LGMT

un progetto Free Software per la gestione di dati su Directory Server

Marco Marongiu

bronto@crs4.it

CRS4

Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna

Chi sono

Marco Marongiu lavora come Amministratore di Sistema presso il CRS4, un centro di ricerca situato vicino Cagliari. Il suo lavoro, combinato con la sua pigrizia e il suo interesse per le tecnologie del Web lo hanno alla fine trasformato in una sorta di programmatore Perl. Minacciato dalla noia di rimanere per troppo tempo a fare le stesse cose cambia continuamente identità, facendosi chiamare ora Dottor BOFH, ora Mr. JAPH[a].Dopo tanti anni di ingrato uso e abuso del free software ha finalmente deciso di contribuire qualcosa alla causa.

[a] BOFH: B... Operator From Hell; JAPH: Just Another Perl Hacker

Che cos'è LGMT

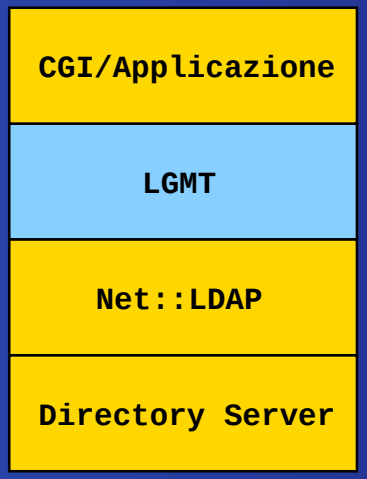
- LGMT è un framework per la gestione di dati su Directory Server attraverso il protocollo LDAP.
- Fornisce una interfaccia semplificata rispetto a Net::LDAP, su cui è basato;
- È un insieme di classi scritte in Perl che nascondono completamente all'utente le transazioni LDAP sottostanti.

Che cos'è LGMT

- Il directory server viene visto come se fosse un database, facilmente accessibile attaverso comuni script Perl.
- LGMT fornisce inoltre una serie di "Widget HTML" per rendere più facile e veloce la creazione di CGI.

Che cos'è LGMT

Concettualmente, LGMT è uno strato di interfaccia fra le applicazioni e la libreria Net::LDAP, il cui scopo è semplificare l'accesso al DS.



Free Software perché...

- può essere utile in diverse realtà (p.e.: gestione distribuita di address book aziendali, di database di utenti, di caselle e-mail per ISP...);
- è utilizzabile (almeno un minimo);
- ora il codice è presentabile (almeno un minimo);

I directory server

Un directory server è un database di tipo *gerarchico*. I dati sono aggregati in *entry*, strutturate in directory. All'interno di una entry i dati sono ordinati in associazioni chiave/valori; le chiavi sono dette *attributi*. Quali attributi siano specifici di una entry e quali di essi debbano essere obbligatoriamente definiti dipende dalle *object class* a cui afferisce la entry.

I directory server

top
account
posixAccount
person
organizationalPerson
inetOrgPerson
mailRecipient
nsLicenseUser
nsMessagingServerUser

