Progetto: **Piattaforma client-server per un provider di servizi mobile computing**
Proponente: **Rino Goller / Metacortex s.r.l.**

Design della piattaforma per lo sviluppo di servizi mobili

Versione **0.1**
Data 15 December 2011
Autori **Rino Goller, Maurizio Barazzuol, Katiuscia Reversi, Carlo Mirabassi, Cristiano Carlevaro.**

Abstract Capitolo che ripercorre i vari design ideati e progettati della piattaforma Metaplanet considerando nel dettaglio l'architettura ideata.

Bando 5/2009: **Interventi attraverso partnership tra imprese e organismi di ricerca per lo sviluppo della ricerca e dell'innovazione**

Legge provinciale n. 6 del 1999

Indice

INTRODUZIONE

Dalle precedenti analisi siamo riusciti a delineare una serie di requisiti generali che ci hanno permesso di arrivare ad una definizione accurata del design della piattaforma. Inoltre descriveremo una primo approccio all'architettura generale del sistema e successivamente illustreremo più in dettaglio le altre due architetture interne. L'architettura generale è del tipo Client-Server ciò sta a significare che i dispositivi mobili (Client) dialogheranno attraverso la rete mobile o internet con un Server che gestisce le connessioni in ingresso, dando come risultato l'abilitazione all'utilizzo dei nostri servizi. L'architettura Client risulta essere molto semplice data la minor potenza di calcolo dei dispositivi, spostando così la complessità computazionale a lato Server. L'architettura Server è basata su una classica struttura MVC (Model View Controller) ma notevolmente più articolata e di maggior complessità.

Riassunto requisiti di sviluppo

Vi sono alcune caratteristiche generali che vanno sempre tenute in considerazione. Fra i requisiti generali elenchiamo i seguenti:

- Cellulare o smartphone con connessione ad internet GPRS/UMTS/3G: la connessione risulta fondamentale per il funzionamento di tutto il sistema
- Gestione utenti: l'applicazione dovrà poter avere una gestione degli utenti ovvero ogni utente verrà registrato nel nostro sistema.
- Autenticazione: per accedere al nostro sistema bisognerà autenticarsi. L'utente avrà la possibilità di registrarsi tramite cellulare o anche tramite internet.
- Sistemi Operativi: data la varietà del mercato sono da considerare tutti i sistemi operativi ora in commercio per cellulari e smartphone anche se verranno presi in considerazione prima gli ambienti di sviluppo open-source.
- Server: alla base del sistema ci sarà un server centrale presso la Metacortex s.r.l che dialogherà con i client. Il linguaggio di programmazione scelto per l'applicativo lato server è Java.
- Protocollo comunicazione: la comunicazione avverrà tramite HTTP e inoltre il server comunicherà con il Client tramite l'utilizzo dello standard JSON.
- Sicurezza: per i dati sensibili vi sarà un apposito sistema di cifratura.
- Espandibilità: deve risultare facile creare dei servizi per la nostra applicazione per chiunque voglia farlo.

- Comprensione: il metodo e il metalinguaggio dai noi creato deve essere di facile comprensione ed utilizzo.

Approccio generale e architettura

L'architettura generale del sistema si basa sull'idea che i nostri servizi siano accessibili attraverso la rete internet da più devices. Come mostrato in figura 1 i devices(il nostro interesse sarà concentrato sulla categoria Mobile Phone e Pad) possono collegarsi tramite la rete internet al nostro WebServer che permette l'accesso alla nostra piattaforma Metaplanet.

