Treball Final de Carrera: Creació del nucli d'un servidor web

Memòria del projecte

Alumne: Antoni-Eric Corvera Bello

Consultor: Maria Isabel March Hermo

Curs 2011-2012, Semestre de Primavera
Juny de 2012

Aquesta versió del treball és la presentada al tribunal.
Trobareu qualsevol futura revisió al lloc web
http://tfc.corvera.eu

Historial de revisions d'aquest document:
1.1: Llicències finals
1.0: Versió presentada

Resum

Aquest TFC, emmarcat dins de l'àrea de xarxes de computadors, ha consistit en el desenvolupament d'un nou servidor web, procés detallat en la present memòria.

En termes generals, un servidor és un programa informàtic que comunica amb un altre, anomenat client, mitjançant unes regles ben definides, anomenades protocol.

En l'àmbit del web els clients més habituals són els anomenats *navegadors web* i el protocol en qüestió s'anomena HTTP. Un servidor web, doncs, és el programa amb què el navegador comunica quan visitem una plana web.
El desenvolupament d'un servidor web, en conseqüència, ha de produir un programa que *parli* la part de servidor d'aquest del protocol.

El protocol HTTP consisteix en un petit conjunt de funcionalitats obligatòries i una sèrie d'optatives. El servidor desenvolupat durant aquest TFC implementa una versió relativament extensa del protocol, incloent funcions poc habituals, en particular suporta els diferents modes de funcionament (anomenats mètodes), a més de característiques més comunes com la protecció mitjançant contrasenya o la compressió de continguts per estalviar ample de banda. El programa resultant, a més, compleix amb els requisits de ser multi-fil i multi-plataforma.

El desenvolupament del programari s'ha enfocat com un projecte formal en què s'ha seguit un procediment estructurat. Sobre la base d'un estudi del què ha de fer un servidor, s'ha dissenyat una arquitectura que satisfés aquests requisits per, sobre una base sòlida, procedir a la implementació per obtenir el producte final

Per a la implementació s'ha utilitzat el llenguatge C++ i un conjunt reduït de biblioteques de suport.

Índex de continguts

Índex d'il·lustracions

Índex de taules

Introducció

L'objectiu d'aquest projecte és desenvolupar un servidor web, un servidor que implementi el protocol HTTP d'acord amb l'especificació d'aquest.

Tot i que hi ha múltiples servidors web disponibles, el mercat es divideix entre molt pocs, en bona part degut a l'exigència que aquesta mena de servidors han de satisfer: han de tenir un bon rendiment, ser estables i, sobre tot, segurs. Com a encarregats de servir pàgines web, són en molts casos l'eina que hi ha darrera del *mostrador* de cara als clients: un mal servidor, un servidor que no funcioni, que no suporti moltes visites o què exposi dades privades, dona mala imatge.
És més, un servidor web és un programa inherentment multi-fil, cal atendre més d'un client alhora, i la programació multi-fil és tradicionalment força complicada per sí sola, fet que fa crear un producte de qualitat encara més difícil.

El present projecte pretén obtenir un servidor de qualitat acceptable però, sobre tot, un servidor que vagi una mica més enllà de l'habitual. Els servidors existents sovint no implementen totes les característiques del protocol, i aquí és on aquest projecte pretén aprofundir. No es planeja implementar el protocol sencer, el temps disponible no és suficient, però sí que s'implementaran parts molt poc comunes d'aquest.
Les característiques més destacables del servidor obtingut seran el suport dels diferents mètodes HTTP[1]: els comuns GET, HEAD i POST i els menys habituals PUT, DELETE, TRACE i OPTIONS; el suport de protecció i prohibició d'accés[2]; l'execució de CGIs[3] i la compressió dels continguts enviats al client[4].

Justificació i context

La tria per la meva part d'aquesta temàtica de projecte respon al meu interès per les tecnologies involucrades en la comunicació en xarxa, àmbit en què el *world wide web* és una de les formes de comunicació més populars.

La possibilitat d'aprofundir en el funcionament del web em va atraure especialment i d'entre els possibles projectes entorn aquesta àrea, la creació d'un servidor web em va semblar especialment interessant per poc freqüent; en efecte, com comentaré al llarg de la introducció, la quantitat de servidors web existents és força reduïda.

El fet de treballar en un servidor també em va atraure, ja que fins ara només havia treballat en programari client, i les idiosincràsies d'un servidor en són molt diferents.

Al començament del projecte només tenia una idea aproximada del funcionament del protocol HTTP i d'un servidor web, i nocions bàsiques de les tècniques necessàries per a la programació multi-fil i portable entre plataformes. Per tant per a mi el projecte no consistia només en completar el projecte de manera mecànica si no què, també, aprendre pel camí sobre aquestes àrees.

Objectius

Generals

L'objectiu general d'aquesta àrea de TFC és desenvolupar un projecte de principi a final relacionat amb l'àrea de xarxes de computadors, en aquest cas un servidor web, aplicant els coneixements adquirits durant els estudis. Per a assolir-ho caldrà:

- Estudiar al situació actual del servidors web
- Adquirir coneixements del protocol HTTP i tecnologies relacionades
- Comprendre, concretar i documentar les característiques o funcionalitats esperades d'un servidor web, mitjançant l'observació d'altres productes i dels estàndards que han de complir

1 Els mètodes HTTP es comenten a l'apartat homònim de la pàgina 19.
2 Les restriccions d'accés es comenten a l'apartat *Autenticació i seguretat*, a la pàgina 17.
3 El concepte de CGI es comenta a l'apartat *CGIs* (pàg. 15) i a l'entrada del glossari corresponent (pàg. 45).
4 La compressió de continguts es menciona breument en l'apartat *Codificacions i Transformacions* de la pàgina 23.

- Adquirir coneixements pràctics de disseny i programació de programes client-servidor i multi-fil (o multi-procés)
- Dissenyar i implementar el nucli d'un servidor web multi-fil (o multi-procés), i documentar aquest procés

Específics

Els objectius anteriors es concreten de cara a completar el present treball en els següents objectius específics:

- Redactar una introducció al funcionament dels servidors web, incloent un breu resum de la situació actual dels servidors existents
- Dissenyar i implementar el nucli d'un servidor web multi-fil i multi-plataforma
- Documentar el procés de disseny
- Evitar re-dissenyar/re-implementar funcionalitats ja proporcionades per les biblioteques, excepte les definitòries del projecte[5]

A continuació es defineixen les tasques concretes que el nucli del servidor web resultant haurà de ser capaç de dur a terme:

- Tractament de tots els tipus de peticions (GET, HEAD, POST, PUT, DELETE, TRACE i OPTIONS, vegeu la pàgina 19) d'acord amb l'especificació
- Suport de clients simultanis (multi-fil)
- Configurabilitat (port, nombre de fils simultanis, dades d'autenticació i permisos d'accés). Tant des de la línia de comandes com des de fitxers de configuració
- Compressió de dades automàtica quan el client ho suporti
- Acceptació de peticions amb *pipelining*[6]
- Implementació d'algun tipus de suport d'*scripting* (CGI(Glossari: CGI) o SSI(Glossari: SSI))

Enfocament i procediment seguit

Per a la realització del present projecte he dividit el treball en diferents fases temàtiques. La present memòria l'he anat redactant a mida que avançava en el projecte. Abans de començar-lo pròpiament vaig realitzar una planificació temporal aproximada que he intentat seguir, i que resumeixo en el següent apartat.

En una primera fase he procedit a recavar informació rellevant per a la realització del projecte, com són les especificacions del protocol HTTP, i m'he informat sobre la situació actual dels servidors web.

A partir d'aquesta informació he redactat una introducció a l'àrea del projecte amb la intenció de presentar una aproximació simplificada al funcionament dels servidors web.

Un cop definits els elements que conformen un servidor web, he procedit a realitzar l'anàlisi i disseny, emprant tècniques orientades a objectes. Inicialment vaig optar per una metodologia clàssica aplicant els conceptes del *Rational Unified Process*[7].
Però el poc marge per error de la planificació temporal, juntament amb la meva inexperiència en un projecte d'aquesta mida, em va convèncer que un mètode tant formal era poc adient i em vaig proposar treballar de manera més propera a la meva forma de pensar[8].

Vaig adoptar un model de disseny incremental, amb elements sense acabar de definir d'entrada i obert a canvis, entenent que cada fase del desenvolupament podia, i de fet ho ha fet, mostrar errors o decisions

5 Per exemple implementar el comportament del protocol HTTP és central al projecte, mentre que implementar la comunicació TCP/IP no ho és.
6 El pipelining s'explica a l'apartat *Autenticació i seguretat*, pàg. 17
7 Metodologia basada en els materials propis dels estudis, [Campderrich, 2004]
8 Metodologia basada en [MPW, 2007]

millorables de les fases anteriors. En aquest sentit

Per últim, i a partir del disseny obtingut anteriorment he procedit a la implementació, alhora que refinava el disseny i acabava de definir els elements pendents en successius cicles reduïts d'anàlisi i disseny.

Per a la implementació he començat per les funcionalitats bàsiques i he anat expandint a partir d'aquestes. He tingut cura d'afegir comentaris i documentació al codi, i de definir proves unitàries per algunes classes fonamentals.
Els RFCs[9] han estat la referència principal per a la implementació del comportament del servidor, amb ocasionals cerques a Internet per aclarir algun punt.

En tot moment he anat provant el codi contra navegadors web (principalment Firefox[10] i Chrome[11]) per comprovar que el comportament fos raonablement correcte, i he utilitzat Wireshark[12] per comprovar quines dades s'intercanvien realment entre navegador i servidor. El desenvolupament l'he dut a terme principalment sobre Linux i ocasionalment he canviat a Windows per validar la portabilitat del codi.

Primer he implementat l'escolta de peticions, l'anàlisi d'aquestes i la generació de respostes; començant per les peticions GET.

A continuació he implementat els índexs de directoris. La implementació d'aquesta funcionalitat bàsica ja implicava haver definit la majoria d'interfícies pel que a partir d'aquí les bases per implementar i provar la resta de mètodes ja estaven assentades.

A continuació he dissenyat i implementat els sistemes de configuració i de *logging*.

Arribats a aquest punt el servidor ja era funcional pel que he optat per començar amb les modificacions necessàries per obtenir un servidor multi-fil i, un cop completat aquest punt he acabat amb el disseny i implementació del suport de CGIs i de la comprovació de permisos.

Amb tots els elements del disseny implementats he finalitzat amb un repàs al funcionament del servidor per a assegurar que el comportament es correspon a l'especificació del protocol.

Planificació

Fase 1: Recollida de documentació

Duració aproximada: 2 setmanes (12 de març a 25 de març). **En paral·lel a la documentació de requisits.**

Tasques:

- Recollida de documents rellevants sobre el protocol HTTP i les seves extensions i possibles protocols relacionats
- Anàlisi d'altres servidors existents. Característiques i funcionalitats ofertes
- Recollida de documentació sobre el desenvolupament multithread i multiprocés
- Recollida de documentació sobre el desenvolupament d'aplicacions client-servidor
- Anàlisi inicial de possibles biblioteques de suport
  En aquesta fase n'hi haurà prou de trobar possibles alternatives. En les fases de disseny i implementació caldrà aprofundir per a aprendre a usar-les i integrar-les.
- Memòria: Inici

9 [RFC 1945], [RFC 2068], [RFC 2616]
10 Mozilla Firefox: http://www.mozilla.org/ca/firefox/
11 Google Chrome: http://www.google.com/chrome
12 Wireshark: http://www.wireshark.org/

- Versió inicial del primer capítol (introducció)
- Planificació inicial dels altres capítols
- Primers elements del glossari, bibliografia i possibles annexos

Fase 2: Documentació de requisits

Duració aproximada: ~2 setmanes (14 de març a 25 de març). **En paral·lel a la recollida de documentació.**

Tasques:

- Recollida de requisits en base la documentació recollida anteriorment
- Categorització de funcionalitats necessàries i funcionalitats desitjables. Aquestes només s'implementarien si el temps i la càrrega de treball ho permeten, com per exemple el suport d'SSIs(Glossari: SSI) o de CGIs(Glossari: CGI), que no formen estrictament part de les funcionalitats mínimes del nucli d'un servidor web.
- Identificació i documentació d'actors, casos d'ús i relacions
- Memòria: Primer esborrany
  - Explicació del procés seguit en aquesta fase, casos d'ús identificats i modelat inicial, en la memòria
  - Addicions a glossari, bibliografia i annexos si cal

Fase 3: Anàlisi

Duració aproximada: 10 dies (26 de març a 4 d'abril). **Encavalcada amb fase de disseny.**

Tasques:

- Definició de les classes d'anàlisi
- Elaboració de diagrama estàtic d'anàlisi
- Especificació formal dels casos d'ús
- Memòria: Primera versió (PAC1)
  - Descripció del procés seguit en l'anàlisi
  - Integració dels documents i diagrames d'anàlisi obtinguts
  - Addicions a glossari, bibliografia i annexos si cal

Fase 4: Disseny

Duració aproximada: 10 dies (30 de març a 8 d'abril). **Encavalcada amb fase d'anàlisi.**

Tasques:

- Identificació i integració de possibles patrons de disseny útils
- Integració en el disseny de possibles restriccions imposades per les biblioteques en ús
- Disseny dels casos d'ús
- Revisió del diagrama estàtic de disseny, adaptant-lo a característiques del llenguatge
- Memòria: Segona versió (PAC2)
  - Descripció del procés seguit en el disseny

    - Integració dels documents i diagrames de disseny obtinguts
    - Bibliografia: Referència a patrons utilitzats
    - Addicions a glossari i annexos si cal

Fita temporal: PAC1 (data límit 8 d'abril). Anàlisi i disseny complets

Fita temporal: PAC2 (data límit 6 de maig). Implementació començada (~50%)

Fase 5: Implementació

Duració aproximada: ~9 setmanes (9 d'abril a 7 de juny)

Tasques:

- Definició de jocs de proves i proves unitàries
- Implementació del disseny (ordre aproximat d'implementació)
    - Escolta i atenció de connexions (fil únic)
    - Analitzador de peticions
    - Validació de petició
    - Comprovació de permisos
    - Aplicació de capçaleres modificadores de la conducta (p.e. GET condicional)
    - Obtenció del recurs sol·licitat
        - Per a recursos dinàmics, generació del contingut
    - Extensió de la implementació al paradigma multi-fil
    - Memòria: Afegits de la fase d'implementació
        - Documentació del procés d'implementació
        - Problemes i solucions trobades
        - Possibles desviacions del disseny i classes de suport necessàries
        - Bibliografia: Referència a possibles documents extra consultats de cara a la implementació
        - Addicions a glossari i annexos si cal
        - Documentació de configuració i ús
        - Documentació d'instal·lació

Fase 6: Preparacions finals de la memòria

Duració aproximada: 2 setmanes (28 de maig a 10 de juny). **Encavalcada amb fase d'implementació i en paral·lel a la preparació de la presentació.**

Tasques:

- Memòria: Versió definitiva
    - Revisió de la memòria
    - Tria i indexació d'informació (i reorganització, si cal)

- Organització de possibles annexos

Fita temporal: Entrega de producte i memòria (data límit 10 de juny)

Fase 7: Preparació de la presentació

Duració aproximada: 3 setmanes (28 de maig a 17 de juny). **Encavalcada amb fase d'implementació i finalització de la memòria.**

Fita temporal: Presentació (data límit 17 de juny)

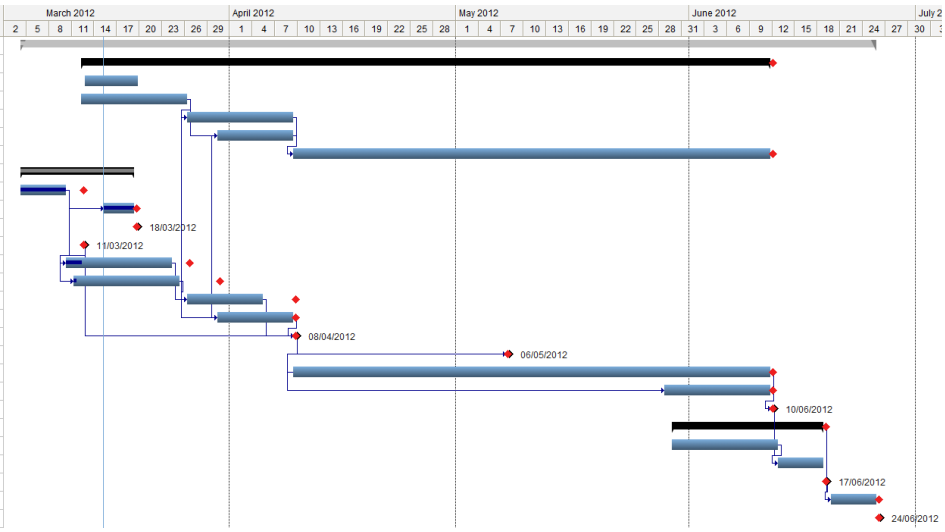
Fase 8: Preparació de respostes al tribunal

Temps: 1 setmana (18 de juny a 24 de juny)

Fita temporal: Respostes al tribunal (data límit 24 de juny)

Diagrama de Gantt

	Name	Duration	Start	Finish
0	⊟TFC: Servidor Web	113d?	03/03/2012	24/06/2012
1	⊟Memòria	91d?	11/03/2012	10/06/2012
2	Inici	1w	12/03/2012	19/03/2012
3	Documentació dels requisits	2w?	11/03/2012	25/03/2012
4	Documentació de l'anàlisi	2w?	25/03/2012	08/04/2012
5	Documentació del disseny	10d?	29/03/2012	08/04/2012
6	Documentació de la implementació	9w?	08/04/2012	10/06/2012
7	⊟0. Pla de treball	15d?	03/03/2012	18/03/2012
8	Pla de treball	6d	03/03/2012	09/03/2012
9	Pla de treball revisat	4d?	14/03/2012	18/03/2012
10	Lliurament de pla revisat	0d	18/03/2012	18/03/2012
11	§ Lliurament de pla de treball	0d	11/03/2012	11/03/2012
12	1. Recollida de documentació	2w?	09/03/2012	23/03/2012
13	2. Documentació de requisits	2w?	10/03/2012	24/03/2012
14	3. Anàlisi	10d?	25/03/2012	04/04/2012
15	4. Disseny	10d?	29/03/2012	08/04/2012
16	§ Lliurament PAC1	0d	08/04/2012	08/04/2012
17	§ Lliurament PAC2	0d	06/05/2012	06/05/2012
18	5. Implementació	9w?	08/04/2012	10/06/2012
19	6. Finalització de memòria	2w?	27/05/2012	10/06/2012
20	§ Lliurament de producte i memòria	0d	10/06/2012	10/06/2012
21	⊟7. Preparació de presentació	2.86w	28/05/2012	17/06/2012
22	Preparació inicial	2w	28/05/2012	11/06/2012
23	Realització sobre producte acabat	6d	11/06/2012	17/06/2012
24	§ Presentació	0d	17/06/2012	17/06/2012
25	8. Preparació de respostes	6d	18/06/2012	24/06/2012
26	§ Lliurament de respostes	0d	24/06/2012	24/06/2012



Desviacions del pla

Com he comentat, un cop encetada la fase d'anàlisi i disseny vaig optar per canviar el pla: Un cop acabades les fases d'anàlisi i disseny encara quedaven per definir alguns punts del disseny que vaig posposar per permetre'm començar la implementació abans.

