Web Scraping: extracció i anàlisi de dades

Manuel León Mondéjar
Enginyeria en Informàtica

Jordi Ferrer Duran

12 Juny, 2011. Curs 2010 / 2011 (Segon semestre).

DEDICATORIA i AGRAÏMENTS

El present treball és el resultat de l'execució d'una sèrie d'activitats realitzades dins l'assignatura "Projecte Final de Carrera" i durant el segon semestre de l'any acadèmic 2010 / 2011.

Durant tot aquest període són diverses les persones que d'una forma o altra han col·laborat i ajudat en la consecució dels objectius definits al començament de l'assignatura.

Gràcies al totes les persones, i molt particularment als pares i germans, per la seva comprensió donat que moltes vegades aquest treball ha suposat una dedicació menor de temps cap a ells.

Gràcies també a totes aquelles persones que de forma desinteressada escriuen i donen suport a diferents portals, blogs, fòrums, etcètera, a Internet. Perquè moltes vegades, trobar l' informació adequada al lloc adequat suposa un important estalvi de temps i una petita ajuda a la millora de la solució lliurada.

Finalment, gràcies al consultor de l'assignatura Jordi Duran pel seu suport continuat durant el desenvolupament de tot el projecte, per la seva eficàcia a l'hora de resoldre dubtes i, en general, per la seva ajuda de cara a facilitar els diferents lliuraments del PFC.

RESUM

Aquest document conté tota la documentació necessària per a entendre el tipus de projecte desenvolupat, els objectius definits, la metodologia que s'ha emprat a l'hora de donar-li solució i tots els productes relacionats.

L'àrea de treball en la qual es centra aquest projecte final de carrera (d'ara endavant PFC), es l'**àrea de Compiladors**. Aquesta àrea es present es desenvolupa a les assignatures Compiladors I i Compiladors II del pla d'estudis d'Enginyeria Informàtica ofert per la UOC. Òbviament, a un treball de síntesi com aquest no és suficient amb el coneixement assolit a aquestes dues assignatures i es posen en pràctica coneixements teòrics i pràctics assolits en altres assignatures com "Metodologia i gestió de projectes informàtics", "Enginyeria del programari orientat a l'objecte" o "Enginyeria del programari de components i sistemes distribuïts".

Aquesta memòria comença amb un capítol d'introducció en el qual s'introdueix el tema del projecte, els objectius específics del projecte i s'explica l'enfocament que es durà a terme per a complir amb els objectius definits a l'enunciat. També dóna informació a alt nivell de la descomposició en tasques del projecte, segons l'enfocament donat, i per tant de la planificació completa del projecte.

El cos de la memòria el formen els capítols 2, 3, 4, 5 i 6 que detallen diferents parts de desenvolupament del projecte.

Finalment, es dóna un glossari de diferents termes que sorgeixen al document així com la bibliografia durant el desenvolupament de tot el projecte.

El document finalitza amb un apartat d'annexos que donen altra informació d'aspectes globals del PFC.

ÍNDEX DE CONTINGUTS

ÍNDEX DE FIGURES

MEMÒRIA

1.1.- Introducció

1.1.1 Justificació del PFC i context en el qual es desenvolupa

Aquest projecte serveix per a posar en pràctica a un mateix treball coneixements adquirits al pla d'estudis d'Enginyeria Informàtica fent us també de tècniques i eines de gestió de projectes, que varen ser tractades a l'assignatura de "Metodologia i gestió de projectes informàtics". Tot aquest treball es documenta a la present memòria i a un document de presentació final que també es dóna com a part del lliurament final.

Partim d'un context en el qual gairebé tothom veu a Internet com una eina d'us en la seva vida quotidiana: Correu electrònic, xarxes socials, mitjans de comunicació digitals, banca electrònica, etc. Internet dóna accés a un immens conjunt d'informació que es troba dispersa arreu del món i es dins aquest context on operen determinades eines automatitzades que rastregen part d'aquesta informació per a filtrar informació útil i per a extreure estadístiques globals com pot ser, per exemple, els llibre més cercat a Internet. Un exemple molt popular d'aquest tipus de sistema es l'aranya de Google.

Amb la nova web semàntica, les pàgines webs no son un simple conjunt de fitxers HTML desordenat sinó que poden ser vistes com a fonts d'informació per a la gran base de dades d'Internet. Per exemple, una aplicació pot accedir a través d'una eina automatitzada a l'API que ofereixen altres aplicacions, unificant després el conjunt d'informació rebut i presentant-lo a l'usuari d'una forma més amigable, que no haurà de visitar les diferents planes que son font d'informació per separat.