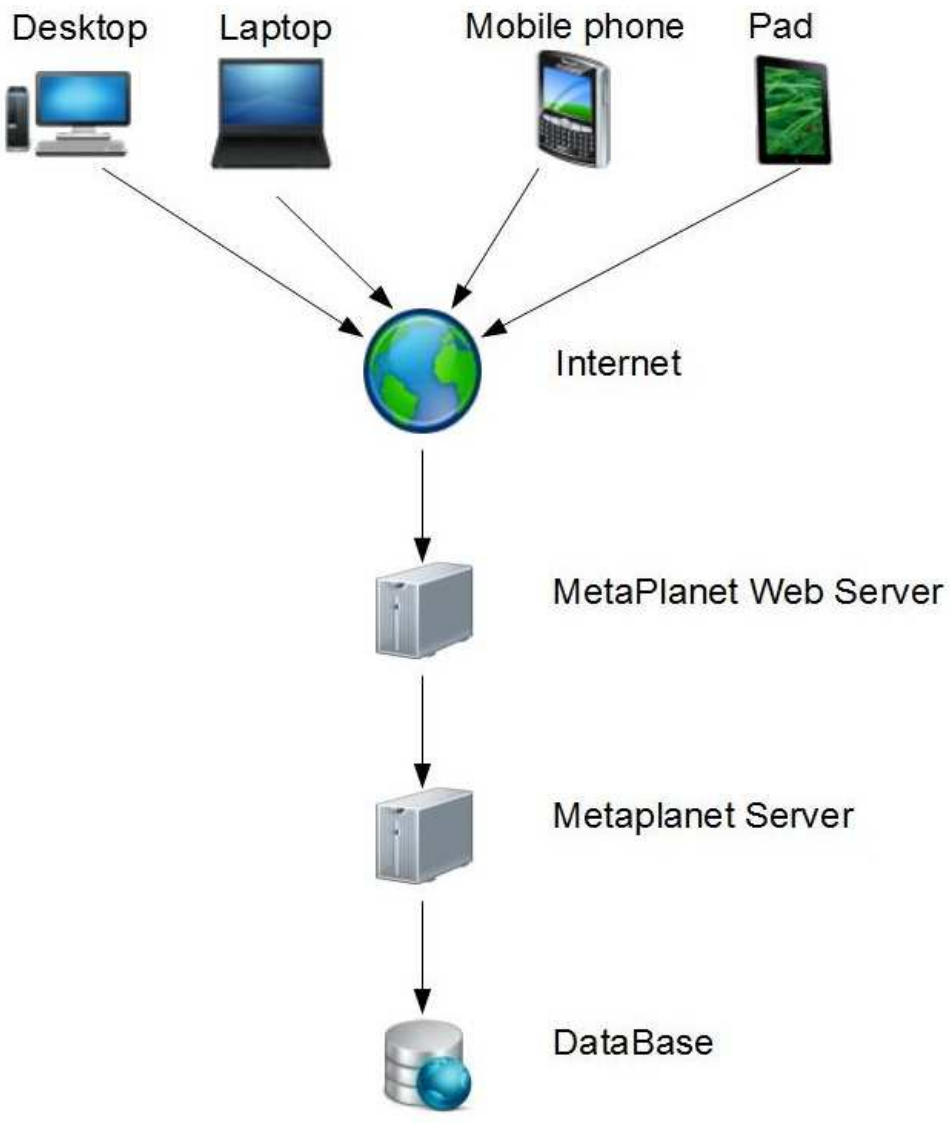
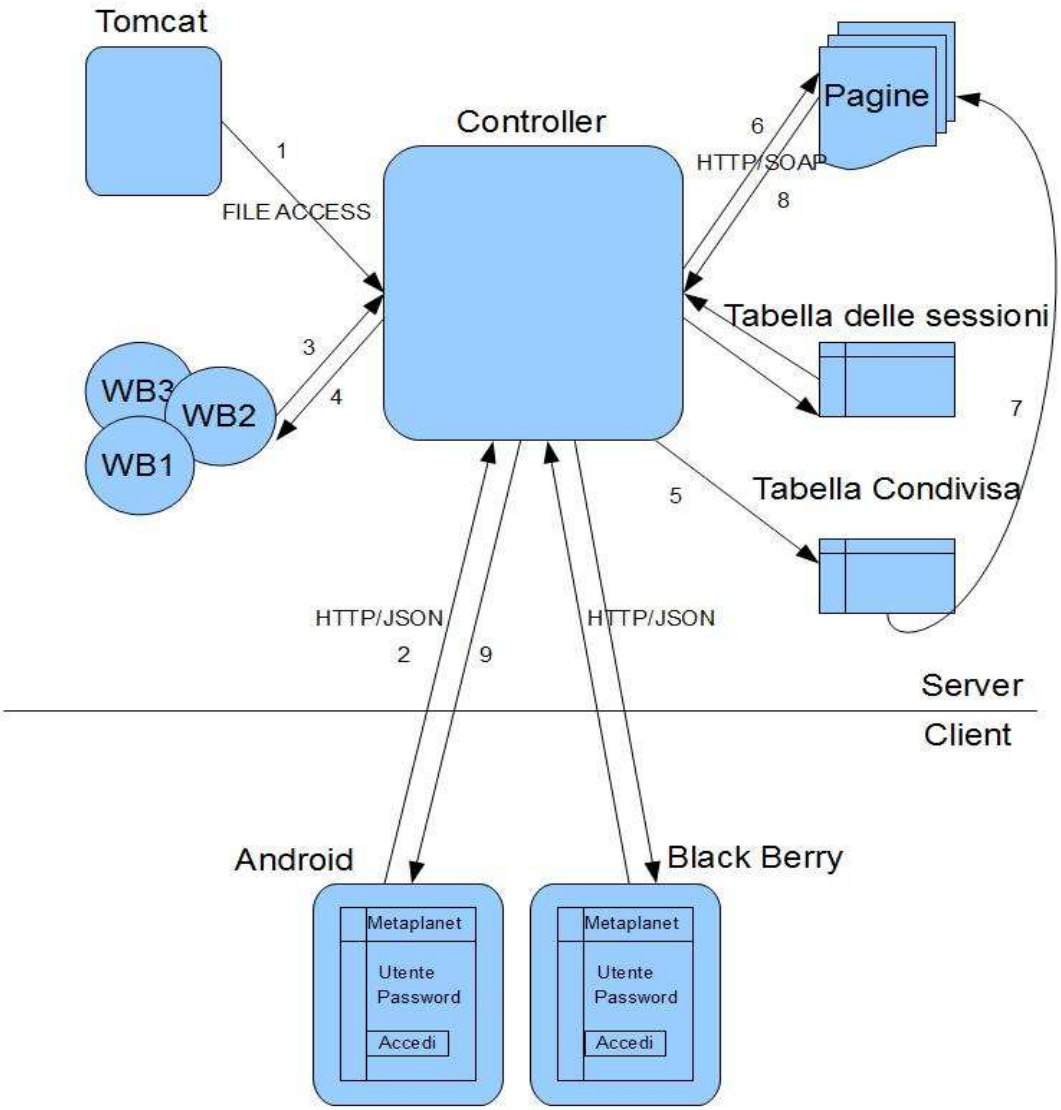