Login name
Nome completo
Cognome
UID Unix
GID Unix
Home directory

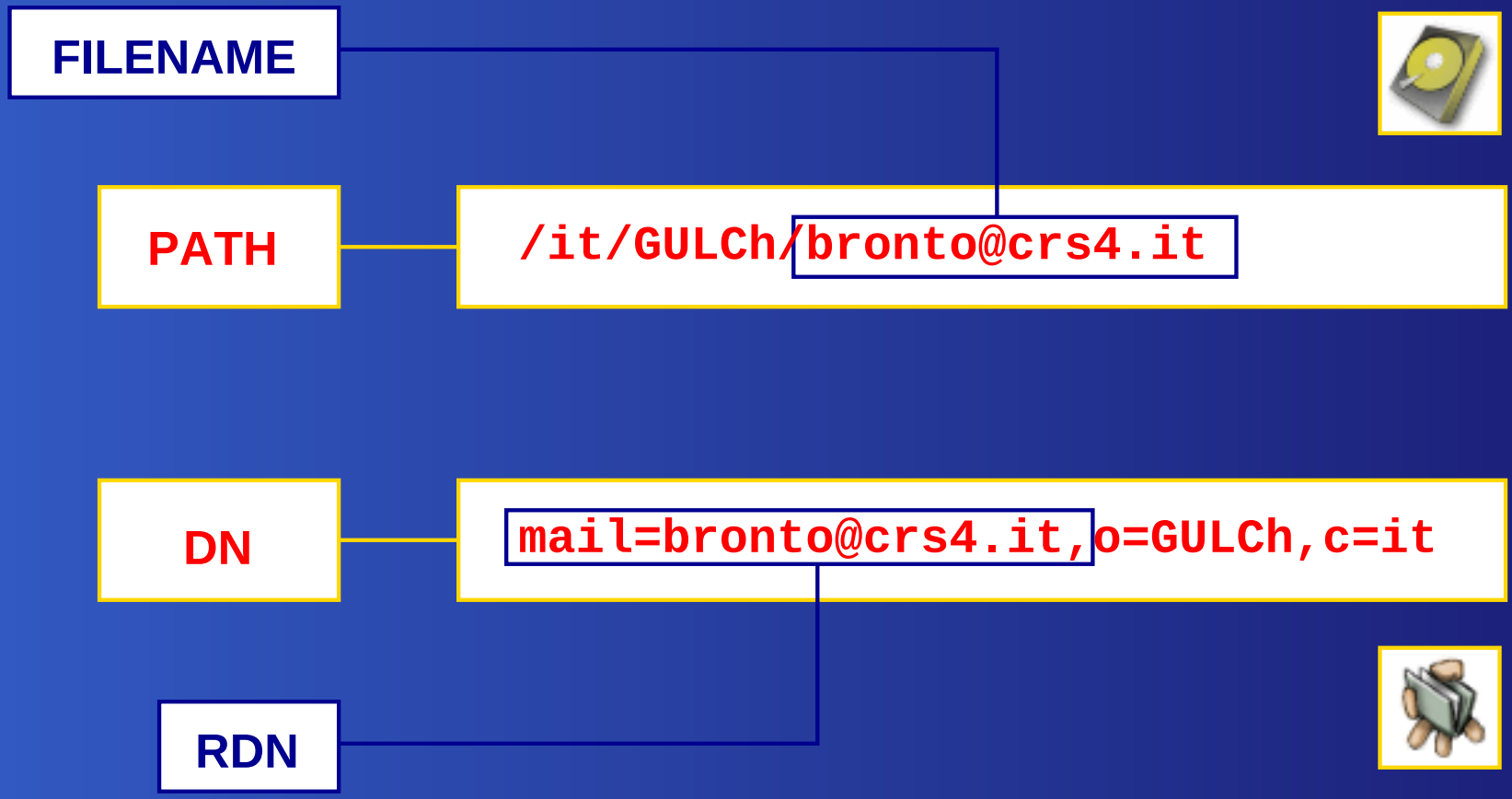
Workstation utilizzata
Descrizione
Organizzazione
Divisione
Shell di login
Password
Numero di telefono
Numero di fax
Indirizzo
Citta‘
Stato o provincia

Matricola
Foto in JPEG
URL Home page
Indirizzo di email
Numero di stanza
Nome della segretaria
Numero della patente
Iniziali
Alias di email
Altri indirizzi di email

I directory server

All'interno della singola directory ciascuna entry è riconoscibile attraverso il suo *Relative Distinguished Name* (RDN). All'interno del directory server ogni entry è distinguibile dalle altre attraverso il suo *Distinguished Name* (DN), formato dalla concatenazione degli RDN della entry e di tutte quelle precedenti, fino alla radice.