La planificació temporal la he respectat bastant. Em vaig endarrerir una setmana amb el disseny i vaig decidir que era preferible continuar treballant la implementació fins al dia d'entrega (però amb cura de fer només modificacions lleus en els últims dies). A part, un cop vist que la presentació ha de consistir en un resum breu de la memòria he optat per deixar-la completament per la setmana posterior a l'entrega d'aquesta i així no restar-li temps.

Respecte el pla inicial també vaig canviar l'ordre proposat d'implementació per prioritzar les àrees en principi més problemàtiques i deixar per al final les funcionalitats no vitals (com ara l'execució de CGIs o la implementació de GETs parcials), així, per exemple, un cop vaig obtenir un servidor mono-fil amb la funcionalitat mínima, el vaig modificar per a ser multi-fil.

De les funcionalitats objectiu definides inicialment, la única (opcional) que no he completat en el temps de desenvolupament ha estat el suport d'SSIs. La manca d'estandardització junt al fet que caldria escriure codi d'anàlisi textual em van fer pensar que requeririen força més temps del que en podria dedicar.

Alguns dels aspectes opcionals del protocol tampoc han quedat implementats per manca de temps, en particular les peticions GET condicionals i parcials (vegeu el mètode GET a la pàgina 20).

Productes obtinguts

Els productes resultants d'aquest treball són tres:

- Producte final: Programari de servidor web, en forma de codi compilable. Implementa el protocol HTTP des del costat servidor, amb capacitat d'atendre diferents clients de forma simultània (multi-fil) i pot funcionar en diferents sistemes operatius (multi-plataforma). Suporta les peticions HTTP de tipus GET, HEAD, POST, OPTIONS, TRACE, PUT i DELETE, amb *pipelining*, comprimeix automàticament les respostes i permet configurar els permisos d'accés.
- Memòria: Aquest document. Es documenta el context de treball. Es recull i resumeix el procés seguit per a l'elaboració del programari, incloent les conclusions personals i un manual del producte.
- Presentació virtual: Síntesi del procés recollit a la memòria, en forma de presentació.

Capítols de la memòria

- Capítol I: Els servidors web (pàg. 13): S'explica en què consisteix un servidor web i com funciona el protocol HTTP en què es basa. Es proporciona una visió resumida del *mercat* dels servidors web.
- Capítol II: Desenvolupament: Recollida de requisits (pàg. 24): Primera fase del cicle de vida del projecte. Es documenta la fase inicial del desenvolupament en què es definiren els requisits que ha de complir el producte final, i possibles requisits addicionals. A més es comença el modelat.
- Capítol III: Desenvolupament: Anàlisi (pàg. 27): S'hi documenta el desenvolupament del projecte a partir de la informació recollida en el capítol II. Aquesta fase inclou la definició de classes de què constarà el programari.
- Capítol IV: Desenvolupament: Disseny (pàg. 31): S'hi documenten els canvis realitzats a la jerarquia de classes resultant del capítol III, un cop s'apliquen les particularitats o restriccions del llenguatge i les biblioteques utilitzades.
- Capítol V: Desenvolupament: Implementació (pàg. 33): S'hi documenten els detalls propis de la implementació i els canvis introduïts en aquesta.
- Capítol VI: Desenvolupament: Iteracions incrementals del disseny (pàg. 40): Detalla les fases posteriors de l'anàlisi i el disseny, dutes a terme en paral·lel a la implementació, a mida que l'aprofundiment en el projecte presenta noves àrees que cal dissenyar.
- Capítol VII: Possibles millores futures (pag. 45): Proposa algunes possibles funcionalitats que es podrien implementar en futures iteracions.
- Conclusions (pàg. 45): Impressions finals personals sobre el projecte.

I. Els servidors web

Un servidor és un sistema informàtic (programari i/o maquinari) encarregat de *servir* informació a altres sistemes, anomenats clients, en base a un protocol de comunicació que defineix com ha de dur-se a terme aquest intercanvi d'informació.

Es coneix com a servidor web aquell que implementa la part de servidor del protocol HTTP (*Hypertext Transfer Protocol*). L'HTTP és un protocol d'aplicació especificat per a treballar sobre el protocol TCP[13]. Per defecte el port TCP assignat al protocol HTTP és el 80.

HTTP és un protocol simple per disseny, orientat a l'intercanvi de *recursos*(Glossari: Recurs), generalment fitxers, de qualsevol tipus. Una sessió només consta d'una o més peticions per part del client i de les respostes corresponents per part del servidor; a diferència d'altres protocols no hi ha un intercanvi múltiple de missatges o negociació entre client i servidor[14].

L'especificació del protocol és explícitament permissiva, permetent una comunicació exitosa en molts casos encara que no s'implementi correctament.

Fins ara hi ha hagut dos revisions del protocol, la primera, HTTP/1.0, es defineix en l'[RFC 1945], de l'any 1996. Aquest document també defineix amb caràcter retroactiu una versió més simple del protocol, anomenada HTTP/0.9, que recull pràctiques anteriors a l'especificació formal d'aquest.

La versió actual i més habitual del protocol, HTTP/1.1, fou definida a l'[RFC 2068] el 1997, i posteriorment revisada i matisada (però sense canviar el número de versió) en l'[RFC 2616].

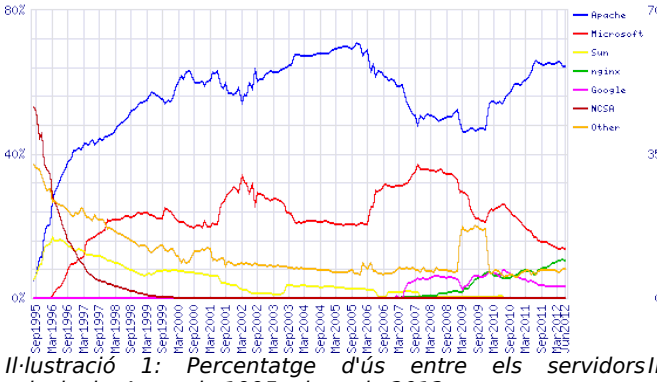
13 L'especificació d'HTTP només imposa l'ús d'un protocol de transport fiable, però alhora, només especifica l'ús de TCP – [RFC 1945], pàg. 8

14 Tècnicament en una mateixa sessió sí que es poden intercanviar múltiples missatges però constitueixen *transaccions* independents petició-resposta, vegeu l'apartat *Persistència de connexions i pipelining*, a la pàgina 16, per a més detalls.

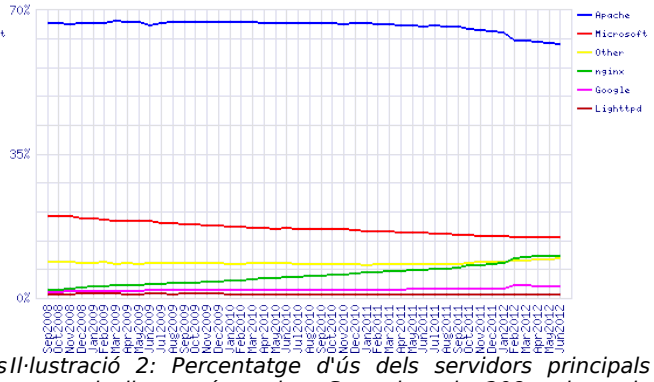
Ús actual dels servidors web

Segons les dades de Netcraft[15], a 6 de juny de 2012, els tres servidors més utilitzats són Apache, Microsoft/IIS i nginx; i els dos primers ho han sigut des de fa més d'una dècada ([Del Río, 2005], [Fontanals, 2003]).

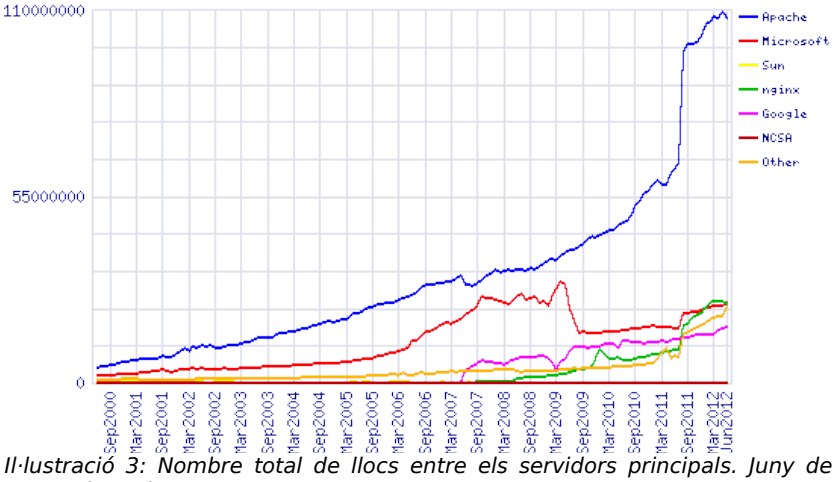
Com a curiositat, el quart més popular és el servidor web de Google, que només es fa servir internament en aquesta empresa (no està publicat), la seva presència entre els més populars es deu a la proliferació de pàgines allotjades als seus servidors (com ara els blogs a blogger)[16].



Il·lustració 1: Percentatge d'ús entre els servidors principals. Agost de 1995 a juny de 2012.



Il·lustració 2: Percentatge d'ús dels servidors principals entre els llocs més actius. Setembre de 208 a juny de 2012.



Il·lustració 3: Nombre total de llocs entre els servidors principals. Juny de 2000 a juny de 2012.

Gràfiques extretes dels resultats de juny del 2012 del web de Netcraft[17].

El servidor Apache[18], clar líder del mercat, és un programa de codi obert, publicat per primera vegada el 1995 a partir del codi d'un servidor anterior (l'NCSA HTTPd, un dels primers servidors web). Funciona en multitud de plataformes diferents si bé tradicionalment és més popular en plataformes POSIX (com la familia BSD i Linux).

El servidor IIS[19] (Internet Information Services), publicat per Microsoft, és gratuït i forma part del sistema

15 Netcraft és una empresa de serveis d'Internet que publica periòdicament, des de fa anys, estudis del web en què detalla l'ús dels diferents servidors web entre altres dades . El seu lloc web és http://www.netcraft.com/

16 Vegeu el primer informe de Netcraft que inclou Google entre els servidors principals per a més detalls: http://news.netcraft.com/archives/2007/09/03/september_2007_web_server_survey.html

17 http://news.netcraft.com/archives/2012/06/06/june-2012-web-server-survey.html

18 http://httpd.apache.org/

19 http://www.iis.net/

operatiu Windows des de la versió NT 3.51 (entorn el 1995), i també existeix una versió independent i reduïda anomenada IIS Express.

Per últim, el servidor nginx[20] és un programa de codi obert publicat per primera vegada el 2004 i la seva popularitat es deu, en part, al seu èmfasi en el suport de webs amb nombres elevats de visites simultànies.

Conceptes relacionats amb el protocol HTTP

El protocol HTTP utilitza conceptes d'altres especificacions que és recomanable conèixer per sobre abans d'entrar en els detalls del protocol. A continuació es comenten les codificacions de caràcters i els *media types*, solucions per a poder indicar com interpretar dades arbitràries, textuals i no-textuals respectivament.

Codificacions de caràcters

En un sistema informàtic el text es pot representar, a nivell binari, de diferents formes. La forma de representar el text s'anomena codificació de caràcters. Diferents idiomes fan servir diferents codificacions i, un mateix idioma pot fer servir múltiples codificacions. A més els diferents sistemes operatius també poden fer servir diferents codificacions per un mateix idioma.

Per fer-ho encara més complicat no tots els idiomes poden representar tots els seus símbols amb la mida mínima d'informació (un byte, 256 possibles caràcters diferents), i necessiten múltiples bytes per a representar un únic caràcter *lògic*.

Per últim existeixen les codificacions de l'estàndard Unicode (que inclou la majoria de caràcters en un únic conjunt) i que utilitzen (o poden utilitzar) múltiples bytes per a la seva representació.

Per aquestes raons, a l'hora de treballar amb dades textuals cal conèixer la codificació emprada, altrament és possible que s'interpreti el text de forma incorrecta (utilitzant els símbols incorrectes).

Media Types

Els media types s'utilitzen per identificar el "tipus" d'unes dades de forma unívoca. Són una cadena de text de la forma "`tipus/subtipus`" que defineixen com interpretar les dades a les que fan referència, que sense conèixer-ne el tipus només són dades arbitràries. A més es defineix un sistema per extendre els media types amb formats no registrats.

Un sistema alternatiu, utilitzat per exemple en l'SO Windows, és l'ús de l'extensió d'un fitxer per indicar el seu tipus: un fitxer de nom "`fitxer.txt`" és un fitxer de text sense format, "`fitxer.html`" és un fitxer en format HTML o "`fixter.mov`" és un vídeo en format quicktime.

Els exemples anteriors tindrien, respetivament, els tipus "`text/plain`" (text/planer), "`text/html`" i "`video/quicktime`".

Els *media types* es defineixen a la segona part de l'estàndard MIME(Glossari: MIME), l'[RFC 2046].

CGIs

Els CGIs(Glossari: CGI) són programes que segueixen una especificació que els permet col·laborar amb els servidors web, de manera que prenen part de les responsabilitats del servidor quan s'hi accedeix. El programa CGI serà el que processarà les dades enviades pel client i el què generarà les que se li enviaran.

L'ús de CGIs és una de les formes d'implementar webs dinàmiques, és a dir, pàgines web que canvien en funció de determinades condicions. Una pàgina web tradicional és estàtica en el sentit que el seu contingut es fixa en el moment de desar-la.

20 http://www.nginx.org/

Com a exemple simple, un CGI podria generar una pàgina web que indiqués quant espai lliure té el servidor, perquè un CGI és un programa que s'executa en cada visita.

Altres

Altres conceptes re-utilitzats, com són URIs o el format dels missatges HTTP (bàsicament missatges MIME) s'inclouen en l'explicació del protocol (a l'apartat següent) ja que en són una part fonamental.

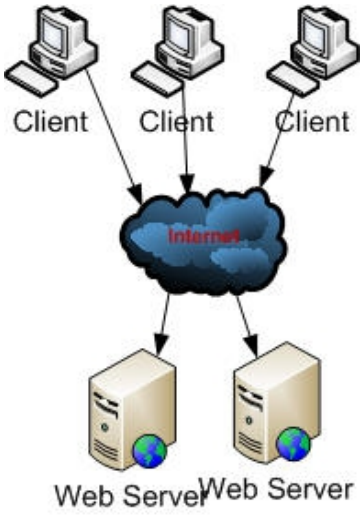
Funcionament del protocol HTTP

L'objectiu d'aquesta secció és presentar una explicació abreujada del funcionament del protocol HTTP. No s'incidirà en detalls com els formats o la sintaxi correcta de les dades intercanviades ni s'entrarà en detalls de tots els aspectes del protocol, en aquest sentit es recomana la lectura dels documents d'especificació del protocol ([RFC 1945], [RFC 2068] i/o [RFC 2616]).

Com ja s'ha mencionat el protocol HTTP és simple i segueix un model de petició i resposta: En una sessió HTTP, el client estableix la connexió cap al servidor i a continuació formula una petició sobre un recurs, a la qual el servidor respon amb el resultat corresponent.

A continuació, en HTTP/1.0 es tanca la connexió, tornant a començar el procés, mentre què en HTTP/1.1 es pot mantenir oberta per continuar enviant i rebent peticions i respostes.

Els missatges HTTP[21] tenen un format fortament influenciat per l'estàndard MIME(Glossari: MIME), utilitzat en el correu electrònic, amb algunes *relaxacions*[22] ja que a diferència del correu electrònic el protocol HTTP sí que suposa un transport segur[23].



Il·lustració 4: Visualització de comunicació client-servidor web
Imatge extreta de codeproject.

Persistència de connexions i pipelining

En HTTP/1.1 les connexions entre client i servidor, per defecte, es mantenen obertes (anomenades connexions persistents). Aquesta característica fa possible re-aprofitar una mateixa connexió per successives comunicacions entre client i servidor, reduint la càrrega de la xarxa i optimitzant recursos.