Illustrazione 1: Architettura generale Metaplanet



Nella figura riportata qui sopra abbiamo voluto illustrare l'architettura del sistema e la sequenza di azioni che vengono svolte dai singoli moduli del sistema per la richiesta e successivamente l'erogazione del servizio.
Lo schema proposto ha uno spiccato riferimento al modello MVC.
MVC è l'acronimo di Model View Controller ed è un pattern architetturale molto diffuso nello sviluppo di interfacce grafiche di sistemi software object-oriented. Originariamente impiegato dal linguaggio Smalltalk, il pattern è stato

esplicitamente o implicitamente sposato da numerose tecnologie moderne, come framework basati su PHP (Symfony, Zend Framework, CakePHP), su Ruby (Ruby on Rails), su Python (Django, TurboGears, Pylons, Web2py), su Java (Swing, JSF e Struts), su Objective C o su .NET.
A causa della crescente diffusione di tecnologie basate su MVC nel contesto di framework o piattaforma middleware per applicazioni Web, l'espressione framework MVC o sistema MVC sta entrando nell'uso anche per indicare specificamente questa categoria di sistemi (che comprende per esempio Ruby on Rails, Struts, Spring, Tapestry e Catalyst).

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- il model fornisce i metodi per accedere ai dati utili all'applicazione;
- il view visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- il controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti

Questo schema, fra l'altro, implica anche la tradizionale separazione fra la logica applicativa (in questo contesto spesso chiamata "logica di business"), a carico del controller e del model, e l'interfaccia utente a carico del view.
I dettagli delle interazioni fra questi tre oggetti software dipendono molto dalle tecnologie usate (linguaggio di programmazione, eventuali librerie, middleware e via dicendo) e dal tipo di applicazione (per esempio se si tratta di un'applicazione web, o di un'applicazione desktop). Quasi sempre la relazione fra view e model è descrivibile anche come istanza del pattern Observer. A volte, quando è necessario cambiare il comportamento standard dell'applicazione a seconda delle circostanze, il controller implementa anche il pattern Strategy.

