Consiglio Nazionale delle Ricerche

Meta ψ:uno strumento di web-metacomputing per realizzare Problem Solving Environments

Ranieri Baraglia, Giancarlo Bartoli, Domenico Laforenza

Technical Report
CNUCE-B4-2001-011

CNUCE

Pisa

Metaψ: uno strumento di web-metacomputing per realizzare Problem Solving Environments

Ranieri Baraglia, Giancarlo Bartoli, Domenico Laforenza
CNUCE - Istituto del Consiglio Nazionale delle Ricerche
Via Moruzzi 1, loc. S. Cataldo 56100 Pisa (Italy)
e-mail: (Ranieri.Baraglia, Giancarlo.Bartoli, Domenico.Laforenza)@cnuce.cnr.it

Abstract

Questo rapporto descrive gli obiettivi di progetto e le caratteristiche di funzionamento di *Meta*ψ per la realizzazione di Problem Solving Environments in ambiente web.

Metaψ è stato implementato utilizzando il linguaggio *Java*^{TM} di *Sun Microsystem*, in modo da garantire la sua portabilità sul maggior numero possibile di macchine.

1 Introduzione

Esistono problemi scientifici e commerciali caratterizzati da un'elevata complessità e richiedenti elevate capacità computazionali. Per una loro soluzione è necessario utilizzare un insieme di risorse computazionali eterogenee, interconnesse da una rete di comunicazione. Questo insieme di risorse, coordinate da una struttura software adeguata (*middleware*), può essere visto come un'unica macchina virtuale parallela a memoria distribuita.

Le problematiche emerse dalla gestione di una siffatta macchina virtuale hanno portato alla definizione di un nuovo paradigma computazionale: il *metacomputing* [SC92]. Questo approccio offre notevoli vantaggi rispetto ai tradizionali ambienti di calcolo parallelo e calcolo distribuito, quali, ad esempio: potenza di calcolo derivante dall'aggregazione delle risorse computazionali costituenti il metacomputer; espandibilità della configurazione (il numero e il tipo delle macchine può variare a seconda delle specifiche necessità dell'applicazione e della disponibilità delle risorse), specializzazione dei moduli (ciascun modulo dell'applicazione può avvalersi di una macchina dedicata ad esso più "affine"). Pertanto, dovendo eseguire su un supercomputer un'applicazione parallela che consta di porzioni di codice di tipo diverso (SIMD, MIMD, ecc.), la maggior parte del tempo verrebbe speso nell'esecuzione di codice non ottimizzato per l'architettura usata [Freu89]. Per contro, la disponibilità di un insieme di risorse eterogenee, rende possibile l'esecuzione di un modulo sulla macchina ad esso più affine.

Le risorse collegate in rete permettono di accedere a strumenti di elaborazione dislocati in vari siti, rendendo meno importante il luogo in cui il ricercatore svolge fisicamente la propria attività.

Un *metacomputer* è composto da un insieme di macchine, in generale eterogenee, interconnesse da una rete ad alte prestazioni e relative interfacce nodi-rete, da protocolli di comunicazione, sistemi operativi e ambienti di sviluppo delle applicazioni. A causa dei ritardi introdotti dalle comunicazioni inter-nodo, tale paradigma computazionale non è adatto per lo sviluppo di applicazioni con parallelismo *fine-grain* e *medium-grain*.

Inizialmente le applicazioni candidate per essere eseguite da un metacomputer erano alcune delle *Grand Challenges*, ovvero problemi di interesse scientifico di elevata complessità, che richedono ingenti risorse computazionali. In seguito si è constatato che anche altre tipologie di problemi si adattano a questo paradigma computazionale.

Alcuni dei tipici problemi presi in considerazione dai ricercatori sono:

- dinamica molecolare e stellare;
- data mining;
- progettazione aerospaziale;
- visualizzazione interattiva 3D di grandi *data sets*;
- elaborazione di modelli climatici su scala globale;
- sistemi di gestione del territorio;
- ecc.

I metacomputer hanno molte caratteristiche in comune con i sistemi paralleli e quelli distribuiti, ma differiscono da questi in vari aspetti. Come per un sistema distribuito, un metacalcolatore deve integrare un variegato insieme di risorse interconnesse da reti (potenzialmente inaffidabili) e spesso localizzate in domini amministrativi diversi. Come per le applicazioni parallele, quelle per i metacomputer devono gestire accuratamente le comunicazioni per non degradare le proprie prestazioni. Tuttavia la natura eterogenea e la disponibilità mutevole delle risorse in un metacomputer limita l'applicabilità degli strumenti e delle tecniche sviluppate per i modelli paralleli e distribuiti, pertanto queste devono essere riviste per trattare adeguatamente le seguenti problematiche [GW96][FK97]:

- facilità d'uso;
- servizio di informazione;
- gestione delle risorse;
- predizione delle prestazioni;
- sicurezza;

- tolleranza ai guasti;
- scalabilità;
- interoperabilità;
- communicazione e accesso ai dati.

La crescita di Internet che si è avuta negli ultimi anni, basti solo pensare ai 93 milioni di calcolatori interconnessi nel luglio 2000 [ISC], ha portato a considerare il World Wide Web come una potenziale risorsa per supportare il metacomputing. La costruzione di un metacalcolatore partendo da questa risorsa inerentemente distribuita, eterogenea e potenzialmente inaffidabile, presenta problematiche diverse da quelle tipiche delle reti locali e costituisce un'evoluzione degli strumenti "classici" di metacomputing basati tipicamente sull'uso di strumenti quali PVM e MPI.

L'introduzione del linguaggio Java, che permette di scrivere codice indipendente dall'architettura, ha dato vita a una serie di progetti che cercano di realizzare un sistema per metacomputing "economico" che riesca a sfruttare la potenza computazionale di una macchina virtuale che potenzialmente può essere costituita da milioni di host. I problemi che si incontrano sono quelli tipici, quali, ad esempio, programmabilità, ambiente di esecuzione dinamico, eterogeneità, portabiltà, sicurezza e alcuni specifici del Web, quali, ad esempio, mancanza di un file system condiviso, bassa larghezza di banda e alta latenza.

In generale, nè il programmatore nè l'utente finale vogliono confrontarsi con un ambiente dinamico e imprevedibile come il Web; infatti, agli utenti non interessa se i programmi vengono eseguiti localmente o su macchine remote. Pertanto, il modello di programmazione deve essere separato dall'ambiente di esecuzione, ovvero i programmi devono essere sviluppati per una macchina virtuale uniforme e prevedibile e il runtime system deve realizzare tale machina virtuale. Il Web è costituito da vari domini amministrativi, dove la disponibilità di un host può venire a mancare repentinamente e i ritardi associati alla rete sono imprevedibili, occorre, pertanto, prevedere dei servizi adeguati di bilanciamento del carico e di tolleranza ai guasti. I problemi connessi con la sicurezza riguardano sia chi cede ad uno "sconosciuto" le proprie risorse di calcolo, sia chi deve ricevere i risultati di una computazione.

In quest'ultimi anni numerosi sono stati i progetti svolti per realizzare strumenti di metacomputing, in cui il ruolo del Web passa da quello passivo di fornitore di informazioni a quello attivo di agente di calcolo (Web-Metacomputing). Tra questi si citano: Atlas (University of California, Berkeley e University of Texas, Austin) [BBB96], DISCWorld (University of [HBC+98] [HJS+98], Charlotte (New York University) [BKK+96] [BKK+98b], Bayanihan (MIT) [Sarm98a],[Sarm98b] SuperWeb (University of California) [AIS+97], WebFlow (Northeast Parallel Architecture Center) [BBC+97] [HAF+99].

Il prototipo *Meta*ψ costituisce una "prova di concetto" tesa a dimostrare la fattibiltà dell'approccio di questo nuovo paradigma computazionale (metacomputing basato su Web). *Meta*ψ permette l'esecuzione remota di applicazioni, computazionalmente complesse, tramite Web. Esso costituisce un primo passo verso la costruzione di un metacomputer e di fatto

il prototipo può essere visto come un ambiente mediante il quale costruire *Problem Solving Environment* (PSE) che fornisce alcune delle principali funzionalità di calcolo necessarie a risolvere una determinata classe di problemi.

Nella sezione 2 é descritta l'architettura di *Meta*ψ, nella sezione 3 e 4 sono descritti, rispettivamente, i livelli di sicurezza e i meccanismi di tolleranza ai guasti presenti nel sistema *Meta* ψ, nella sezione 5 sono descritti gli strumenti usati per l'implemetazione del sistema e nella sezione 6 é descritta l'interfaccia utente. Infine nella sezione 7 sono riportate le conclusioni e gli sviluppi futuri.

2 *Meta*ψ: descrizione del sistema

Questo strumento nasce con la finalità di fornire accesso a risorse di calcolo ad elevate prestazioni tramite Web. L'idea che sta alla base di *Meta*ψ è quella di fornire un servizio completamente "trasparente" all'utente, che non si deve fare carico di problematiche quali, ad esempio, localizzazione e allocazione delle risorse. Queste funzionalità sono state realizzate nel server Web, opportunamente esteso. Quando è stato possibile, si è cercato di usare degli strumenti standard per realizzare le funzionalità aggiuntive del server, in particolare si è fatto ricorso all'uso di Servlet Java [Java][HC98], del Directory Service basato su LDAP [WHK97] e della *Application Programming Interface* (API) messa a disposizione da JNDI [SUN00] come interfaccia fra Java e LDAP. Nella sezione 4 viene data una descrizione dettagliata di questi strumenti. Inoltre, le varie componenti del sistema sono state implementate in maniera da garantire facile manutenibilità ed estendibilità.

2.1 Architettura generale

L'architettura di *Meta*ψ, come illustrato in Figura 1, è quella tipica di un'applicazione a tre livelli:

- *Client*, costituito da un browser Web;
- *Middleware*, costituito da un server Web; esteso da Servlet Java e dalle funzionalità offerte dal server LDAP;
- *Backend* di risorse computazionali.

Ad un utente sul *client*, dopo una fase di autenticazione, viene visualizzata la pagina, generata dinamicamente da un *servlet* (programma Java in esecuzione sul server, vedi sul server Web, in cui viene presentata la lista delle applicazioni disponibili nel *backend*. Ad ogni applicazione corrisponde un *link*, che, se attivato, invoca un *servlet* che genera dinamicamente la pagina relativa alle modalità di immissione dei dati di input all'applicazione selezionata. *Meta*ψ, dopo aver ricevuto i dati di ingresso, deve localizzare le macchine disponibili che offrono il servizio richiesto. In *Meta*ψ è necessario un servizio di *directory*, che consenta di recuperare efficacemente varie categorie di informazioni. Tale servizio si basa su *Lightweight Directory Access Protocol* (LDAP) (ved.sezione 5).

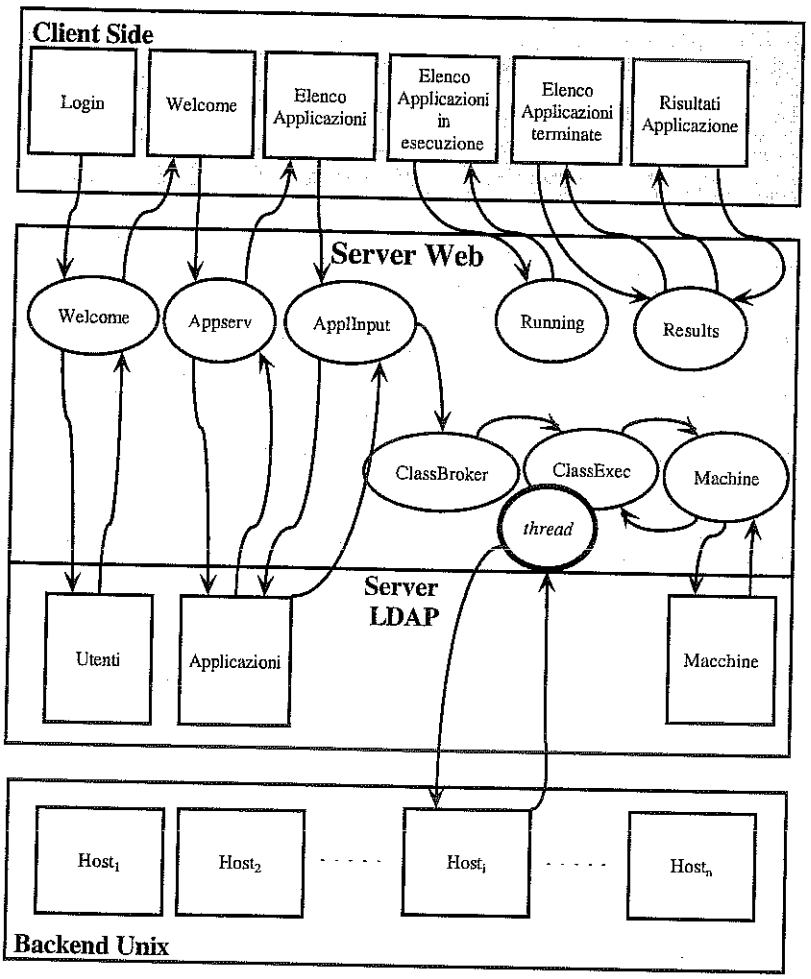

Figure 1: Schema generale della architettura in cui sono messe in evidenza le interazioni principali tra le diverse componenti del sistema.