I directory server



I directory server

A differenza di un comune database relazionale, un directory server è progettato per rendere veloci le ricerche, non per essere robusto. In particolare, non è previsto niente di analogo alle transazioni. Per interrogare un directory server non si usa un linguaggio come SQL, ma un vero e proprio protocollo: LDAP (Lightweight Directory Access Protocol).

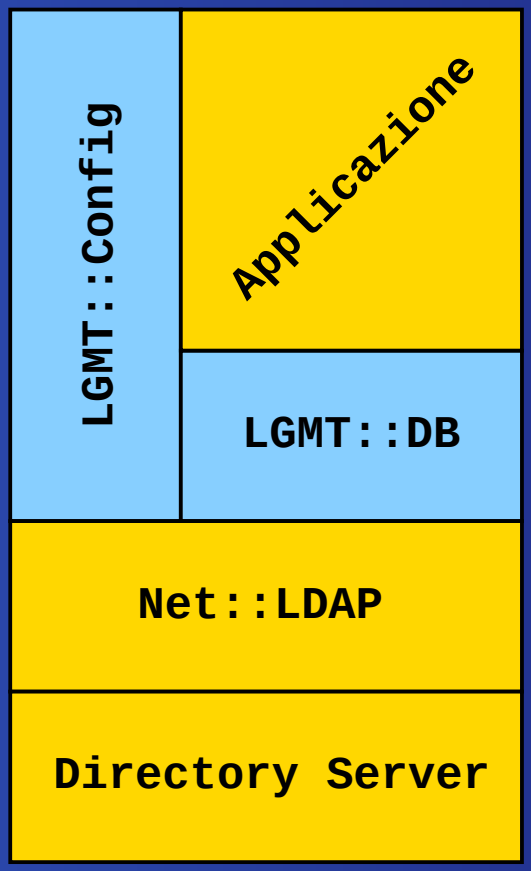
Il problema della gestione dei dati

La gestione dei dati su directory server è resa difficoltosa da due ordini di problemi. Una interfaccia più o meno comune è costituita da programmi a linea di comando, utile ma inadatta all'amministrazione day-to-day. I prodotti proprietari (p.e. iPlanet) dispongono di interfacce grafiche, che però rimangono ancora ad un livello troppo basso e sono estremamente pesanti.

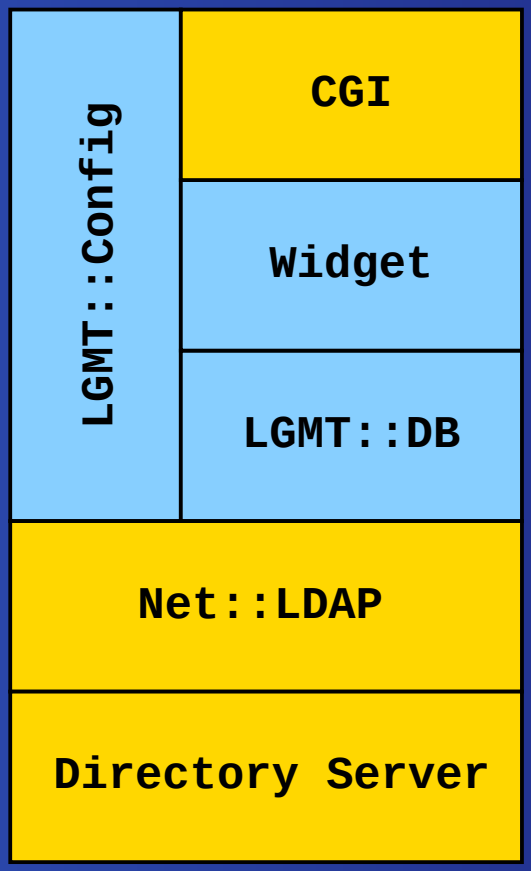
Case study: gestione NIS al CRS4

- migrazione da plain NIS al DS iPlanet + SUN NIS extenstions
- interfacce grafiche in Java pesanti e poco adatte a utenti inesperti
- impensabile l'uso di file LDIF e comandi `ldap*`
- necessità di un sistema distribuito, facile, flessibile e estensibile