Web Scraping es una tècnica que extrau informació de llocs web però que ho fa d'una manera més rudimentària que la que s'ha vist anteriorment. El concepte de Web Scraping es simple d'entendre: es realitzen connexions amb altres servidors i, simulant el que seria una navegació humana, s'extrau (es "raspa") el codi HTML i altra informació d'aquesta navegació. Aquesta informació extreta de forma totalment automatitzada s'analitza posteriorment, fent comparacions entre les diferents fonts d'informació i extraient informació útil segons els objectius definits.

Serà objectiu fonamental d'aquest projecte l'aprenentatge de tècniques de Web Scraping aplicades al cas concret que ens ocupa: cerca d'informació de preus d'hotels i de vols. Al següent apartat es dóna una llista més acurada dels objectius d'aquest PFC.

1.1.2 Objectius del PFC

L'organització VTC (Viatja en Temps de Crisi) vol que els hi construïm un cercador web que serà accessible des de la seva aplicació web. Aquest cercador es connectarà a diferents operadors d'hotels i de vols, farà una cerca de preus per a uns determinats valors dels camps d'entrada del cercador i finalment, amb aquesta informació de preus, farà el càlcul de les cinc combinacions possibles de vol més hotel que en resultin en total més econòmiques. Aquest cercador que haurem d'implementar serà el producte principal d'aquest PFC.

Amb aquest projecte, i considerant el tipus de pla d'estudis i l'àrea d'àmbit del PFC, es pretenen assolir els següents objectius:

- Posar en pràctica coneixements del pla d'estudis, aplicant-los a un cas complex i d'utilitat dins l'Enginyeria Informàtica
- Adquirir coneixements relacionats amb l'exercici professional de l'Enginyeria Informàtica
- Posar en pràctica habilitats de comunicació oral i escrita, enfocant els esforços en la realització de la memòria del projecte i en la comunicació amb el promotor del projecte (consultor)
- Utilitzar eines i mecanismes de gestió de projectes: planificar amb detall el desenvolupament del projecte, definir fites, lliurables i realitzar tasques de control i seguiment
- Estudiar llibreries de Web Scraping i aplicar-les de forma successiva a la solució del projecte
- Aprendre tècniques d'extracció d'informació útil de documents HTML segons uns patrons de cerca determinats
- Fer us de llenguatges de marcat com XHTML, XML i XSL, ja estudiats a l'assignatura Compiladors II
- Combinar totes les tecnologies esmentades a l'hora de filtrar i mostrar les combinacions de preus més econòmiques

1.1.3 Enfocament

En relació a l' implementació del cercador, s'ha decidit utilitzar J2EE com a tecnologia d'implementació. Aquesta decisió fixarà totes les diferents tècniques i llibreries que s'utilitzaran a l' implementació del cercador.

Centrant-nos a l'estructura del sistema, es poden veure tres parts (actors) involucrades:

- Aplicació web: Serveix d'interfície entre l'usuari que accedeix a la plana de VTC per a consultar preus d'hotels i vols i el motor de cerca. Presenta un formulari a l'usuari el qual emplena amb un determinat filtre de consulta. Aquest formulari realitza una petició HTTP contra el back-end del sistema. Una vegada obté la resposta, la visualitza al navegador de l'usuari.
- Motor de cerca: Part central i objecte fonamental del desenvolupament. De fet, l'enunciat especifica la conveniència de centrar els esforços en construir un motor potent i de qualitat per sobre d'altres elements que més aviat només aporten valor afegit com pot ser l'aplicació web. El motor de cerca analitza la petició de l'aplicació web, que rep per HTTP, recull l' informació útil dels diferents operadors, processa els resultats i finalment retorna la resposta a l'usuari.

- Operadors externs: Servidors webs externs que reben peticions HTTP adaptades segons la petició inicial de l'usuari i que retornen l' informació HTML que visualitzaria un usuari que estigués visualitzant les mateixes pàgines a aquest servidor i amb el mateix filtre de cerca.

Aquests actors junt amb la relació entre ells queda present a la següent figura:

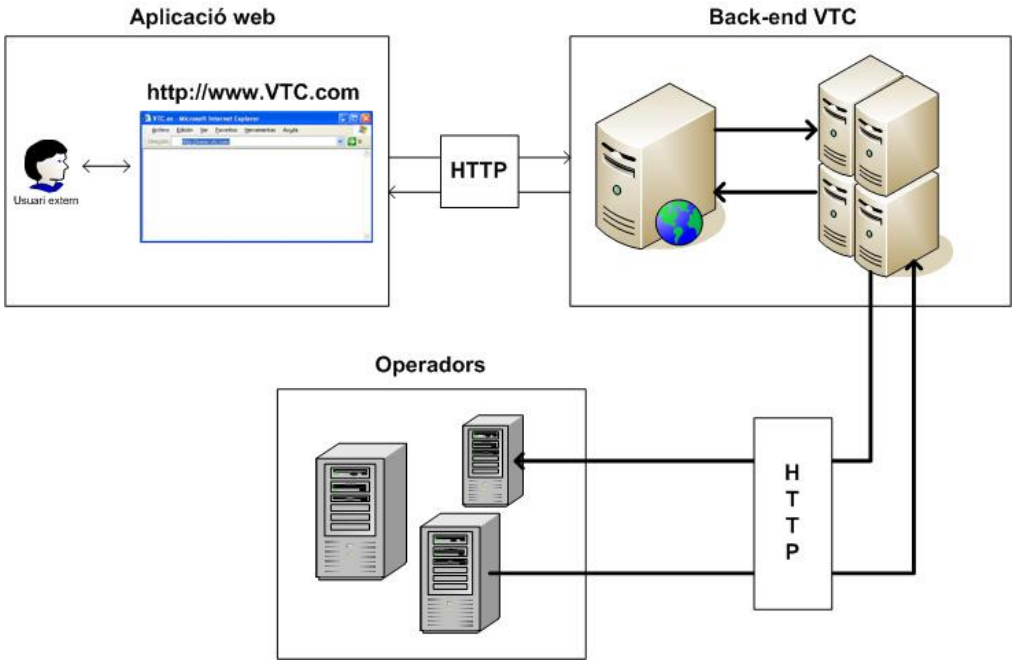


Figura 1. *Actors del sistema.*

Aclarit el flux general d'execució, es defineix ara amb més detall el flux d'execució que haurà de seguir el motor de cerca, el component que consulta els preus de vols i estàncies en hotels, ordena els preus, calcula les combinacions i retorna les cinc combinacions més econòmiques en una resposta amb format HTML:

1. Connexió amb els operadors externs: El cercador rep una petició (*request*) per HTTP relativa a una consulta de cerca d'un usuari. Analitza la petició separant els camps útils i analitzant-los, retornant un missatge d'error si qualque camp no es correcte (per exemple, la data d'inici es posterior a la data de fi). Si els paràmetres del request son correctes, estableix connexions HTTP amb els servidors web dels diferents operadors i obté l' informació HTML que visualitzaria un usuari humà amb el mateix filtre de consulta que el rebut per part del cercador. Una vegada finalitzen les connexions per a un operador determinat, l' informació consultada s'analitza (següent pas).
2. Filtre i ordenació dels resultats: De l' informació consultada es filtren els camps útils: només interessen per a cada operador els 5 resultats més econòmics i dins cada resultat alguns camps com el preu i el nom del hotel. Segons es va llegint aquesta informació, es va construint un objecte de dades per a cada operador a fi de, finalment i amb tota l' informació ordenada de tots els operadors, obtenir les 5 combinacions totals més econòmiques de vol + hotel.
3. Crear fitxer XML: L'objecte de dades final es parseja amb un codificador XML i es crea un fitxer XML que conté l' informació representativa de les 5 combinacions.
4. Transformació XSLT: Finalment, s'aplica una transformació XSLT a l' XML i s'obté un HTML de sortida. Aquest HTML es el que l'usuari finalment visualitzarà al seu navegador.

Gràficament podem veure tot aquest procés a la figura de la plana següent, on es divideix el servidor d'aplicacions on s'executa el cercador en altres quatre segons aquesta seqüència d'activitats definida anteriorment:

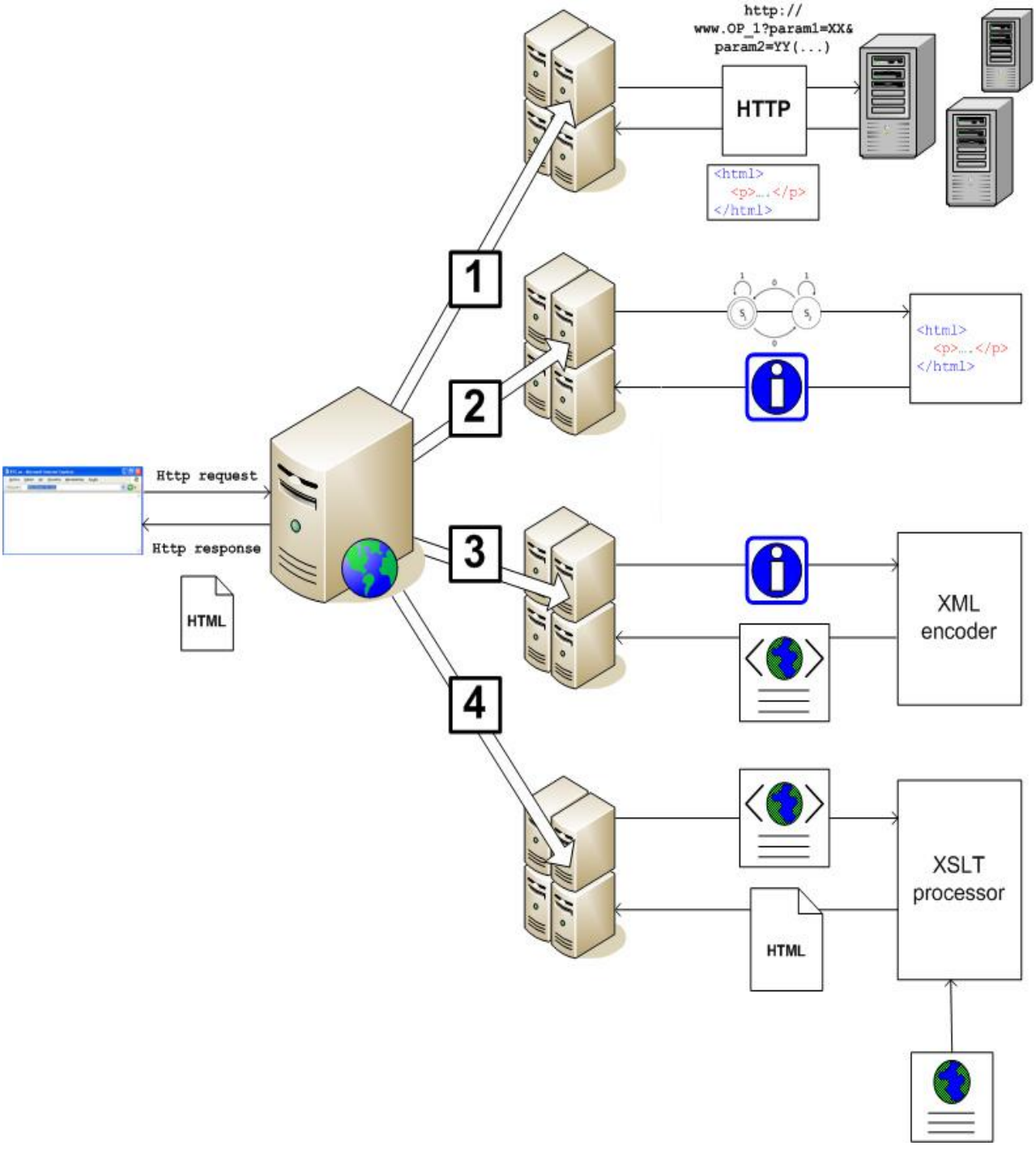


Figura 2 *Etapes del cercador*

1.1.4 Planificació del projecte

S'ha planificat el projecte dividint-lo en tasques, definint les estimacions temporals i establint les diferents fites.

Les tasques definides son les següents:

0	Instal·lació i configuració de l'entorn
Descripció	Instal·lació i configuració de l'entorn: servidor d'aplicacions, entorn de desenvolupament, eines d'edició, etc. També es confecciona (índex i estructura) el document que servirà de base a la memòria.

1.1	Estudi de l'API HttpClient
Descripció	Instal·lació, estudi i execució d'exemples utilitzant l'API d' Apache HttpClient.

1.2	Cerca i elecció dels servidors web
Descripció	Es cercarà per la web pàgines web que retornin informació útil de reserves de vols i hotels. Això inclou conèixer exactament el tipus d'informació cercada i estudiar com podem connectar les connexions per obtenir les dades en format HTML.

1.3	Lectura dels resultats externs de cerca
Descripció	Programació del mòdul que es connecta amb els servidors web externs i recupera l' informació. Construcció d'un petit formulari de dades que envia els paràmetres de cerca a aquest mòdul i que serveix d'interfície.

2.1	Estudi de les expressions regulars
Descripció	Estudi de la construcció d'expressions regulars i la seva aplicació en Java.

2.2	Estudi de l'estructura del codi HTML
Descripció	Estudi més detallat de l'estructura del codi HTML capturat amb l'objectiu de definir les expressions regulars que encaixen amb l' informació útil.

2.3	Aplicació de les expressions regulars
Descripció	Implementació del codi que aplica les expressions regulars al codi HTML.

3.1	Construcció de l'objecte de dades
Descripció	Es defineix un objecte amb atributs que representen l' informació útil d'una reserva. Aquest objecte s'inicialitza amb l' informació extreta a l'apartat anterior.

3.2	Estudi de l'API JAXB
Descripció	Estudi dels fonaments de JAXB i de la seva aplicació al nostre cas concret.

3.3	Construcció del codificador XML
Descripció	Construcció del codi que llegeix un objecte Java i ho emmagatzema a un fitxer XML utilitzant l'API JAXB.

4.1	Estudi de l'API JAXP
Descripció	Estudi dels fonaments de JAXP i del funcionament del processador XSLT Saxon.

4.2	Construcció de la transformació XSLT
Descripció	Construcció de la plantilla XSLT i de l'aplicació de la transformació XSLT.

5	Programació de l' interfície web
Descripció	Construcció de les planes web, de la crida al motor de cerca i de la visualització de resultats.

