6 : Approccio TOP-DOWN

Pro e contro dell'approccio TOP-DOWN

I test cases possono essere definiti in termini di funzionalità del sistema..

Vantaggio: si inzia dall'interfaccia utente

Svantaggi:

- Scrivere gli stub può essere di icile: gli stub devono consentire tutte le possibili condizioni da testare.
- É possibile che un grande numero di stub sia richiesto, specialmente se il livello più in basso del sistema contiene molti metodi,

Una soluzione per evitare molti stub: top-down modificato

- Testa individualmente ogni layer della decomposizione prima di fare il merge dei layer
- Svantaggio: sono necessari sia test stub sia test driver

Approccio SANDWICH

Combina la strategia top-down con quella bottom-up

Il sistema è visto come se avesse 3 layers

- Un layer target
- Un layer sopra il target
- Un layer sotto il target

Il testing converge al layer target.

Come selezionare il layer target se ci sono più di 3 layers?

- Si può tentare di minimizzare il numero di stub e driver

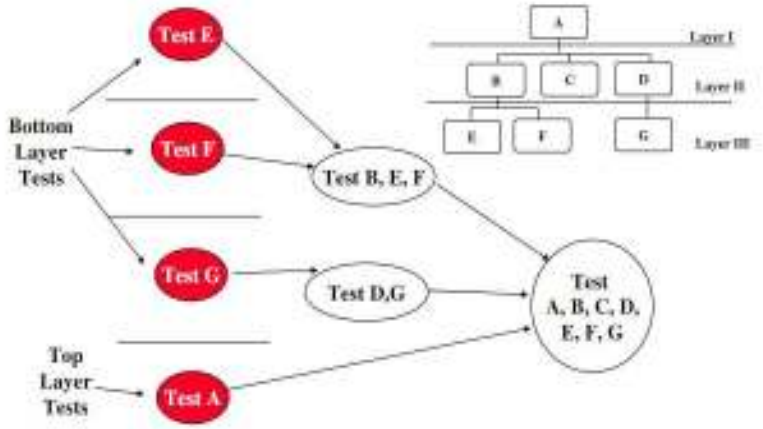

Figura 6.7 : Approccio SANDWICH

Pro e contro dell'approccio SANDWICH

I test dei layer al top e al bottom possono essere effettuati in parallelo.

Problema: non sono testati i sottosistemi individuali prima dell'integrazione.

APPROCCIO USATO

Per il test di integrazione abbiamo usato l 'approccio Big-Bang.

L'obbiettivo del test era verificare il coretto funzionamento dell'intero sistema. Verificare quindi se i servizi sviluppati si interfacciano correttamente con il mondo esterno (Anagrafica utente, Integration frame work, Database, fornitore esterno di servizi).

Ognuna delle applicazioni con le quali la nostra applicazione deve interagire sono installate in ambiente di collaudo e offrono delle interfacce web per poter effettuare dei test e verificare il comportamento dei singoli servizi offerti via EJB.

Ognuna delle applicazioni sono quindi state testate singolarmente per poi essere testate insieme come un singolo sistema.

Una volta che è andato a buon fine il test d'integrazione vengono effettuati il test prestazionale e lo stress test.

6.3 Test prestazionale e stress test

Il test prestazionale è specificamente effettuato per verificare che il software soddisfi i requisiti di prestazioni specificati, ad esempio i livelli di capacità di utilizzo ed i tempi di risposta.

Questo test è stato effettuato dal middleware che dispone di materiale adeguato per poterlo eseguire. Avendo a disposizione i tempi di esecuzione / risposta dei vecchi servizi , sono stati fatti dei confronti con i tempi di quelli nuovi. Confronti che hanno avuto un esito positivo.

Lo stress test sollecita il software al livello massimo di carico progettato, ed oltre. Anche questo test è stato eseguito dal middleware.

6.4 Test di accettazione utente (USER ACCEPTANCE TEST)

Il testing di accettazione controlla il comportamento del sistema rispetto ai requisiti (comunque espressi) del committente. I clienti effettuano, o specificano, attività tipiche di uso del sistema, per controllare che i loro requisiti siano stati soddisfatti. Questa attività di test può coinvolgere o meno gli sviluppatori del sistema.

Testing condotto in un ambiente operativo reale per determinare se un sistema soddisfa i propri criteri di accettazione (ad esempio i requisiti iniziali, e le attuali esigenze dei suoi utenti) e per permettere al cliente di determinare se accettare o meno il sistema .

Una volta che l'applicazione è pronta per il rilascio ,il passo cruciale è quello del test di accettazione (User acceptance test detto UAT). Questo test è quasi sempre eseguito prima del rilascio dell'applicazione in ambiente di produzione.

Prerequisiti

Prima di eseguire il test di accettazione dell'applicazione, questa deve essere interamente sviluppata e tutti gli altri test devono già essere stati eseguiti (unit, integration, sistema) . Dato che questi test sono già stati eseguiti , la maggiore parte dei bug tecnici sono già stati rilevati e risolti.

Cosa testare ?

Per assicurare uno svolgimento efficiente del test vanno creati dei test case. Questi ultimi possono derivare dagli use cases identificati durante la fase di definizione dei requisiti. I test cases identificati devono assicurare la copertura della maggioranza dei scenari durante i test.

Questo tipo di test si focalizza sull'utilizzo dell'applicazione nel mondo reale. Il test viene eseguito in un ambiente che simula l'ambiente di produzione. Nel nostro caso è stato giustamente eseguito nell'ambiente di validazione.

Come testare ?

UAT è di solito un test di tipo black-blox. In altre parole si focalizza sugli aspetti funzionali e sulà'usabilità dell'applicazione piuttosto che sugli aspetti tecnici.

I passi da compiere per eseguire un UAT sono i seguenti :

1. Piano UAT
2. Disegno degli User Acceptance Test cases.
3. Selezione di un team per l'esecuzione dei test
4. Esecuzione dei test.
5. Documentazione delle incongruenze o errori rilevate.
6. Risoluzione di eventuali bug.
7. Decisione.