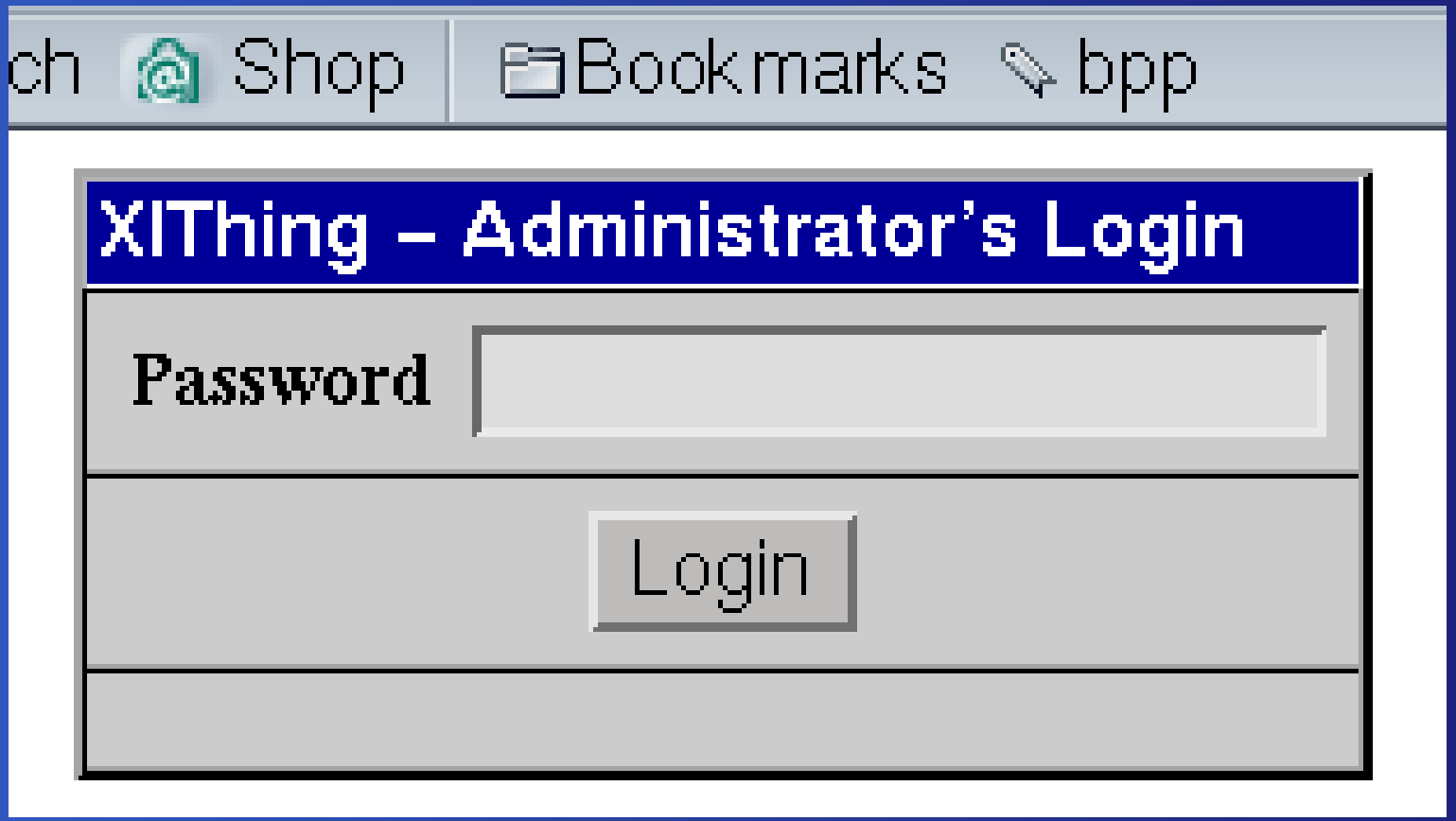
Applicazioni con LGMT



CGI con LGMT



I widget: uno snapshot di esempio



Prerequisiti per l'installazione

- Net::LDAP
- CGI::Pretty (opz.)
- Apache (opz.)
- mod_perl (opz.)