Un possibile caso d'uso attraverso la nostra piattaforma potrebbe essere quello definito in figura n.x dai numeri accanto alle frecce di comunicazione.
Per prima cosa l'utente farà richiesta al Web Server Tomcat di accedere alla piattaforma Metaplanet dalla suo device (Android, BlackBerry, etc..).
Il web server dialogherà con il controller dell'applicazione server che invierà la pagina
di autenticazione. La pagina verrà richiesta tramite web service ad un altro server o presa direttamente dal server locale. Le info richieste verranno salvate
in una tabella condivisa sul nostro Database MySql. Quindi verranno inviate le pagine da visualizzare al controlle che le smisterà al Client. Il Client visualizzerà la pagina di autenticazione.

Una volta compilata con i dati personali i dati verrano inviati al server che registrerà l'utente e la sessione nella tabella condivisa e nella tabella dei log. Nella tabella delle sessioni verrano salvate
tutte le sessioni esistenti e tutto il percorso eseguito dall'utente.

Tomcat

È il web server sul quale gira il sistema. La scelta di tomcat deriva essenzialemente dal fatto che questo web Conteiner implementa la specifiche JavaServer Pages (JSP) e Servlet di Sun Microsystems e di conseguenza è una perfetta piattaforma per l'esecuzione di applicazioni web scritte in Java.

Controller metaplanet

È il core della piattaforma. Di qui passa ogni tipo di informazione sul client connesso, e da qui arrivano le istruzioni su come l'utente può navigare all'interno dell'applicazione. Il controller dirige il traffico dati e sa cosa può e cosa non può fare l'utente.

DB

È l'archivio principale della piattaforma. Al suo interno troviamo delle sezioni che contengono informazioni sui profili utente e sulle attività del client. Le sezioni del Database sono la tabella delle sessioni e la tabella condivisa.

JSP Collection

Si tratta essenzialmente dell'insieme delle pagine che contengono la logica e il codice con il quale sono stati creati i servizi. Ogni JSP contiene il codice relativo ad un servizio.

Web services

Si tratta di tutti quei servizi web che vengono sviluppati da terzi e che vengono poi integrati nella piattaforma Metaplanet.

Client

Sono i dispositivi mobili sui quali si potranno visualizzare e utilizzare i nostri servizi.
Principalmente si tratterà di Pad e Smartphone, ma l'accesso a Metaplanet non è prettamente riservato a queste due tipologie di dispositivo. È possibile infatti connettersi alla piattaforma anche da computer desktop o laptop.

Di seguito mostriamo in dettaglio l'architettura del sistema con i suoi componenti principali.