Gli UAT sono stati eseguiti da uno staff della banca in ambiente di validazione. Gli eventuali bug erano segnalati usando un'applicazione del gruppo bancario dedicata alla risoluzione dei bug.

In quanto esterno ho accesso all'applicazione dedicata alla gestione dei bug. Le segnalazioni arrivano via mail con una breve descrizione ed un codice. Connettendosi all'applicazione ed usando il codice del bug si ha accesso ad informazioni dettagliate sul bug. Dopo aver risolto il bug viene cambiato lo stato della segnalazione e lo staff dedicato ai test può di nuovo eseguire il test per il controllo della modifica effettuata.

6.5 Passaggio in produzione

Dopo l'esito positivo di tutte le fasi di test, l'applicazione viene portata in ambiente di produzione.

Maggiore attenzione è data a questa fase data che è un'attività altamente rischiosa coinvolgendo numerosi team in azienda .

Il passaggio in produzione è un'attività altamente sincronizzata e che va pianifica nei minimi dettagli :

- Tutte le applicazioni che interagiscono devono essere portate in produzione almeno quelle che ancora non lo erano
- Le applicazioni che erano già in produzione ma che hanno effettuato degli sviluppi nuovi per poter interagire devono anche esse portate in produzione con le modifiche apportate.
- Anche il database va portato in produzione.
- Le applicazioni vengono portate in produzione in un ordine ben definito.

Ho dovuto dare supporto durante il passaggio in produzione per un minimo di due settimane lavorando sia il sabato che la domenica.

6.5 Schermate dell'applicazione in produzione

Seguono alcune immagini di una delle applicazioni in ambiente di produzione che usano i servizi sviluppati da me .