21 Tant les peticions com les respostes són missatges HTTP. Es farà servir aquest nom per a referir-s'hi indistintament.
22 Les diferències entre la versió corresponent de l'estàndard MIME i els missatges HTTP es recullen en els successius RFCs. Per exemple l'apèndix C de l'[RFC 1945] (HTTP/1.0) ho fa respecte l'[RFC 1521], mentre que l'apartat 19.4 de l'[RFC 2616] (HTTP/1.1) ho fa respecte l'[RFC 2045].
23 En contraposició, el protocol utilitzat per transferir correus electrònics està orientat a transferir dades textuals, pel que cal definir mètodes per transferir de forma segura dades "binàries",

Sobre el concepte de la persistència de connexions es defineix el de *pipelining*: un client no necessita esperar a rebre una resposta del servidor per enviar la següent petició re-aprofitant la connexió, de manera que el client pot enviar d'una sola vegada les peticions que consideri oportú. Per a garantir la coherència el servidor les ha de respondre en l'ordre en què les rep i només pot començar a respondre una després d'acabar amb l'anterior.

Autenticació i seguretat

Per a restringir l'accés a un recurs, un servidor pot sol·licitar l'autenticació al client, en forma de nom d'usuari i contrasenya (credencials), i denegar-li l'accés al recurs en qüestió si aquests no es reconeixen.

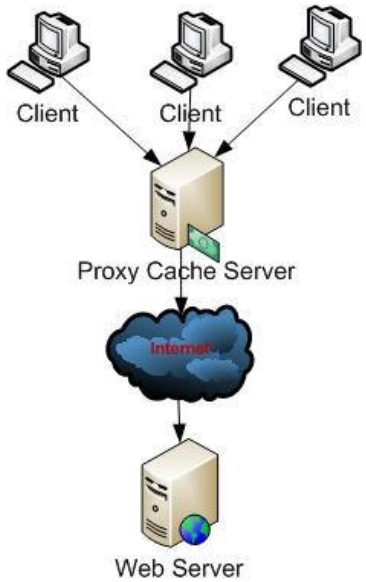
És important destacar que aquesta és una característica de privacitat i no de seguretat. La restricció d'accés no comporta cap mena de protecció de les dades intercanviades, que s'intercanvien de la mateixa manera que si no l'accés fos públic quan les credencials s'accepten. És a dir, algú amb capacitat per veure el tràfic entre client i servidor veurà aquestes dades sense protecció.

Per a proporcionar protecció respecte a escoltes o manipulació de dades intercanviades cal recórrer a l'HTTP segur (SSL/TLS), ja que el protocol HTTP no contempla l'intercanvi segur d'informació.

Amb l'ànim de solucionar aquesta limitació, Netscape va afegir al seu navegador una combinació del protocol HTTP amb SSL(Glossari: SSL) l'any 1994 ([Walls, 2005]) per a permetre encriptar el protocol HTTP, combinació que s'anomenà HTTP segur o HTTPS, combinació detallada en l'[RFC 2818]. A diferència de l'HTTP, el protocol HTTPS fa servir el port TCP 443 per defecte. L'estudi d'aquest queda fora dels objectius d'aquest treball i en conseqüència no s'aprofundirà més en ell.

Proxies

Un servidor també pot actuar de punt intermig (*proxy*), fent de pont entre el client i un altre servidor (que també pot ser un altre *proxy*, creant una cadena de *proxies* col·locats entre un client i el *servidor final*). La implementació d'aquesta funcionalitat implica que el *proxy* haurà de poder funcionar com un client de cara a connectar amb el següent servidor o *proxy*.
Aquest ús del protocol queda fora de l'àmbit d'aquesta pràctica i per tant no s'hi incidirà.



Il·lustració 5: Visualització de connexió entre servidor i client amb proxy/caché
Imatge extreta de codeproject.

Cachés

Una altra característica que un servidor (o proxy) pot implementar és la de "*caché*". Un caché emmagatzema còpies de les respostes generades pel servidor permetent respondre més ràpid si la petició es repeteix. Per exemple el primer proxy d'una cadena pot emmagatzemar el resultat d'una petició i quan se li torni a fer evitarà haver de continuar la cadena de proxies fins al servidor.

Per a garantir que no es retornen respostes incorrectes (respostes que ja no són vàlides) o inadequades (com dades privades d'un usuari a un altre), els RFCs del protocol defineix un mecanisme i una sèrie de normes per a determinar si una resposta es pot *cachejar* o si, un cop a la caché, segueix vigent.

Aquesta característica també queda fora de l'àmbit del projecte, la seva utilitzat és molt més marcada en proxies que en servidors finals. Cal mencionar que la implementació de GETs condicionals (el mètode GET s'explica a la pàgina 20) és una tasca associada que serveix a la implementació de cachés en els clients. Aquesta última característica sí es pretenia implementar però no s'ha dut a terme per manca de temps.

Elements d'un missatge HTTP

Una petició HTTP està composada per:

- Una línia de petició, que defineix les característiques bàsiques de la petició: el mètode de petició (vegeu l'apartat *Mètodes HTTP* a la pàgina 19), la URI sol·licitada (vegeu l'apartat *URIs* a sota), i la versió HTTP del missatge.
- Una sèrie de capçaleres HTTP (si s'escau) que defineixen característiques de la petició o del client. Vegeu l'apartat *Capçaleres* a la pàgina 20).
- Un cos de missatge (si s'escau). En el cas més comú la petició ja queda definida amb la línia de petició i les capçaleres, però algunes peticions poden incloure dades addicionals.

Una resposta HTTP està composada per:

- Una línia d'estat, que defineix el resultat de la petició: la versió HTTP del missatge, i un codi i frase d'estat (vegeu l'apartat *Codis i frases d'estat* a la pàgina 21).
- Una sèrie de capçaleres HTTP que defineixen característiques de la resposta o del servidor. Vegeu l'apartat *Capçaleres* a la pàgina 20).
- Un cos (si s'escau). En el cas més comú la resposta a una petició inclourà dades per presentar al client, com el contingut d'un fitxer, la sortida d'un programa o un missatge d'error.

URIs

Els URIs (*Uniform Resource Identifiers*, *Identificadors Uniformes de Recursos*) pretenen ser, com el seu nom indica, identificadors de recursos (elements qualssevol) amb un format estandarditzat. En el seu sentit més ampli els URIs poden definir qualsevol element del món (per exemple, un llibre d'una biblioteca, però també la localització geogràfica d'aquesta biblioteca). En el protocol HTTP, els URIs es fan servir per a identificar recursos(Glossari: Recurs) a la xarxa, i poden referenciar tant el recurs en qüestió com el servidor en què resideix (i ambdós alhora).

Un subconjunt dels URIs són els URLs (*Uniform Resource Locators*, *Localitzadors Uniformes de Recursos*), que se centren en el context de la comunicació en xarxa. En aquest context sovint es fan servir els termes URI i URL de forma intercanviable.

Un URI HTTP pot tenir diferents aparences ja que la majoria d'elements són opcionals. Els RFCs que defineixen el protocol HTTP també especifiquen formalment quin format han de tenir els URIs[24] a continuació es mostren alguns exemples:

`http://example.com`

Exemple bàsic. Protocol i nom de servidor. El número de port 80 és implícit al protocol HTTP.

`//example.com:80/recurs`

El protocol és implícit, s'especifica el número de port i la ruta dins el servidor del recurs.

`http://usuari:contrasenya@www.example.com:80/ruta/recurs.html#fragment`

S'inclou un nom d'usuari i contrasenya. *#fragment* és un identificador local que li correspondrà interpretar al client.

Alguns dels elements d'aquest exemple no s'enviaran al servidor tal qual: el fragment només és rellevant pel client (per tant no s'envia), mentre que el nom d'usuari i contrasenya s'indiquen

24 Vegeu l'apartat 3.2 de l'[RFC 2616] per a la especificació formal més recent del format utilitzat en HTTP

mitjançant capçaleres i no s'envien com a part de la URI.

Codificació d'URIs

Per a assegurar la correcta interpretació d'URIs ambigües o que utilitzen caràcters fora del rang segur de caràcters[25] les URIs, a nivell del protocol HTTP, s'envien amb una senzilla codificació que re-emplaça els caràcters problemàtics. Cada un d'aquests se substitueix per el signe de percentatge seguit dels dos caràcters hexadecimals[26] que es corresponen amb el caràcter en la taula ASCII.

Així si, per exemple, cal accedir al recurs "recurs # 3", caldrà codificar els espais (problemàtics) i el coixinet per evitar que s'interpreti com el començament d'un fragment. L'espai és el caràcter 20h ASCII, mentre que el coixinet és el 23h, per tant "recurs # 3" s'enviarà com a "`recurs%20%23%203`", eliminant tota ambigüitat.

Mètodes HTTP

Una petició HTTP pot tenir diferents intencions, com ara accedir un recurs o enviar dades al servidor. Per a definir la intenció del client les peticions inclouen el que es coneix com a mètode, definit l'especificació del protocol, i que determina què es vol fer amb el recurs identificat per l'URI de la petició. Els mètodes s'identifiquen per un nom (de vegades també anomenat verb). És pràctica habitual escriure els noms del mètodes en majúscules.

Posteriors revisions del protocol poden afegir nous mètodes, i per tant noves funcionalitats, sense afectar a la semàntica prèviament definida. Per exemple HTTP/1.1 va afegir dos mètodes (`PUT` i `DELETE`) que permeten als clients modificar els recursos del servidor, mentre que cap dels mètodes definits en HTTP/1.0 permetien modificar les dades del servidor.

La *pseudo-versió* 0.9 d'HTTP només té un mètode (per a la obtenció d'un recurs, equivalent al mètode GET explicat a continuació). En conseqüència el format de la petició és més simple i només indica quin recurs es vol obtenir.

25 Subconjunt de l'US-ASCII, vegeu l'apartat *Codificacions de caràcters*, a la pàgina 15
26 Amb dos caràcters hexadecimals es poden representar tots els possibles valors d'un byte

Mètodes HTTP/1.0	
`GET`	És el mètode bàsic: Sol·licita obtenir un recurs. Existeix una variant, *GET condicional*, en que només es retorna el recurs si aquest ha estat modificat des d'una data especificada pel client. HTTP/1.1 afegeix una altra variant, *GET parcial*, que permet obtenir només part del recurs (utilitzat, per exemple, per continuar una descàrrega interrompuda)
`HEAD`	Funcionament idèntic a GET, però sense incloure el contingut (cos) del missatge. Cal, però, retornar exactament les mateixes capçaleres que es retornarien amb un GET. Possibles usos inclouen la comprovació d'URIs o obtindre la mida d'un recurs sense descarregar-lo.
`POST`	La petició inclou un cos amb dades, que se sol·licita al servidor que processi en *relació* al recurs sol·licitat. Per exemple, si el recurs és un formulari, el cos contindria les dades resultants d'omplir-lo. L'acció que el servidor ha de dur a terme realment no queda definida i dependrà del recurs. A més no cal que el servidor conservi les dades rebudes més enllà del seu processament. Les respostes a aquest mètode no s'han de cachejar, excepte si inclouen capçaleres de control de caché que ho permetin.
Mètodes HTTP/1.1	
`DELETE`	Se sol·licita al servidor que esborri el recurs indicat. Una resposta exitosa no implica que el recurs s'hagi esborrat, només la intenció del servidor d'esborrar-lo.
`OPTIONS`	Obté del servidor les possibles opcions aplicables a un recurs. El recurs pot ser “*”, per indicar les opcions que suporta el servidor, sense aplicar-les a cap recurs en particular.
`PUT`	La petició inclou un element que se sol·licita al servidor que emmagatzemi en la URI del recurs indicada. Si ja existeix, es considera que és una modificació.
`TRACE`	El servidor final ha de retornar el cos de la petició en la resposta. Permet al client veure què es rep exactament a l'altre extrem, per a proves i diagnosis.
`CONNECT`	Reservat a l'[RFC 2616] per ús en proxies.

Taula 1: Mètodes HTTP

Tots els servidors HTTP tenen la obligació d'implementar almenys els mètodes `GET` i `HEAD`. La resta són d'implementació opcional i, cas de rebre un mètode no implementat, s'ha de comunicar aquesta situació al client.

El pla per al servidor documentat en la present memòria és implementar tots els mètodes excepte `CONNECT` (que no és aplicable ja que la funcionalitat de proxy no s'implementarà).

Capçaleres

Els missatges HTTP poden incloure una sèrie de capçaleres que defineixen informació addicional sobre el contingut del missatge i sobre el sistema emissor del missatge, tant en forma de meta-informació com de paràmetres modificadors del comportament.

La quantitat de capçaleres definides en les especificacions del protocol és notable, en aquest document només s'enunciaran les més comunes o rellevants.

Authorization	Utilitzada per a accedir a recursos que requereixin autorització.
Content-Encoding	Identifica una transformació inherent al recurs (per exemple, compressió). Vegeu l'apartat Codificacions i Transformacions (pàg. 23).
Content-Length	Mida del recurs.
Content-Type	Tipus de mitjà (*media type* o *tipus MIME*) del recurs. Per als tipus textuals també es pot indicar la codificació de caràcters. Vegeu els apartats Codificacions de caràcters (pàg. 15) i Media Types (pàg. 15).
Date	Data d'enviament del missatge.
Expires	Data de caducitat (de cara a cachés).
Host	Indica el nom de servidor i port extrets de la URI sol·licitada (degut a la resolució de noms, el servidor no podria conèixer aquests valors automàticament). Permet fer correspondre més d'un nom de servidor a una única adreça IP. Aquesta capçalera és un requisit en les peticions HTTP/1.1.
Last-Modified	Última data de modificació del recurs.
Referer	Indica l'adreça del recurs el qual ha portat al recurs actual (per exemple en seguir un enllaç).
Server	Identificador del servidor (per exemple nom i versió).
Transfer-Encoding	Identifica una transformació aplicada al recurs (per exemple, compressió). Vegeu l'apartat Codificacions i Transformacions (pàg. 23).
User-Agent	Identificador del client (per exemple nom, versió i sistema operatiu).
WWW-Authenticate	Indica el mètode d'autenticació necessari per accedir a un recurs.

Taula 2: Capçaleres HTTP més rellevants

(amb fons ombrejat, capçaleres introduïdes en HTTP/1.1)

Codis i frases d'estat

Les respostes del servidor han d'indicar, en la primera línia (*línia d'estat*), quin ha estat el resultat de la petició. El format utilitzat inclou un codi de tres xifres estandarditzat i una frase descriptiva (no estandarditzada, si bé sovint s'utilitzen les frases recomanades als RFCs).

La primera xifra del codi d'estat identifica el tipus d'estat, mentre que les altres dos identifiquen l'estat en concret. Un client que no conegui l'estat en qüestió pot utilitzar la primera xifra per saber quin tipus genèric de resultat ha obtingut. Els codis d'estat acabat amb dos zeros es fan servir com a codi d'estat genèric del tipus corresponent.

1xy	Informació. En HTTP/1.0 no utilitzat però reservat.
2xy	Èxit. Petició rebuda, interpretada i acceptada.
3xy	Redirecció. Cal realitzar alguna acció extra per a completar l'obtenció del recurs.
4xy	Error de client. La petició no es pot satisfer o és incorrecta.
5xy	Error de servidor. La petició sembla correcta però el servidor no l'ha pogut dur a terme.

Taula 3: Tipus de codi d'estat i significat

100	Continua
101	Canviant protocols
200	OK
201	Creat
202	Acceptat
203	Informació no autoritària
204	Sense contingut
205	Re-inici de contingut
206	Contingut parcial
300	Eleccions múltiples. Aquest codi no es fa servir directament, però serveix com a codi 3xx genèric.
301	Mogut permanentment
302	Mogut temporalment
303	Vegeu altres
304	No modificat
305	Utilitza *proxy*
306	Codi no utilitzat, però reservat. Definit a l'[RFC 2616]
307	Redirecció temporal. Definit a l'[RFC 2616]
400	Petició incorrecta
401	No autoritzat
402	Cal pagament. No utilitzat, reservat per usos futurs.
403	Prohibit
404	No trobat
405	Mètode no permès
406	No acceptable
407	Cal autenticació de *proxy*
408	Temps de petició exhaurit
409	Conflicte
410	Desaparegut
411	Cal indicar longitud
412	Precondició fallida
413	Entitat de petició massa gran
414	*Request-URI* (URI de la sol·licitud) massa llarga
415	*Media type* no suportat
416	Rang sol·licitat no satisfactible. Definit a l'[RFC 2616]
417	Expectació fallada. Definit a l'[RFC 2616]
500	Error intern del servidor
501	No implementat
502	Porta d'enllaç incorrecta
503	Servei no disponible
504	Temps de porta d'enllaç exhaurit
505	Versió d'HTTP no suportada

Taula 4: Codis d'estat estàndard i significat

(Amb fons ombrejat, codis introduïts en HTTP/1.1)

Codificacions i Transformacions

Un missatge HTTP pot incloure una o més *transformacións* del seu cos si el servidor i el client les suporten.

Les transformacions poden estar definides com a codificacions (capçalera `Content-Encoding`) o com a transformacions per a assegurar la integritat en el transport (`Transfer-Encoding`), a la pràctica però, ja que el transport del protocol HTTP garanteix la transmissió literal de les dades, els dos tipus de codificacions són pràcticament intercanviables. Tant `Content-Encoding` com `Transfer-Encoding` indiquen que el recurs està codificat amb el mètode indicat.

Content-Encoding generalment implica que el recurs està emmagatzemat al servidor en forma codificada però què la seva forma des-codificada és la "real".
Per exemple un servidor pot emmagatzemar i enviar les pàgines HTML comprimides i el client les tractarà com si hagués rebut text HTML, descomprimint-les automàticament.

Els `Content-Encoding`'s definits a l'[RFC 2616] són diferents tipus de compressió, a més de la identitat (cap codificació).

Transfer-Encoding indica que el recurs s'ha transformat específicament per a la seva transmissió.

Els `Transfer-Encoding`'s definits a l'[RFC 2616] són els mateixos que els `Content-Encoding`'s, i s'afegeix la transformació *chunked* (esbocinat) que divideix el contingut en múltiples bocins, indicant la mida de cada un i marcant clarament quin és el bocí final. Aquesta transformació és d'implementació obligatòria en clients i servidors HTTP/1.1.