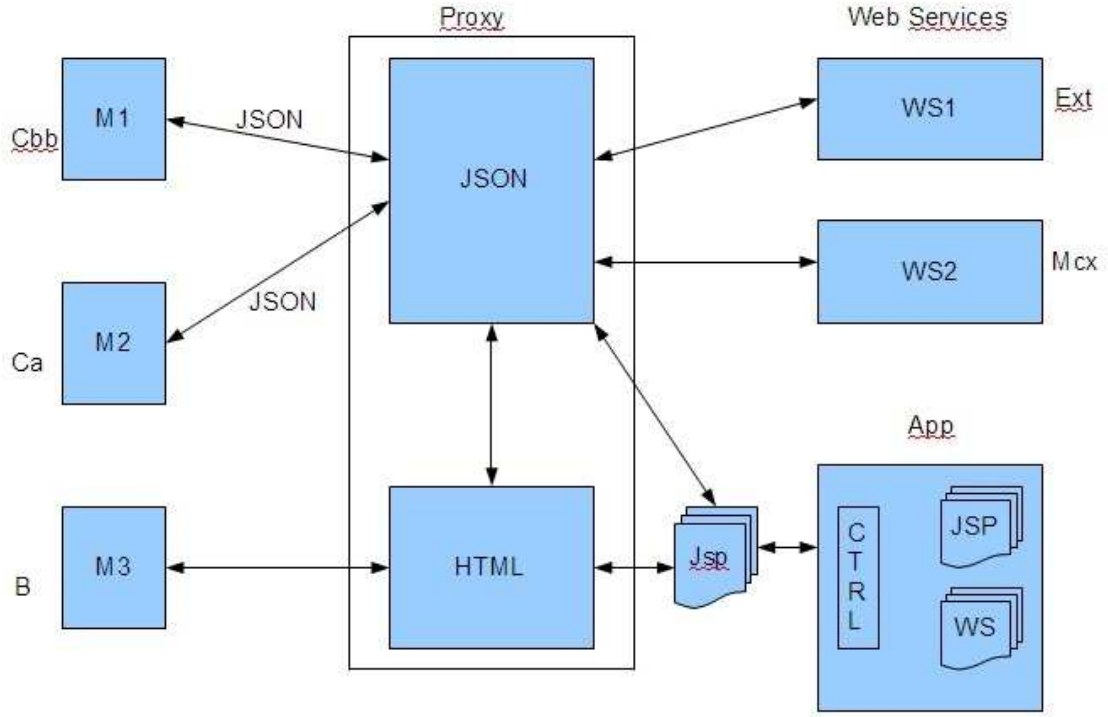


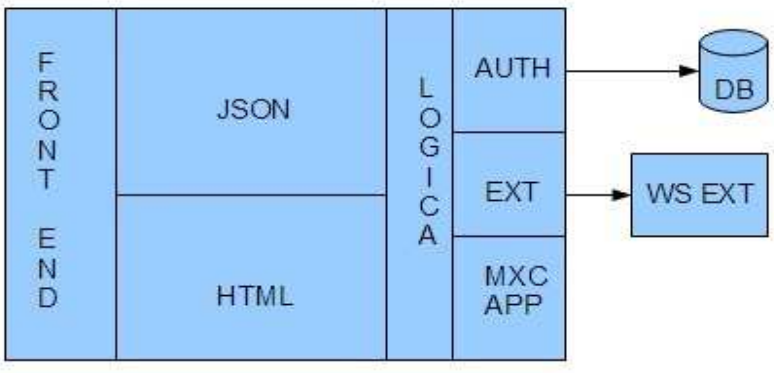
Illustrazione 2: Architettura generale Metaplanet

I dispositivi mobili attraverso il canale internet, sfruttando l'impacchettamento dei dati in formato Json, comunicano con il Server che decodificando il messaggio sarà in grado di fornire attraverso la nostra piattaforma Metaplanet i servizi richiesti. Non è necessario però che i servizi stiano direttamente sul nostro server poiché tramite Web Services è possibile utilizzarli anche se allocati su macchine differenti. Un normale device quale per esempio un Laptop comunica tramite internet, senza l'ausilio del impacchettamento Json, direttamente con il Server attraverso il browser.

Ambiente di esecuzione lato server

In figura vengono mostrati i vari moduli che implementano il server:

- FRONT END: è l'entry point del sistema. Chiunque voglia accedere ai servizi messi a disposizione deve necessariamente passare da qui. Il Front End si presenta all'utente con la pagina di login, all'interno della quale inserire le proprie credenziali per potersi collegare al proprio profilo col quale usufruire dei servizi.
- JSON: è il modulo che si occupa di codificare e decodificare i pacchetti inviati e ricevuti dal dispositivo utente.
- HTML: è il modulo che si occupa di definire il linguaggio con il quale vengono passate le schermate nel caso in cui l'accesso ai servizi avvenga tramite browser.
- LOGICA: è il core del sistema. Qui risiede tutta la logica che permette una corretta navigazione all'interno del sistema e delle singole applicazioni. In questo modulo troviamo tutto il codice java atto a "tenere in piedi" tutto il sistema e che, per motivi di sicurezza, non deve essere accessibile in nessun caso.
- AUTH: questo blocco si occupa dell'autenticazione degli utenti. È in diretto collegamento con l'entry point che visualizza la schermata di login del sistema.
- DB: è il database all'interno del quale vi sono memorizzate le informazioni di tutti gli utilizzatori di MetaPlanet. È sempre in questo database che avviene il controllo utente/password valido per l'autenticazione al sistema.
- EXT: è il modulo che si occupa della connessione tra il server Metacortex e i Web Services esterni progettati e realizzati da terzi.
- WS EXT: fa riferimento ad un modulo esterno all'architettura MetaPlanet che però contiene dei WebServices compatibili con la nostra piattaforma.
- MCX APP: è la parte del server che racchiude tutti i servizi che Metacortex mette a disposizione.