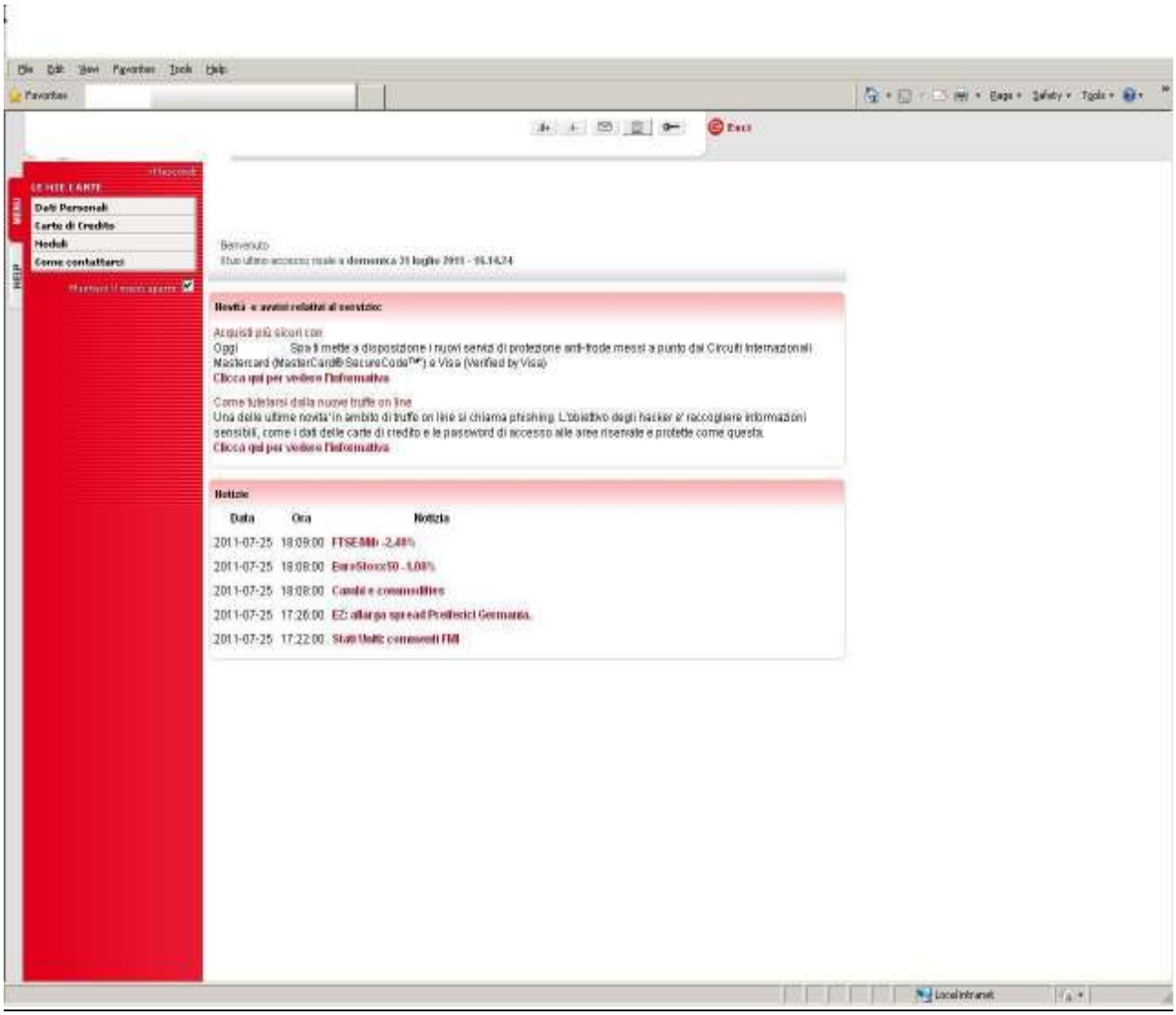

Figura 6.8 : Applicazione in produzione – Menu principale

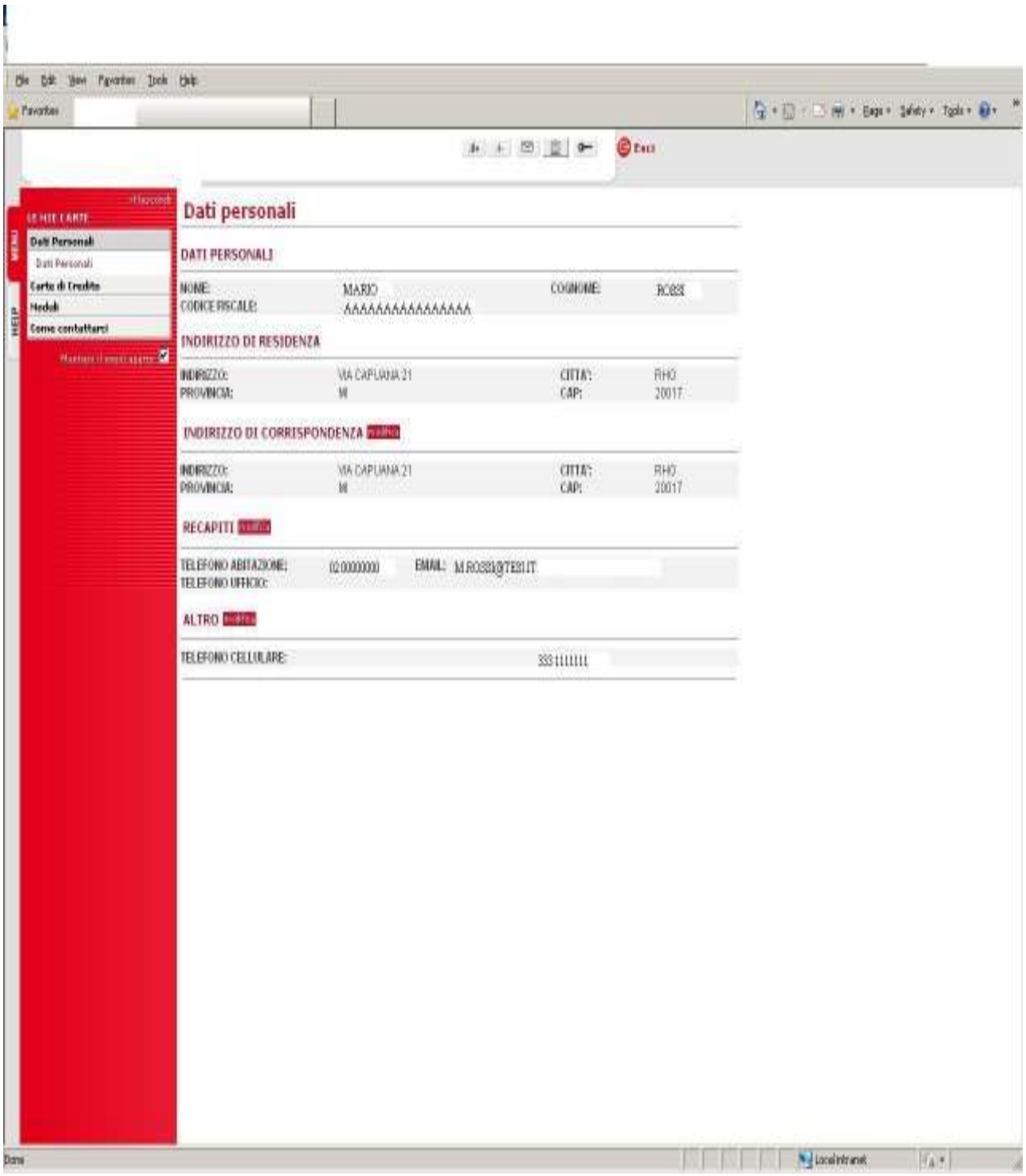