Dopo aver interrogato il server LDAP e aver ottenuto le informazioni sulle macchine disponibili, il server trasferisce i dati sulla macchina selezionata e attiva l'esecuzione dell'eseguibile relativo all'applicazione tramite uno script residente sul backend. Al termine dell'esecuzione dell'applicazione i risultati che questa produce vengono trasferiti sul server e al client viene notificata la terminazione dell'applicazione tramite l'invio di un e-mail.

2.2 Architettura del middleware

Il middleware è costituito da cinque componenti: un server Web (Apache Web Server), un servlet engine (JServ), un server LDAP (OpenLDAP), un insieme di classi Java che implementano varie funzionalità e un'interfaccia Java (JNDI) utilizzata per l'interazione con il server LDAP. La componente Web si occupa dell'interazione con il client e si fa carico di eseguire i servlet che gestiscono le richieste dell'utente. I vari servlet presenti sul server Web gestiscono le funzionalità relative a:

- autenticazione dell'utente e profili utente (**Welcome**);
- gestione profili dell'applicazione (**Appserv**, **ApplInput**);
- gestione esecuzione remota dell'applicazione, che prevede anche la raccolta degli input e la notifica dei risultati (**ClassBroker**, **ClassApp**, **Running**, **Results**);
- localizzazione, allocazione e configurazione delle risorse (**Machine**);
- aggiornamento e mantenimento del sistema (**Initial**, **Updater** e **LdapReanimator**);

Welcome implementa l'algoritmo scelto per autenticare l'identità dell'utente; in Figura 2 vengono illustrate le componenti coinvolte in questo processo e le informazioni scambiate fra queste. A seconda della politica scelta, `Welcome` deve gestire le varie interazioni con il client per stabilire la sua identità. Per realizzare ciò deve fare accesso alla lista degli utenti ammessi a fruire dei servizi, che può essere codificata come un insieme di entry di LDAP. Tra gli attributi che fanno parte dell'oggetto utente (in LDAP), ci sono quelli che identificano il suo profilo, ovvero quali applicazioni può mandare in esecuzione, e alcuni attributi ausiliari, che vengono usati per l'identificazione e che possono variare in relazione alla politica scelta. Per motivi legati alla *privacy* tutte le informazioni confidenziali dell'utente sono state nascoste ad utenti non autorizzati (tutti tranne egli stesso e l'amministratore del sistema) e gli entry sono registrati con un nome completamente anonimo, tipo `usernnnn`, dove con nnnn si indica un valore numerico intero.

Vista l'insufficenza dei metodi di autenticazione di base forniti dal server HTTP e vista la necessità di garantire la riservatezza e l'integrità della password dell'utente che si collega al sistema, si è reso necessario integrare al sistema un modulo software per la codifica e la trasmissione di tali informazioni nella maniera più riservata possibile. In particolare in *Meta*ψ si è scelto di adottare un meccanismo di autenticazione dei messaggi basato su

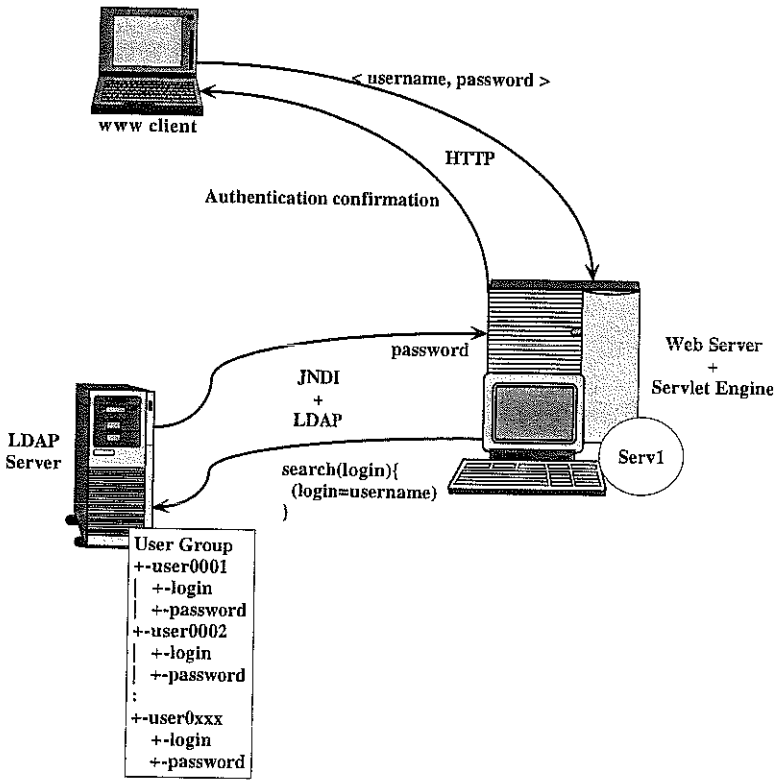

Figure 2: Il processo di autenticazione

HMAC-MD5 [KBC96]. L'algoritmo prevede la conoscenza da parte di entrambi i partecipanti al processo di autenticazione, di una stringa segreta, nel nostro caso la password dell'utente. Tale stringa viene usata assieme a un valore casuale (identificativo, ID, di sessione generato dal servlet) e a un altro valore casuale calcolato in base al tempo trascorso prima che l'utente invii la risposta per generare (sia dal lato client che dalla parte server) un messaggio, chiamato digest, che contiene la codifica della password. Il client invia questo digest al server che lo confronta con quello calcolato da se stesso e verifica in questo modo l'identità dell'utente. In Metaψ il processo di autenticazione viene realizzato mediante un modulo JavaScript che sta nella pagina di login e il servlet Welcome dalla parte del server Web. Il modulo JavaScript viene attivato dalla pressione del tasto che conferma l'immissione della password, applica l'algoritmo HMAC-MD5 prendendo la password come chiave segreta condivisa, lo ID di sessione gli viene passato insieme alla pagina Web dal servlet che la ha generata e inserita in un campo nascosto del form HTML. Dalla parte server invece viene usato un package standard (`java.security`) per le funzioni della generazione dei digest MD5, mentre l'algoritmo HMAC è stato messo a punto seguendo le indicazioni presenti nella documentazione dello stesso (vedi rif. KBC96).

Dal momento che il servlet ha autenticato il client viene attivata una sessione sul server, tale sessione mantiene le informazioni sull'utente, le applicazioni selezionate, per permettere ai servlet di sistema di interagire tra di loro in base alle scelte dell'utente. I profili utente si rendono necessari per stabilire chi può eseguire cosa e dove vanno memorizzati i risultati dell'esecuzione; si presuppone infatti che in uno scenario futuro ci possa essere una specializzazione della tipologia degli utenti e anche in questo caso l'accesso ai dati memorizzati sul server LDAP permette di stabilire a quale categoria appartenga l'utente.

Appserv prepara la pagina web che serve a visualizzare all'utente l'elenco di applicazioni disponibili al sistema, associa a ciascuna di queste un collegamento ipertestuale al servlet successivo (`ApplInput`) passando ad esso il nome e la versione dell'applicazione riferita dal link.

ApplInput, in base all'applicazione selezionata dall'utente, stabilisce le modalità di passaggio dei dati di ingresso.

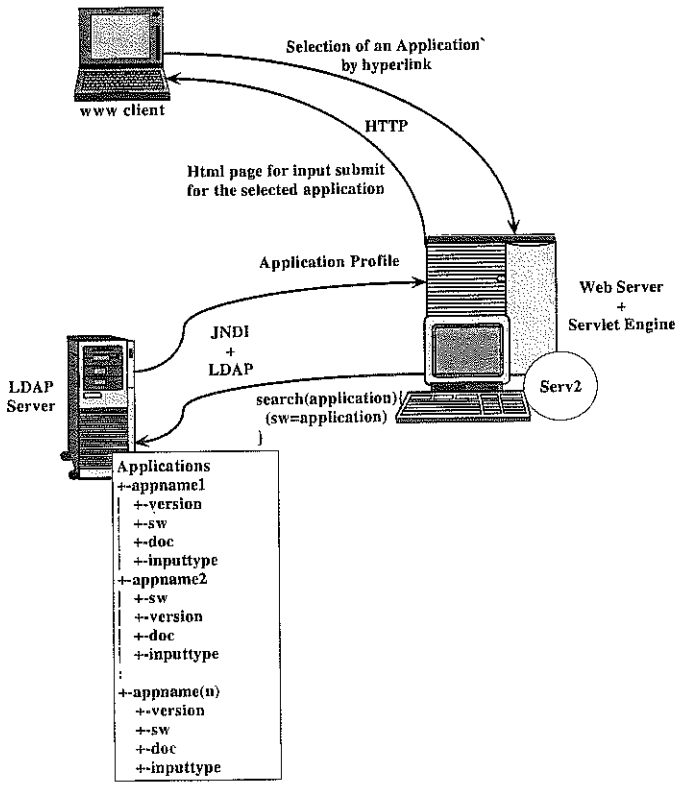

Figure 3: Il servlet `ApplInput` genera una pagina HTML contenente l'elenco delle applicazioni disponibili al sistema mediante informazioni prelevate dal server LDAP.

In Figura 3 vengono illustrate le componenti coinvolte in questa fase e le informazioni scambiate fra le diverse componenti.

Anche questo servlet fa ricorso ai servizi offerti da LDAP per poter stabilire quali sono i formati ammessi per lo input. In LDAP le informazioni sono organizzate in maniera gerarchica, in una struttura ad albero, il *Directory Information Tree* (DIT). Nel DIT sono presenti degli entry relativi ad ogni applicazione presente nel backend, gli attributi stabiliscono in che modo il client deve produrre i dati da passare agli eseguibili. Possono essere inviduate principalmente tre modalità di passaggio dati: attraverso un'interfaccia HTML si offre una maschera per la raccolta dei parametri dell'applicazione, attraverso l'operazione di *upload* di un file sul server oppure attraverso l'indicazione di un host remoto dove il sistema può recuperare i dati di cui ha bisogno. In base alle caratteristiche dell'applicazione `ApplInput` genera dinamicamente una pagina HTML che guida l'utente nell'immissione dei dati. La maschera di raccolta dati e il codice HTML relativo alle operazioni di upload sono memorizzati sul server LDAP come attributi dello entry relativo all'applicazione, da cui segue che ciascuna applicazione ha la propria interfaccia.

ClassBroker, una volta che il server Web ha ottenuto i dati di input dalle informazioni relative all'applicazione selezionata, preleva da LDAP il nome della classe Java associata all'applicazione, la instanzia mediante i meccanismi di *reflection* di Java invocando il suo costruttore, passando ad esso tutti i dati necessari per soddisfare la richiesta dell'utente.