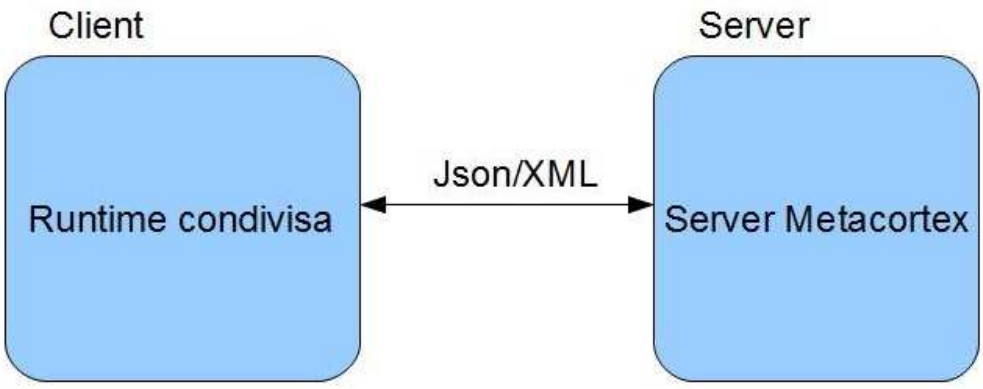


Ambiente di esecuzione lato client

Per dovere di completezza abbiamo scelto di esporre i modelli di architettura lato client che erano stati valutati fin da principio in maniera tale da poter illustrare il percorso fatto e i problemi riscontrati prima di arrivare alla soluzione definitiva.
Per fornire un'idea di massima, l'evoluzione della progettazione del client si può distinguere in 3 fasi, corrispondenti alle 3 versioni progettate.

MetaPlanet Client 1.0

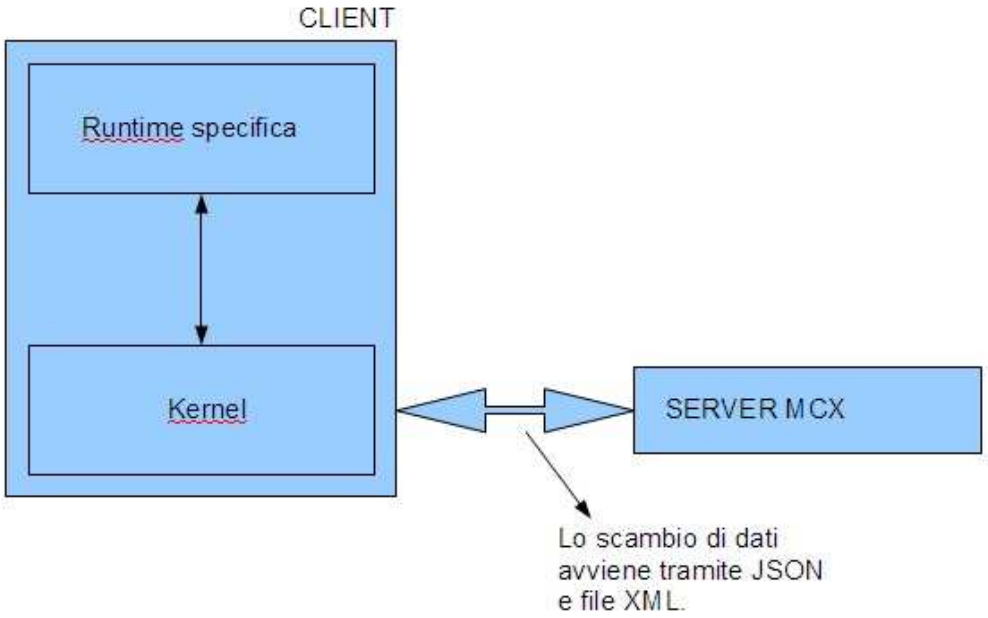
Siamo partiti da un client relativamente semplice con l'idea di renderlo compatibile per qualsiasi dispositivo. Dopo aver creato e testato questa prima versione sull'emulatore presente nell'ambiente di sviluppo siamo passati alla prova sul dispositivo vero e proprio.
I problemi che si sono presentati erano tutti a livello di compatibilità tra il codice sviluppato e il tipo di codice che invece poteva essere supportato dai dispositivi.
Infatti da una piattaforma all'altra vi è stata una ridefinizione delle classi Java che non ha permesso di rendere standard il codice sviluppato per il client.
In figura è possibile vedere come il client sia composto dal singolo componente identificato dalla runtime.
Questo è per l'appunto il nodo centrale del client che contiene tutto il codice che permette al nostro applicativo di connettersi ad internet e di dialogare tramite JSON e XML con il server.