Figura 6.9 : Applicazione in produzione – Dati personali

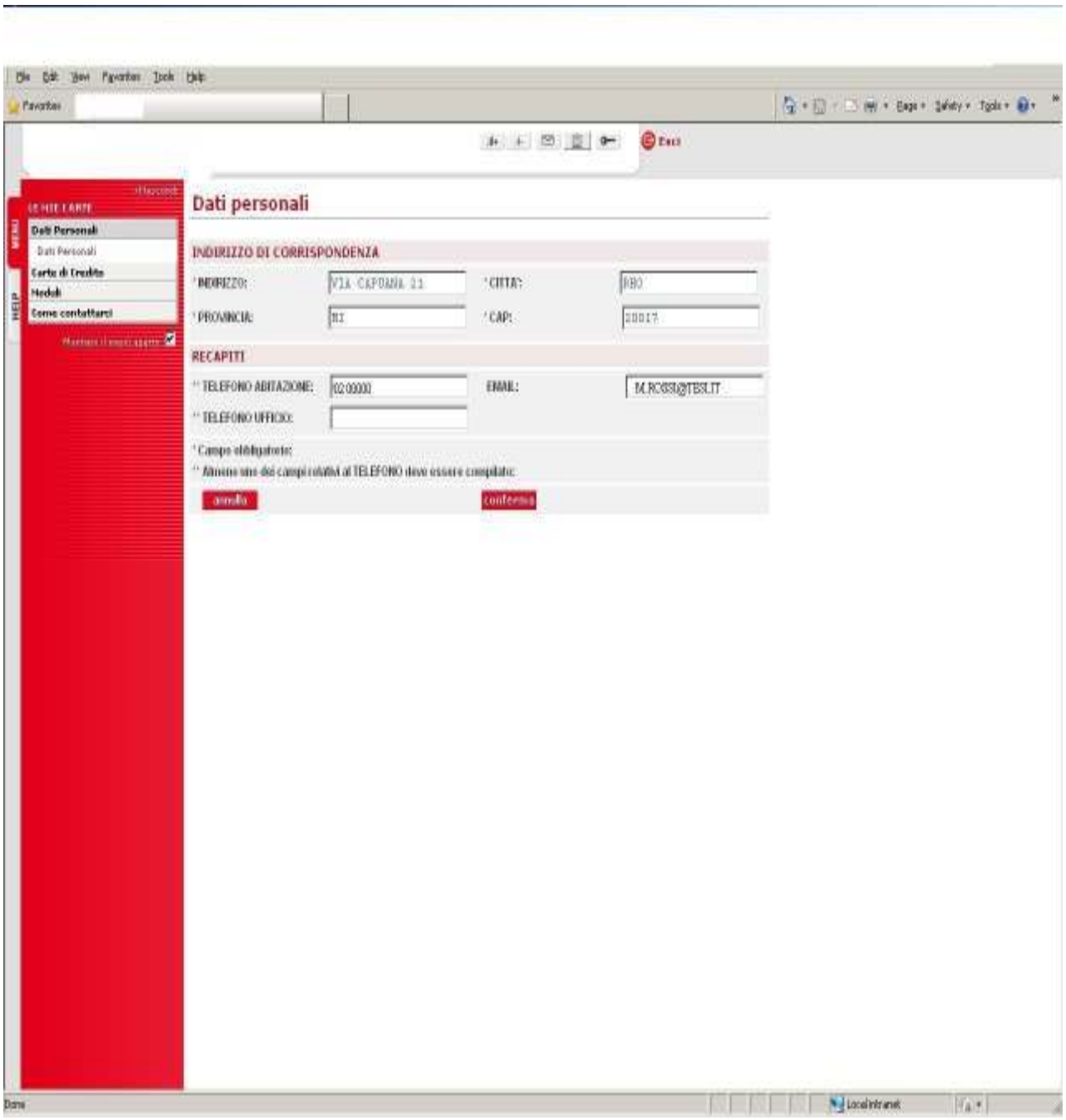

Figura 6.10 : Applicazione in produzione – Modifica dei dati personali