6	Proves
Descripció	Si be es realitzen proves amb cada desenvolupament, es reserva un temps per fer proves del conjunt de l'aplicació: Es testeja la funcionalitat des del punt de vista de l'usuari i es comprova que el funcionament es correcte segons les especificacions i segons diversos camps de cerca. Inclou les tasques necessàries per a corregir errors trobats.

7.1	Realització de la memòria
Descripció	Es realitza la memòria del projecte. Es va realitzant segons s'avança al projecte. Es deixa una setmana de temps per tancar i revisar la memòria.

7.2	Realització de la presentació
Descripció	Es realitza la presentació del projecte. Es la darrera tasca del projecte i finalitza un dia abans del dia de lliurament.

	Es reserva el dia de lliurament final per fer una revisió general i muntar els arxius de lliurament.

8	Debat virtual
Descripció	Uns dies després del lliurament dels productes que composen el PFC s'inicia un debat virtual amb els components del tribunal d'avaluació. Aquest debat té una duració de 3 dies.

9	Seguiment
Descripció	Es una tasca periòdica. Es fa una revisió de l'estat del projecte en relació a la planificació inicial, a l'estat de la documentació i a l'estat d'avanç del projecte. En cas de desviacions es prenen mesures com, per exemple, mes hores de dedicació al projecte o avançament d'altres tasques.

El diagrama de Gantt resultant de la planificació és el de la figura següent:

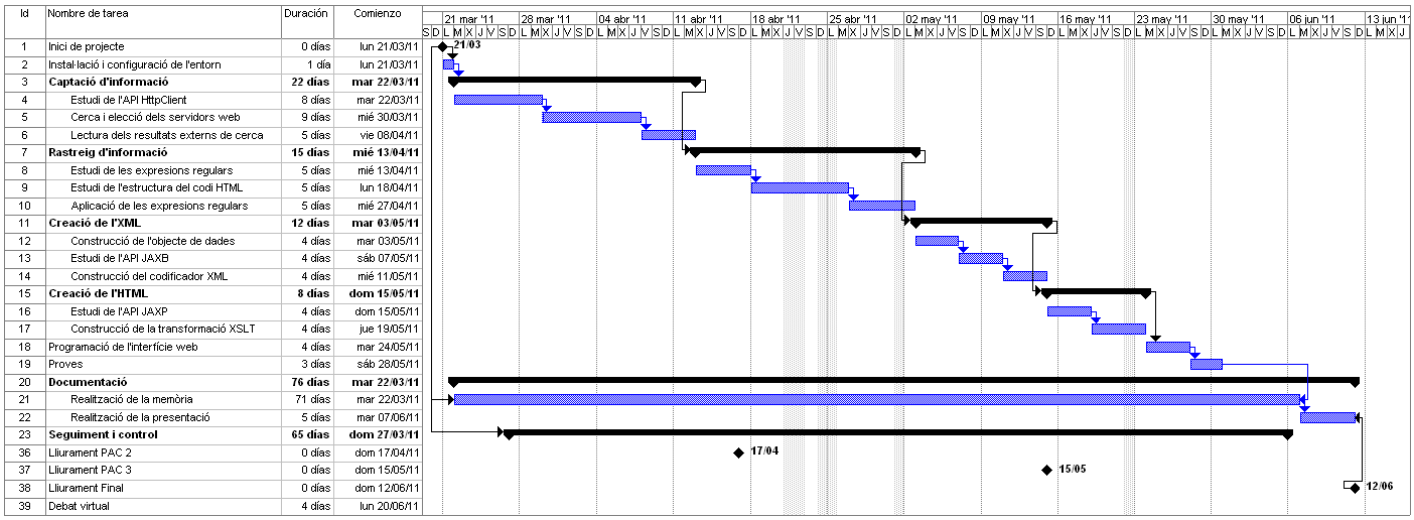


Figura 3. *Diagrama de Gantt.*

Les fites definides son les següents:

Data	Descripció
21 Març	Inici projecte.
17 Abril	Lliurament PAC 2
15 Maig	Lliurament PAC 3
12 Juny	Lliurament Final
23 Juny	Inici debat virtual

1.1.5 Seguiment de la planificació

A la finalització del projecte i en línies generals, podem dir que la planificació feta inicialment s'ha pogut complir sense desviacions molt importants.

De l'execució del projecte, es poden extreure un seguit de conclusions amb relació al seguiment que es va fer a les primeres activitats del projecte. Hi ha hagut també un seguit d'incidències que mereixen unes línies al present apartat A continuació es fan tots aquests comentaris:

- Els dies de lliurament de les PACs 1, 2 no haurien d'haver sigut dies de treball a altres tasques. Per retràs a tasques anteriors, per l'ajornament de les proves o simplement perquè els dies de lliurament de PACs (i també el lliurament final) marquen fites importants al projecte, haurien de ser dies dedicats en exclusiva a la revisió i finalització d'aquests lliuraments. En aquest sentit, pot resultar convenient crear una tasca específica de preparació i finalització d'aquests lliuraments.