La transformació chunked té la utilitat de permetre al client saber si ha rebut correctament tot el missatge en casos en què no es coneix la mida d'aquest, com ara quan s'aplica compressió al vol[27]. En HTTP/1.0 el tancament de la connexió marca el final del missatge (encara què no és condició suficient per garantir-ne la recepció completa) però en HTTP/1.1, amb la persistència de connexions, cal un mètode per indicar el final del missatge si la mida no es coneix d'entrada.

27 L'aplicació de compressió al vol (en el propi servidor) comporta el problema de conèixer la mida del contingut: Aquesta dada no es pot
conèixer fins que la compressió es completi, però completar-la abans de començar l'enviament suposa deixar el client esperant fins que
la tasca es completi i fer un ús potencialment alt d'emmagatzematge temporal.

II. Desenvolupament: Recollida de requisits

Llista de requisits

A partir de l'anàlisi inicial del domini i d'altres servidors podem definir la següent llista de funcionalitats requerides:

- Suport de les versions 1.0 i 1.1 d'HTTP
- Atenció de peticions simultànies en paral·lel (multi-fil)
- Port configurable
- Suport d'autenticació
- Suport de recursos dinàmics en forma de CGI
- Compressió automàtica quan el client la suporti
- Registre (*log*). Sortida simultània per consola i fitxer de registre.
- Funcionar en entorns UNIX/GNU i Windows
  - Necessari: Gestionar/reaccionar correctament als permisos dels arxius
- Suport de *pipelining* de peticions

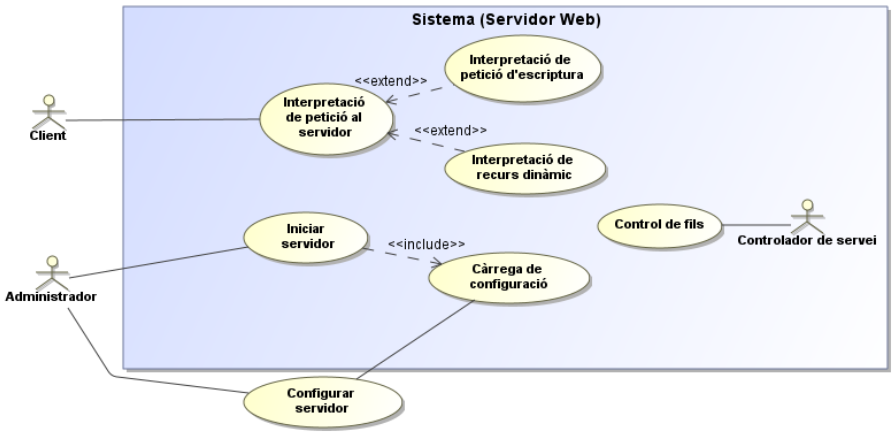
Identificació d'actors

Client. Representat per l'agent d'usuari. No forma part del model a implementar, extrem oposat del model (però el servidor podria actuar com a client en cas d'implementar funcionalitat de proxy o porta d'enllaç).

Administrador. Pot manipular la configuració del servidor. Generalment és un agent extern al sistema (mitjançant arxius de configuració o línia d'ordres). No forma part del model a implementar en ser extern.

Servidor. Aten les peticions del client.

Diagrama de casos d'ús



Il·lustració 6: Diagrama de casos d'ús

Casos d'ús, escenaris

Control de fils

Aquest cas d'ús descriu com es gestionen els fils. Aquí es descriu un model molt simple en què cada

connexió inicia un fil nou, encara que es preveu que el model utilitzat sigui una mica més complex. Més endavant s'aprofundirà en aquest punt.

1. L'administrador inicia el servidor
   Cas relacionat: Lectura inicial de configuració: Aplicació de permisos configurats
2. El servidor queda a l'espera de connexions
3. Un client connecta
4. El servidor crea un nou fil
   1. El nou fil passa a executar un dels cassos d'ús d'atenció de peticions
      Casos d'ús relacionats: Cas principal: Peticions només de lectura (GET, HEAD, POST, TRACE, OPTIONS), Cas alternatiu: Peticions d'escriptura (PUT, DELETE)
   2. El servidor queda a l'espera de connexions
   3. El nou fil termina

Cas principal: Peticions només de lectura (GET, HEAD, POST, TRACE, OPTIONS)

1 El client realitza una petició al servidor
   - Precondició: El servidor està escoltant

2 El servidor rep la petició

3 El servidor analitza la petició

3.1 La petició és incorrecta : **Error**

3.2 El servidor modifica el comportament en funció de les capçaleres rebudes

4 El servidor interpreta la ruta sol·licitada (vegeu *subcas* a continuació)

5 El servidor envia la resposta, que inclou el resultat i el recurs corresponent, si s'escau

5.1 Si el client suporta compressió, s'aplica al recurs transmès

5.2 Si es produeix un error de servidor, s'envia un missatge d'error

6 (HTTP/1.0) El servidor tanca la sessió

Subcas: Anàlisi de la ruta de la petició

A l'hora d'interpretar la ruta, es poden donar aquests supòsits:

4.1. La ruta és incorrecta : **Error**

4.2. L'accés al recurs no es permet : **Error**
Cas relacionat: Lectura inicial de configuració: Aplicació de permisos configurats

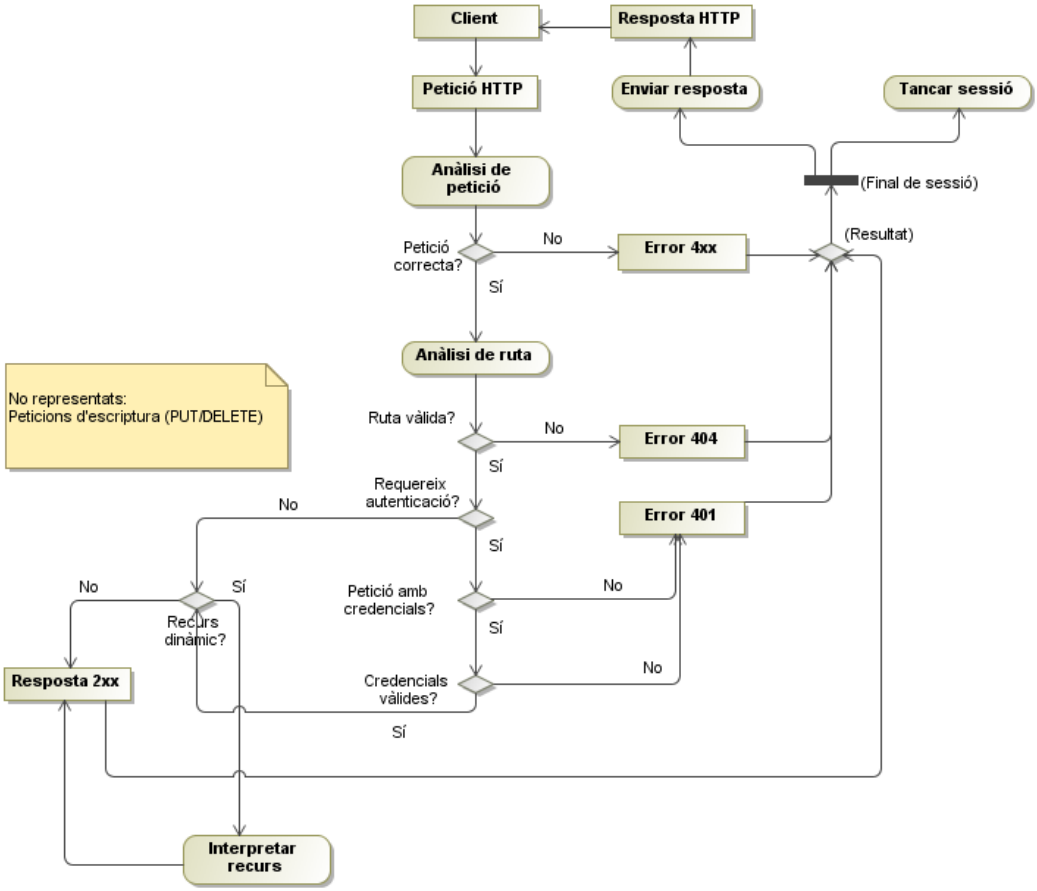
4.3. El recurs requereix autenticació
Cas relacionat: Lectura inicial de configuració: Aplicació de permisos configurats

4.3.1. La petició no inclou credencials o aquestes no són vàlides : **Error**

4.4. El recurs és dinàmic

4.4.1. S'interpreta el recurs per a generar el recurs *real* a retornar

Diagrama de flux d'una petició



Il·lustració 7: Diagrama de flux, cas d'ús principal

Cas alternatiu: Peticions d'escriptura (PUT, DELETE)

4.1. La ruta requereix autenticació
Cas relacionat: Càrrega inicial de configuració
4.1.1. La petició no inclou credencials o aquestes no són vàlides : **Error**
4.2. L'usuari no té permisos d'escriptura : Error

Lectura inicial de configuració: Aplicació de permisos configurats

Precondició: El servidor no està en funcionament

1. L'administrador arrenca el servidor
2. El servidor interpreta la configuració, definint els permisos de les rutes configurades (rutes prohibides, rutes amb autenticació, rutes amb permís d'escriptura)
3. El servidor es queda a l'espera de connexions

Altres consideracions

En el món real ens trobarem amb implementacions incorrectes o incompletes del protocol. Els RFCs reflecteixen aquest fet i proposen una actitud laxa al respecte: ignorar allò que no s'entén i possiblement corregir allò que s'entén encara que sigui incorrecte.

III. Desenvolupament: Anàlisi

A continuació es detalla el procés de definició de l'estructura del projecte. Es farà servir com a eina principal el diagrama de classes per la seva simplicitat i universalitat.

S'enfoca l'anàlisi i disseny acceptant que és gairebé impossible obtenir-ne un plenament satisfactori d'entrada i per tant es procedirà a modificar-lo tant com calgui a mida que s'aprofundeixi en el projecte. A més també cal acceptar que un disseny en contínua evolució pot fer palès que decisions de disseny anteriors no eren les més adients o inclús que eren incorrectes.

Identificació ràpida de classes: Anàlisi textual

En un primer pas de l'anàlisi s'ha aplicat la tècnica d'anàlisi textual, consistent en cercar noms en els casos d'ús[28], que sovint es correspondran amb classes en el sistema (i els verbs sovint ho faran amb mètodes d'aquestes classes). En aquesta primera aproximació apareixen:

> *Administrador, Servidor, Connexions, Client, Fil, Petició, Error, Capçaleres, Ruta, Resposta, Compressió, Resultat, Recurs, Sessió, Autenticació, Credencials, Recurs dinàmic, Configuració, Permisos, Resultat;* (i els noms qualificats *Ruta prohibida* i *Ruta amb permís d'escriptura*).

A més podem observar que *Client* i *Administrador* són elements externs al sistema (actors i iniciadors de casos d'ús), i que *Ruta prohibida* i *Ruta amb permís d'escriptura* pot ser la combinació de *Ruta* amb *Permisos* i *Configuració.*

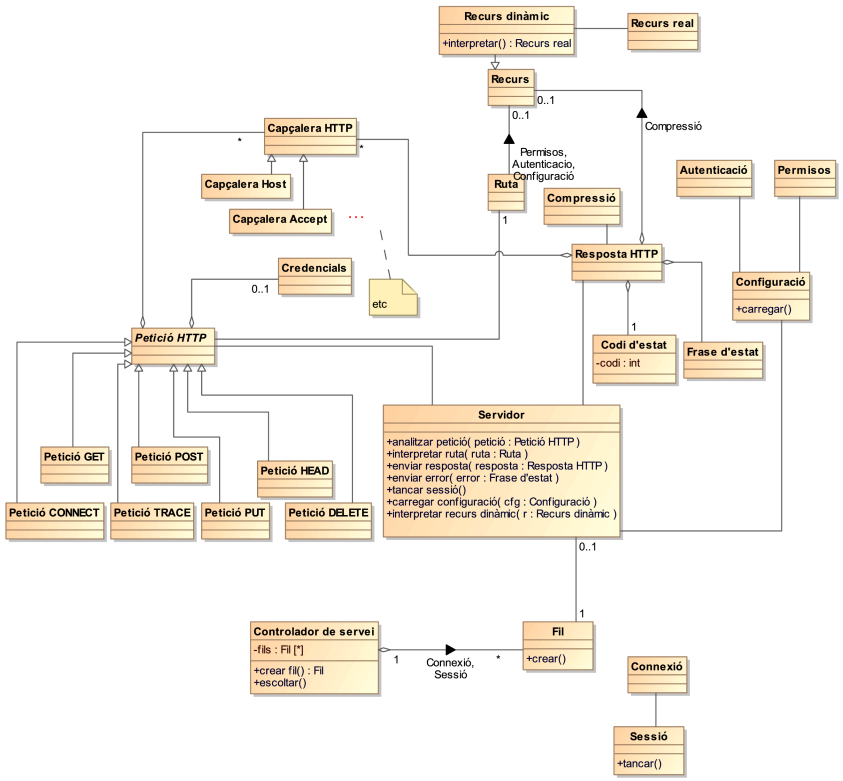
Aplicació dels coneixements sobre el domini

En aquest punt també podem establir algunes jerarquies de classes i identificar tipus que no apareixen explícitament als casos d'ús.

- Com s'explica a la descripció del protocol (pàg. 21) la resposta contindrà un codi d'estat i una frase d'estat, per tant en comptes de *Resultat* tindrem *Codi d'estat* i *Frase d'estat*. *Error* és un tipus de resultat.
- A l'anàlisi dels requisits es menciona que *Servidor* escoltarà peticions, crearà fils per atendre-les i les atendrà. En realitat estem parlant de dos classes diferenciades, amb diferent funcionalitat: el *Servidor HTTP* pròpiament dit, que atendrà les peticions, i el *Servidor* (o dimoni) que esperarà connexions i llençarà fils, per a eliminar aquesta ambigüitat s'ha anomenat aquest segon "*Controlador de servei*".
- Cada tipus de petició HTTP té un comportament diferent, per tant haurem de diferenciar-los: tenim peticions HTTP *GET, HEAD, POST, OPTIONS, TRACE, PUT* i *DELETE*.

28 Malgrat que l'aplicació d'aquesta tècnica és desaconsellada per alguns autors (p.e. [Campderrich, 2004], M5, pàg. 14), el seu ús és comú i d'altres sí que la consideren útil ([MPW, 2007], pàg. 177)

Diagrama de classes de l'anàlisi, aproximació inicial



Il·lustració 8: Diagrama de classes d'anàlisi, primera aproximació
Nota: Hi ha multitud de Capçaleres estàndard, per simplicitat no s'han representat totes en l'esquema.

Interpretació

En aquest diagrama tenim que *Controlador de servei* serà l'encarregat de gestionar les connexions entrants i els fils corresponents. Cada *Connexió* iniciarà una *Sessió* que s'atendrà en un *Fil*, i un *Servidor* serà l'encarregat de processar la *Petició HTTP* rebuda i enviar una *Resposta HTTP* corresponent a partir de la *Configuració*, que mitjançant *Permisos* i *Autenticació* definirà si l'accés a la *Ruta* sol·licitada en la *Petició* és permès i si calen *Credencials* (què són una característica de la *Petició*).

Les *Peticions* i les *Respostes* contenen *Capçaleres HTTP* per a definir el seu comportament.

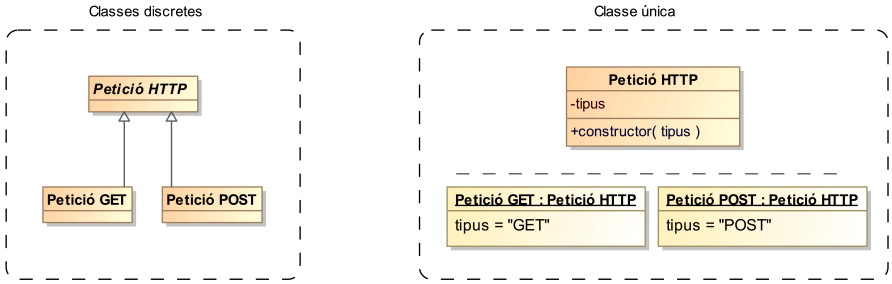
La *Ruta* a la que fa referència la *Petició* es pot correspondre a un *Recurs*, i un cas particular de *Recurs* és el *Recurs dinàmic*, que un cop *interpretat* produeix el *Recurs real* que s'enviarà en la *Resposta*.

La *Resposta* consistirà en un *Codi d'estat*, una *Frase d'estat* i opcionalment el *Recurs* sol·licitat en la *Petició*, que pot enviar-se comprimit si el client suporta la *Compressió*.

Refinaments

El diagrama de classes anterior proporciona una primera aproximació i una visió general del disseny, a partir de l'estudi d'aquest podem aplicar uns primers refinaments i identificar detalls passats per alt fins ara:

- Es pot veure que la quantitat de classes de Petició és innecessàriament gran: totes elles funcionen de la mateixa manera i el diferent comportament que representen recaurà sobre *Servidor*, per tant no cal tenir una subclasse per a cada tipus de petició. *Petició HTTP* deixa de ser abstracta i podem eliminar les diferents variants, i representar-les com un atribut:



Il·lustració 9: Reducció de nombre de classes de petició

- Amb *Capçalera* passa el mateix, els estàndards defineixen un nombre considerable de capçaleres i cada una només representa una associació nom-valor (la semàntica dependrà del servidor), per tant es poden eliminar els subtipus de *Capçalera.*
- *Recurs dinàmic* és un tipus especial de *Recurs*, que un cop executat produeix el *Recurs real*. Per tal de reduir l'ambigüitat del nom, reanomenarem aquest com a *Recurs generat*.
- *Tanmateix Credencials representa en realitat una capçalera concreta (“`Authorization`”, [RFC 1945], pàg. 49).*

Subdivisions

Com s'ha mencionat es pretén seguir un model iteratiu i incremental. En aquest sentit podem subdividir les classes identificades fins ara en diferents paquets funcionals, cada un dels quals aportarà noves funcionalitats al programari. A partir d'aquesta divisió aprofundirem en el desenvolupament de la funcionalitat mínima necessària, deixant esbossada la resta de funcionalitats amb la idea d'aprofundir-ne en posteriors iteracions. Cal mencionar que aquesta divisió en paquets funcionals només té la funció de facilitar el desenvolupament incremental, i que per tant no es plasmarà en el producte final.