MetaPlanet Client 2.0

Visti i problemi di compatibilità tra il codice scritto e quello supportato ci siamo scostati leggermente dal modello iniziale e abbiamo pensato di procedere in questo modo: realizzare con delle classi base un kernel di base valido per tutti i dispositivi e poi su questo costruire una runtime specifica per ogni tipologia di dispositivo.
In questa maniera si poteva ancora aspirare alla scalabilità del kernel su più dispositivi e poi creare un applicazione ad hoc per ogni dispositivo che permettesse di integrare nell'applicativo anche icone e pulsanti tipici della marca del produttore del sistema operativo. L'immagine sottostante illustra per l'appunto come il client sia suddiviso in due moduli:

- Kernel: è l'insieme di classi java base che sono valide per ogni client.
- Runtime specifica: è l'insieme di classi che invece fornisono elementi più dettagliati e di carattere grafico che permettono di abbellire l'interfaccia utente.



MetaPlanet Client 3.0

La terza versione del client invece è stata modificata ulteriormente.
Abbiamo notato che nonostante le classi java usate per il kernel nella versione 2.0 fossero quelle di base erano diverse in base al sistema operativo sulle quali si cercavano.
Nella fattispecie sia Blackberry che Android hanno rilasciato versioni della loro SDK che non presentano somiglianza alcuna con le classi base del linguaggio Java.
Infatti vengono chiamate diversamente e soprattutto hanno al loro interno elementi diversi, nonostante siano state implementate per un determinato tipo operazioni.
Si tratta di una runtime a sé stante per ogni dispositivo. Sempre scritta in Java e sempre compatibile con il server progettato.

Questa applicazione, una volta installata, all'avvio si connette al server col quale dialoga e al quale chiede di volta in volta, e in base alla scelta dell'utente la pagina successiva da visualizzare.

L'utente deve ovviamente essere registrato. In caso contrario non sarà abilitato al download dell'applicativo né quanto meno all'utilizzo dei servizi offerti dalla nostra piattaforma.

MetaPlanet Client 4.0

La quarta e ultima versione del nostro client è quasi completamente distaccata da quella che era l'idea principale e con la quale siamo partiti per realizzare questo progetto.
Dopo esserci scontrati con svariate problematiche a livello software a causa dei limiti imposti dai marchi proprietari e dopo aver analizzato la diffusione sempre maggiore di smartphone con sistemi quali Blackberry, Android e soprattutto iOS, abbiamo pensato che fosse doveroso supportare anche quei sistemi per i quali i brand proprietari hanno imposto parecchi blocchi a livello software.
Per questo per poter garantire massima compatilità, massima scalabilità, massima facilità di accesso ai servizi e massima facilità di utilizzo abbiamo modificato ulteriormente il client rendendolo un client web.
In questo modo qualsiasi dispositivo disponga di un browser internet all'interno del quale è abilitato il linguaggio javascript potrà connettersi e accedere ai servizi MetaPlanet, senza la necessità di scaricare ex novo un'applicazione che comporta una certa dimestichezza nella configurazione del dispositivo qualora esso non sia predisposto per connettersi alla rete.
L'utente apre il browser ed ha accesso, dopo l'autenticazione, ai servizi offerti da MetaPlanet.

Idee per l'implementazione

In fase di design e progettazione sono stati ipotizzati diversi modelli per la realizzazione del sistema. La vasta portata del progetto ha ovviamente comportato tante modifiche in itinere causate da problemi di incompatibilità che si sono verificati durante lo sviluppo.
Il design del sistema ad alto livello è abbastanza essenziale ed intuitivo: il dispositivo mobile si connette tramite l'applicativo al server Metacortex.

Android

L'architettura di Android

Android ha un'architettura che comprende tutto lo stack degli strumenti per la creazione di applicazioni mobili di ultima generazione. Tra questi strumenti troviamo un sistema operativo, un insieme di librerie native per le funzionalità core della piattaforma, un'implementazione della VM e un insieme di librerie Java. Gli sviluppatori hanno a disposizione un'architettura a layer, dove i livelli inferiori offrono servizi ai livelli superiori, offrendo un più alto grado di astrazione. Esamineremo tutti i componenti di questa architettura, facendo riferimento all'immagine che rappresenta l'architettura di Android.