- Les tasques "*Estudi de les expressions regulars*" i "*Estudi de l'API JAXP*" han estat una mica sobredimensionades. Junt amb l'estudi de les tecnologies va donar temps a realitzar proves amb casos relacionats amb el PFC i, en aquest sentit, avançar tasques següents. A l'estudi de l'API JAXB també es va donar això, i la tasca d'estudi de l'API va avançar un parell de dies de treball de la tasca "Construcció del codificador XML".

- Les tasques "Cerca i elecció dels servidors web" i "Lectura dels resultats externs de cerca" varen sofrir un petit retràs (complicacions a la cerca dels operadors, en conjunt massa operadors trobats amb detalls molts específics per cadascun d'ells) que va afectar a l' inici de les tasques següents. La finalització d'aquestes tasques va finalitzar amb el lliurament de la PAC 1,

- La realització de la memòria és una tasca periòdica i que s'ha anat actualitzant a cada tasca del projecte. Tasques com la "Programació de l' interfície web" haurien de contemplar qualque dia més doncs aquest treball s'ha d'incloure. Aquesta petita desviació ha anat en detriment del temps planificat a finalitzar la memòria, dins el període entre el 31 de maig i el 6 de juny.

- La confecció de la presentació no ha estat la darrera tasca, després d'aquesta tasca es varen fer altres modificacions a la memòria (concretament les seccions d'agraïments i aquesta mateixa. També es va fer un repàs ortogràfic en conjunt). Aquesta tasca va estar sobredimensionada a un parell de dies que es varen dedicar al repàs ortogràfic de la memòria.

- Una tasca molt important durant el projecte han estat les proves. Crec que hagués sigut convenient incloure una tasca de proves tant per al lliurament de la primera PAC com per al lliurament de la segona. Al lliurament de la segona PAC es varen detectar segons quines errades a les expressions regulars (en part per falta de proves anteriors i per canvis al codi HTML de l'scraping) que varen suposar una major dedicació de temps al projecte. Finalment, i també amb relació a les proves de la solució, crec que a un projecte com aquest on bona part del funcionament depèn del codi HTML llegit dels operadors (que pot canviar, i de fet ha canviat) és important no fer la darrera fase de proves amb massa antelació, com es va planificar.

1.1.6 Productes obtinguts

Els productes obtinguts amb la finalització del PFC son els següents:

Cercador: Empaquetat que conté el codi font i els compilables relatius al cercador web i a l'aplicació web que serveix d'interfície entre l'usuari i el cercador.

Memòria: Document que seguint una determinada estructura documenta tot el projecte i sintetitza tot el treball realitzat detallant la metodologia utilitzada en la solució del projecte.

Document de presentació: Document de presentació que sintetitza el treball realitzat al llarg del semestre.

1.1.7 Descripció dels capítols de la memòria

Es comenten breument els diferents capítols que constitueixen el cos de la memòria:

Capítol 2. Cerca d'informació

Introducció i especificació de la solució que cobreix el flux d'execució del cercador en relació a l'obtenció del codi HTML amb informació útil dels diferents operadors de vols i hotels seleccionats.

Al diagrama de Gantt, la seva finalització coincideix amb la PAC 1.

Capítol 3. Captura de dades

Introducció i especificació de la solució que cobreix el flux d'execució del cercador des que s'han consultat els preus dels operadors fins que es té un objecte de dades (vector) amb les combinacions més econòmiques de vol + hotel de totes les consultades.

Capítol 4. Creació de l' XML

Introducció i especificació de la solució que cobreix el flux d'execució del cercador des que es rep un objecte de dades fins que es crea un arxiu XML conseqüència de parsejar aquest objecte de dades.

Al diagrama de Gantt, la seva finalització coincideix amb la PAC 2.

Capítol 5. Creació de l' HTML

Introducció i especificació de la solució que cobreix el flux d'execució del cercador en relació a la transformació de l' XML a un document HTML de sortida.

Capítol 6. Interfície web

Introducció i especificació de la tasca de desenvolupament de l' interfície web. Es la darrera tasca dins la part d'implementació i consisteix a construir el cercador que visualitzaran els usuaris al seu navegador i a convertir l'aplicació d'escriptori de treball (dins un entorn local) a una aplicació web.

1.2 Capítol 2. Cerca d'informació.

1.2.1 Introducció

Aquest primer capítol inclou la descripció de totes les tasques necessàries per a llegir el codi HTML amb l' informació de la consulta de preus en tots els operadors. Es a dir, tindrem a variables temporals el codi que un usuari que es connectés directament a la plana dels diferents operadors visualitzaria al seu navegador. Òbviament, sempre i quan el filtre de consulta sigui el mateix.