ClassApp è la superclasse da cui vengono estese le classi Java registrate con le applicazioni. Essa fa da collante tra l'applicazione e il resto del sistema, ovvero si fa carico di mantenere informazioni relative all'utente che ha chiesto l'esecuzione, la sessione nella quale è stata richiesta l'operazione, aggiorna i log di sistema, fornisce un metodo per l'attivazione del modulo per la selezione della macchina migliore per eseguire l'applicazione e si occupa di interagire col backend per l'attivazione vera e propria dell'applicazione.
La classe che estende `ClassApp` si deve preoccupare solo di definire i parametri legati all'applicazione, estendendo i metodi messi a disposizione dalla superclasse nel caso in cui l'applicazione abbia bisogno di particolari configurazioni prima dell'esecuzione, ad esempio la preparazione di uno script per la pre-elaborazione dei risultati.
Tra le operazioni svolte dal modulo `ClassApp`, troviamo:

- **Allocazione delle risorse** localizzate precedentemente. Come illustrato in Figura 4, mediante meccanismi di remote shell (SSH nel nostro caso), viene attivata una sessione sullo account utente messo a disposizione dalla macchina del backend, vengono trasferiti i file di input in una determinata directory di questa macchina, viene configurato lo script di attivazione dell'eseguibile relativo all'applicazione;

- **Esecuzione remota dell'applicazione**, che consiste nell'attivare l'esecuzione dello script che gestisce l'esecuzione dell'applicazione. I meccanismi di sistema (*pvm daemon*, *mpirun*, *Condor*, ...) locali alla macchina prescelta si occupano dell'esecuzione dell'applicazione e della memorizzazione su file dei risultati;

- **Raccolta e notifica dei risultati**, `ClassApp`, o meglio il thread attivato dalla classe derivata da essa, rimane in attesa della terminazione dell'applicazione per reperire, sempre tramite remote shell, i risultati prodotti da questa, quindi aggiorna lo stato di

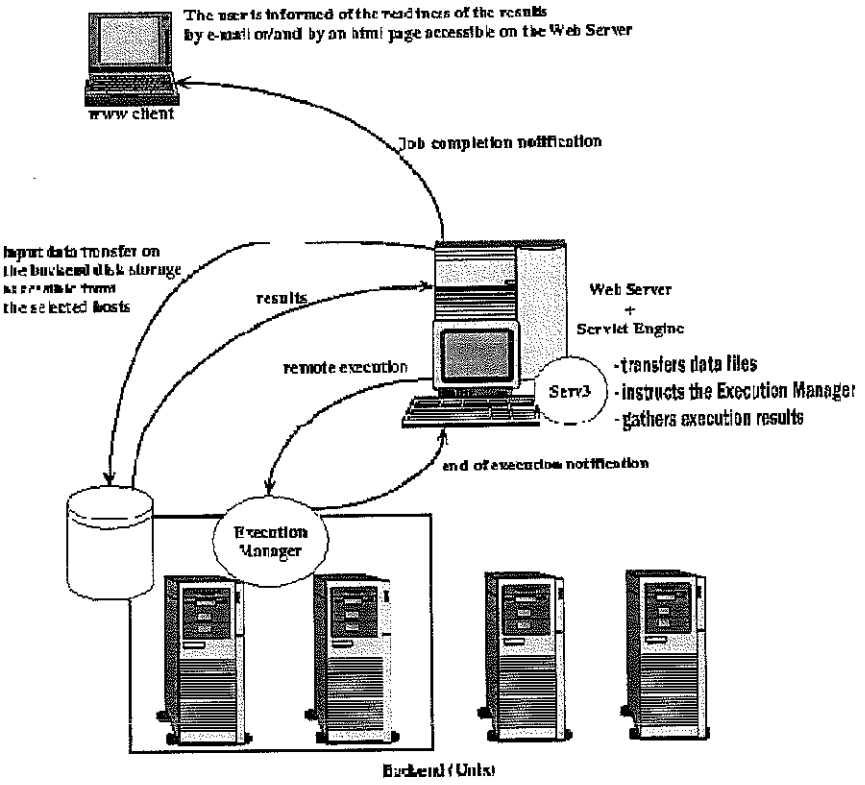

Figure 4: Esecuzione remota dell'applicazione.

avanzamento dell'applicazione, indicando anche il nome del file contenente i risultati, e informa l'utente del completamento dell'esecuzione dell'applicazione mediante l'invio di un e-mail all'indirizzo da questi indicato durante la registrazione.

Results si occupa dell'inoltro dei risultati all'utente. Nel momento in cui l'utente vuole vedere i risultati prodotti da un'applicazione attiva il link corrispondente alla pagina dei risultati. Questa pagina viene generata dinamicamente dal servlet Results e contiene inizialmente un elenco di applicazioni terminate e una serie di link che permettono di vedere i risultati relativi. Ovviamente la visualizzazione dei risultati dipende dal tipo di risultati. Nel caso in cui il risultato dell'applicazione fosse un'immagine compatibile coi formati per il Web, esiste un modulo che permette al servlet di visualizzarla in una pagina separata. Nel caso più generico in cui i risultati dell'applicazione sono numeri da rielaborare mediante altri programmi (fogli di calcolo, generatori di grafici, Matlab, Gnuplot), allora è importante dare all'utente la possibilità di scaricare il file, mediante un hyperlink, direttamente sulla macchina su cui questi dati possono essere ulteriormente elaborati.

Running è un servlet che, allo stato attuale ha solo valore informativo sullo stato di avanzamento dell'applicazione, generando un report su quello che il sistema sta facendo

istante per istante.

Machine, come illustrato in Figura 5, deve interagire con il server LDAP per individuare quali macchine hanno la disponibilità del software richiesto dall'utente.

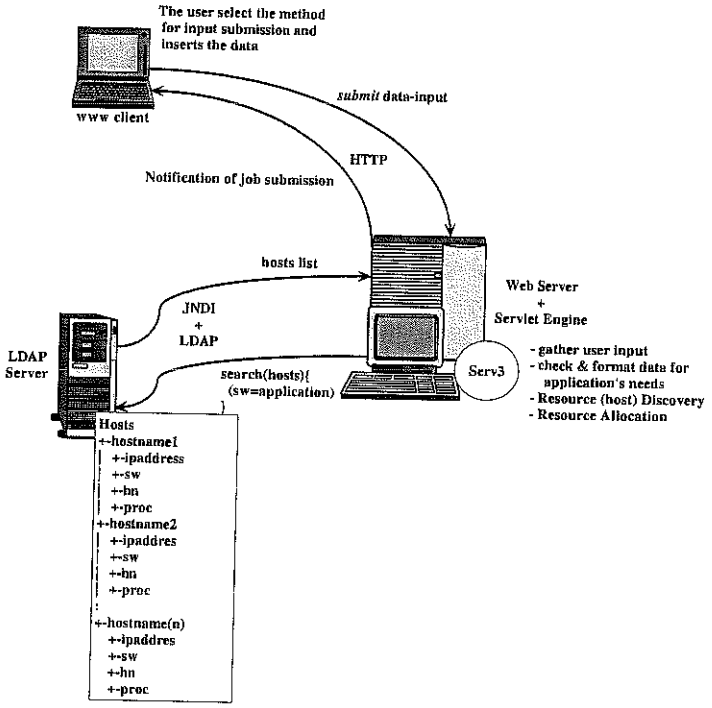

Figure 5: Visione d'insieme delle funzionalità di **Machine**.

Se l'applicazione selezionata è disponibile su più macchine del backend si deve effettuare una selezione in base ad un qualche criterio. Attualmente *Meta*ψ fa la sua scelta basandosi solo sul carico rilevato sulle macchine candidate e viene allocata quella che presenta il carico più basso. Questa è ovviamente una soluzione approssimativa, non garantisce l'ottimizzazione dell'utilizzazione delle risorse, ma la messa a punto di un'euristica di *mapping* rappresentava un compito non banale e troppo oneroso nell'economia della realizzazione del nostro prototipo. L'adozione di un algoritmo di mapping più "sofisticato" renderebbe necessario solo la riscrittura della classe Java che gestisce la selezione e mantenendo l'interfaccia della classe attuale, occorrerebbe solo una ricompilazione per integrarla nel sistema.

Updater. Occorre notare che le informazioni contenute in LDAP sono inerentemente statiche, pertanto bisogna prevedere dei meccanismi che permettano di monitorare lo stato delle risorse del backend e che aggiornino conseguentemente gli entry del DIT garantendo la consistenza delle informazioni in esso contenute

Fra gli attributi che descrivono le macchine facenti parte del backend, `isActive` e `avgload` indicano, rispettivamente, se la macchina è attiva (o più in generale raggiungibile) e il suo carico computazionale, indicato con un numero decimale.
È la classe Java **MachineUpdater** (MU), instanziata dalla classe `Updater`, che si fa carico di aggiornare queste informazioni; periodicamente, MU esegue una *uptime* su tutte le macchine del backend, se queste sono attive aggiorna il valore dell'attributo `avgload` tramite JNDI, se si verifica un errore marca la macchina come non disponibile ponendo uguale a *false* il valore di `isAlive`;

2.3 Caratteristiche del backend

Il backend può essere costituito da risorse computazionali ad alte prestazioni, sistemi multiprocessore, reti di workstation, che mettono a disposizione del metacalcolatore delle specifiche applicazioni. Il prototipo attuale dello strumento richiede che sulle macchine del backend sia installato il sistema operativo Unix o un qualche suo "dialetto" e che su queste sia presente SSH per poter permettere l'esecuzione remota di applicazioni. Le organizzazioni proprietarie delle risorse devono fornire un account utente all'amministratore di *Meta*ψ e questi deve configurare opportunamente lo script necessario per la gestione dell'esecuzione dell'applicazione per adattarlo alle caratteristiche dell'ambiente di esecuzione.

Inoltre deve essere presente il programma *uptime*, che viene utilizzato per l'aggiornamento degli entry relativi alle macchine nel server LDAP.

3 Sicurezza

A causa della struttura a tre livelli dell'architettura di *Meta*ψ, possono verificarsi inconvenienti legati alla sicurezza nelle interazioni fra le varie parti del sistema. Queste riguardano principalmente le comunicazioni fra:

- client e server Web;
- server Web e servlet Engine;
- server Web e server LDAP;
- server Web e backend.

Come è già stato descritto nel paragrafo 2.2, l'ausilio dell'algoritmo HMAC-MD5 permette di criptare le informazioni che vengono scambiate fra client e server nella fase di autenticazione, garantendo la riservatezza delle informazioni scambiate. Il meccanismo delle sessioni permette di limitare l'accesso alle pagine Web di *Meta*ψ solo ad utenti accreditati che abbiano superato l'autenticazione con il sistema; qualora qualcuno cercasse di accedervi senza diritti riceverebbe un messaggio di avvertimento contenente un link alla pagina di login.

Occorre notare che le interazioni fra client e server Web successive all'autenticazione avvengono su canale insicuro, ma l'adozione di protocolli quali, ad esempio, lo HTTPS [Resc99], potrebbe ovviare a questo inconveniente.

Nell'attuale configurazione, JServ è stato installato sulla stessa macchina su cui è in esecuzione il server Apache, pertanto le comunicazioni del protocollo AJP non passano su rete TCP/IP e pertanto non sono esposte ad attacchi di tipo *packet sniffing* e/o *packet spoofing*.

Per motivi di sicurezza si è scelto di mantenere le pagine Web sul server LDAP. Inoltre, questa scelta è stata dettata anche dalla considerazione che, se in futuro si volesse sostituire l'attuale unico server Web con una rete di server, in maniera tale da distribuire fra questi il carico generato dai client, tale peculiarità permetterebbe di facilitare la messa a punto di tale miglioramento, in quanto non sarebbe necessario dover replicare le pagine HTML su ogni server Web. Occorre precisare che anche il contenuto del DIT di LDAP può essere partizionato e/o replicato su vari server LDAP, il tutto può essere fatto semplicemente settando le relative direttive nei file di configurazione.

Le comunicazioni tra server Web e server LDAP non avvengono su canale sicuro, in quanto l'implementazione attuale di LDAP (OpenLDAP) usata nel nostro prototipo non supporta l'uso della crittografia nelle comunicazioni. Esistono implementazioni commerciali dello standard LDAP che annoverano tale caratteristica, pertanto volendo aggiungere questo ulteriore livello di protezione occorrerebbe solo rimpiazzare OpenLDAP con una diversa implementazione e settare alcuni parametri nelle classi che, tramite JDNI, interagiscono con il Directory Server.

Le interazioni con il backend avvengono tramite l'ausilio del protocollo SSH e di `scp` (*secure copy*), pertanto il livello di sicurezza di queste interazioni è quello garantito dal suddetto protocollo.

4 Tolleranza ai guasti

Nella messa a punto di *Meta*ψ si è cercato di provvedere dei meccanismi che garantissero tolleranza ai guasti, almeno verso alcuni tipi di malfunzionamento. I possibili malfunzionamenti che sono stati trattati riguardano:

- il server Web;
- le interazioni fra server Web e server LDAP;
- le interazioni fra server Web e macchine del backend.

Il servlet Engine (JServ) del **server Web** (Apache) in fase di attivazione (*startup*) permette di eseguire automaticamente un servlet e noi abbiamo sfruttato questa caratteristica per fare effettuare a questo servlet (`Initial`) oltre che le inizializzazioni necessarie per il funzionamento di *Meta*ψ anche una procedura di ripristino nel caso in cui il server venga riavviato a seguito di un malfunzionamento. `Initial` controlla se in una determinata directory del *filesystem* della macchina che ospita il server Web è presente un file (`accrocco.tmp`). Se non c'è significa che è la prima volta che viene eseguito *Meta*ψ, altrimenti vuol dire che si è verificato un malfunzionamento e che il server Apache è stato riavviato. `accrocco.tmp` contiene un oggetto Java serializzato (la serializzazione è un meccanismo messo a disposizione

dal linguaggio Java per gestire la persistenza di un oggetto) che implementa una struttura dati realizzata per tenere traccia delle applicazioni la cui esecuzione non è ancora terminata. Questa struttura dati viene aggiornata e salvata su disco (serializzata) ogni volta che viene accettata una nuova richiesta utente (inserimento di un nuovo riferimento) e ad ogni notifica di terminazione di un'applicazione (eliminazione di un riferimento). `Initial` ricostruisce l'oggetto contenuto in `accrocco.tmp` e riavvia tutte le applicazioni in esso contenute. In pratica la politica che si è adottata è quella di considerare non valide le istanze di esecuzione delle applicazioni non ancora terminate al momento del guasto del server Web, pertanto vengono di nuovo localizzate e riallocate le risorse necessarie e viene riattivata l'escuzione di tali applicazioni.