Per installarli il consiglio è il solito: usate il modulo CPAN!

```
perl -MCPAN -e shell
```

Installazione

Una delle deficienze della versione alpha attuale è che *NON* c'è una procedura di installazione (è una delle cose da fare). Si devono copiare i moduli nella directory delle librerie locali (p.e.: `/usr/local/lib/site_perl`)

Installazione

Per chi volesse usare LGMT per le interfacce web, le classi sono state scritte in modo da poter funzionare anche in ambiente mod_perl mediante `Apache::Registry`. Questo consente un notevole incremento di velocità di esecuzione, una volta che il web server ha cache-ato il codice.

Installazione

Nella directory `cgi` trovate le CGI `useradmin`, `homeadmin` e `aliasadmin`. Sono quelle che stiamo utilizzando al CRS4 per la gestione delle informazioni sugli utenti, le home e gli alias di e-mail rispettivamente. Ho cercato di commentare il codice il più possibile, quindi dovrebbero essere dei buoni punti di partenza.

Scrivere applicazioni con LGMT

Il dialogo fra le vostre applicazioni e l'interfaccia di LGMT verso il directory server (`LGMT::DB`) viene mediato dal modulo `LGMT::Config` e dalle sue sottoclassi. Nella classe base sono contenute le informazioni che tutte le sottoclassi devono condividere (p.e.: qual'è il directory server e la porta a cui connettersi). Le sottoclassi devono essere scritte insieme alle applicazioni. Anche di queste ce ne sono alcune nella distribuzione.

Qualche dettaglio sul codice

Per la parte che segue è necessaria una conoscenza, almeno minima, del Perl. Stiamo per illustrare alcuni brevissimi stralci delle classi che compongono LGMT, per dare un'idea di come scrivere una configurazione.

Dentro il codice: LGMT::Config

LGMT::Config contiene le informazioni che tutte le sottoclassi devono conoscere e condividere:

```
{
  my %DefaultConfig =
    ( LDAPServer     => 'ldapserver.domain.it',
      LDAPServerPort => undef,
      RootDN         => 'cn=SuperMan',
      RootPW         => 'WonderWoman',
  ) ;

...
```

Le sottoclassi di LGMT::Config

Definite le informazioni generali e fondamentali, bisogna definire tutto il resto, ad esempio:

- in quale directory si trovano le informazioni?
- che significato hanno gli attributi?
- Quanti valori posso assegnare? Uno? Molti? Uno/molti all'interno di un insieme predefinito?
- sono obbligatori? In quale contesto?

“Voi siete qui” e altre sciocchezze...

```
package LGMT::Config::People ;
...
use base 'LGMT::Config' ;
...
my %Config =
  (
   BaseDN           => 'ou=People,dc=crs4',
   RDN              => 'cn',
   DefaultOC        => [qw(top account...
 organizationalPerson inetOrgPerson...
```

Attributi: significato e tipi

A ogni attributo è associato un array di due o tre elementi: il nome, il tipo e, eventualmente, una reference a un hash.

```
Attributes      => {
   givenname => ['Nome','single'],
   nslicensedfor => ['Licenza uso','multi'],
   gidnumber => ['GID','slist',
{ generator => \&gidnumber_gen }],
```

I tipi *mlist* e *slist*

I tipi *single* e *multi* non hanno bisogno di grandi spiegazioni. Più interessanti sono invece le *mlist* (multiple selection list: elenco di valori per l'attributo con possibilità di selezionarne più di uno) e *slist* (single selection list: come sopra, ma con una sola selezione permessa).