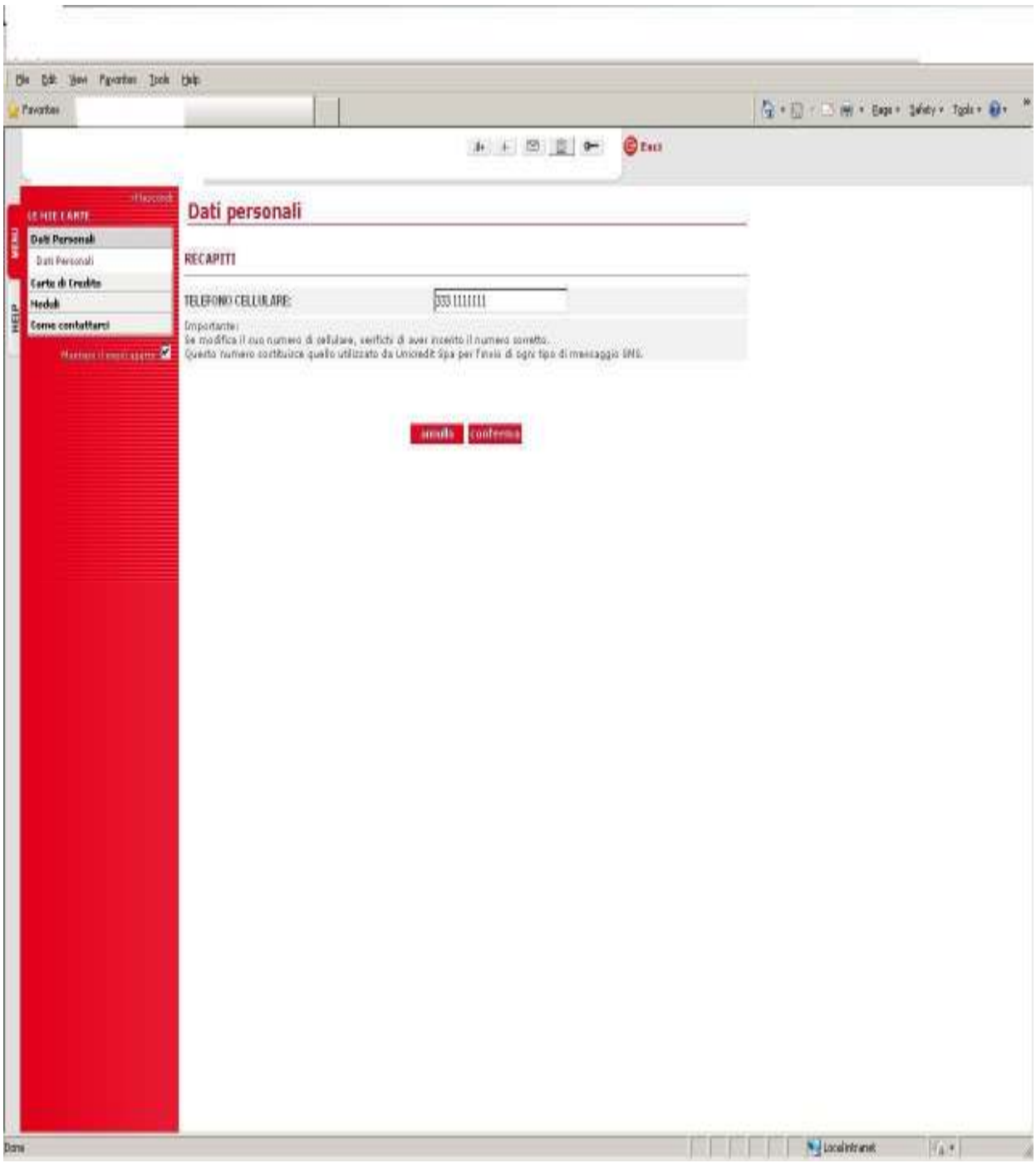

Figura 6.11 : Applicazione in produzione – Modifica del numero di cellulare

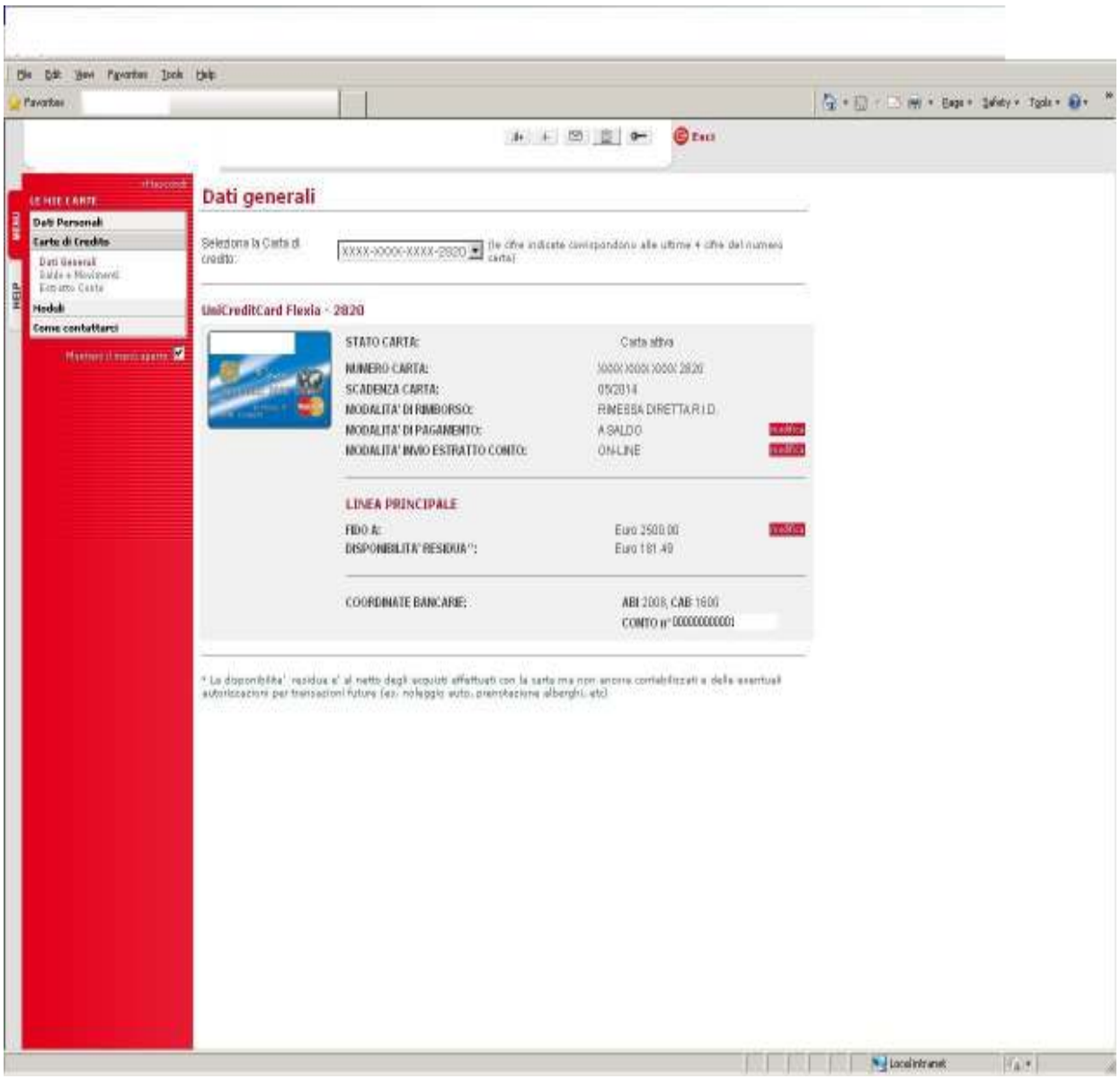