Paquets principals. Funcionalitats centrals del projectes:

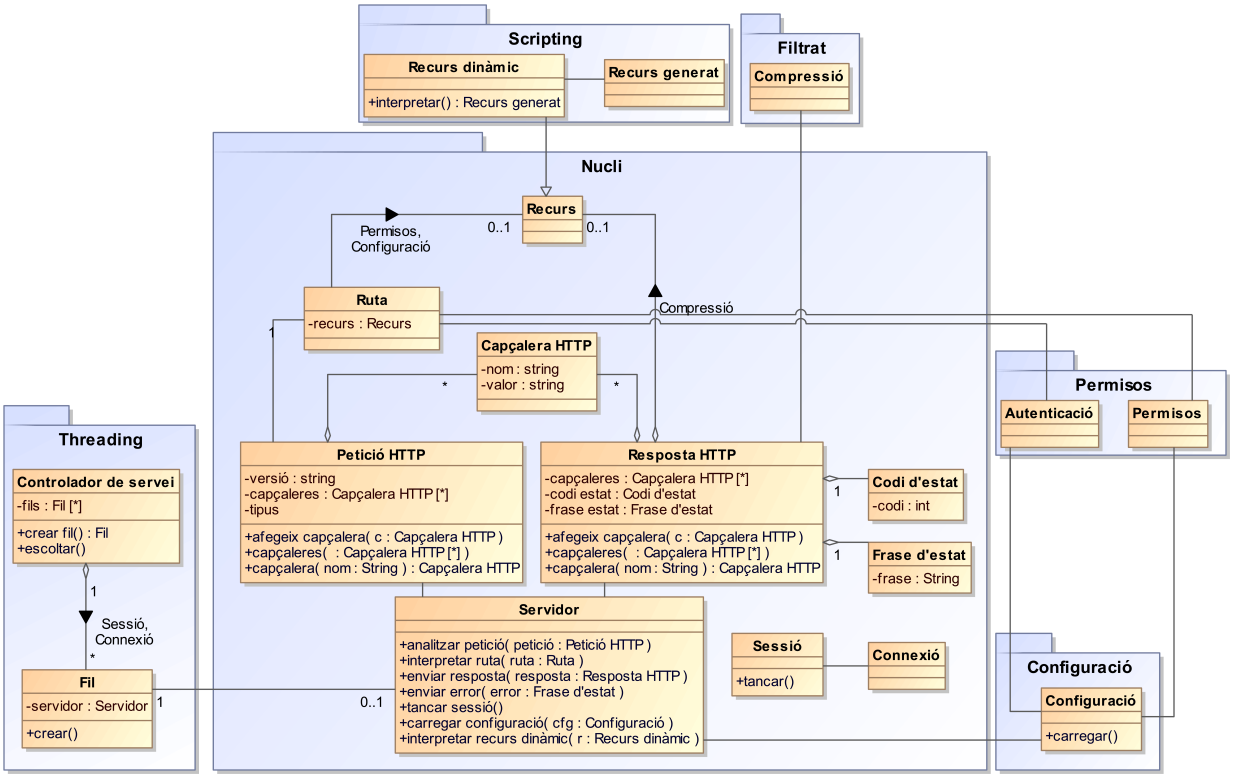
- **Nucli**: Funcionalitat bàsica del servidor. Amb només aquest paquet el servidor ja seria funcional.
- ***Threading*** (fils)**:** Capacitat per atendre peticions simultànies. Aquesta funcionalitat és vital en un servidor en producció.

Paquets secundaris. Elements del projecte desitjables però no primordials:

- **Configuració**
- **Permisos**
- **Filtrat** (compressió)
- ***Scripting***: Suport de recursos dinàmics

Diagrama de classes d'anàlisi definitiu

A partir dels refinaments anteriors i la divisió de funcions obtenim un nou disseny. Els paquets en què no hem aprofundit fins ara queden esbossats de cara a posteriors iteracions:



Il·lustració 10: Diagrama de classes d'anàlisi
(Nota: S'han obviat els mètodes «setters» i «getters» dels atributs per simplicitat)

IV. Desenvolupament: Disseny

En la fase de disseny es procedirà a refinar l'estructura del projecte a partir de consideracions del món real, com les restriccions imposades per el llenguatge de programació o les estructures de dades predefinides disponibles.

A partir dels refinaments introduïts a continuació i els canvis incrementals detallats en el capítol VI (pàgina 40) s'anirà modificant el disseny anterior; al final d'aquest (pàg. 43) es presenta un diagrama de classes definitiu que recull aquests canvis.

Elements introduïts en el disseny

Tria del llenguatge de programació

L'enunciat del TFC proposava l'ús de Java o C, però s'ha optat per l'ús de C++, el qual també compleix la condició principal, que és permetre assolir els objectius fixats a l'inici del projecte.

Respecte C, l'aplicació de tècniques de disseny orientades a objectes fa que el suport nadiu del paradigma d'orientació a objectes sigui desitjable per reduir les tasques de reinterpretació, fent una mica C menys adient; a més, C++ permet eliminar la major part dels problemes típics de la programació en C, com ara la gestió correcta de memòria dinàmica[DevX, 2008].

La elecció de C++ sobre Java respon més a un criteri personal. Ambdós llenguatges són opcions igualment vàlides donats els objectius del present projecte. Ambdós disposen de biblioteques madures i de qualitat, ja siguin les estàndard o suplementàries.
Al llarg dels estudis el llenguatge en què s'han treballat les pràctiques sempre ha estat Java. Triar C++ suposa un repte afegit i la possibilitat d'aprofundir en els coneixements d'un llenguatge i un paradigma diferent.

Adaptació de noms de classes

Caldrà adaptar el disseny anterior a les normes de sintaxi del llenguatge, per exemple les classes "*Petició HTTP*" i "*Capçalera HTTP*" passen a anomenar-se "*PeticioHTTP*" i "*CapsaleraHTTP*" respectivament.

Biblioteques

Un dels objectius del projecte és no *re-inventar la roda* més enllà de la construcció del nucli del servidor, és a dir, sempre que sigui possible es farà ús de biblioteques, excepte en la implementació de la funcionalitat central del projecte.

És ben sabut que el llenguatge C++ només proporciona una biblioteca estàndard bàsica (que inclou principalment entrada/sortida, contenidors i algorismes); a més de la biblioteca estàndard de C. Per aquesta raó sovint s'utilitza C++ en conjunció amb una biblioteca independent.

La biblioteca més representativa és **Boost**, que de fet és una col·lecció de biblioteques que cobreixen molts més camps que la biblioteca estàndard (com ara comunicacions en xarxa o interpretació d'arguments) . Les biblioteques que en formen part estan escrites en el mateix estil que la biblioteca oficial, són d'alta qualitat i en molts casos són candidates a integrar-se en l'estàndard (de fet algunes ja han format la base de funcions afegides en C++11). A més les biblioteques Boost abstrauen les interfícies pròpies de cada sistema operatiu per operacions de baix nivell. Per aquestes raons es tindran en compte, de cara al disseny, com si fossin part de la biblioteca estàndard.

Només hi ha una funcionalitat necessària per al projecte que Boost no cobreix: la gestió de processos fills (amb interfícies neutrals al sistema operatiu), necessària per a la implementació del suport de CGI. La biblioteca *Boost Process*, que implementa aquesta funcionalitat, ha estat una candidata a formar part de Boost (rebutjada) i arrel d'això hi ha múltiples versions disponibles. Atès que la funcionalitat necessària és

mínima (crear processos definint-ne l'entorn i llegir-ne la sortida) qualsevol implementació correcta serveix.

Per un últim aspecte també es farà servir una biblioteca addicional: la compressió[29]. S'ha optat per zlib, biblioteca molt popular i contrastada que implementa l'algorisme *deflate*.

Patrons de disseny identificats

Thread-pooling (o *Worker Threads*): El patró ThreadPool és una de les implementacions més populars per a la gestió de fils, utilitzat molt sovint en programari client-servidor. És ideal quan el nombre de tasques a realitzar i el nombre de fils a utilitzar no es pot predir, o quan la durada i la càrrega de treball de les tasques no és necessàriament equivalent. Vegeu més detalls a l'apartat *El patró Thread Pool*, a la pàgina 40.

Singleton: El patró singleton és útil quan té lògica tenir una única instància d'un objecte. Consisteix en no permetre la instanciació d'una classe i proporcionar un mètode de classe per accedir a la única instància existent. Aquest patró s'utilitzarà en la implementació del sistema de *logging* (detalls a la pàgina 41).

Revisió de tipus

Sobre la base d'un llenguatge real i la col·lecció de biblioteques triades podem introduir els tipus de dades propis d'aquests en disseny:

- Trets comuns de *Petició HTTP* i *Resposta HTTP*: Aquestes dues classes comparteixen el fet de contenir una sèrie de *Capçaleres HTTP*. Encara que representen conceptes independents, representen dos casos de *missatges HTTP*. Podem introduir una classe base anomenada *MissatgeHTTP* que amb els trets comuns (el fet d'estar compostos per capçaleres i, opcionalment, tenir un cos).
- *Tipus de petició*: S'ha optat per una enumeració d'entre les possibles representacions (una altre possibilitat seria una cadena), que anomenarem *TipusPeticioHTTP*.

  ```
  enum TipusPeticioHTTP { GET, POST, HEAD, OPTIONS, TRACE, PUT, DELETE, desconegut };
  ```
- *Codi d'estat*: No ens cal una classe pròpia per aquest concepte ja que només cal representar un codi numèric d'entre la sèrie de codis predefinits que el servidor implementi. A més és un component de la Resposta HTTP i no té operacions pròpies. En aquest cas també es pot definir com a enumeració, *CodiEstatHTTP*.

  ```
  enum CodiEstatHTTP { indefinit, CODI_100, CODI_101, CODI_200, /* ... */ };
  ```
- Frase d'estat: Les frases d'estat també són un conjunt prefixat i un component de la resposta, per tant no és estrictament necessari definir una classe discreta.
  ```
  map<CodiEstatHTTP, string> FrasesEstatHTTP;
  ```
- *Connexió*: representa la connexió entre el Servidor i un client, es correspon a la classe *socket* de boost.

A més la classe *Sessió* és innecessària ja que la seva tasca, representar una sessió, ja la du a terme la classe Servidor amb el disseny actual (per a cada connexió es crea un Servidor per atendre-la).

L'inici de la implementació

En aquest punt es comença la implementació del projecte; aquesta posarà de relleu detalls *reals* que no s'hagin tingut en compte fins ara, bé perquè no han aparegut explícitament en els casos d'ús o bé perquè són problemes eminentment pràctics.

A més queden per concretar detalls del disseny que s'han posposat a *posteriors iteracions*. Aquest procés iteratiu es detallarà en el capítol VI, *Desenvolupament: Iteracions incrementals del disseny* (a partir de la pàgina 40). Abans, però, es procediran a detallar els detalls propis de la implementació.

29 De fet Boost inclou codi per a comprimir com a part de Boost Iostreams, però l'abstracció de les interfícies combinada amb la seva documentació superficial i el poc temps disponible per a implementar aquesta funcionalitat m'han fet optar per una biblioteca més simple.

V. Desenvolupament: Implementació

Pla d'implementació

Per a la implementació se seguirà amb el pla de treballar de manera iterativa i incremental, implementant noves funcionalitats que ampliïn les anteriors en successives *passades*. Addicionalment s'escriuran tests unitaris, i s'elaboraran jocs de proves, en base a les especificacions per ajudar a assegurar que la implementació sempre es correspon al desitjat.

En una primera iteració s'implementarà un servidor mono-fil del servei HTTP bàsic, sense les modificacions introduïdes en la versió 1.1.

La implementació ha consistit en les següents fases:

- Implementació simple d'HTTP/1.0: Mètodes GET, HEAD i POST; mono-fil i sense mecanismes de seguretat ni configuració.
- Configurabilitat (port, nombre de fils)
- Sistema de registre (*logging*)
- Implementació multi-fil (en el pla inicial aquest era un pas força posterior però es va adelantar per la seva relativa complexitat)
- Mètodes HTTP restants (PUT, DELETE, TRACE i OPTIONS)
- Modificacions d'HTTP/1.1: manteniment de connexions obertes (re-ús i *pipelining*)
- Configurabilitat de la resta d'opcions: autenticació (nom d'usuari, contrasenya i realme), permisos de rutes
- Mecanismes d'accés: denegació de rutes configurades com a prohibides, autenticació i gestió dels permisos POSIX
- Mecanismes de filtrat de contingut: compressió
- Generació de recursos dinàmics: interfície CGI

Detalls a tenir en compte en l'ús de C++

Val la pena mencionar un parell de detalls a tenir en compte a l'hora d'implementar el projecte en C++.

Programant en «C++ amb estil C» és fàcil caure en paranys típics de la programació en C (p.e. un punt particularment problemàtic de C és la facilitat amb què els punters i els arrays es poden intercanviar). Fer un ús intensiu de l'STL i de Boost ajuden a reduir aquest tipus de problemes.

L'altre gran problema típic dels programes en C++ és la gestió incorrecta de la memòria dinàmica i les conseqüents fugues de memòria. L'ús sistemàtic de la tècnica RAII[30] combinada amb punters intel·ligents ha d'eliminar completament aquest problema en un projecte com aquest ([DevX, 2008], pàg. 4).

C++11 (anteriorment C++0x)

Recentment (agost de 2011) s'ha aprovat una nova revisió del llenguatge (anomenada C++11, reemplaçant C++98) que incorpora novetats significatives. Ja que aquesta revisió ha estat anys en desenvolupament, els compiladors actuals ja n'inclouen un suport parcial relativament madur[31,32], per aquesta raó es faran servir algunes d'aquestes noves funcionalitats allà on siguin útils (i disponibles en els compiladors principals).

30 *Resource Acquisition is Initialization*. Tècnica consistent bàsicament en reservar els recursos en la construcció dels objectes i alliberar-los en la destrucció.
31 *C++0x/C++11 Support in GCC* <http://gcc.gnu.org/projects/cxx0x.html>
32 *C++0x Core Language Features In VC10: The Table* <http://blogs.msdn.com/b/vcblog/archive/2010/04/06/c-0x-core-language-features-in-vc10-the-table.aspx>, *C++11 Features in Visual C++ 11* <http://blogs.msdn.com/b/vcblog/archive/2011/09/12/10209291.aspx>

Entorn de desenvolupament i eines

Encara que l'objectiu és que el producte resultant sigui multi-plataforma, per motius pràctics el desenvolupament es farà i provarà principalment sobre GNU/Linux (en concret Debian Sid per arquitectura amd64).

Com a compiladors principals en Linux es faran servir tant *GCC* (l'elecció habitual atesa la plataforma) com *Clang*, aquest darrer aporta uns missatges d'error sovint més clarificadors i fàcils de llegir, facilitant-ne la solució de problemes.

En entorns Windows s'utilitzarà *Mingw* (port de GCC)[33].

L'ús de diferents compiladors hauria d'ajudar a l'hora d'evitar codi ambigu o particularitats d'una eina concreta.

En entorns POSIX es farà servir el sistema *autotools*, el qual permet compilacions incrementals ràpides i el canvi simple de compilador[34].

A més, com a regla per incrementar la correctesa del codi s'estableix la norma de compilar amb el nivell màxim de *warnings*, i no permetre compilacions amb *warnings*. Addicionalment es posarà el compilador en mode *pedant*[35].

Jocs de proves

Proves unitàries

Al directori `tests`, del codi font, s'inclouen algunes proves unitàries per validar la coherència de les implementacions.

Hi ha definides proves unitàries per validar les classes Base64 (pàg. 38), CapsaleraHTTP, DataHTTP, Log (pàg. 24), PeticioHTTP, RespostaHTTP, TipusMIME i Uri.

A més hi ha un parell de programes de mostra per a comprovar el funcionament del control de fils (pàg. 40) i les transformacions de text (pàgs. 23 i 42).

Mètodes HTTP

Per a provar el correcte funcionament a nivell de protocol del programa s'han definit uns *scripts* de mostra[36] (escrits en Bash) dels diferents mètodes HTTP. Aquests tests estan pensats per executar-se contra tfcweb corrent en en directori src i amb capacitat per escriure-hi. Els tests es detallen a continuació; de totes maneres per a evitar possibles errors d'interpretació de l'especificació la millor prova de comportament és utilitzar un navegador.

`TestConnect.bash`	Mètode CONNECT. Resultat correcte: Error 501, el mètode CONNECT no s'implementa.
`TestDelete.bash`	Mètode DELETE. Resultat correcte: Confirmació d'esborrament.
`TestGet.bash`	Mètode GET. Resultat correcte: Índex del directori arrel del servidor.
`TestMetodeFictici.bash`	Mètode inexistent. Resultat correcte: Error 501 (mètode no implementat).
`TestOptions.bash`	Mètode OPTIONS. Resultat correcte: Resposta 200 sense cos, amb

33 També s'emprarà l'entorn de desenvolupament *Microsoft Visual C++ Express 11*. Per motius d'aprofitament del temps, però, no es garantirà la correcta compilació sobre aquest compilador.

34 Per exemple:
Compilació incremental: `$ make`
Amb compilador GCC: `$ make CXX=g++`
Amb compilador Clang: `$ make CXX=clang++`
...

35 En aquest mode (opció *-pedantic*) GCC (també Clang) intenta obligar a adherir-se a l'estàndard i produeix avisos i errors en fer servir extensions no permeses per aquest.

36 Els *scripts* de prova s'inclouen en el directori `tests/metodes`.

Content-Length 0 ([RFC 2616], pàg. 52).