{*m,s*}*list*: valori, label, generatore

Questi tipi sfruttano la hash reference associata all'attributo. Le chiavi di questo hash possono essere tre: `labels`, `values` e `generator`. Le prime due vengono utilizzate quando i valori possibili sono statici:

```
sex => ['Sesso','slist',
     { labels => ['Maschio','Femmina'],
       values => ['m','f'] },
```

{*m,s*}*list*: `generator`

Quando la lista dei valori deve essere creata o aggiornata dinamicamente (p.e.: da informazioni presenti su un directory server!) si può specificare la key `generator`, associandole come valore un puntatore ad una subroutine.

```
gidnumber => ['GID','slist',
    { generator => \&gidnumber_gen }],
```

La subroutine restituisce un hash del tipo precedente.

{*m,s*}*list*: generator

```
sub gidnumber_gen {
...
  foreach my $entry (@entries) {
    push @names,$entry->getattr('cn') ;
    push @gids,$entry->getattr('gidnumber') ;
  }

  return ( labels => \@names,
           values => \@gids ) ;
}
```

Quando LDAP non basta più

I concetti di attributo obbligatorio (cioè il cui valore debba essere necessariamente impostato) e opzionale è previsto nel protocollo LDAP. Vi sono però alcune situazioni particolari che non sono (e non potevano essere) previste dal protocollo.

Quando LDAP non basta più

- attributi con valori di default da impostare alla creazione di una entry;
- attributi obbligatori oltre a quelli definiti nello schema utilizzato;
- attributi che devono essere necessariamente impostati in fase di creazione di una entry;

Quando LDAP non basta più

Tutte queste situazioni vengono gestite dalle sottoclassi di LGMT::Config. Si veda, ad esempio, dove in LGMT::Config::People vengono definite le chiavi di configurazione `DefaultSet`, `MandatoryAttrs`, `MandatoryAtCreat`, `PublicAttrs` e `PrivateAttrs`, `CreationAttrs`.

Stato attuale

- attualmente ci sono tutti gli strumenti per gestire qualunque directory di dati; ogni directory è vista come un database;
- le CGI finora realizzate consentono di gestire i dati relativi ad alcune mappe NIS;
- tutto funziona, ma il codice è stato scritto di getto ed a più riprese distanti fra loro: deve essere migliorato e ripulito;

Stato attuale

- i directory server sono progettati per essere veloci, non per essere robusti; dove occorre la robustezza (p.e.: accesso concorrente) bisogna implementarla a monte;
- autenticazione e autorizzazione proprie dei directory server non sono sfruttate; di conseguenza alcuni dati “sensibili” (p.e.: password di amministrazione del directory server) sono visibili in chiaro sui moduli;

Stato attuale

- nella nomenclatura delle classi rimangono ancora delle tracce di LGMT+1, che devono essere eliminate;
- il progetto è iniziato poco prima dell'uscita del Perl 5.6, ed è quindi scritto in Perl 5.005; il porting a 5.6 può migliorare ulteriormente la qualità del codice;
- la documentazione POD è poca, e quella che c'è è, in alcuni casi, obsoleta;

LGMT: dove, come, quando

I sorgenti di LGMT sono liberamente scaricabili dall'indirizzo

`ftp://ftp.crs4.it/pub/lgmt`

e presto potrebbe andare anche su CPAN.

LGMT: dove, come, quando

È stata attivata una mailing list per gli sviluppatori:

`lgmt-dev@crs4.it`

L'iscrizione si effettua inviando un messaggio con subject vuoto e la parola `subscribe` nel corpo all'indirizzo

`lgmt-dev-request@crs4.it`

Informazioni su questo documento

Questo documento è stato realizzato con LaTeX utilizzando il package `prosper`; i grafici sono stati realizzati con `XFig`; gli screenshot sono stati realizzati con `The GIMP`.