Il kernel di Linux

Il layer di più basso livello e rappresentato dal kernel Linux nella versione 2.6. La necessita iniziale era quella di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante, attraverso la definizione di diversi driver. Possiamo quindi notare la presenza di driver per le gestione delle periferiche multimediali, del display, della connessione Wi-Fi, dell'alimentazione e della ash memory.

Conclusione

A conti fatti si è arrivati -grazie anche ad un'analisi dettagliata e approfondita degli scenari- a capire quale fosse il miglior approccio per creare un'applicazione che fosse utile e che non comportasse per l'utente più complicazioni che benefici.
Gli studi effettuati hanno permesso di arrivare a disegnare un applicazione calibrata sull'utente meno esperto in maniera tale da non limitare l'utilizzo ai soli utenti con esperienza di browsing e installazione applicazioni elevata.

Il client, semplice ed intuitivo, garantisce un accesso ai servizi rapido senza trascurare la sicurezza e la protezione dei dati sensibili quali username, password, numeri di conto corrente e/o carte di credito.
Il server invece contenendo al suo interno la complessità e la logica di navigazione tra le applicazioni e all'interno delle applicazioni è il punto nevralgico del sistema. Scendendo nel dettaglio possiamo notare come in realtà il server comprenda al suo interno dei moduli i cui ruoli sono ben definiti.
Abbiamo ideato -sulla base dello standard MVC- una parte di controller che fosse in grado di gestire l'entry point del nostro sistema, cioè che fosse in grado di discernere tra utenti registrati e non, che fosse in grado di capire quali utenti possono accedere a quali servizi e che fosse in grado di guidare l'utente all'interno dell'applicazione senza possibilità di generare errori.
Parallelamente al controller è stato studiato un ambiente che fungesse da registro nel quale andare a catalogare tutti gli accessi al sistema in maniera tale da garantire sicurezza in più per gli utenti.
Questo modulo del sistema è stato realizzato tramite una tabella nel DB collegato al sistema che viene automaticamente alimentata dagli spostamenti che fa un utente attraverso il sistema.
La mentalità open source e il voler condividere questa piattaforma ci ha permesso di tenere in considerazione un aspetto importante: lo sviluppo di servizi da parte di terzi. La problematica principale venuta alla luce è stata la seguente: come possiamo fare in modo che uno sviluppatore “esterno” a Metacortex sia in grado di realizzare un servizio e poi condividerlo?
Ebbene la soluzione sta tutta nel fatto che il modulo di controller è in grado di accedere ad eventuali servizi esterni in maniera da poter ampliare e includere nella rosa di servizi offerti anche quelli sviluppati da terzi.
Successivamente però si è presentato un problema in ambito sicurezza.
Infatti è stato necessario considerare l'intenzione “malevola” di chi sviluppa un servizio per la nostra piattaforma. Infatti dare la possibilità a chiunque di fare il deploy sul server Metacortex di un qualsiasi servizio potrebbe comportare, nel caso in cui non venisse fatta un'analisi ben dettagliata del codice, un disservizio su tutta la rete di servizi.
Ragion per cui il sistema è pensato per ospitare solo i file di navigazione xml mentre la jsp collection nella quale sono presenti i comandi veri e propri può rimanere sul server di chi sviluppa, azzerando quindi la possibilità che vi siano parti di codice dannose opportunamente mascherate.
Concludiamo con un'ultima considerazione in merito al percorso effettuato per arrivare al design definitivo della nostra piattaforma.
Siamo partiti con l'idea di realizzare un sistema in grado di offrire dei servizi ad utenti che disponessero di uno smartphone. Il design da zero di una piattaforma di questo tipo ha richiesto molte prove e molti cambi in itinere, che però ci hanno portato ad ottenere una piattaforma in grado di erogare servizi su dispositivi mobili. Inizialmente i problemi legati alle differenze tra

sistemi operativi mobili avevano minato la possibilità di poter realizzare questo progetto, ma con l'ultimo client l'obiettivo è stato raggiunto.
Le diversità tra i sistemi operativi non sono più un fattore da considerare, poiché la piattaforma è utilizzabile anche da quei dispositivi che prima non pensavamo nemmeno di supportare, iPhone compreso.