La classe `Initial` è stata realizzata tramite l'ausilio di meccanismi di *multithreading* di Java. Infatti, questa implementa l'interfaccia *Runnable*, che permette, quando possibile, di effettuare delle operazioni in parallelo, cercando di ridurre i tempi di completamento dell'operazione.

Nelle interazioni con il **server LDAP** potrebbero verificarsi delle inconsistenze dovute a malfunzionamenti del directory server. Queste condizioni sono rilevabili a tempo di esecuzione, tramite i meccanismi standard di Java per il trattamento delle eccezioni. Infatti, se un'interrogazione di LDAP genera un'eccezione, questa viene "catturata" e viene richiamato un metodo della classe `LdapReanimator` per cercare di riattivare il server LDAP. Dopo un numero massimo di tentativi infruttuosi viene generato un messaggio di errore e viene informato il client dell'attuale impossibilità di utilizzo di *Meta*ψ.

Dopo aver attivato lo script per la gestione dell'esecuzione dell'applicazione sul **back-end**, la classe per la gestione dell'esecuzione remota dell'applicazione attiva un *thread* che aspetta la notifica dei risultati. Se il thread rileva un errore sul canale dello standard output allora viene eseguita di nuovo la procedura di localizzazione e allocazione delle risorse e l'applicazione viene nuovamente eseguita.

5 Descrizione degli strumenti utilizzati

Questa sezione descrive dettagliatamente gli strumenti utilizzati per l'implementazione del middleware di *Meta*ψ. Per ciascuno di essi sono riportate le motivazioni che hanno portato alla loro scelta.

5.1 Il server Apache

La scelta del server HTTP è ricaduta sull'implentazione della Apache Software Foundation (ASF) [Apac96]. Il motivo che ci ha portato a scegliere questo server è dovuto principalmente alla sua larghissima diffusione in Internet (soprattutto in ambienti Unix), la sua disponibilità per i più diffusi sistemi operativi, tra i quali Windows NT, Linux, Solaris e OpenVMS, il fatto che sia distribuito gratuitamente e che fa parte della categoria di software denominato *open source*.

Tra le alternative commerciali citiamo il Netscape Server [NETS00] e il Java Web Server di Sun Microsystems [SUN00b]. Mentre tra le alternative non commerciali da ricordare il primo posto lo merita sicuramente Jigsaw della W3C [W3C], al quale abbiamo preferito Apache per i motivi citati sopra: massima diffusione da cui deriva il fatto di essere un prodotto ampiamente testato, mentre Jigsaw funziona solo su quelle architetture che supportano una macchina virtuale Java.

Uno degli aspetti positivi di Apache che deriva dal fatto di essere *open source* è dato dall'estendibilità mediante moduli aggiuntivi. Apache fornisce, infatti, un API per il programmatore che gli permette di far interagire un programma scritto in C, Java o anche Perl con il motore del server. Tra i moduli citiamo solo alcuni tra i più usati: moduli per il supporto dei Servlet Java, modulo di estensione per il protocollo HTTPS tramite SSL, modulo per il supporto CGI, per PHP e per le pagine ASP.

Per quel che riguarda gli aspetti sulla sicurezza, Apache è il server HTTP che più di tutti rispetta lo standard, quindi propone di base alcuni servizi per impedire l'accesso al server da domini internet non riconosciuti (firewalling), da indirizzi IP, oppure garantisce l'accesso solo a utenti o gruppi autorizzati mediante passaggio di credenziali. Pur non supportando lo HTTPS prevede comunque un modulo di estensione specifico che si affida allo SSL presente sulla macchina host su cui viene installato.

5.2 Servlet

I *servlet* [Java][HC98] sono dei programmi Java usati per estendere le funzionalità di un server Web, in pratica possono essere considerati l'omologo degli Applet dalla parte client. Differiscono da questi ultimi per il fatto che non vengono eseguiti in un browser e non fanno uso di interfacce utente grafiche. Interagiscono con il browser del client tramite meccanismi di *request/response* mediate da un Servlet Engine in esecuzione sul server Web e basati sul protocollo standard HTTP.

Come funzionalità sono molto simili ai programmi CGI, ma si differenziano da questi per:

- indipendenza dalla piattaforma: i servlet possono essere eseguiti su una qualsiasi piattaforma dove è installato un *servlet engine*; gli script CGI invece, se sono scritti in linguaggi ad alto livello, come per esempio il C, possono aver bisogno di modifiche e devono essere ricompilati;
- prestazioni: i servlet riducono i ritardi di *startup* perchè una volta attivati possono eseguire più richieste, a differenza degli script CGI che possono soddisfare solo una richiesta per ogni attivazione;
- estendibilità: usano un API standard che si interfaccia con la maggiorparte dei server Web, inoltre possono contare su un'ampia libreria di pacchetti disponibili per Java per la realizzazione delle loro funzionalità.

5.2.1 Architettura

Tutti i servlet Java devono implementare l'interfaccia `javax.servlet.Servlet` per poter essere eseguiti in un *servlet engine*, che è un'estensione di un server Web che gli permette di supportare i servlet.

L'interfaccia citata definisce i metodi che vengono invocati durante l'attività del servlet. Il servlet engine istanzia e carica un servlet, dal file system locale o remoto, avvalendosi dei meccanismi di caricamento delle classi di Java. Fatto ciò occorre inizializzare (una sola volta per tutto il suo ciclo di vita) il servlet invocando il suo metodo `init()`, che prende un oggetto `ServletConfig` come parametro.

Dopo che è avvenuta l'inizializzazione il servlet può iniziare a gestire le richieste client, che il servlet engine gli passa tramite il metodo `service()`, che ha due parametri, `servletRequest` e `servletResponse`. Il primo contiene la codifica della richiesta del client, il secondo viene usato per definire la risposta del servlet al client.

Per come sono stati progettati i servlet gestiscono automaticamente il *multithreading*, che permette la gestione simultanea di più richieste da client diversi. Inoltre, se il servlet engine è distribuito, è possibile eseguire un'istanza del servlet su ogni server su cui è in esecuzione lo engine (distribuzione del carico).

Quando il servlet engine decide di terminare il servlet invoca il suo metodo `destroy()`.

5.2.2 Sessioni HTTP

Un altro vantaggio dei servlet nei confronti di CGI deriva dalla gestione delle sessioni. La sessione è un meccanismo utilizzato dal server per tenere traccia delle operazioni svolte per soddisfare le richieste di un singolo utente. La necessità di utilizzare le sessioni nasce soprattutto quando le interazioni tra il server e l'utente sono gestite da un insieme di servlet cooperanti. Tale situazione se affrontata con i meccanismi del CGI (ad esempio effettuando salvataggi su file dei log delle operazioni oppure facendo rimbalzare tali informazioni dal server al client e viceversa), può portare, oltre che complicazioni di carattere tecnico per chi deve sviluppare il programma, anche problemi di sicurezza e riservatezza.

Java Servlet mette a disposizione un'interfaccia appositamente studiata per la gestione delle sessioni. Una sessione viene attivata e considerata nuova quando il client si collega per la prima volta al servlet; in questa fase viene generato un valore numerico che viene associato al client e che verrà usato per le successive interazioni per verificarne l'identità. Le informazioni relative alla sessione sono conservate sul server in un oggetto Java residente in memoria temporanea e per un periodo di tempo specificato dal programmatore del servlet (tempo di sessione), alla scadenza del quale la sessione viene considerata scaduta e l'oggetto legato alla sessione viene rilasciato. In questo modo si ottiene una maggiore riservatezza poichè i dati viaggiano dal client al server una volta sola per ogni sessione, inoltre essi sono conservati sul server, sottoforma di oggetto Java, garantendo una maggior sicurezza rispetto alla memorizzazione dei dati stessi su file, come è consueto fare con CGI.

5.2.3 Il servlet engine di Apache: JServ

JServ è un *servlet engine*, sviluppato da ASF, che rispetta le specifiche definite da Sun Microsystems con le API per Servlet della versione 2.0.

JServ è stato progettato secondo il modello di un'applicazione a tre livelli. Il servlet engine non è parte del server Web, ma un è un *application server*. Quando il server Web accetta la richiesta di esecuzione di un servlet, si comporta da client, inoltrando tale richiesta tramite il protocollo *Apache JServ Protocol* (AJP) al servlet engine che deve elaborarla. Jserv è costituito da due componenti principali: *mod_jserv* e *Apache JServ*.

mod_jserv è un modulo per Apache Web server, è scritto completamente in C e il suo compito principale è quello di inoltrare una richiesta HTTP fatta dal server Web al sevlet engine tramite il protocollo AJP. Quando il servlet engine elabora una richiesta AJP i suoi risultati sono convertiti nel formato dello HTTP e sono inviati al server Web.

Apache JServ è completamente scritto in Java e non richiede un server in particolare per essere eseguito. Elabora le richieste AJP e invia lo output del servlet al server Web.

Il processo dell'esecuzione di un servlet può essere riassunto come segue:

- Un client contatta il server Web Apache richiedendo l'esecuzione di un servlet;
- Apache gestisce la richiesta HTTP tramite il modulo mod_jserv;
- mod_jserv traduce la richiesta HTTP del client in una richiesta AJP e contatta il servlet engine JServ tramite TCP/IP;
- JServ ha già effettuato il suo processo di inizializzazione ed è in grado di gestire le richieste AJP;
- JServ traduce la richiesta AJP in un oggetto *ServletRequest* e crea un oggetto *ServletResponse* usato dal servlet per restituire i risultati al richiedente;
- durante l'esecuzione tutti i dati passati a *ServletResponse* sono convertiti in formato AJP e sono inviati a mod_jserv;
- mod_jserv traduce i dati in formato AJP in un *response* HTTP e li inoltra al server Web.

L'uso di un'architettura a tre livelli in JServ si traduce in una basso utilizzo di risorse da parte di mod_jserv e nella possibilità di contattare vari servlet engine residenti su host diversi da quello su cui è installato Apache. L'uso di TCP/IP per interconnetere le due componenti di JServ permette una migliore utilizzazione delle risorse, dando la possibilità di bilanciare il carico fra più servlet engine.

JServ può essere usato in due modi:

Locale: server Web e servlet engine sono in esecuzione sulla stessa macchina. È probabilmente la modalità di utilizzo più comune di JServ ed è anche quello più veloce, in quanto AJP non deve ricorrere a TCP/IP per trasmettere i dati al servlet engine, mod_jserv è in grado di attivare automaticamente la macchina virtuale Java e il servlet engine durante la

fase di *startup* del server Web ed è in grado di controllare lo stato della macchina virtuale e di riattivarla in caso di malfunzionamenti;

Remoto: server Web e servlet engine risiedono su due sistemi diversi. In questa modalità mod_jserv non può attivare e controllare JServ, ma è stato scritto (da ASF) un programma di utilità, lo *Standalone Wrapper*, che si fa carico di controllare l'esecuzione del servlet engine.

Queste due possibilità di utilizzo possono portare a diverse topologie nella connessione dei moduli di JServ:

One-to-One: è il caso più semplice e si ha quando un solo server Web è connesso con un solo servlet engine;

One-to-Many: permette ad un unico server Web di connettersi a più servlet engine. Questa configurazione si rivela utile in situazioni in cui o le risorse sono frammentate su host diversi o l'esecuzione del servlet sarebbe troppo onerosa su un'unica macchina e il carico viene ripartito su più host;

Many-to-One: permette a più server Web di essere connessi ad un singolo servlet engine, dando la possibilità di condividere i servlet;

Many-to-Many: in questo scenario un insieme di server Web può connettersi ad un insieme di servlet engine.

5.2.4 Sicurezza

La sicurezza è un aspetto molto importante da trattare in JServ, in particolare in riferimento al trasferimento di dati su TCP/IP fra server Web e servlet engine. Sono stati individuati due diversi aspetti: sicurezza interna e sicurezza esterna [MF98].

Sicurezza interna: i meccanismi per garantire la sicurezza interna devono impedire a persone non autorizzate l'esecuzione di servlet. Jserv prevede una lista di controllo degli accessi, basata su indirizzi IP, da includere nei file di configurazione e un algoritmo di autenticazione basato su MD5. Non sono stati previsti meccanismi di crittografia delle informazioni scambiate tramite AJP, pertanto il sistema risulta essere esposto ad attacchi di tipo *packet sniffing* e *packet spoofing* [MF98].

Sicurezza esterna: il servlet engine partiziona lo spazio di esecuzione dei vari servlet in *Servlet Zones* (SZ), garantendo che i servlet vengano eseguiti in questo ambiente confinato, non potendo interferire con altri servlet e negando l'accesso alle altre risorse del sistema su cui è in esecuzione il JServ. In altre parole, dal punto di vista dei servlet, i SZ sono visti come delle macchine virtuali astratte. In questo modo gli sviluppatori di questo servlet engine hanno garantito che i servlet non possano compromettere la consistenza di dati manipolati da altri servlet e/o le risorse del sistema.