`TestPostChunked.bash` Mètode POST amb codificació "chunked" en la petició. Resultat correcte:

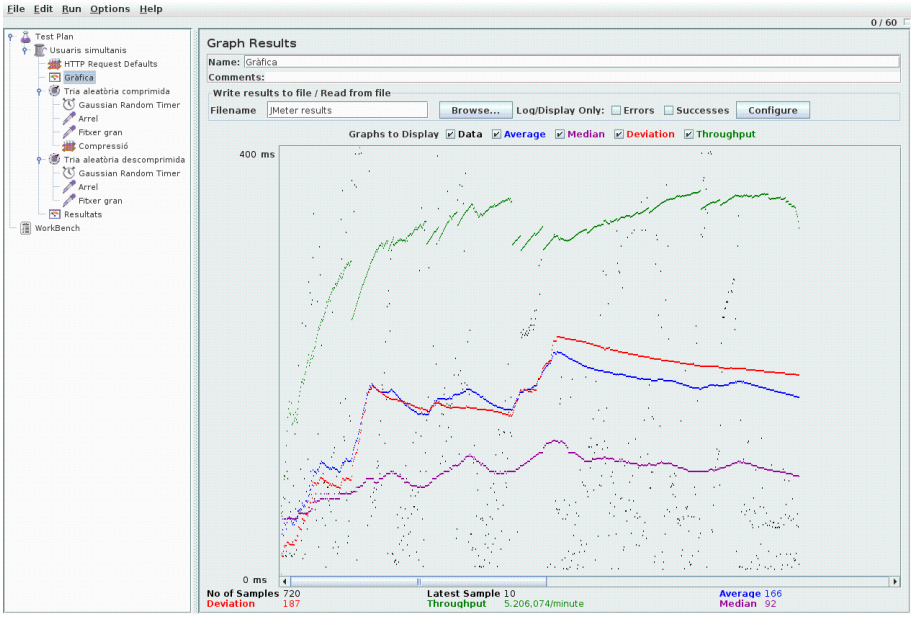
`TestPut.bash` Mètode PUT. Resultats correctes:
Amb permís d'escriptura: resposta 201 en la primera crida, 204 en successives
Sense permís: resposta 405

`TestPutWinie.bash` Mètode PUT a l'estil de Winie[37] (HEAD seguit de PUT). Resultats correctes: 200 o 404 per al HEAD i resultats iguals que amb `TestPut.bash` per al PUT.

`TestTrace.bash` Mètode TRACE. Resultat correcte: Resposta 200 amb les capçaleres de la petició en el cos de la resposta.

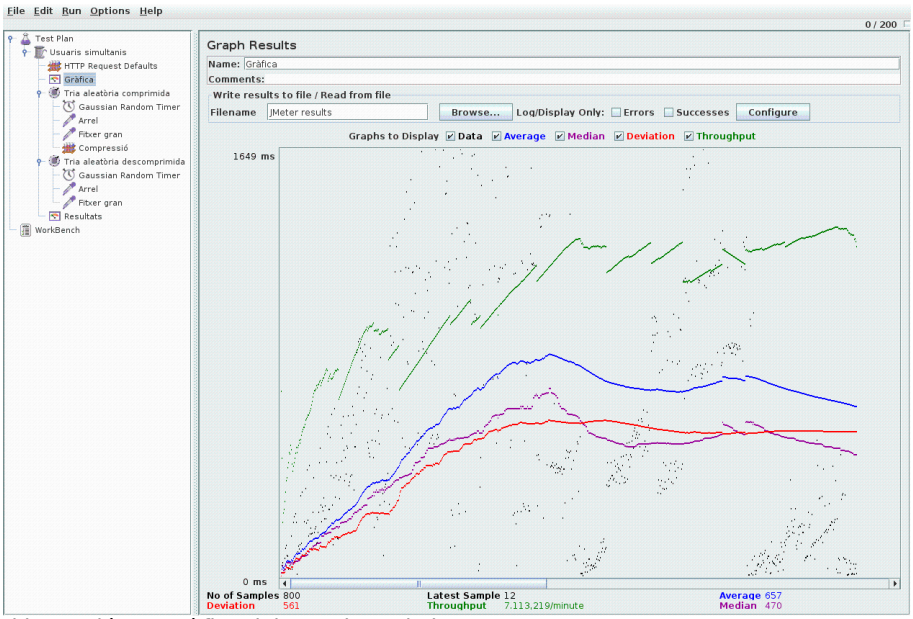
Test de sobrecàrrega

Per a comprovar el comportament del servidor quan s'hi connecten múltiples clients de manera simultània s'ha utilitzat jMeter[38]. El rendiment alt no ha estat entre els objectius i per tant no s'ha tingut massa en compte, les proves però, mostren que un cop s'estabilitza la càrrega, les respostes se serveixen a un ritme raonablement estable.



Execució amb 60 fils simultanis en 6 iteracions.

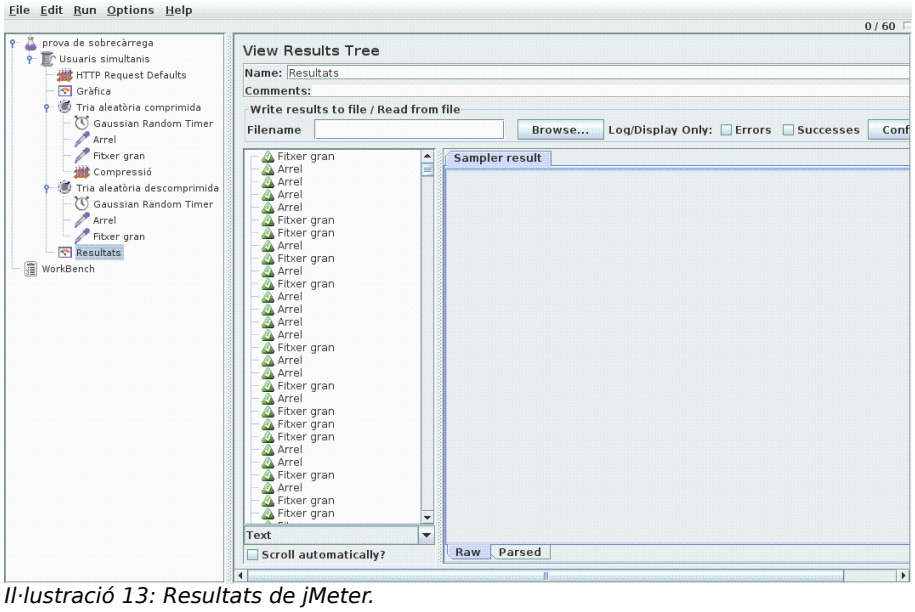
37 Winie és una eina del W3C que implementa els mètodes PUT i DELETE en mode client: http://jigsaw.w3.org/Winie/
38 Programa per a comprovar el comportament d'una aplicació sobrecarregant-la de feina, http://jmeter.apache.org/



Il·lustració 12: Gràfica dels resultats de jMeter, II

Execució amb 200 fils simultània en 2 iteracions.

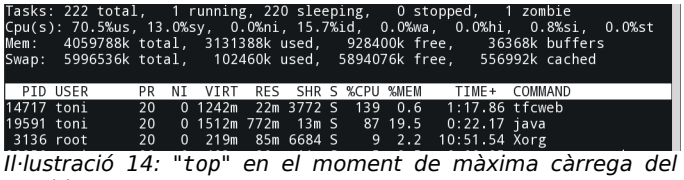
El que sí és important d'aquestes proves es comprovar que el servidor respon correctament i no rebutja peticions innecessàriament ni es *cau* un cop se'l sobrecarrega.



Il·lustració 13: Resultats de jMeter.

Cap resultat incorrecte.

Càrrega de recursos del sistema, el servidor ha respost correctament:



Il·lustració 14: "top" en el moment de màxima càrrega del servidor

El sistema en qüestió té tres nuclis, per tant el màxim ús de CPU en top és 300%.

Val a dir que en la realització de les proves al càrrega que el propi jMeter impossa al sistema ha estat un problema.

Test de fugues de memòria

Per a comprovar que no hi ha fugues de memòria s'ha executat el servidor sota Valgrind[39]. Com s'ha mencionat l'ús correcte de les eines proporcionades per C++ i les biblioteques associades hauria d'impedir qualsevol fuga.

Resultat de mostra, s'ha executat una prova amb jMeter[40] en paral·lel per carregar el servidor:

```
$ valgrind --tool=memcheck -v -- ./tfcweb --fils 10
==21145==
==21145== HEAP SUMMARY:
==21145==     in use at exit: 12,342 bytes in 91 blocks
==21145==   total heap usage: 660,249 allocs, 660,158 frees, 140,622,191 bytes
allocated
==21145==
==21145== Searching for pointers to 91 not-freed blocks
==21145== Checked 81,498,440 bytes
==21145==
==21145== LEAK SUMMARY:
==21145==    definitely lost: 0 bytes in 0 blocks
==21145==    indirectly lost: 0 bytes in 0 blocks
==21145==      possibly lost: 3,286 bytes in 19 blocks
==21145==    still reachable: 9,056 bytes in 72 blocks
==21145==         suppressed: 0 bytes in 0 blocks
==21145== Rerun with --leak-check=full to see details of leaked memory
==21145==
==21145== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 10 from 6)
--21145--
--21145-- used_suppression:     10 dl-hack3-cond-1
==21145==
==21145== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 10 from 6)
```

La línia *total heap usage* indica que s'han reservat dinàmicament, al llarg de l'execució del programa, un total de 134 megues.

Les línies "*definitely lost: 0*" i "*indirectly lost: 0*" indiquen que no s'ha perdut cap bloc de memòria. Les línies *possibly lost* i *still reachable* indiquen que alguns blocs no s'havien alliberat en sortir del programa, la qual cosa és un comportament estàndard de les biblioteques utilitzades.

Modificacions del disseny introduïdes en la implementació

Com era de preveure, la fase d'implementació ha tret a la llum alguns elements del disseny que a la pràctica necessitaven re-dissenys, que es detallen a continuació.

39 Sistema d'anàlisi de programes, especialment utilitzat per a detectar errors en la gestió de memòria, http://valgrind.org/

40 Degut a la instrumentalització que fa valgrind del programa, l'execució s'alenteix molt pel que la prova de jMeter ha estat només amb un nombre de fils suficients per superar els fils de servidor disponibles i sense repeticions

Cos de les respostes (i dels missatges HTTP en general)

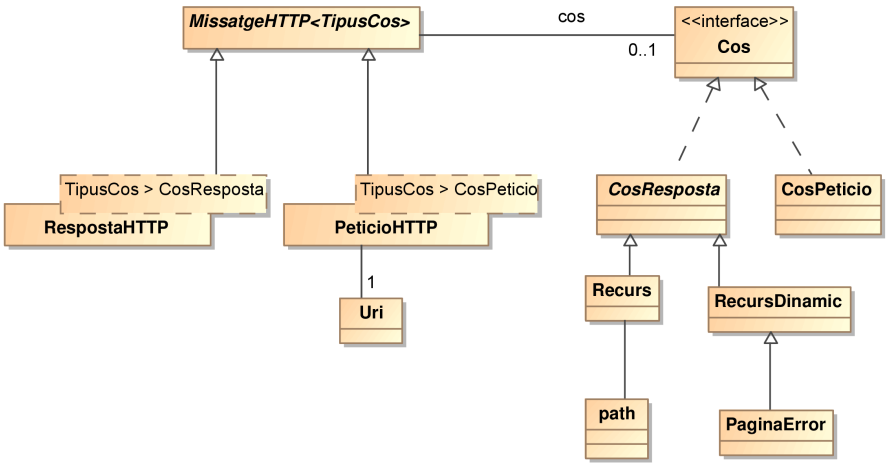
Tant les peticions com les respostes poden contenir un «cos», en el cas de les peticions el servidor simplement rebrà un flux de bytes però en les respostes, generades pel servidor, aquest cos pot correspondre a un recurs estàtic, un recurs dinàmic o un missatge error (què a la pràctica també és un tipus de recurs dinàmic), un fet que no queda reflectit correctament en el disseny actual (el cas del missatge d'error de cara a l'usuari client, de fet, no s'havia tingut en compte). Part del problema és que actualment la funció de cos està, parcialment, integrada en la classe `Recurs`, que representa tant un fitxer com la interfície cap al cos d'una resposta.

Per a solucionar-ho caldrà separar les dues funcions de la classe `Recurs`: representació d'un recurs (fitxer o directori) i cos d'un missatge HTTP. Introduïm un atribut `cos` en la classe `MissatgeHTTP`, que ha de ser capaç de contenir els casos mencionats, per tant `Cos` serà una interfície comuna implementada per `Recurs`, `RecursDinamic` i la nova classe per presentar pàgines HTML d'error (`PaginaError`). Amb la separació de funcions de `Recurs`, `RecursDinamic` deixa de ser-ne una especialització.

En la pràctica el cos d'una petició i d'una resposta són totalment diferents encara que en el fons representin el mateix concepte, per facilitar-ne l'ús en la implementació (evitant coercions tot proporcionant accés directe a la variant de cos adient) s'ha optat per definir MissatgeHTTP com una plantilla parametritzada per el tipus de cos:

A més cal clarificar que un Recurs (fitxer, directori, ...) tindrà una ruta en el disc i una ruta d'accés, que corresponen a conceptes diferents: la ruta d'accés és la URI per la qual s'hi accedeix i és una atribut de la petició, mentre que la ruta en disc és un atribut del Recurs.

La ruta d'accés es representarà amb una classe *Uri* mentre que la ruta en disc es correspon amb boost::path.



Il·lustració 15: Introducció de classes Cos en els missatges HTTP

Codificació Base64

L'intercanvi de credencials es fa codificant-les en Base64, una codificació simple definida en l'estàndard MIME ([RFC 2045], apartat 6.8: *Base64 Content-Transfer-Encoding*) per a l'intercanvi de dades binàries en missatges de correu electrònic (és a dir, en format textual)[41].

Per a proporcionar aquesta funcionalitat definim una classe Base64 amb mètodes per codificar i descodificar.

Enviament progressiu del cos de la resposta

El cos de la resposta, que pot correspondre's amb un fitxer o la sortida d'un *script*, pot ser de mida

41 Aquesta codificació pren els bits de 6 en 6 (d'aquí el 64, 2^6=64 possibles valors) i els assigna un caràcter de text d'una taula estàndard.

arbitrària. Per aquesta raó col·locar-lo en memòria al complet és una mala idea i convé un mètode per enviar-lo al client de manera progressiva, en blocs.

Deleguem en la implementació concreta del cos per fer aquesta tasca, definint un mètode *envia* en *CosResposta*, que s'encarregui directament d'enviar-se al client. Això fa redundant la classe *RecursGenerat*, que precissament actuava de contenidor en memòria del resultat d'un recurs dinàmic.

VI. Desenvolupament: Iteracions incrementals del disseny

Com s'ha comentat al capítol IV, les àrees que quedaven per acabar de dissenyar s'han completat en petits cicles d'anàlisi-disseny-implementació, i es recullen a continuació.

Indexació de directoris

La norma quan se sol·licita la URL corresponent a un directori és enviar al client un llistat dels continguts del directori quan no existeix un fitxer d'índex[42] (aquest comportament és de fet opcional, i alternativament es pot mostrar un error 403 –"prohibit"– indicant que no es permet el llistat de directoris).

Per implementar aquesta funcionalitat introduïm una classe, *AutoIndex*, que és una especialització de RecursDinamic, i que genera el codi HTML corresponent.

Configuració i permisos

Per a la configuració simplement definim una classe per emmagatzemi els valors definits. Caldrà extreure'ls de la línia de comandes i/o fitxers de configuració, ambdues tasques es poden implementar sobre la biblioteca Boost Program Options.

Per a la gestió dels permisos definim una classe que actuï de *porter*, a partir de la configuració establerta determinarà si una ruta del disc és o no accessible i si cal proporcionar credencials.

Control de fils

El patró Thread Pool

Per al control de fils s'ha optat per implementar un Thread Pool. Aquest consisteix en mantenir un nombre determinat de fils oberts a l'espera de tasques a realitzar, combinats amb una estructura que permeti *programar* tasques. Qualsevol fil que romangui a l'espera es farà càrrec d'una tasca programada i quan la completi tornarà a estar a l'espera. I quan es programi més d'una tasca alhora, un fil diferent s'encarregarà de cada tasca de forma simultània.

L'avantatge principal d'aquest patró és que el nombre de fils és fix i per tant els fils només es creen en un moment concret i no sota demanda, a mida que apareguin noves tasques. És ideal per a mantenir els recursos sota control: obrir un fil és una tasca molt més costosa que deixar un fil a l'espera, i el nombre constant de fils fa que la programació de *massa* tasques no sobrecarregui el sistema, un cop tots els fils estiguin treballant, les tasques programades hauran d'esperar el seu torn per ser ateses.

La implementació d'aquest patró requerix l'ús d'una estructura *cua* amb capacitat d'ús simultani des de diferents fils, ja que els diferents fils treballadors el consultaran i n'extrauran les tasques disponibles.

Ni l'STL ni Boost proporcionen una estructura que satisfagui aquestes característiques, pel que s'ha d'implementar, l'anomenarem CuaConcurrent, i contindrà tasques.

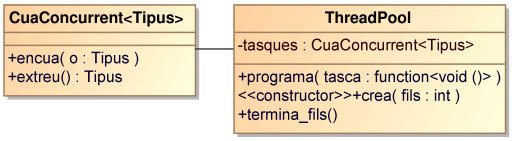
Implementació

Per a implementar la *CuaConcurrent* s'ha optat per un model amb bloqueig: modificar la cua en bloca l'accés de manera que dues modificacions simultànies no siguin possibles (la segona es quedarà l'espera).

Les tasques es representen com a objectes *cridables* genèrics (amb boost::function), de manera que es

42 El fitxer d'índex (sovint `index.html`) és un fitxer que se serveix per defecte quan existeix, en indicar la URL d'un directori.

pugui programar qualsevol tasca que es pugui cridar com una funció.



Il·lustració 16: Diagrama de «Thread Pool»

Respecte al disseny actual, la classe ControladorServei es correspon a la funció main() de C++ i a un ThreadPool, mentre que la classe Fil és boost::thread.