Figura 6.12 : Applicazione in produzione – Dati generali

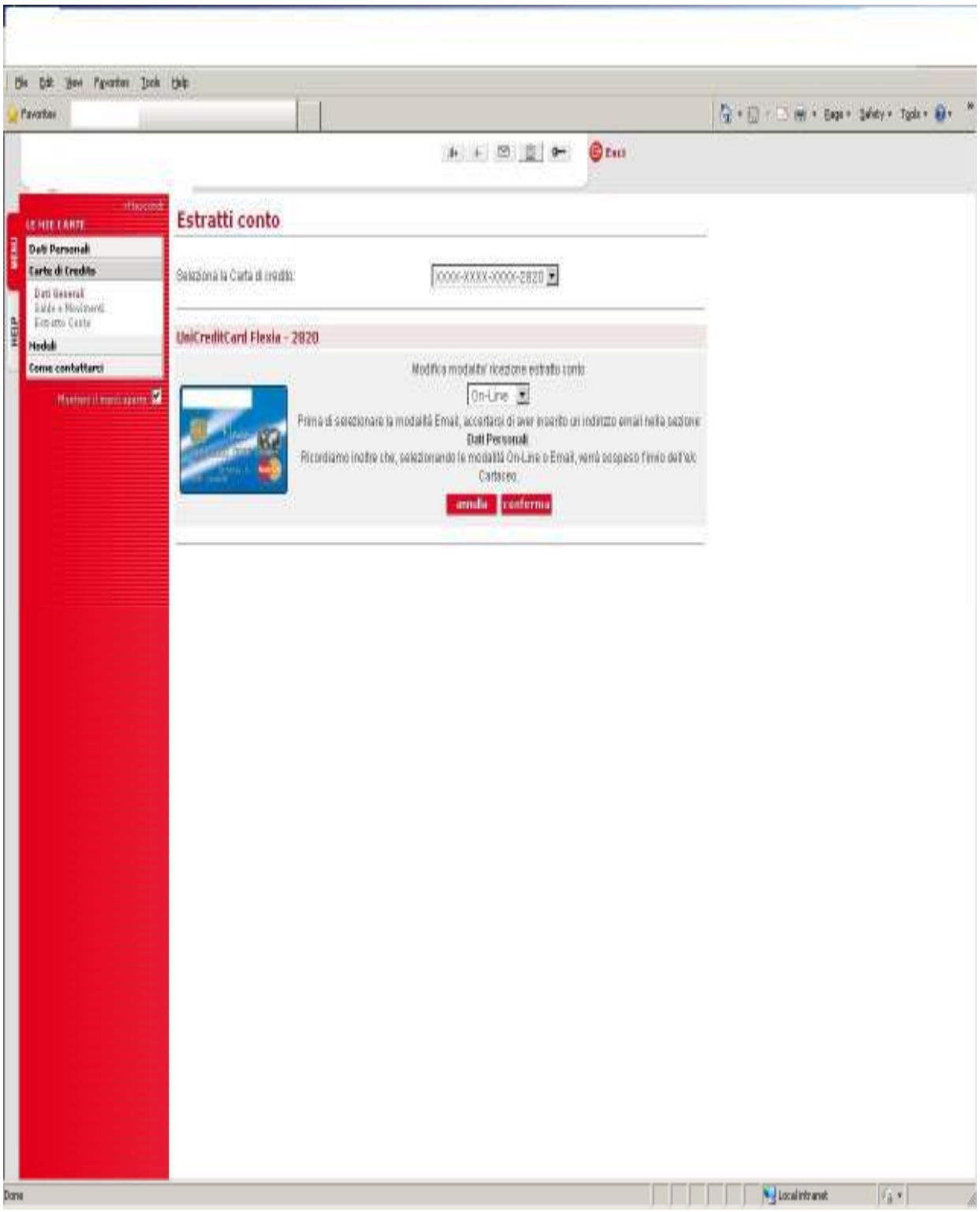

Figura 6.13 : Applicazione in produzione – Modifica modalità di ricezione estratto conto