`$Id: slides.tex,v 1.7 2001/12/08 16:35:50 bronto Exp $`

LGMT+2 ChangeLog

Come nasce

- Nasce come progetto personale per permettere la gestione del database dei soci del GULCh;
- la speranza era anche quella di razionalizzare la gestione delle mailing list, degli alias, dell'elenco dei soci. . . ;
- GMT+1 (GULCh Management Tools, versione 1) ha un frontend CGI; il backend è un file DBM;

GMT+1

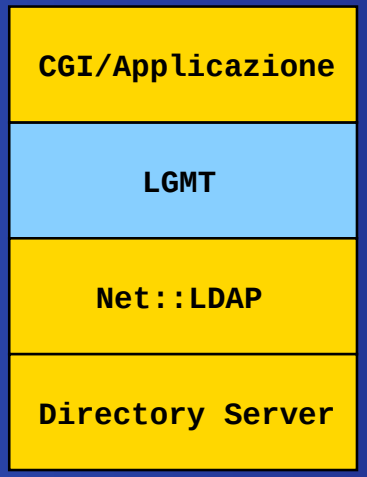
- i dati dei soci sono indicizzati tramite l'indirizzo di e-mail, tutti gli altri dati sono serializzati e conservati come valore associato all'indirizzo di e-mail;
- funziona, ma l'interfacciamento con il MTA e le mailing list è un pò "artigianale";
- non consente ai soci di aggiornare autonomamente i propri dati;

da GMT+1 a LGMT+1

- “scoperta” dei directory server, del protocollo LDAP e del formato LDIF;
- “scoperta” della possibilità di alcuni MTA (p.e.: postfix, qmail,. . .) di interfacciarsi direttamente ai directory server tramite LDAP;
- “scoperta” di un directory server aperto: OpenLDAP;
- “scoperta” della libreria `Net::LDAP`;

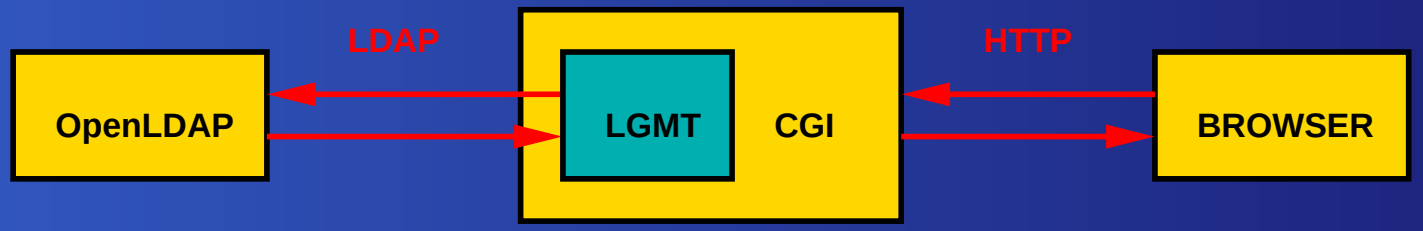
da GMT+1 a LGMT+1

- le scoperte mi stuzzicano, ma i moduli Net::LDAP sono un pò macchinosi da usare: decido di creare uno “strato di interfaccia” fra le mie CGI e il directory server;



LGMT+1

- nasce così la primissima versione di LGMT (LDAP GMT);
- è una serie di classi scritte in Perl e di alcune CGI;
- l'astrazione di LGMT dagli strati adiacenti è ancora approssimativa: le classi e le CGI sono ancora "interconnesse";



da LGMT+1 a LGMT+2

Lo sviluppo di LGMT subisce una piccola svolta quando al CRS4 ci si rende conto che serve uno strumento di quel genere per gestire le informazioni sugli utenti (DS iPlanet + SUN NIS extensions).

- il progetto viene “adottato” dal gruppo CNS (oggi NSM);
- la strutturazione diversa (e dispersa) delle informazioni costringe ad una revisione abbastanza radicale;

da LGMT+1 a LGMT+2

- la revisione di LGMT lo porta nella sua giusta collocazione: un framework slegato dalle CGI o da una particolare strutturazione dei dati;
- viene comunque mantenuta la libreria di “Widget HTML”;
- il frontend CGI viene anch’esso adottato dal CRS4 e continua ad essere sviluppato.