5.3 Directory Service

In un metacalcolatore la capacità di individuare le risorse necessarie a tempo di esecuzione, è di fondamentale importanza. Ciò implica che chi gestisce il metacalcolatore abbia la possibilità di accedere in lettura ad un database contenente informazioni sulle risorse disponibili

al metacomputer. Questo tipo di database sarà comunque limitato a contenere riferimenti a macchine, reti e a tutte e sole le risorse necessarie all'elaborazione di programmi. Quindi, un servizio di directory service leggero e maneggevole si pone decisamente meglio di un servizio database tradizionale.

In un directory service le informazioni sono conservate spesso in maniera grezza e comunque pratica. Queste informazioni, nel nostro caso particolare, sono informazioni relative a organizzazioni, personale e risorse. Tra i principali servizi di directory su Internet tra quelli che meritano menzione in quanto divenuti standard, ricordiamo: DNS, Whois/Whois++, X.500 e LDAP. Ognuno ha, ovviamente, i suoi pregi e difetti, al nostro scopo però X.500 e LDAP sono risultati essere i più adatti in quanto maggiormente configurabili e interfacciabili mediante un API.

5.4 Descrizione di X.500 e LDAP

X.500 [WRH92] è uno standard OSI che definisce un servizio di directory, comprendente: un modello per le informazioni, uno spazio dei nomi, un modello funzionale e l'autenticazione degli accessi. Inoltre definisce un protocollo *ad hoc*, anche esso OSI, il Directory Access Protocol (DAP), che viene sfruttato solo in parte dalle applicazioni *client*.

DAP è molto più complicato del corrispondente protocollo basato su TCP/IP e richiede molto codice e potenza di calcolo per poter essere usato, ed è quasi impossibile utilizzarlo su macchine di modeste dimensioni, come PC o Macintosh. Questa è una delle caratteristiche di X.500 che ne ha limitato la diffusione.

LDAP [WHK97] (Lightweight Directory Access Protocol) è stato sviluppato per "alleggerire" il carico che i *client* X.500 devono sopportare, dando la possibilità di accedere ai servizi di directory anche a macchine con capacità computazionali modeste. LDAP si affida completamente a TCP/IP per il trasporto, semplifica molte delle funzioni di X.500 e ne elimina alcune poco usate, emulandole con altre, ottenendo così un overhead minore per l'accesso al directory rispetto a X.500.

Entrambi i protocolli adottano il modello del *client* che richiede una operazione ad un *server*, che ha il compito di effettuare questa operazione e al completamento della stessa restituisce al *client* o il risultato dell'operazione oppure un messaggio di errore.

I componenti che costituiscono lo X.500 sono:

- modello per le informazioni: determina la struttura e le caratteristiche delle informazioni contenute nel directory;
- spazio dei nomi: specifica come vengono organizzate e riferite le informazioni contenute nel directory;
- modello funzionale: determina quali operazioni possono essere effettuate sulle informazioni;
- struttura per l'autenticazione: permette l'accesso alle informazioni solo a *client* autorizzati;

- modello operazionale distribuito: specifica come vengono distribuite le informazioni su più *server* e come vengono gestite le operazioni su queste.

Il modello per le informazioni è basato sul concetto di *entry*, composti da attributi che hanno un tipo e uno o più valori. Il tipo dell'attributo ne determina la sintassi e l'insieme dei valori ammissibili. Per ogni *entry* esiste un oggetto (`objectClass`) che al momento della creazione di un nuovo elemento, effettua dei controlli sulla consistenza dei valori immessi.

I vari entry sono organizzati in una struttura gerarchica ad albero detta *Directory Information Tree* (DIT), distribuita fra più server. Ogni entry viene identificato da un *Distinguished Name* (DN) e da un *Relative Distinguished Name* (RDN) che è il DN dello entry privato della parte di DN dello entry genitore a livello superiore. Per semplificare la visita del DIT è possibile definire degli alias che puntano direttamente a specifiche entry di un sottoalbero.

Le operazioni di base che X.500 definisce sugli entry sono di tre categorie: ricerca (*search*) e lettura (*read*), modifica (*modify*) e autenticazione (*bind*). La *read* restituisce gli attributi di un entry di cui è noto il nome; la *list* elenca i figli di un dato entry; la *search* seleziona entry appartenenti ad un sottoalbero in base a criteri di selezione stabiliti da un filtro di ricerca.

Nella seconda categoria X.500 definisce quattro operazioni per modificare il directory: *modify* è usato per cambiare un entry esistente, permettendo di aggiungere o rimuovere attributi o valori; *add* e *delete* sono usati per inserire e rimuovere entry dal directory; *modifyRDN* permette di modificare il nome di un entry.

L'ultima categoria definisce l'operazione *bind* che permette ad un client di identificarsi e di iniziare una nuova sessione. Sono supportati più meccanismi di autenticazione, dalla password *clear-text* a metodi basati su chiave pubblica; *unbind* termina una sessione con il directory; *abandon* interrompe un'operazione in corso.

Ogni operazione effettuata sugli entry può essere firmata per assicurarne l'integrità. Il protocollo di firma usato si basa sullo scambio di chiavi pubbliche tra server e client.

X.500 permette di distribuire l'informazione tra più server, chiamati Directory System Agents (DSA) che collaborano tra loro per restituire al client l'informazione richiesta oppure possono inviare al client un riferimento ad un server su cui reperire l'informazione.

LDAP adotta lo stesso modello per le informazioni e lo stesso spazio dei nomi di X.500. Il modello funzionale di LDAP è un sottoinsieme di quello di X.500 e la codifica dei filtri di ricerca è molto più semplice che in DAP. I parametri di ricerca sono sempre codificati come stringhe di caratteri e non come strutture complesse. Le risposte al client sono inviate una alla volta in pacchetti separati. Per quanto riguarda l'autenticazione degli accessi, LDAP prevede un sottoinsieme delle possibilità di X.500: un'autenticazione semplice, tramite passaggio di credenziali in chiaro; meccanismi di autenticazione basati su Kerberos versione 4; altri meccanismi sono adattabili grazie al fatto che l'operazione di bind di LDAP prevede la scelta del meccanismo di autenticazione.

5.5 Problemi implementativi

Il modello dati è stato completamente basato su LDAP, TCP/IP e un insieme di server distribuiti che mantengono l'informazione; ogni server è responsabile per un singolo e intero DIT. Questo porta però ad avere degli svantaggi in termini di prestazioni:

- chi fornisce l'informazione è un singolo server: tutta l'informazione di un DIT deve essere mantenuta e gestita da un unico server.
- architettura cliente/servente: LDAP richiede almeno un overhead di comunicazione con il server per ogni accesso. Accessi frequenti sono proibitivi e necessitano meccanismi di caching.
- visibilità dei dati: ogni informazione è accessibile ovunque nella rete (nei limiti del controllo agli accessi).

Si tratta tuttavia di limitazioni del protocollo LDAP e non della sua API. Quindi, per la realizzazione di un directory service per la gestione di un metacalcolatore, è necessaria un'implementazione *ad hoc* di LDAP con prestazioni superiori e più fornitori di informazioni.

5.6 Uso di LDAP in *Meta*ψ

Diamo ora una descrizione della struttura del DIT che abbiamo adottato all'interno di Metaψ per fornire un'astrazione di utenti, applicazioni e macchine. Quanto si è fatto ci ha permesso di creare un *repository* di informazioni di diverso tipo, a cui i servlet del sistema accedono tramite la medesima interfaccia (fornita da JNDI), nelle diverse fasi dell'elaborazione di una richiesta utente.

La struttura ad albero del DIT realizzato riprende quella che è la struttura gerarchica delle informazioni disponibili in Metaψ, in cui la radice è costituita dal solo *Distinguished Name* (DN) della pseudo-organizzazione che fa capo al sistema, come si può vedere nella Figura 6 e dal seguente estratto dal file di configurazione del DIT:

```
dn:  o=Overall, c=it
o:  Overall
o:  CNUCE
objectclass:  organization
```

I nodi del primo livello sono costituiti da entry che sono usati per descrivere:

- il manager del sistema che ha l'accesso alle risorse di LDAP ed ha pieni diritti nella modifica del DIT; egli è colui che registra, manualmente, sul server LDAP le organizzazioni che vogliono offrire risorse e applicazioni, registra e amministra i nuovi utenti nel gruppo utenti esistente;

- un superutente al quale sono permessi unicamente la lettura e scrittura di un sottoinsieme delle informazioni contenute sul server LDAP. Infatti il sistema deve interagire con il server LDAP per aggiornare le informazioni dinamiche contenute in esso (carico delle macchine e loro disponibilità), in pratica il superutente non è altro che un'astrazione delle classi Java che, tramite JNDI, interagiscono con LDAP;
- gli attributi relativi alle pagine Web che vengono usate dai servlet come modelli per la generazione dinamica delle pagine da restituire al client;
- il gruppo di utenti registrati a cui è permesso l'uso delle risorse di sistema;
- altri tipi di informazioni, ognuna rappresentante l'organizzazione che fornisce le macchine e le applicazioni su queste installate.

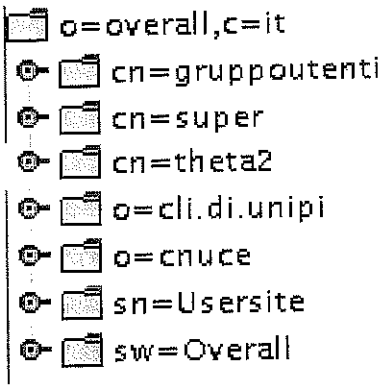

Figure 6: Primo livello del DIT di LDAP, dove sono visibili elementi che rappresentano: il gruppo utenti del sistema (gruppoutenti), l'entry del super utente (super), quello del manager (theta2), gli entry di due organizzazioni che hanno offerto le loro macchine (cnuce e cli.di.unipi), il repository per le pagine HTML (usersite) e un entry descrittivo il software (Overall).

A loro volta nei livelli sottostanti sono presenti altri entry, per esempio:

- come sotto-entry di ciascuna organizzazione c'è un entry per il manager dei sistemi e un inisieme di entry ciascuno rappresentante una macchina disponibile, questi sono a loro volta a capo di un altro livello di sotto-entry rappresentanti le applicazioni installate sulle macchine;
- lo entry rappresentante il gruppo utenti al livello precedente ha come sotto-entry un elenco di utenti e le informazioni ad essi correlate, quali ad esempio, la password, l'indirizzo e-mail e la username registrata con il sistema.

Per costruire la struttura del DIT sono state definite, mediante la sintassi fornita da OpenLDAP, delle nuove classi di entry (`ObjectClass`), specificando tutti gli attributi che si rendevano necessari all'implementazione del DIT.

Il principale ostacolo nell'utilizzo di LDAP è rappresentato dal fatto che le informazioni fornite da un entry di classe `host` (rappresentanti le macchine disponibili), possono variare nel tempo senza possibilità alcuna di previsione a priori e dipendono, ad esempio, da guasti riguardanti la connessione di rete della macchina o la disponibilità stessa della macchina nella rete. Questo ha complicato la gestione del directory service, il quale è stato sviluppato per gestire informazioni tipicamente statiche e non offre quindi meccanismi per interagire con le risorse per mantenere aggiornate le relative informazioni. Per risolvere questo problema si è proceduto in due passi: per prima cosa, sono state individuate le informazioni che si è reputato fosse importante dovere aggiornare dinamicamente, questo è stato fatto dopo aver scelto l'algoritmo di allocazione delle risorse, implementato poi nella classe `Machine`, si sono introdotti in ogni entry rappresentante gli host, due attributi, `isAlive` e `avgload`, rappresentanti rispettivamente la disponibilità della macchina e il carico di lavoro presente su di essa.

Come secondo passo, si è intervenuto mediante lo sviluppo, di specifiche classi Java (`Updater` e `MachineUpdater`) che, sempre attive sul server, periodicamente (il periodo viene impostato dall'amministratore del sistema) accedono alle macchine per testarne la disponibilità e il carico di lavoro, aggiornando il valore degli attributi (`isAvailable` e `avgload`) del relativo entry nel DIT, i cui valori vengono utilizzati dalla classe `Machine` che si occupa della scelta della macchina su cui eseguire l'applicazione prescelta dall'utente.

Attribute	Value
hn	brunello
overallpasswd	BINARY (33b)
overallaccount	theta2
objectclass	host
objectclass	mpiHost
mpirun	/usr/local/mpich-1.2/f77-npf90/bin/mpirun
avgload	0.08
overallhome	/home/brunello/theta2/OVERTMP
isactive	true
ipaddress	brunello.cnuce.cnr.it

Figure 7: Lista degli attributi dello *objectclass* `host`.

Tra gli altri attributi di un `objectclass host`, come illustrato in Figura 7, citiamo anche:

- `ipaddress` che specifica l'indirizzo IP dello host;
- `overallAccount`: specifica il nome utente assegnato a Metaψ dall'amministratore dello host;
- `overallPasswd`: la password dello account assegnato a Metaψ;
- `overallHome`: il percorso assoluto, riferito al filesystem dello host, che indica lo spazio riservato a Metaψ;

Queste informazioni sono utilizzate dai servlet del sistema che accedono alla macchina selezionata per attivare l'esecuzione di un'applicazione.