Sistema de registre (logging)

Requisits

El programari de servidor sovint funciona de forma desatesa i en segon pla, per tant és habitual proporcionar un mecanisme que registri els esdeveniments (com errors) en un, o múltiples, fitxers, per a posteriors diagnòstics.

Idealment un sistema de registre ha de poder enviar les seves entrades a múltiples destinacions (com ara un arxiu i la consola), i cada missatge ha de tenir associades algunes meta-dades, com la hora en què es produeix i la seva naturalesa (error, avís, missatge d'estat...).

Detalls de disseny

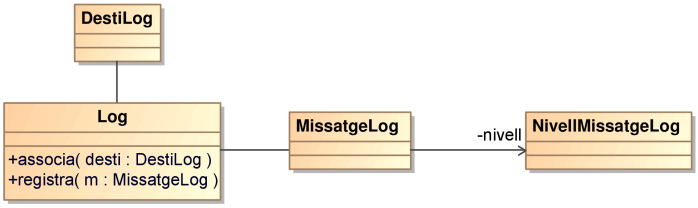
Cal tenir en compte que els missatges de registre poden generar-se des de qualsevol fil, per tant cal assegurar que crides simultànies no interfereixen entre elles, serialitzant-les.

Per a poder generar fàcilment missatges de registre des de qualsevol punt del programari s'ha optat per utilitzar el patró singleton, de manera que hi haurà un únic objecte Log que rebi tots els missatges de registre.

Implementació

Donades les característiques del llenguatge C++, seria desitjable implementar un mecanisme d'inserció (operador "<<"), ja que d'aquesta manera s'evita al codi usuari del registre haver de fer les conversions a cadena de text manualment. Definir un operador d'inserció comporta, però, que la recepció d'un missatge de registre no sigui *atòmica*, per tant no té sentit blocar l'accés al registre quan es rebi un missatge per evitar interferències entre fils.

Per a assolir els requisits plantejats es defineix una classe MissatgeLog que representa una entrada del registre, i que es pot anar construint incrementalment mitjançant operacions d'inserció. Un cop complet s'envia a l'objecte Log, que l'afegirà a les destinacions configurades.



Il·lustració 17: Diagrama de classes del sistema de registre

Els detalls de la la construcció gradual dels missatges de registre s'han amagat proporcionant una interfície similar a la sortida estàndard:

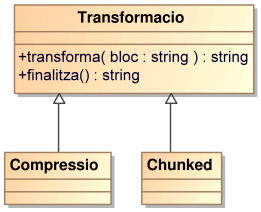
```
    log() << NivellDebug << "Això és el missatge de registre núm. " << 1 << commit;
```

On `log()` construeix el missatge i `commit` l'envia la classe Log.

Transformacions

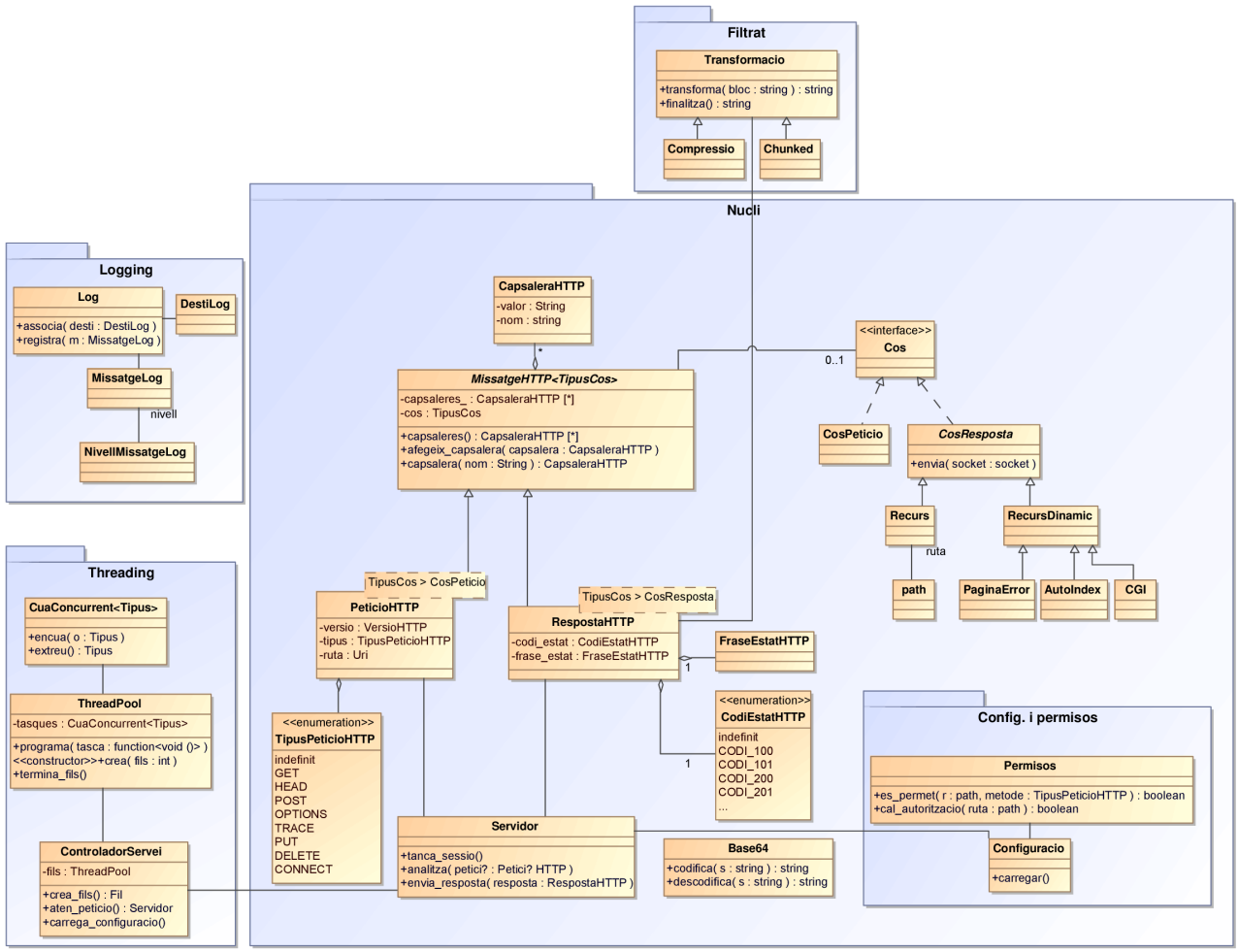
Per a implementar les transformacions de text mencionades a l'apartat *Codificacions i Transformacions* (pàg. 23) cal tenir en compte, com en l'enviament de dades al client, que les dades a transformar són de mides arbitràries i per tant cal una interfície que faci transformacions en blocs. També cal tenir en compte que alguns tipus de transformacions requereixen un tractament especial al final.

Definim la classe Transformació amb la interfície comuna i una subclasse per a cada tipus de transformació:



Il·lustració 18: Diagrama de classes de transformacions de dades

Diagrama de classes definitiu



Il·lustració 19: Diagrama de classes definitiu

Interpretació

Com en el diagrama original *ControladorServei* s'encarrega de rebre les connexions entrants dels clients, però ara és *ThreadPool* qui gestiona els fils. En rebre una connexió, *ControladorServei* programa la seva gestió mitjançant una instància de *Servidor* que, a través de *ThreadPool*, serà atesa per el primer fil disponible.

Servidor interpretarà la *PeticioHTTP* i generarà una *RespostaHTTP* corresponent, ambdues representen especialitzacions de MissatgeHTTP i per tant contenen *CapsaleresHTTP* i poden contenir un *Cos*. La resposta adequada es generarà a partir dels *Permisos* definits a la *Configuracio*.

El cos de les respostes, *CosResposta*, pot representar el contingut d'un fitxer (*Recurs*) o contingut generat dinàmicament (*RecursDinamic*), exemples d'aquest últim cas són *PaginaError*, per els missatges d'error, *AutoIndex*, per a mostrar el contingut d'un directori, o *CGI*, per a la sortida d'un programa CGI.

Quan el client ho suporti, el contingut de les *Respostes* pot ser manipulat mitjançant una *Transformacio*, de les quals s'implementen *Compressio*, per a estalviar ample de banda, i *Chunked*, per enviar en blocs.

A nivell global hi ha disponible el sistema de *logging*, que permet enviar missatges de registre

(*MissatgeLog*) als destins (*DestiLog*) configurats, mitjançant la classe *Log*.

VII. Possibles millores futures

La funcionalitat d'un servidor HTTP pot anar molt més enllà del què s'ha creat en aquest TFC, a continuació es recullen algunes idees de possibles característiques addicionals a implementar.

Ja s'ha comentat que algunes funcionalitats extra d'un servidor com la de proxy i caché, i algunes característiques opcionals del protocol com les peticions GET condicionals o parcials no s'han implementat, per tant aquestes són les candidates més òbvies a futures millores.

Una característica inherent al programari de servidor que no s'ha implementat és l'execució en mode *dimoni*, mode en què el servidor funciona en segon pla, de manera independent del terminal des del qual s'executa. A més, un cop el servidor funciona en segon pla, seria recomanable poder configurar el sistema de *logging* per definir on i de quin tipus cal enviar els missatges, les bases d'aquesta funcionalitat estan implementades però ara per ara no es pot configurar.

Una altra característica menys evident és el suport de *hosts virtuals* (de vegades anomenats *vHosts*). La introducció de la capçalera Host (pàg. 21) de forma obligatòria en les peticions HTTP/1.1 va en aquest sentit: el fet que a una mateixa adreça IP li pugui correspondre més d'un nom de domini fa que un mateix servidor pugui servir múltiples llocs web, com si es tractés de diferents servidors, fent servir el nom de host per identificar a quin s'està accedint.

La observació de servidors com Apache fa paleses un parell de limitacions del sistema de configuració actual que seria interessant corregir: la configuració per directori i la diferenciació entre rutes virtuals i rutes canòniques[43].
La configuració per directori, en el cas d'Apache, es fa incloent un fitxer de configuració en el directori en qüestió, que es llegeix en cada accés a aquest i permet re-definir la configuració que en fa referència.
Quant a les rutes, pot interessar configurar els permisos d'una ruta en funció de la URI utilitzada per accedir-hi en comptes de la seva ruta corresponent al sistema de fitxers.

Per últim, una de les grans característiques dels servidors principals és el suport de connectors (o mòduls) d'extensió, que permeten afegir-ne noves funcionalitats mitjançant la càrrega opcional d'un arxiu de biblioteca. Per exemple, en Apache, el mòdul SSL afegeix suport per al protocol HTTPS, o el mòdul PHP (desenvolupat per un equip independent) proporciona la capacitat d'executar *scripts* en llenguatge PHP.

Conclusions

Abans de començar amb aquest projecte ja estava una mica familiaritzat amb els conceptes bàsics del protocol HTTP i tenia una idea aproximada del seu funcionament. L'estudi de l'especificació necessari per realitzar aquest projecte m'ha servit per a aprofundir molt més.

El protocol HTTP és un protocol de funcionament simple i explícitament definit de manera que amb un mínim de requisits ja he pogut tenir un servidor funcional, de fet en pocs dies ja hi podia accedir sense problema des d'un navegador, un detall molt d'agrair. A partir d'aquí, però, els RFCs estan plens de petits detalls que el compliquen i que fan aquesta primera impressió una mica enganyosa. Tot i això, en l'estat actual, molt lluny de completar tota la funcionalitat detallada en les especificacions, el funcionament del servidor és prou satisfactori.

La posada en pràctica de les tècniques de disseny en un projecte real també m'han ajudat a valorar-les més. En efecte, a partir d'un disseny que, com s'ha vist, encara necessitava força refinaments, he pogut realitzar la implementació amb molta soltura i tenint en tot moment una idea prou clara de com enfocar la tasca.

Aquesta feina d'implementació també m'ha servit per aprofundir en l'ús de C++, que sempre he trobat un llenguatge especialment interessant per la seva riquesa semàntica. L'ús intensiu de les biblioteques estàndard i de Boost m'han permès aprendre força en molt poc temps, i cap al final del projecte m'he trobat utilitzant amb prou soltura abstraccions que al principi del mateix no acabava d'entendre.

En conjunt estic satisfet amb el resultat i he de dir que ha estat molt gratificant veure el programa en el seu estat final servint pàgines al navegador.

43 Una ruta canònica és la ruta real i normalitzada d'un fitxer o directori, és a dir, la ruta absoluta i sense enllaços simbòlics.

Glossari

Agent d'usuari: Programari del client (p.e. navegador)

Cachejar: Emmagatzemar en la caché. Vegeu l'apartat Funcionament del protocol HTTP, a la pàgina 16.

CGI: Common Gateway Interface (*Interfície d'entrada comuna*). Especificació per permetre a servidors web i aplicacions externes comunicar-se, de manera que l'aplicació pugui processar una petició.

Hipertext: Text que pot incloure *hiperenllaços*, enllaços a altres textos o recursos relacionats.

HTML: Hypertext Markup Language (*Llenguatge de marcat d'hipertextos*). Llenguatge estàndard de les pàgines web. És un llenguatge destinat a estructurar i enllaçar hipertextos, mitjançant l'ús d'*etiquetes* semàntiques entorn el text al que afecten, aquestes etiquetes indiquen a l'agent d'usuari el contingut i aquest és l'encarregat de presentar-lo a l'usuari d'una forma adequada.

HTTP: Hypertext Transfer Protocol (*Protocol de Transferència d'Hipertext*). És el protocol de base dels servidors web, un protocol simple destinat a la transferència eficient de recursos multimèdia i en particular pàgines HTML. El protocol HTTP s'especifica en succesius RFCs, el més recent dels quals és [RFC 2068].

IETF: Internet Engineering Task Force (*Grup Especial d'Enginyeria d'Internet*) organització que desenvolupa els estàndards utilitzats a Internet.

MIME: Multipurpose Internet Mail Extensions (*Extensions de correu Internet multipropòsit*). Definit inicialment en l'RFC 1341 (l'última revisió és compren els RFCs des del 2045 fins al 2049). Estàndard que defineix un format de missatges que permet la transmissió de dades que no fan servir la codificació US-ASCII, codificació estàndard dels missatges a Internet.

Recurs: En el context del protocol HTTP sovint es fa servir la forma "recurs" per referir-se als elements als què el servidor proporciona accés. En el cas més habitual un recurs es correspon amb un fitxer, però també pot tractar-se de dades produïdes a petició per el servidor o per un programa extern.

RFC: Request For Comments (*Petició de comentaris*). Documents tècnics utilitzats per l'IETF. Cada RFC documenta una proposta de protocol o tecnologia relacionada amb Internet.

SSI: Server Side Includes (*Inclusions en el costat del servidor*). Llenguatge interpretat simple que s'executa en el servidor per afegir dinamisme a pàgines web.

SSL: Secure Sockets Layer (*Capa de sòcol segur*). Protocol que proporciona criptografia asimètrica per a la comunicació segura com a capa addicional a un altre protocol. La seva combinació amb HTTP s'anomena HTTP segur o HTTPS.

TLS: Transport Layer Security (*Seguretat en la capa de transport*). Protocol SSL en l'actualitat. L'IETF va fer-se càrrec del protocol SSL i el re-anomenà TLS.

Bibliografia

[Campderrich, 2004] : **Campederrich Falgueras, Benet**, *Enginyeria del Programari*, 2004
[Del Río, 2005] : **del Río Medina, Ángel**, *Creación del núcleo de un servidor web*, 2005
[DevX, 2008] : **Kalev, Danny**, *The State of the Language: An Interview with Bjarne Stroustrup*, 2008. http://www.devx.com/SpecialReports/Article/38813
[Fontanals, 2003] : **Fontanals Rofes, Miquel**, *Creació del nucli d'un servidor Web: XicHttpd*, 2003
[KMK, 1999] : **Krishnamurthy, B.; Mogul, J. C.; Kristol, D. M.**, *Key Differences between HTTP/1.0 and HTTP/1.1*, 1999
[MPW, 2007] : **McLaughlin, B. D.; Pollice, G., West. D**, *Head First Object-Oriented Analysis and Design*, 2007
[RFC 1521] : **Borenstein, N.; Freed. N**, *RFC 1521: MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, 1993
[RFC 1945] : **Berners-Lee, T.; Fielding, R.; Frystyk, H.**, *RFC 1945: Hypertext Transfer Protocol - HTTP/1.0*, 1996
[RFC 2045] : **Freed, N.; Borenstein, N.**, *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, 1996
[RFC 2046] : **Freed, N.; Borenstein, N.**, *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, 1996
[RFC 2068] : **Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Berners-Lee, T.**, *RFC 2068: Hypertext Transfer Protocol - HTTP/1.1*, 1997
[RFC 2616] : **Fielding, R.; Mogul, J.; Frystyk, H.; Masinter, L.; Berners-Lee, T.**, *RFC 2616: Hypertext Transfer Protocol - HTTP/1.1*, 1999
[RFC 2818] : **Rescorla, Eric**, *RFC 2818: HTTP Over TLS*, 2000
[Walls, 2005] : **Walls, Colin**, *Embedded software: The Works*, 2005

Annex I. Manual d'ús

Vegeu l'annex II per a detalls de com compilar el codi font. En el paquet de l'entrega s'inclouen executables ja compilats per a Windows i algunes versions de Linux, dins el directori `acorvera_producte/executable`.

Arrencada del servidor

Per arrencar el servidor amb les opcions per defecte n'hi ha prou d'executar la comanda corresponent des del directori en què es trobi (recent compilat, l'executable es troba en el directori `src`).

UNIX/Linux: `./tfcweb`
Windows: `tfcweb-mingw.exe`

(S'utilitza el port 8000 per defecte)

Notes addicionals:

Sistemes Windows: L'executable proporcionat pot donar error si manquen les biblioteques necessàries al sistema, en aquest cas executeu el fitxer `vcredist.exe` per instal·lar-les.

Sistemes Linux: A causa del sistema d'enllaçat utilitzat en Linux, els executables proporcionats només funcionaran en un petit conjunt de versions del sistema operatiu, per la resta caldrà compilar el programa.