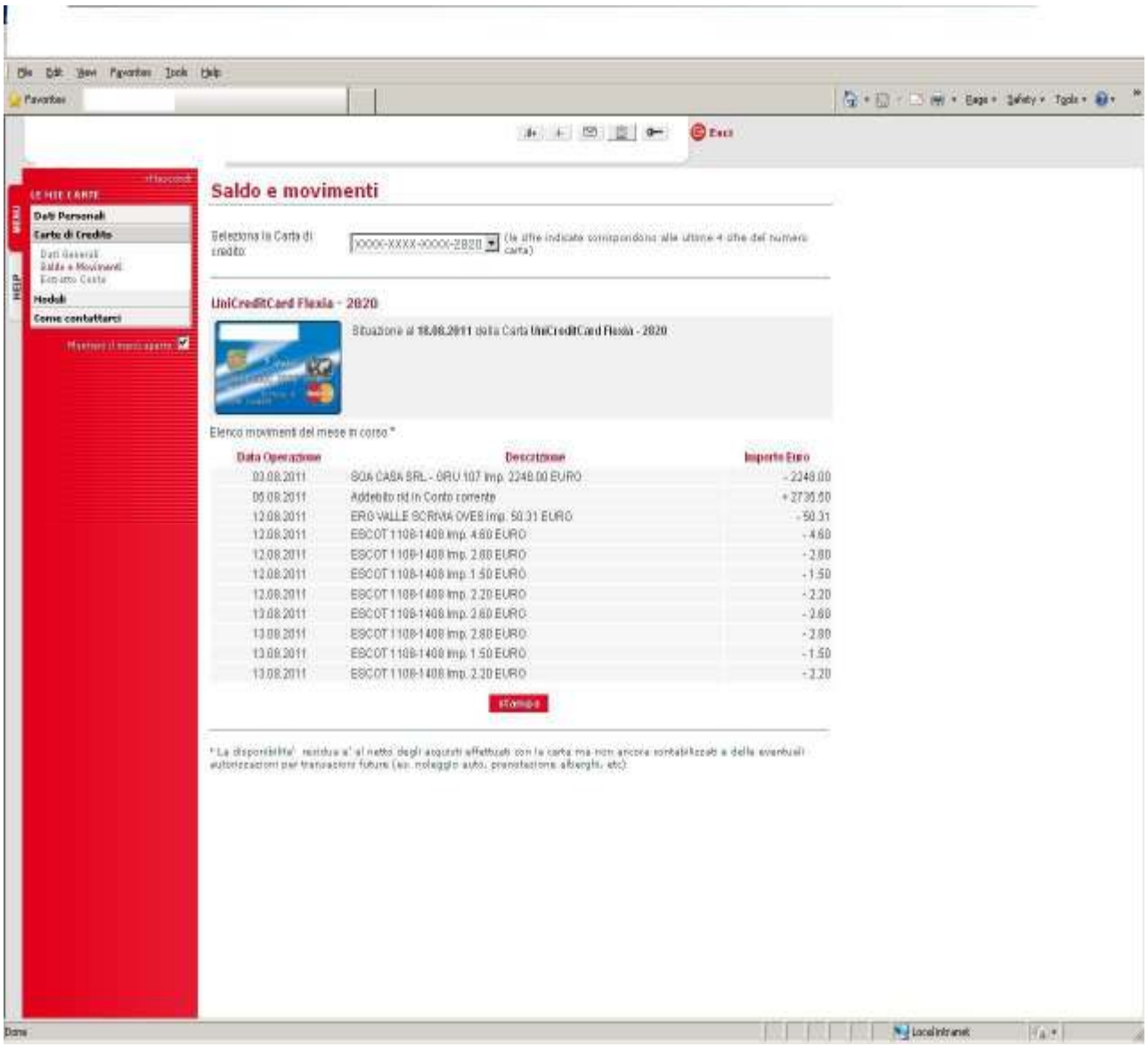

Figura 6.14 : Applicazione in produzione – Saldo e movimenti

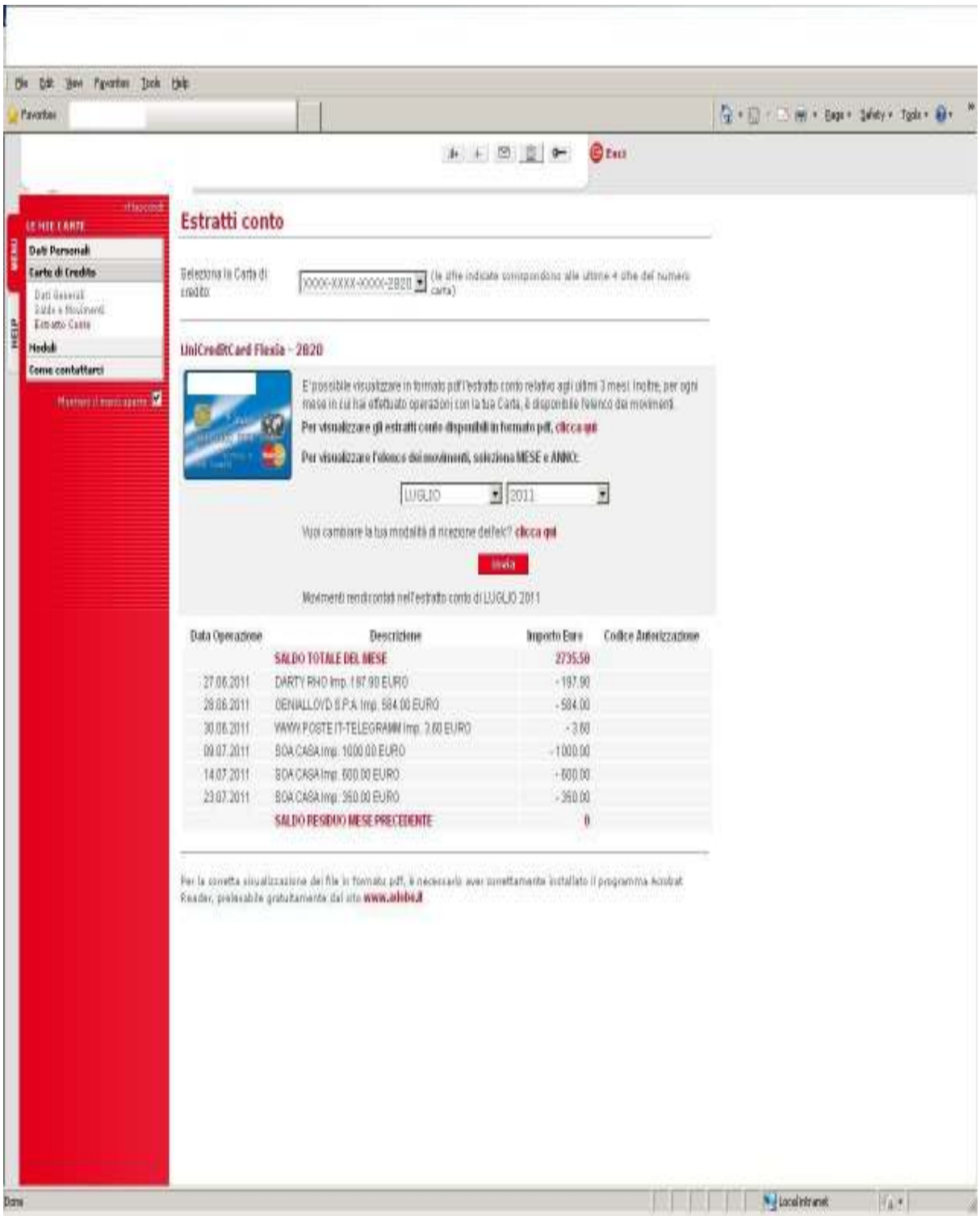

Figura 6.15 : Applicazione in produzione – Estratto conto

CAPITOLO 7

CONCLUSIONE

L'obbiettivo del presente lavoro era proporre ed implementare una soluzione tecnologica per fornire ai vari canali che la banca mette a disposizione degli utenti una fonte unica di servizi che implementano una serie di funzionalità richieste per la gestione delle carte di credito, prestiti e similari ideati per il finanziamento di acquisti di beni di consumo.

Un canale non ha più quindi la necessità di sviluppare una funzionalità propria anche se questa non è messa a disposizione degli altri canali. E' presente quindi un unico team di sviluppo responsabile dello sviluppo di servizi nuovi, della risoluzione di eventuali bug.
Abbiamo quindi un vantaggio nell'implementazione di eventuali evolutive non dovendo svolgere un lavoro n volte il numero di canali qualora si decidesse modificare l'implementazione di una funzionalità.

L'obbiettivo del lavoro era anche quello di arrivare ad un'architettura semplice con un'unica tecnologia per fornire i servizi .
Nell'architettura complessiva proposta a tre livelli , l'applicazione da noi sviluppata fa da *middleware* tra i canali messi a disposizione dei clienti e i sistemi di back-end dell'azienda .

L'applicazione è stata sviluppata secondo lo standard aziendale XFRAME e si compone di tre moduli

- Un modulo WS (web service) dove vengono implementati i servizi è anche stato sviluppato un web service *handler* che permette di loggare le chiamate SOAP in ingresso ed in uscita.
- Un modulo BF (Business framework) dove vengono sviluppati gli *entreprise java bean* che saranno richiamati dai web service per accedere ai sistemi di back-end aziendale in scrittura o lettura oppure chiamare funzionalità messe a disposizione da altre applicazioni dell'azienda sempre via *entreprise java bean*.
- Un modulo di utilità che contiene classi usate dai moduli precedenti.