A ciascun entry rappresentante un host, come evidenziato in Figura 8, sono anche assegnati dei sub-entry che rappresentano le applicazioni installate e configurate (espressamente per interagire con Metaψ, mediante lo script del backend) per l'esecuzione su quella specifica macchina.

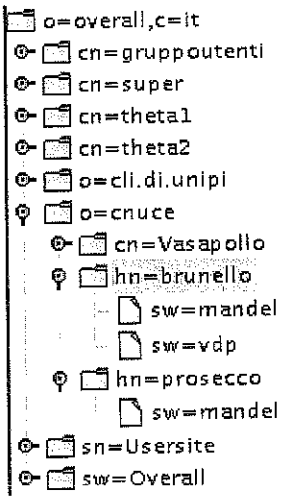

Figure 8: Dettaglio delle applicazioni presenti su un particolare host.

La scelta di associare le applicazioni alle macchine su cui queste sono installate è legata al fatto che i programmi e le macchine restano sempre di proprietà degli enti (o organizzazioni) che le mettono a disposizione di Metaψ. Ciascuno di questi enti può scegliere di non fare eseguire sulle proprie macchine codice non proprietario dell'organizzazione, oppure impedire che il proprio codice possa essere eseguito su macchine di altre organizzazioni.

Un entry "applicazione" ha degli attributi che ne definiscono le caratteristiche: un nome, la versione, la directory (riferito allo host su cui è installata) in cui è installata e un attributo, `jclass`, che specifica la classe Java che estende il modulo `ClassApp` di Metaψ.

Allo stato attuale del progetto, è permesso l'accesso alla configurazione del directory service solamente ad un amministratore (che può essere l'amministratore del sistema su cui è installato) il quale può intervenire su di esso unicamente utilizzando gli strumenti forniti di serie con il server, ovvero delle utilty che permettono di effettuare tutte le operazioni di ricerca, lettura e modifica del DIT. Inoltre puó essere usato lo strumento visuale, LDAP Browser/Editor [Gawo99] La configurazione del server LDAP per il nostro caso specifico è finalizzata a garantire la riservatezza delle informazioni contenute in esso. Ad esempio, come illustrato in in Figura 6, si può notare che allo entry che specifica il manager (`cn=theta2`), che è l'amministratore di Metaψ, è stato affiancato il superuser (`cn=super`), che di fatto è un'astrazione delle classi Java che accedono al server LDAP. A superuser sono stati concessi diritti di lettura sulle informazioni del *directory* e diritti di modifica solo su un sottinsieme

di queste, mentre il manager può sia accedere che modificare ogni informazione contenuta sul server LDAP. Tutto ciò ha permesso di non utilizzare le credenziali dell'amministratore nelle classi Java che accedono al *directory*.

Nel file di configurazione sono stati impostati dei vincoli di accesso ai vari livelli del DIT, mediante la sintassi del file di configurazione di OpenLDAP. Questa speciale sintassi permette di controllare l'accesso a singoli attributi di una particolare classe di entry stabilendo vincoli che sono dipendenti dal client (utente oppure host) che richiede le informazioni relative al particolare entry che si vuole controllare. In particolare, sono state vietate la visualizzazione di informazioni quali le password, le configurazioni delle macchine e del software a utenti non autorizzati (che non siano quindi il manager o il super utente) ed è stato protetto anche il codice HTML e JavaScript contenuto nello entry di LDAP che rappresenta il *repository* per i modelli di pagina utilizzati dai servlet del sistema.

Un possibile miglioramento al sistema potrebbe essere l'integrazione di una nuova componente software di supporto all'attività del manager, fornendo un'interfaccia utente che semplifichi il lavoro dell'amministratore nella riorganizzazione del DIT. Questa interfaccia potrà anche essere fornita ad un utente dello strumento, con funzionalità più limitate, per effettuare modifiche personalizzate al DIT (almeno di quelle parti del DIT non condivise con altri utenti), inserendo tra i calcolatori facenti parte del backend, ad esempio, un proprio server o il proprio PC come destinazione delle elaborazioni e per la raccolta dei dati e dei risultati.

5.7 JNDI

Java Naming and Directory Interface (JNDI) [SUN00] è un Application Program Interface (API) che mette a disposizione delle applicazione scritte usando il linguaggio Java funzionalità di *naming* e di *directory* (paragrafo 2.4). JNDI è stato progettato per essere indipendente dall'implementazione del directory service, pertanto permette l'accesso trasparente ad un insieme di directory service.

L'architettura di JNDI consiste di un API e di un *Service Provider Interface* (SPI). SPI permette di interfacciare più servizi di naming e di directory, permettendo alle applicazioni di accederli tramite lo API. JNDI è inclusa nella SDK 1.3 di Java 2 ed è anche disponibile come estensione standard per la versione 1.1 della JDK. Nella SDK 1.3 sono attualmente supportati tre service provider:

- LDAP;
- naming service di CORBA (COS);
- Registry di Remote Method Invocation (RMI).

5.8 SSH

Secure Shell (SSH) è stato sviluppato da SSH Communication Security [SSH] ed è un protocollo per effettuare *login* su macchine remote, per eseguirvi dei comandi e per spostare

file da una macchina ad un'altra. Attualmente coesistono due standard, SSH1 e SSH2, che sono di fatto due standard diversi.

SSH garantisce autenticazione e comunicazioni sicure su canale insicuro [YKS+00] ed è stato realizzato per sostituire *telnet*, *rlogin*, *rsh*, *rcp* e in SSH2 è presente anche *sftp* che va a rimpiazzare FTP.

In SSH l'autenticazione può avvenire in vari modi: tamite *password*, tramite chiave pubblica (RSA o DSA a seconda dell'implementazione), tramite Kerberos (in SSH1) o tramite *host* (in SSH1 tramite il file *.rhosts* o */etc/host.sequiv*, chiave pubblica in SSH2). SSH garantisce protezione nei confronti delle seguente tipologie di attacco:

- **IP spoofing**, in cui un host remoto invia pacchetti in cui è stato falsificata l'identità del mittente, sostituendola con quella di un host "fidato";
- **DNS spoofing**, dove un hacker falsifica i record del nome del server;
- **Intercettazione** di password non criptate e altri dati da parte di host intermedi;
- **Alterazione** di dati da parte di host intermedi;

SSH1 e SSH2 sono due protocolli completamente diversi, in pratica SSH2 è una riscrittura di SSH1 in cui sono stati migliorati aspetti legati a sicurezza, prestazioni e portabilità [YKS+00].

5.8.1 Architettura di SSH

SSH è costituito da tre componenti principali: *Transport Layer Protocol* (TLP), *User Authentication Protocol* (UAP) e *Connection Protocol* (CP). Descriviamo brevemente le caratteristiche e le funzionalità di queste componenti:

Transport Layer Protocol è un protocollo di trasporto sicuro di basso livello e fornisce un insieme di servizi di rete, fra cui autenticazione del server, crittografia "forte" e protezione dell'integrità. Permette di negoziare vari parametri necessari per l'interazione fra client e server, quali, ad esempio, metodo di scambio delle chiavi, algoritmo di chiave pubblica e algoritmo di autenticazione dei messaggi.

Authentication Protocol ha il compito di effettuare l'autenticazione del client e parte dal presupposto che ci sia già un livello di trasporto sicuro (stabilito da TLP), che il server sia già stato autenticato, che sia stato già stabilito un canale di comunicazione criptato e sia stato associato un identificativo alla sessione.

Connection Protocol si basa su TLP e UAP, permette di stabilire sessioni di *login* interattive, l'esecuzione di comandi sulle macchine remote, l'inoltro di connessioni TCP/IP e X11 e può anche permettere al server di eseguire comandi sul client.

6 Descrizione dell'interfaccia web

Nella presente sezione viene descritta l'interfaccia grafica utente (GUI) dello strumento realizzato. Per poter usufruire dei servizi offerti da *Meta*ψ l'utente deve disporre di un

browser che supporti JavaScript e le pagine HTML sono ottimizzate per Internet Explorer versione 4.0 o superiore e Netscape Navigator 4.4 o superiore.

L'idea di questa parte del progetto era quella di rendere il più semplice possibile l'uso di *Meta*ψ a persone che non devono farsi carico dei problemi connessi alla localizzazione e allocazione delle risorse.

6.1 I linguaggi usati

Per realizzare l'interfaccia si è fatto uso dei linguaggi HTML [W3C] e JavaScript [Nets]. Entrambi sono attualmente i linguaggi più usati per la realizzazione di interfacce basate su tecnologie Web; l'alternativa sarebbe stata l'inserimento di componenti realizzate mediante Applet Java [AG96], ma avrebbero potuto rappresentare un sensibile carico computazionale per la macchina del client; riteniamo che gli Applet Java siano utili, ma, solo nel caso in cui non sia possibile realizzare una funzionalità con i mezzi forniti da HTML e JavaScript.

6.2 Il linguaggio HTML

HyperText Markup Language (HTML) è un linguaggio interpretato che utilizza delle particolari etichette (*tag*) per marcare parti del testo che l'interprete, incorporato in un browser Web, converte in elementi grafici. Ciascun elemento presente in una pagina Web è controllato da un tag di HTML, per esempio: la dimensione del carattere, la presenza di un'immagine, una tabella. Tra i vari tag, da ricordare, c'è il tag *A* che permette di definire un collegamento ipertestuale (*link*) ad un altra pagina Web. Da notare che la destinazione di un collegamento non per forza deve essere un'altra pagina, ma può anche essere la richiesta di eseguire un programma sul server (server side). Oltre a quelle predisposte al semplice controllo del testo, il linguaggio HTML offre, per la realizzazione di interfacce grafiche, numerose componenti (non tutte sono state sfruttate nel presente lavoro) quali ad esempio: bottoni, caselle di testo, campi lista, *switch* e *checkbox*. Questi controlli, coadiuvati dall'uso del normale testo e dei link, raccolti in moduli (*form*) si sono dimostrati sufficenti a realizzare in modo semplice (sia per chi l'ha realizzata che, si spera, per chi dovrà usarla) l'applicazione.

6.3 Il linguaggio JavaScript

Il linguaggio JavaScript è anch'esso interpretato da un componente interno (l'interprete JavaScript) del browser Web. JavaScript aumenta l'interattività tra una pagina Web e l'utente. Il suo utilizzo più comune è, attualmente, quello di arricchire le pagine mediante animazioni, sfruttando assieme al linguaggio le più recenti caratteristiche dello HTML. Un utilizzo, meno ludico, riguarda lo sviluppo di applicazioni Web nei casi in cui è necessario far fare al browser operazioni che è inutile o inappropriato far fare ai server, come, ad esempio, scegliendo alcune tra le funzionalità implementate nel nostro progetto: controllo sulla correttezza dei dati inseriti in un modulo, applicazione di algoritmi di crittografia delle password e richieste di conferma di alcune operazioni.

JavaScript è stato sviluppato inizialmente da Netscape Communications e ricalca per la sintassi e per la maggiorparte dei costrutti il linguaggio Java, ma differisce da questo per la minor rigidità delle regole di controllo sui tipi, fatto questo che permette a JavaScript di avere un minor numero di tipi primitivi e un supporto run-time più leggero rispetto al Java. È un linguaggio a oggetti e quindi ha un'ampia gamma di espandibilità.

Un modulo JavaScript viene integrato in una pagina HTML mediante un particolare tag di questo linguaggio (SCRIPT) e in generale viene attivato da un evento generato da un utente che interagisce con un elemento dell'interfaccia, come, ad esempio: la pressione di un bottone, la scrittura di una parola in una casella di testo o il click con il mouse su un collegamento ipertestuale.

Quasi tutti i tag di HTML hanno infatti dei controlli aggiuntivi (parametri) che definiscono quale modulo JavaScript attivare nel momento in cui l'utente interagisce con esso: il browser cattura l'evento ed esegue la funzione (modulo JavaScript) ad esso associata.

La funzione JavaScript da attivare viene specificata come valore di un parametro che rappresenta l'evento e viene attivata nel momento in cui si verifica quel particolare evento sul testo o l'oggetto marcato dal tag HTML.

6.4 Le pagine web

Viene descritta ora la struttura delle pagine Web che costituiscono l'interfaccia di *Meta*ψ. Dove necessario saranno descritte le soluzioni adottate al fine di superare probemi che di volta in volta venivano a presentarsi.

6.4.1 Pagina iniziale

Per poter usufruire dei servizi offerti da *Meta*ψ, un utente deve contattare il manager del sistema per ottenere da questi account e password per poter accedere alle risorse del metacalcolatore; attualmente non è stato sviluppato un protocollo che automatizzi tale procedura ed è necessario un contatto diretto (verbale, telefonico, e-mail...) con l'amministratore del sistema.