Opcions suportades

Utilitzeu `tfcweb --help` per a mostrar una llista de les opcions disponibles. Es detallen a continuació:

`--help`	Mostra la llista d'opcions i els seus valors per defecte.
`--version`	Mostra informació sobre la versió de les biblioteques i compilador emprats.
`--port NUM`	Defineix el port.
`--config ARXIU`	Carrega la configuració de l'arxiu indicat, les opcions que s'accepten són les mateixes que a la línia de comandes excepte `--help`, `--version`, i `--config`.
`--arrel DIRECTORI`	Defineix el directori base a l'hora de servir documents.

`--usuari USUARI`
`--contrasenya PASS`
`--realme REALME`
	Defineixen el nom d'usuari i contrasenya que s'esperarà en rutes protegides, i l'identificador de la zona protegida.

`--fils`	Defineix el nombre de fils simultanis que s'empraran.

`--privat RUTA`
`--protegit RUTA`
`--cgi RUTA`
`--escribible RUTA`
`--llegible RUTA`
	Especifica els permisos assignats a a una ruta en el disc.

Taula 5: Llistat d'opcions de configuració de tfcweb

Parada del servidor

Per aturar el servidor no es proporciona un mecanisme directe, el servidor s'aturarà quan rebi els senyals `USR1` o `INT`. Des de la consola on s'executa un CTRL+C li enviarà un `INT`, mentre que l'script "`atura.sh`" li enviarà en `USR1` en sistemes UNIX/Linux.
En sistemes Windows també es pot tancar la finestra de terminal des de la que s'estigui executant per a

tancar el servidor.

Annex II. Manual de compilació

Com ja s'ha comentat s'han fet servir algunes de les biblioteques *Boost*, que hauran d'estar instal·lades en el sistema per poder compilar el programa. A més, l'ús de construccions de C++11 requereix un compilador relativament recent.

L'entorn de desenvolupament ha estat Linux i en conseqüència aquest és l'entorn més testat, tot i això el programa s'ha compilat correctament en plataformes Windows.

Compilació en entorns UNIX/Linux

Dependències: biblioteques boost (almenys 1.48), zlib, autotools (autoconf, automake) i g++ (almenys 4.4) o clang (almenys 3.0).

El projecte utilitza el sistema *autotools*, per tant per a compilar (o comprovar si es satisfan les dependències) n'hi haurà prou amb:

```
$ cd codi_font    # Canviem al directori que contingui el codi font
$ ./configure     # Detecció de dependències i preparació per compilació
$ make            # Compilació
```

Sistemes Debian i derivats: Paquet DEB

Per a sistemes Debian es proporcionen els scripts necessàries per a la construcció i l'empaquetat automàtic:

```
# apt-get install devscripts   # Instal·lació d'eines necessàries. Calen permisos d'administrador.
$ cd codi_font                 # Canviem al directori que contingui el codi font
$ debuild                      # Contrucció. Si no es satisfan les dependències indicarà quins
                                    paquets falten per instal·lar (instal·lar-los amb apt-get).
```

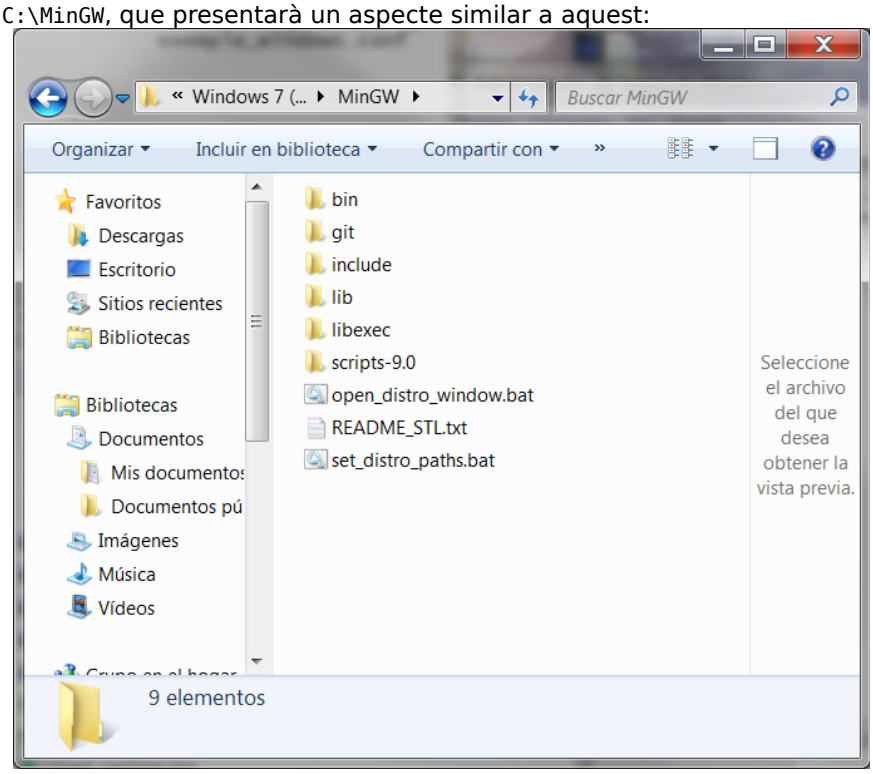
Nota: Degut a les dependències comentades, cal una distribució relativament recent. En el moment de l'entrega, Debian Stable no és apta.

Compilació en Windows (amb MinGW)

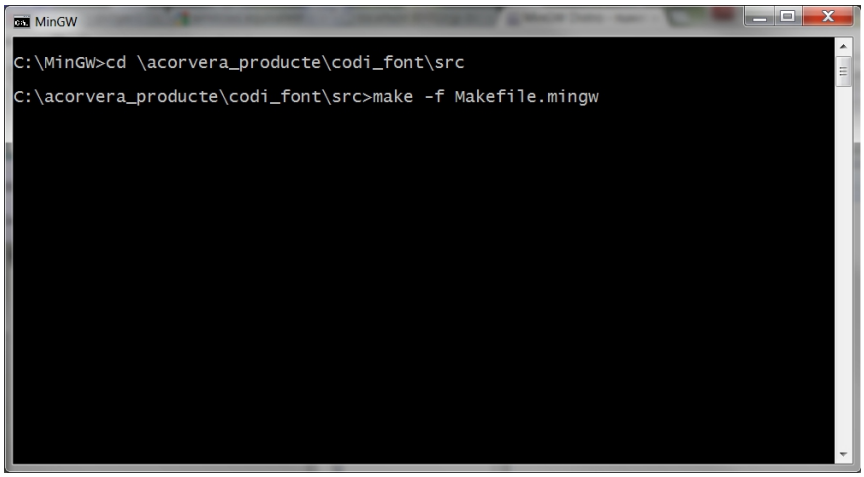
Vegeu l'annex III per instruccions per a la instal·lació de MinGW.

Un cop completats els passos de l'annex III es pot utilitzar MinGW per a compilar el codi d'aquest projecte. Suposant que el codi està en la carpeta `C:\acorvera_producte\codi_font` i que MinGW s'ha instal·lat en la ruta per defecte (`C:\MinGW`) seguirem els següent passos.

Obriu la carpeta `C:\MinGW`, que presentarà un aspecte similar a aquest:



Feu doble clic a "`open_distro_window.bat`" per obrir una finestra ja configurada per a poder compilar fàcilment. En aquesta, canvieu al directori del codi font (`cd C:\acorvera_producte\codi_font\src`).
Per a compilar el projecte, executeu la comanda « `make -f Makefile.mingw` ».



Quan el procés acabi, si no s'ha produït cap error, l'executable del projecte s'anomenarà `tfcweb-mingw.exe`.
Cal tenir en compte que per executar-lo necessitareu tenir a la mateixa carpeta l'arxiu de DLL "`zlibwapi.dll`", que s'inclou en el paquet del codi font. La construcció correcta el copiarà la carpeta de construcció, també el podeu copiar manualment des de la carpeta `win32`. Per a copiar-lo a la carpeta actual executeu la comanda:

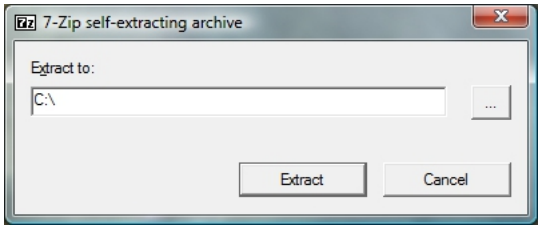
« `copy acorvera_producte\codi\win32\zlib-1.2.5\dll32\zlibwapi.dll .` »

Annex III. Instruccions d'instal·lació de MingGW en Windows

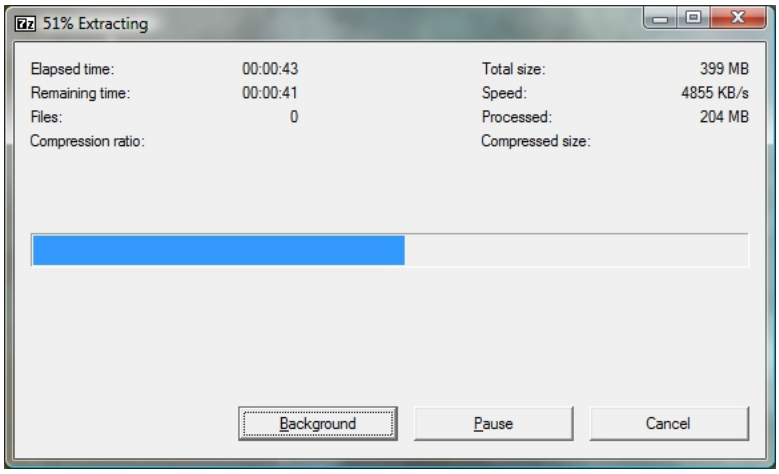
MinGW és una versió per a Windows del compilador GCC, utilitzat habitualment en entorns UNIX. Per a simplificar la seva instal·lació jut amb les biblioteques Boost es recomana la utilització de la a distribució de preparada per Stephan T. Lavavej (http://nuwen.net/mingw.html), que ja inclou aquestes biblioteques pre-compilades i per tant permet compilar el programa sense cap pas extra.

Per instal·lar la MinGW el primer pas és obtenir l'instal·lador, disponible en l'adreça http://nuwen.net/files/mingw/mingw-9.0.exe. Un cop descarregat cal executar-lo i es procedirà a la instal·lació. A continuació es mostra el procés en forma de captures de pantalla:

Primer pantalla: Lloc d'instal·lació. MinGW crearà una carpeta (*MinGW*) dins la ruta que si indiqui aquí. Per defecte `C:\` (per tant s'instal·larà en `C:\MinGW`). Premeu "Extract" per a continuar.



Segona pantalla: Instal·lació. En aquest punt s'està instal·lant en la ruta indicada. El procés portarà algun temps degut principalment a la mida de les biblioteques Boost. Espereu a que acabi.



Quan acabi, MinGW ja estarà disponible en el sistema, la carpeta C:\MinGW contindrà el compilador i les biblioteques instal·lats. Vegeu el següent annex per als detalls per al seu ús.

Imatges extretes de http://nuwen.net/mingw.html.

Annex IV. Estructura d'exemple

Per a mostrar el servidor en funcionament s'inclou una estructura de carpetes i una configuració d'exemple amb permisos configurats i un CGI d'exemple, que es poden utilitzar de la següent manera:

UNIX/Linux

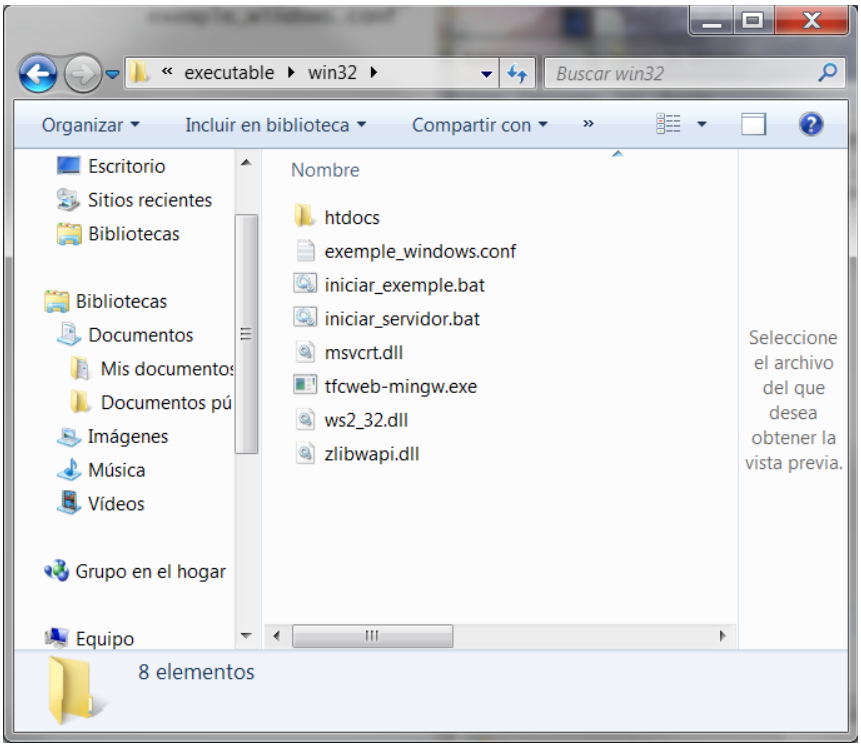
L'script `genera_htdocs.sh`, al directori principal del codi, generarà aquesta estructura. Des del directori `src/` podeu utilitzar-la carregant la configuració `exemple.conf`:

```
$ cd src
$ ../genera_htdocs.sh
$ tfcweb --config exemple.conf
```

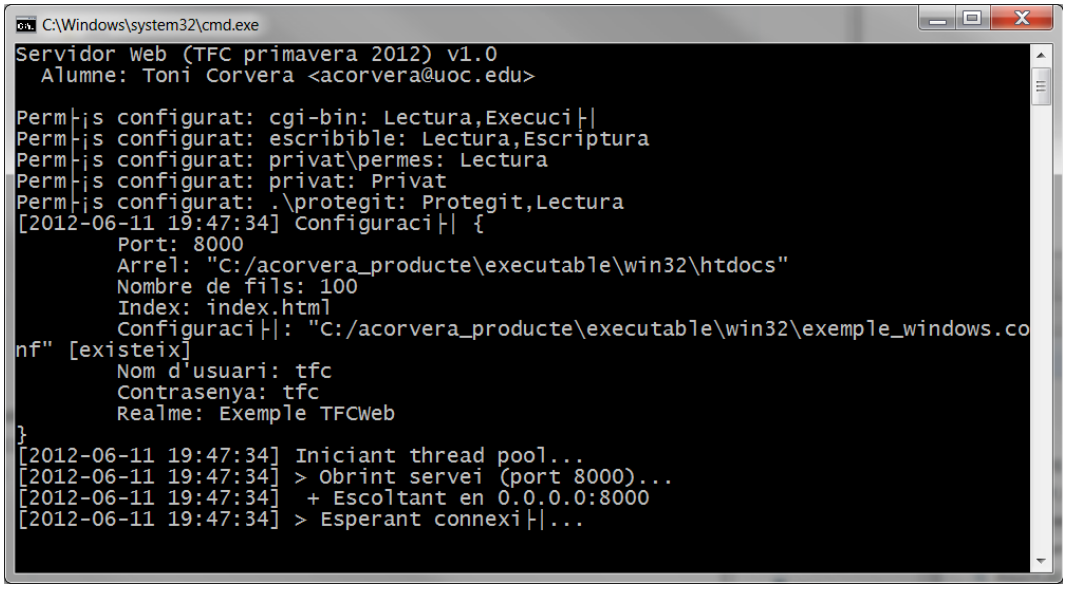
Un cop el servidor arrenqui, accediu a http://localhost:8000/LLEGIU-ME.html per a més detalls.

Windows

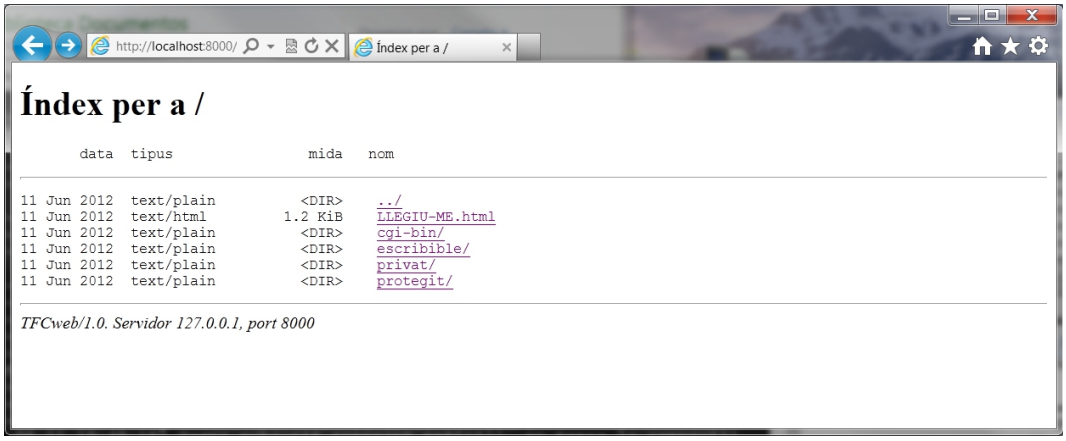
La carpeta de l'entrega ja conté la estructura per a Windows, junt amb una versió llesta per utilitzar del servidor, dins `\acorvera_producte\executable\win32`. En el seu interior el fitxer `iniciar_exemple.bat` iniciarà el servidor carregant la configuració d'exemple.



Fent doble clic en `iniciar_exemple.bat` es mostrarà la finestra del terminal en què el servidor estarà funcionant. Tancar aquesta finestra tancarà el servidor:



El servidor estarà disponible en el port 8000, que podiu accedir des de http://localhost:8000:



Accediu a http://localhost:8000/LLEGIU-ME.html per a més detalls.