E' stata usata la tecnologia dei web service per fornire servizi ai vari canali.
Questa tecnologia ha i seguenti principali vantaggi :

- Disaccoppiare il client dal server per cui l'implementazione della funzionalità può avvenire senza modifica al cliente finché l'interfaccia del servizio non cambia
- Il client può essere sviluppato usando una tecnologia diversa da quella usata per sviluppare il server.

L'applicazione dopo le varie fasi di test è stata portata in produzione e abbiamo fatto vedere una serie di screenshot in produzione per cui finora il progetto è un successo.

Questo progetto mi ha arricchito dal punto di vista professionale perché mi ha permesso di lavorare in ambito bancario in cui ci sono requisiti e tempi da rispettare.
Dal punto di vista formativo mi ha permesso :

- di approfondire certi meccanismi come quelli di sicurezza informatica soprattutto in ambito bancario dove è una cosa fondamentale,
- di approfondire lo studio di sistemi sconosciuti o poco conosciuti come AS400,
- di lavorare in ambiente in cui l'architettura informatica è abbastanza complessa
- di studiarmi un nuovo frame work anche se proprietario.

Sarebbe stato sicuramente interessante per me partecipare alla fase dei test prestazionali e stress test per maturare esperienze anche in quel campo e partecipare alla messa a punto dei meccanismi di sicurezza che è un campo per conosciuto solo al livello teorico.

Nel mondo reale un web service non offre solo servizi semplici e stateless come ad esempio il controllo di una carta di credito, ma questi possono essere combinati insieme o riusati e possono prevedere più interazioni con l'utente per offrire altri servizi. Quindi un sviluppo possibile potrebbe essere la composizione di web service singoli per poter fornire funzionalità più complesse.

BIBLIOGARFIA

SERVICE LOCATOR

http://java.sun.com/blueprints/patterns/ServiceLocator.html

http://www.mokabyte.it/2002/10/pattern_sl.htm

SAML (SECURITY ASSERTION MARKUP LANGUAGE)

http://it.wikipedia.org/wiki/Security_Assertion_Markup_Language

http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

JAX-RPC

http://www.ibm.com/developerworks/webservices/library/ws-soa-jaxrpc/

http://download.oracle.com/docs/cd/B32110_01/web.1013/b28975/appjaxrpcmapping.htm#ASAWS1200

http://www.deitel.com/newsletter/20020523/articles/JavaWebServices_JAX-RPC.pdf

http://www.developer.com/java/ejb/article.php/10931_2109561_3/Working-with-JAX-RPC.htm

http://my.safaribooksonline.com/book/programming/java/0596007345/web-services/jboss-chp-10-sect-8#X2ludGVybmFsX0ZsYXNoUmVhZGVyP3htbGlkPTA1OTYwMDczNDUvMjI1

http://www.developerfusion.com/article/84349/j2ee-and-jaxrpc-web-services/

http://download.oracle.com/javaee/1.4/tutorial/doc/JAXRPC.html

WEB-SERVICES

http://www.informit.com/guides/content.aspx?g=java&seqNum=162

http://docs.jboss.org/jbossas/jboss4guide/r2/html/ch12.html

http://www.mokabyte.it/2003/05/jws-4.htm

http://docs.jboss.org/jbossas/getting_started/v5/html/ws.html

http://www.mokabyte.it/2004/01/jws-faq.htm

WEB-SERVICE HANDLER

http://java.boot.by/wsd-guide/ch04s07.html

http://www.developer.com/services/article.php/10928_3503766_2/Processing-RequestResponse-Messages-of-a-Web-Service-Using-Handler-Chain.htm

http://www.mokabyte.it/2003/09/WSJ2EE-1.htm

http://www.javastaff.com/index.php?option=com_content&view=article&id=161

ENTREPRISE JAVA BEANS

http://www.dmi.unict.it/~tramonta/lessons/sd/ejb.pdf

http://java.html.it/guide/lezione/3427/session-bean/

http://jonas.ow2.org/JOnAS_4_7/doc/PG_Session.html

http://rollerjm.free.fr/pro/EJB_UML.html

http://www.giuseppesicari.it/articoli/session-bean-ejb/

CONTROLLO DI VERSIONE

http://it.wikipedia.org/wiki/Concurrent_Versions_System

http://www.agilejournal.com/resources/toolspotlight/1773-serena-software-dimensions-cm

http://it.wikipedia.org/wiki/Controllo_versione

TESTING

http://www.ebizq.net/topics/soa_management/features/3307.html

http://searchsoa.techtarget.com/answer/Integration-testing

http://blogs.msdn.com/b/ploeh/archive/2006/11/15/integrationtestingprinciples.aspx

http://www.analisi-disegno.com/testing/testing_glossario.htm

http://www.cs.colorado.edu/~kena/classes/7818/f06/lectures/WebTest.pdf

http://www.ing.unisannio.it/dilucca/GSSW/materiale09/testing_OO_09.pdf

http://it.wikipedia.org/wiki/Unit_testing

http://www.testinggeek.com/integration-testing

http://www.technologytransfer.it/?cis=4;1&rec=92&yy=2011&mm=3

http://www.exforsys.com/tutorials/testing/what-is-user-acceptance-testing.html

RINGRAZIAMENTI

Desidero innanzitutto ringraziare la prof.ssa Francalanci per i preziosi insegnamenti durante i due anni di laurea e per le numerose ore dedicate alla mia tesi. Inoltre , ringrazio l'ing. Veronesi per la grande disponibilità e cortesia dismostratemi , e per tutto il tempo che mi ha dedicato durante la stesura di questo lavoro.

Un sentito ringraziamento a mio papa deceduto (ti voglio bene da morire) e a mia madre per tutto quello che hanno fatto , stanno facendo e faranno per me.

Desidero ringraziare la mia meravigliosa fidanzata che mi è stata vicina sia nei momenti difficili, sia nei momenti felici (sei la mia ispirazione).

Desidero inoltre ringraziare i mie fratelli , la mia sorellina e tutti i miei amici che mi sono sempre stati vicini .