L'utente registrato avvia il proprio browser Web e lo indirizza all'indirizzo del server Web, noto a priori (ad esempio: http://novello.cnuce.cnr.it:8080/overall), dove è ospitata la pagina iniziale del sistema.

Funzionalità: La schermata di presentazione (vedi Figura 9) illustra brevemente le caratteristiche dello strumento, a cosa serve, quali sono le attuali potenzialità e come fare per accedere al sistema. La pagina presenta anche link alle pagine Web degli istituti che hanno reso possibile lo sviluppo di *Meta*ψ (l'area pisana della ricerca, CNR e l'istituto CNUCE) e alle organizzazioni di open source che hanno fornito il software necessario per realizzarlo (Apache, OpenLDAP, Linux).

Implementazione: La pagina è stata realizzata utilizzando una tabella HTML senza bordo e riempendo le diverse celle con l'effettivo contenuto della pagina. L'uso delle tabelle senza bordo, non solo in questa pagina ma anche nelle altre, è una tecnica generalmente

usata che permette di posizionare in maniera più precisa gli elementi grafici, rispetto all'uso dei tag di posizionamento elementari dell'HTML. La tabella è suddivisa in 5 righe: la riga superiore formata da tre celle contiene i loghi degli *sponsor* e i link ai rispettivi siti; la seconda riga, di colore viola, fa da divisore tra la prima e quella centrale; la riga centrale costituita da una cella che però copre in larghezza la dimensione della tabella, contiene la descrizione del sito; la quarta riga è anche questa un divisore; la riga inferiore è di tre celle, contiene i loghi dei tool software usati e i link ai rispettivi siti.

Tra le componenti grafiche è presente anche il logo del progetto, il globo che ruota, realizzato usando una immagine animata di tipo GIF, che permette di memorizzare su uno stesso file più fotogrammi in serie e di visualizzarli poi in sequenza circolare (loop).

Interazione: A questo punto l'utente può entrare nel sistema cliccando sul link *enter*.

6.4.2 Pagina di autenticazione

La pagina successiva si presenta all'utente con un form costituito da due campi di testo e un pulsante (vedi Figura 10). Attraverso questa pagina vengono raccolti i dati dell'utente, viene effettuata una codifica della password e viene inviata la codifica al servlet `Welcome` che fa il controllo sulla validità dei dati inseriti. Sono contenuti all'interno della pagina alcuni moduli e funzioni JavaScript.

Funzionalità: Le due caselle di testo servono a raccogliere le informazioni riguardanti lo account e la password dell'utente e il pulsante per confermare l'inserimento.

Implementazione: La pagina, pur apparendo esteticamente, a ogni utente, allo stesso modo, viene creata dinamicamente da un servlet in esecuzione sul server (vedere servlet `Login`). Il motivo è dovuto al fatto che nella pagina è presente un modulo JavaScript, che si occupa della crittografia della password e che necessita a tal scopo di un parametro di volta in volta diverso che viene trasmesso dal server. Il servlet che genera la pagina passa questo parametro al modulo JavaScript utilizzando un campo nascosto (tag `input` di tipo `hidden`) del form. Il valore di questo parametro sarà diverso di volta in volta che la pagina viene nuovamente caricata e rappresenta la caratteristica random utilizzata dall'algoritmo di crittografia implementato dal nostro modulo. L'utilizzo del tag `input` di tipo `hidden` è diffusamente usato nella programmazione Web basata su CGI e Servlet per passare informazioni tra le componenti del sistema, nel nostro caso vengono passati alcuni parametri dal server al modulo JavaScript sul client e viceversa.

Le caselle di testo e il bottone sono contenuti in un form HTML. In particolare, la casella di testo usata per inserire la password è stata creata utilizzando un tag `input` di tipo `password` che è simile al tipo `text` ma, a differenza di questo, mostra degli asterischi mentre si digita la password. Il pulsante è creato mediante un tag `input` di tipo `submit`. La pressione del bottone attiva il modulo che si occupa della crittografia dei dati inseriti. Prima di applicare l'algoritmo il modulo controlla che le caselle di testo non siano vuote, altrimenti si invia un messaggio di errore all'utente chiedendo di immettere nuovamente i dati. Finita la fase di crittografia entra in azione il normale funzionamento di un form HTML che esegue un'operazione POST dello HTTP che serve a inoltrare tutti i dati inseriti

al server, in particolare al servlet che si occupa dell'autenticazione.

Interazione: L'utente registrato inserisce la username e la password che gli sono stati consegnati dopo la registrazione. Premendo il tasto *Login* si confermano i dati inseriti e si aspetta la risposta positiva o negativa del server.

6.4.3 Pagina di benvenuto

Una volta autenticatosi con il sistema, l'utente entra in una sezione introduttiva al vero e proprio lavoro. Vengono riepilogati i passi da percorrere per poter eseguire un'applicazione. La pagina offre, come illustrato in Figura 11, un menu laterale che permette di spostarsi velocemente in altre aree del sito. Da questo momento in poi comincia la sessione di lavoro dell'utente la cui traccia viene mantenuta sul server mediante gli strumenti di *Session Tracking* dei Servlet Java.

Funzionalità: Tra le voci dei menu troviamo i link:

- *Application Selection*: permette di visualizzare la lista delle applicazioni disponibili nel sistema;
- *Running Applications*: permette di visualizzare la lista di applicazioni ancora in esecuzione;
- *Applications Results*: permette di visualizzare la lista delle applicazione terminate con successo;
- *System Log off*: permette di uscire dal sistema interrompendo la sessione di lavoro;
- *System Help Index*: permette di accedere al sommario dello help in linea;
- *System Feedback*: permette di accedere a una pagina in cui l'utente può segnalare eventuali malfunzionamenti o presunti "bachi" del sistema.

Implementazione: La pagina è dinamicamente generata da un servlet (`Welcome`) quando l'utente si autentifica con successo al sistema. La pagina è stata strutturata in maniera da avere il nome del sistema sempre in primo piano, subito in secondo piano è indicato il titolo della pagina. Per l'organizzazione delle sezioni della pagina si è fatto uso delle tabelle HTML. I link del menu laterale attivano l'esecuzione dei servlet sul server Web.

6.4.4 Pagina contenente l'elenco delle applicazioni disponibili

Ha la stessa struttura della pagina precedente e vi si accede selezionando il link *Application Selection* dal menu laterale. Le applicazioni disponibili all'utente sono tutte e solo quelle che gli amministratori dei sistemi, sui quali queste sono installate, hanno concesso al nostro sistema, come illustrato in Figura 12.

Funzionalità: la pagina permette di visionare l'elenco delle applicazioni disponibili nel sistema e di potere quindi selezionare quella che si vuole eseguire.

Implementazione: la pagina è generata automaticamente da un servlet in esecuzione sul server Web il quale raccoglie le informazioni sulle applicazione dal server LDAP. I link che permettono di selezionare un'applicazione, se selezionati, attivano un servlet (`ApplInput`) che permette di passare i dati di input all'applicazione selezionata. Al servlet vengono passati due parametri mediante un *query string* HTTP che sono il nome e la versione dell'applicazione.

Interattività: L'utente sceglie l'applicazione da eseguire cliccando sul collegamento ipertestuale dall'elenco di applicazioni.

6.4.5 Pagina contenente l'elenco delle applicazioni in esecuzione

Ha la stessa struttura della pagina precedente e vi si accede selezionando il link *Running Applications* dal menu laterale.

Funzionalità: se l'utente ha attivato l'esecuzione di applicazioni che sono ancora in esecuzione, queste saranno presenti in questa pagina in un elenco che indica il nome dell'applicazione, la data, con ora, minuti e secondi dell'attivazione dell'applicazione e lo stato di avanzamento attuale dell'esecuzione, come illustrato in Figura 13. Lo stato indica all'utente se l'applicazione è in fase di completamento (trasferimento dei risultati), in fase iniziale (preparazione e trasferimento degli input) o ancora in esecuzione (running). Le applicazioni sono raggruppate in base all'identificativo di sessione (*session Id*) che è stato assegnato all'utente durante la sessione di lavoro in cui l'applicazione è stata eseguita. Questo tipo di ordinamento è stato scelto presumendo che in ogni sessione di lavoro l'utente esegue solo applicazioni utili ad uno stesso fine. Se nessuna applicazione è attualmente in esecuzione, il sistema suggerisce all'utente di andare a visitare la pagina dei risultati presentandogli il link a quella pagina.

Implementazione: Il lavoro di raccolta delle informazioni è fatta da un servlet (`Running`) che scandisce le aree temporanee del sistema per vedere quali applicazioni dell'utente non sono ancora terminate.

6.4.6 Pagina con i risultati delle esecuzioni

La sezione dedicata alla visualizzazione dei risultati è composta da due ben distinte pagine, una che visualizza e permette di gestire l'elenco delle applicazioni terminate con successo (Results Browser) e l'altra alla quale vi si accede per visualizzare il dettaglio di un'applicazione selezionata dall'elenco. Entrambe le pagine sono gestite da un unico servlet (`Results`).

Funzionalità: La prima pagina, illustrata in Figura 14, è *Results Browser*, che viene visualizzata selezionando *Application Results* dal menu laterale di una delle precedenti pagine e contiene l'elenco delle applicazioni completate con successo. Ogni applicazione ha a disposizione oltre alle informazioni sulla data e l'ora di esecuzione due link che permettono di visualizzare (*view*) i risultati dell'applicazione o di rimuovere (*delete*) i risultati dal server quando questi non siano più necessari.

Se si seleziona il comando *view* corrispondente a una delle applicazioni disponibili nell'elenco si accede a una nuova pagina che visualizza alcuni dettagli sui risultati. I dettagli da

visualizzare vengono decisi da chi installa l'applicazione sul sistema in base alla propria esperienza per quel che riguarda il trattamento dei dati di quella specifica applicazione. Questi dettagli potrebbero essere, ad esempio, grafici, tabelle o qualsiasi informazione prevista dall'applicazione.

Implementazione: I comandi a disposizione dell'utente per controllare i risultati sono stati realizzati mediante link allo stesso servlet che ha generato la pagina *Results Browser*. Ad esso sono inviati però dei parametri in più, entrambi prendono il nome dell'utente, il session Id dell'applicazione, il flag numerico che rappresenta la data in millessimi di secondo in cui è stata eseguita l'applicazione, e un'opzione (parametro *op*) che serve a distinguere se viene richiesta la visualizzazione del dettaglio (*op=view*) oppure la eliminizione del risultato (*op=delete*). Nel caso in cui l'utente scelga il comando delete viene attivato un modulo JavaScript che chiede conferma all'utente prima di proseguire.

La pagina usata per visualizzare i risultati di un'applicazione è dipendente dall'applicazione stessa, in quanto ogni applicazione può fare uso di *media* e meccanismi diversi per rendere visibili i risultati; in Figura 15 viene illustrata una pagina contenente i risultati di un'applicazione che restituisce in output dei grafici.

6.4.7 Pagina per l'immissione dei dati

La pagina, come illustrato in Figura 16, mostra il nome dell'applicazione selezionata, la sua versione, sua breve descrizione e come devono essere strutturati i suoi dati in ingresso.

Funzionalità: la pagina viene usata per passare i dati di input all'applicazione scelta per l'esecuzione. Le modalità di input previste sono tre:

- un modulo (*form*) che l'utente deve riempire indicando tutti i parametri necessari all'applicazione;
- *upload* di file, contenente i dati di ingresso per l'applicazione, questo metodo prevede che l'utente conosca il formato dei dati in ingresso all'applicazione;
- sorgente remota tramite collegamenti FTP, HTTP o TCP/IP in generale, a server di informazioni da usare come database online.

Implementazione: Per implementare il modulo di raccolta dati si è ricorso all'uso intenso di JavaScript e HTML. La non totale compatibilità con le specifiche di JavaScript e HTML dei due più diffusi browser Web in circolazione, Microsoft Internet Explorer e Netscape Navigator, ci ha costretto ad usare alcuni stratagemmi per cercare di rendere uniforme l'interfaccia su entrambi i browser e renderla allo stesso tempo il più *user-friendly* possibile. In particolare, una volta selezionato il metodo di immissione dei dati, l'utente vede solo quella parte di interfaccia relativa alla modalità selezionata. Si sono usati quindi tre livelli (`layer` per Netscape e `div` per Explorer) sovrapposti e visibili uno per volta in base alla scelta fatta dall'utente mediante lo *switch* (radio button) posizionato in cima al form. Per nascondere i livelli si sono usate alcune proprietà dei fogli di stile (CSS) [w3c] collegate a `layer` e `div` e resi dinamici mediante l'utilizzo di funzioni scritte in JavaScript. Ciascun

livello è responsabile di un form per l'invio dei dati il cui destinatario risulta lo stesso Servlet (`ClassBroker`) che si occupa della raccolta dei dati e dell'attivazione del codice java che gestisce l'applicazione da eseguire.

Interazione: L'utente deve seguire le istruzioni riportate nella pagina prima di attivare un'applicazione. Usare lo *switch* in alto per scegliere la modalità di input, e poi riempire il modulo che compare con i dati richiesti rispettando quanto viene raccomandato nella descrizione dei formati dei dati.

6.4.8 Pagine di aiuto in linea

L'utente ha sempre a disposizione, sul menù laterale, la voce *Help Index* che permette di accedere alle pagine di aiuto in linea per l'utilizzo del sistema.

Funzionalità: le pagine permettono all'utente di utilizzare al meglio il sistema. Queste costituiscono, in pratica, il manuale del sistema.

Implementazione: le pagine sono statiche e memorizzate sul server web nella zona pubblica. La prima pagina rappresenta un sommario degli argomenti presenti e ogni argomento è rappresentato da un collegamento ipertestuale alla pagina che ne descrive il contenuto. Alcune delle pagine, che sono collegate logicamente tra di loro, contengono collegamenti ipertestuali che permettono di accederle senza passare dal sommario. Il sommario dello *Help* è raggiungibile dal menu laterale, insieme alla voce *Personal Page* che permette di tornare alla pagina di benvenuto, se l'utente è connesso al sistema, oppure alla pagina di login se l'utente non si è autenticato con il sistema.

Interazione: Selezionare *Help index* in qualsiasi pagina per sfogliare lo help-on-line del sistema. Nel sommario cliccare su un link per accedere all'argomento specifico. Il link *Personal page*, che compare nel menu laterale, permette di tornare alla pagina di benvenuto personale o alla pagina di login.

6.4.9 Pagine non direttamente attivabili dall'utente

Tra le pagine Web create per la realizzazione dell'interfaccia ve ne sono alcune non direttamente raggiungibili dall'utente, ma che in caso di errore o malfunzionamenti del sistema possono comunque comparire sul browser dell'utente. Queste pagine rappresentano un modo per far sapere all'utente che si è verificato un problema mentre si cercava di soddisfare una sua richiesta, specificando il più possibile in dettaglio quanto accaduto e invitando l'utente a contattare l'amministratore di sistema mediante il modulo di feedback.

Funzionalità: Ciascuna pagina di errore, generata dinamicamente in base alle eccezione che di volta in volta possono verificarsi nei moduli del middleware, contiene un breve messaggio descrittivo, un codice di errore da indicare eventualmente nel form di feedback e il link alla pagina da caricare per proseguire.

Implementazione: Quando un servlet rileva un problema a run-time genera dinamicamente una pagina HTML per segnalare l'errore all'utente, viene visualizzato il tipo di errore, un messaggio di testo, un codice di errore fisso che serve all'amministratore per individuare il punto in cui questo errore si è verificato.

Interazione: L'utente dovrebbe aprire la pagina contenente il modulo di feedback e segnalare l'errore all'amministratore indicando il codice fornitogli dalla pagina di errore.

6.4.10 Accesso sicuro alle pagine

Generalmente le pagine di un sito Web sono memorizzate sul file system del server Web il quale, in questo modo, le rende disponibili a chiunque ne faccia richiesta specificando il percorso relativo del file, in genere in formato HTML. Nel caso di un sito Web con pagine generate dinamicamente da moduli *server-side* le possibilità di memorizzazione possono essere diverse, soprattutto nel caso in cui si voglia restringere ai soli utenti autorizzati l'accesso a quelle pagine. Nel nostro caso molte delle pagine sono generate dinamicamente da alcuni Servlet Java in esecuzione sul server Web, questi rispondono solamente a richieste di utenti autorizzati. Inoltre, lo sviluppo di pagine Web, che non risultino banali esteticamente e quindi complesse da progettare e implementare, però, non permette o rende difficoltoso l'incorporazione all'interno del codice sorgente dei servlet.

La nostra soluzione è stata quella di creare alcuni modelli di pagine, mediante programmi dedicati (editor HTML), che, di volta in volta, i servlet riempiono con le informazioni rilevanti per l'utente. Questi sono poi stati memorizzati tra gli entry del server LDAP, venendo a far parte del DIT di supporto per la nostra applicazione. In particolare è stato creato un entry *usersite* figlio della radice del DIT i cui attributi sono i modelli delle pagine HTML. Tra gli attributi di questo entry sono stati memorizzati anche i moduli JavaScript più complessi e di particolare importanza per la sicurezza. Tutto questo è stato fatto per cercare di garantire una certa sicurezza per le pagine del sistema e per ridurre al minimo l'accesso (anche se in sola lettura) al *filesystem* del server Web, l'alternativa sarebbe stata infatti quella di memorizzare le pagine in un directory non accessibile dall'esterno ma accessibile solo ai servlet. Un altro motivo che ci ha portato ad adottare questa soluzione è stato la possibile evoluzione del progetto: memorizzare le pagine su un server separato (come LDAP) permette a più server Web di accedere alle pagine che saranno presenti in singola copia su LDAP anzichè replicarli sul *filesystem* di ogni server Web.

7 Conclusioni e sviluppi futuri

Il presente lavoro ha avuto come obiettivo la progettazione e l'implementazione di *Meta*ψ, un prototipo funzionante di un ambiente per Web-Metacomputing per la costruzione di Problem Solving Environment (PSE). *Meta*ψ permette l'esecuzione remota di applicazioni tramite tecnologie standard Internet e, in particolare, Web. Tale approccio costituisce un primo passo verso la costruzione di prototipi di PSE in ambiente di Web-Metacomputing fornendo le funzionalità di calcolo necessarie a risolvere una determinata classe di problemi.

Metaψ è stato realizzato usando HTML e JavaScript per la parte client, il linguaggio Java e, in particolare, i Servlet per realizzare il middleware, LDAP per il servizio di *directory*, JNDI per l'interfaccia fra Java e LDAP e SSH per l'interazione con il backend. La scelta

sul server Web è ricaduta su Apache, per il server LDAP é stata usata l'implementazione di OpenLDAP, come Servlet Engine é stato usato JServ.

Allo scopo di migliorare e rendere più funzionale *Meta*ψ, durante la descrizione del sistema sono stati evidenziati quali componenti del prototipo potrebbero essere migliorati e questi riguardano principalmente quelli relativi a: bilanciamento del carico, sicurezza, creazione di metapplicazioni a partire da moduli preesistenti, creazione di applicazioni da parte dell'utente, possibilità di interazione diretta client-server LDAP e *steering*.

Bilanciamento del carico. In *Meta*ψ la selezione della macchina (o delle macchine, nel caso in cui l'applicazione sia parallela) candidata per l'esecuzione dell'applicazione avviene (se ci sono delle alternative) scegliendo quella che presenta il carico più basso. Il carico delle macchine facenti parte del backend viene monitorato tramite l'esecuzione remota di *uptime* (si è partiti dal presupposto che si abbia a che fare con macchine dotate di un sistema operativo Unix-like) e il valore rilevato viene riportato nel server LDAP aggiornando il campo relativo allo *entry* della machina. L'adozione di un'euristica di mapping più "sofisticata", che cerchi di bilanciare il carico, permetterebbe una migliore utilizzazione delle macchine del backend. Nel nostro prototipo, inoltre, è presente un unico server Web e, qualora ci fosse un numero elevato di client che richiedono i servizi offerti dal sistema, ciò potrebbe costituire un collo di bottiglia, con conseguente degradazione delle prestazioni di *Meta*ψ. La soluzione per ovviare a questo inconveniente consiste nel sostituire l'unico server Web con una rete di server Web, in maniera tale da garantire la ripartizione del carico generato dai client. In tale ottica, già nell'attuale implementazione le pagine HTML sono state memorizzate sul server LDAP, facilitando il processo della replicazione, in quanto non si rende necessario dover mantenere su ogni server della rete le pagine HTML. Un ulteriore miglioramento potrebbe aversi da un partizionamento e/o replicazione dei dati contenuti nel server LDAP; nell'implementazione scelta da noi, OpenLDAP, per fare ciò sarebbe necessario memorizzare i sottoalberi del DIT in vari server LDAP e configurare adeguatamente i demoni *slapd* e *slurpd*.

Sicurezza. Le comunicazioni fra client e server Web avvengono su canale insicuro, ma si potrebbe ovviare facilmente a questo inconveniente adottando un procollo "sicuro" come HTTPS.

L'implementazione scelta per il server LDAP, OpenLDAP, nell'attuale versione non supporta le comunicazioni su canale sicuro, pertanto tutte le comuncazioni fra Apache e LDAP viaggiano su canale insicuro. Per garantire la riservatezza delle informazioni scambiate fra i due server, sarebbe necessario sostituire OpenLDAP con un'altra implementazione di LDAP che annoveri fra le sue funzionalità l'uso di canali sicuri per le comunicazioni (ad esempio l'implementazione di LDAP di Netscape).

L'esecuzione remota di applicazioni residenti sul backend attualmente avviene tramite l'utilizzo di *secure shell* (SSH) e di *script shell*, ma in una versione futura di *Meta*ψ potrebbero essere utilizati anche strumenti di calcolo distribuito come CORBA.

Il prototipo attuale è vincolato all'esecuzione di ben determinate applicazioni, caratteristica che limita la generalità di *Meta*ψ. Capacità quali la **costruzione di meta-applicazioni** a partire da moduli preesistenti e la possibilità di **scrivere proprie ap-**

Figure 11: Pagina di benvenuto

Figure 12: Pagina contenente l'elenco delle applicazioni disponibili

Figure 13: Pagina contenente l'elenco delle applicazioni ancora in esecuzione

Figure 14: Pagina contenente l'elenco delle applicazioni terminate

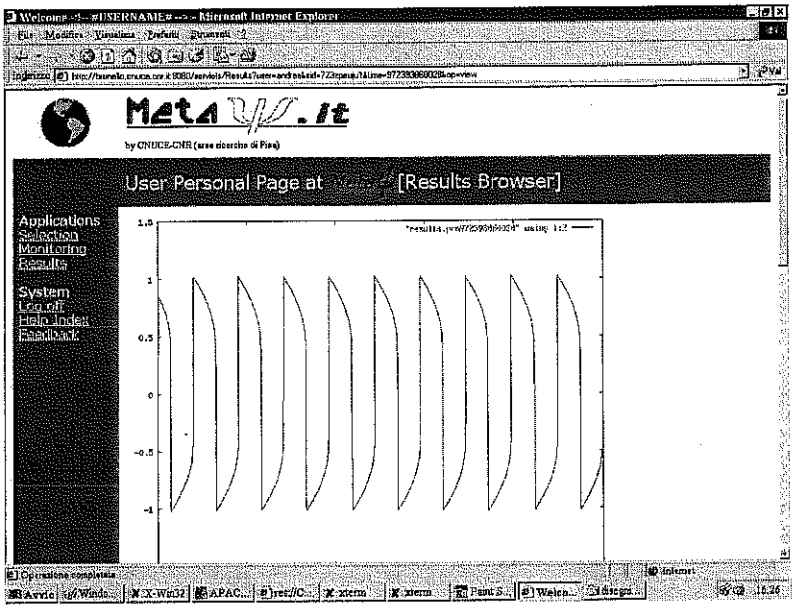

Figure 15: Pagina usata per visualizzare i risultati di un'applicazione

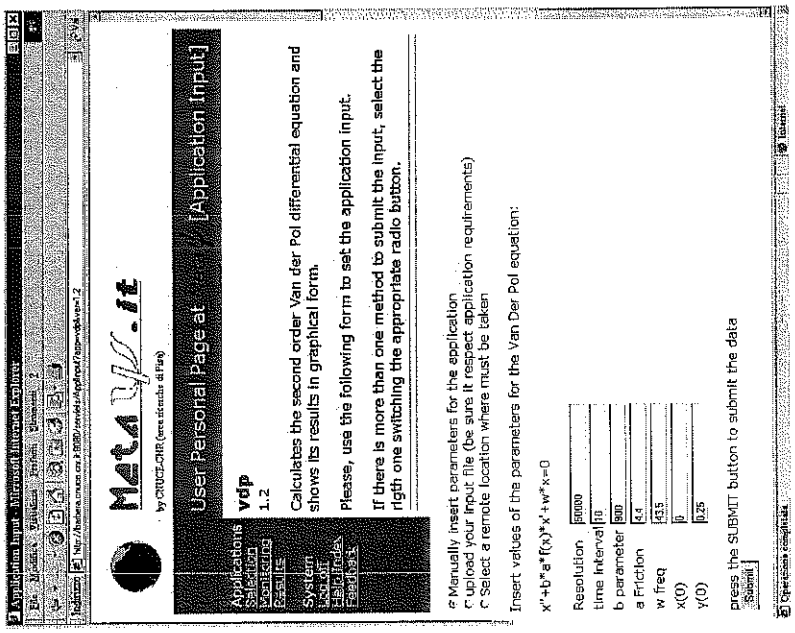

Figure 16: Pagina per l'immissione degli input di una delle applicazioni del sistema