Tot aquest procés ho haurà de fer de forma automatitzada i per a cada operador el cercador que estem dissenyant i que utilitzarà l'organització VTC al seu aplicatiu web.

A la planificació inicial tot aquest treball està estructurat en tres tasques ben diferenciades. Aquestes tasques son les següents:

- Estudi de l'API HttpClient
- Cerca i elecció dels servidors web
- Lectura dels resultats externs de cerca

Com es pot veure, abans d'escollir els servidors web que retornen l' informació útil junt amb la posterior captura del codi HTML i el seu emmagatzemament a variables temporals de l'aplicació hi ha una tasca bastant important i que es la de l'estudi de l'API HttpClient.

Ajudarà molt a l'hora de cercar els servidors webs conèixer quines eines tenim per fer peticions als mateixos i recuperar l' informació, i tot aquest coneixement s'assolirà a la primera tasca d'aquesta primera part del PFC que es comenta a continuació.

1.2.2 Fonaments d' HttpClient

HTTP (*HyperText Transfer Protocol*) es avui dia el protocol més àmpliament emprat a Internet. Cada vegada que fem la petició d'una plana web amb el nostre navegador s'inicia una petició (request) a un servidor remot. Si tot va be, aquest servidor ens respondrà amb una resposta que finalment el nostre navegador rep i haurà de tractar. Tota aquesta comunicació bàsica entre components d'una arquitectura web es fa mitjançant HTTP.

L'actual versió d' HTTP es la 1.1. Aquesta versió queda perfectament definida a la norma RFC 2616 (http://tools.ietf.org/html/rfc2616). RFC és l'acrònim de "Request For Comments", un document publicat per l'organització IETF i que descriu exhaustivament un protocol, comportament o mètode aplicat a la xarxa Internet. Com dèiem, HTTP 1.1 està especificat a la norma RFC 2616, la qual es una actualització de la norma RFC 2068.

Així doncs, necessitem una eina que simuli les peticions i respostes que es fan per HTTP de tal forma que es connecti a servidors webs remots i retorni les respostes a un conjunt de peticions que el nostre cercador haurà de muntar. S'ha escollit HttpClient com a API que automatitza tota aquesta feina i que ens permet despreocupar-nos dels detalls de mes baix nivell relatius al protocol HTTP.

HttpClient dóna molta més flexibilitat i funcionalitats que el package *java.net* disponible a l'API estàndard de Java, motiu pel qual s'ha escollit aquesta llibreria com a API a aquesta primera part del PFC. Es a dir, tenim una eina robusta i testejada que dóna bastants facilitats a l'hora de fer

connexions HTTP amb altres servidors. El fet de que HttpClient sigui un mòdul creat per la Apache Software Fundation fa que a més tinguem la garantia de estar fent us d'un producte amb una qualitat i robustesa més que garantida.

HttpClient dóna suport a el que seran totes les nostres necessitats: Connexions per HTTP, implementació dels mètodes HTTP més comuns (bàsicament GET i POST), abstracció dels dispositius intermedis (proxies, routers i demés), suport per a execucions concurrents, manipulació de cookies, enviament de capçaleres HTTP,…, i tot perfectament compatible i integrat amb la plataforma J2EE. Es important també, dir que HttpClient compleix amb les especificacions de las versions 1.0 i 1.1 d' HTTP.

Hem comentat funcionalitats i avantatges d' HttpClient, però no és or tot el que llueix… HttpClient té com a principal responsabilitat el protocol HTTP: executa peticions i rep respostes, abstraient de determinats detalls de tota la comunicació. Però HttpClient no és una simulació del comportament d'un navegador web: No dóna format ni corregeix informació HTML desestructurada, no fa us de la cache dels navegadors, no dóna suport a altres protocols d'aplicació, no executa codi javascript, no té res a veure amb una interfície web o presentació de contingut HTML… És bàsicament una utilitat que ens ajudarà en la captura d'informació HTML simulant les connexions del navegador, i es tasca d'aquest projecte el analitzar el seguit de peticions i respostes HTTP que haurem d'enviar amb HttpClient per a obtenir finalment l' informació desitjada. La següent figura es bastant il·lustrativa en relació al que es responsabilitat d' HttpClient i el que queda fora del seu abast:

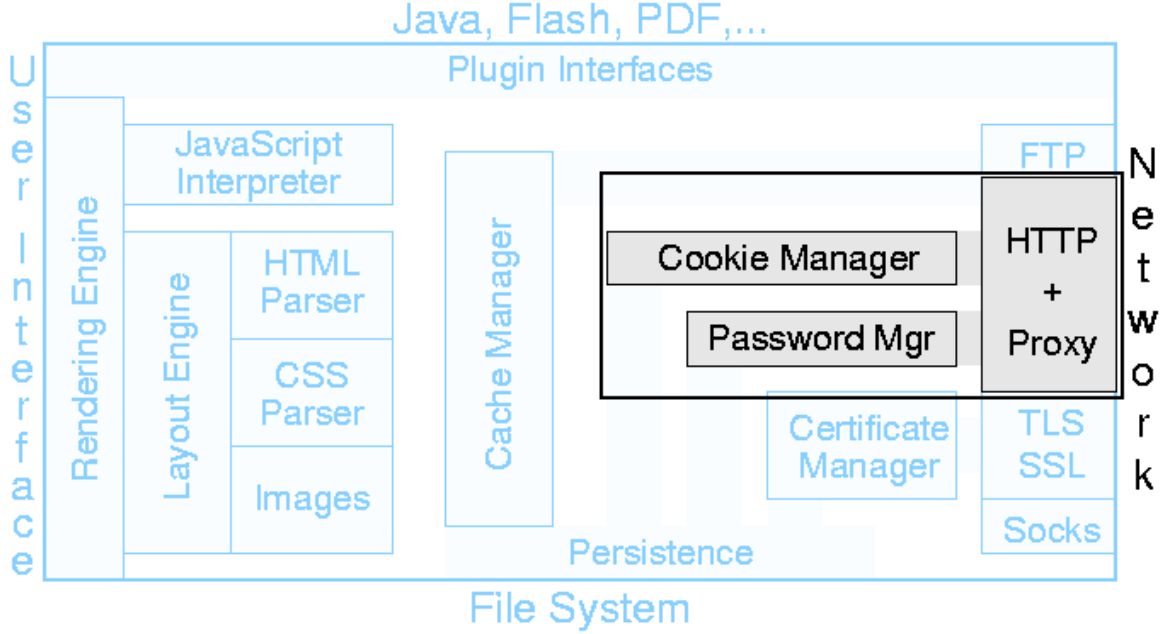


Figura 4. *Àmbit HttpClient*

HttpClient ens servirà per a obtenir l' informació HTML que els servidors web dels operadors envien per a cada tipus de petició, però no forma part de la millor estratègia possible amb relació al disseny del motor de cerca.

Ens servirà per a capturar codi HTML que posteriorment haurem d'analitzar… però aquest codi HTML pot canviar doncs és un contingut dinàmic: Per exemple, pot canviar els nom dels camps/filtres de cerca, les adreces intermèdies fins a mostrar els resultats,…, fins i tot poden canviar les adreces entre successives peticions HTTP. Això ens fa pensar que l'ús d' HttpClient no com a tal sinó com a part d'una estratègia que obté informació útil rastrejant codi HTML es una mala

pràctica i per tant no és la millor possible doncs en cas de que aquesta aplicació tingués un fi comercial, tindríem que estar contínuament revisant-la per si de cas es produeixen canvis a la manera en com els operadors realitzen les crides per HTTP. De totes formes, per al que son els objectius d'aquest PFC i fonamentalment per a l'aprenentatge del Web Scraping, aquesta solució es força vàlida. Altra solució, més estable en cas de que aquesta aplicació tingués un fi comercial, hauria sigut fer us d'APIs privades que donessin accés a diferents cercadors d'estàncies en hotels i de vols. Això facilita molt la tasca, doncs es clar que el resultat de la crida a una determinada funció d'aquest tipus d'API no variarà des del punt de vista del client, i per tant ens estalvia la preocupació d'estar pendents de si el codi HTML dels servidors webs ha canviat o no.

Donem a continuació un seguit de definicions que seran útils a l'hora d'entendre posteriors comentaris:

- **Missatge HTTP (*HTTP Message*)**: Format per una capçalera i, opcionalment, una entitat. Existeixen dos tipus de missatges:
    - **Petició HTTP (*HTTP Request*)**: Missatge enviat d'un client al servidor. La primera línia inclou l' URI (*Uniform Resource Identifier*) a la qual es fa la petició i el mètode que haurà d'executar el servidor. Els mètodes més comuns son el GET i el POST.
    - **Resposta HTTP (*HTTP Response*)**: Missatge enviat d'un servidor a un client en resposta a una petició. La primera línia inclou un codi que indica l'estat de tractament de la petició (*success, not found, failure, etc*)
- **Capçalera**: La capçalera forma part d'un missatge HTTP i està formada per un conjunts de parells nom/valor. Existeix un ampli conjunt de possibles noms pels camps d'una capçalera (com per exemple el tipus de navegador que envia una petició) i no tots els tipus de missatges tenen els mateixos possibles camps. Per exemple, un missatge de resposta té camps a la capçalera que una petició no en té.
- **Entitat**: L' informació inclosa a un missatge HTTP. Per exemple, a una petició hi poden anar tots els camps enviats a un formulari i a una resposta el contingut d'una imatge. Com es veu, el tipus de dada pot variar i es el camp MIME type de la capçalera el que dóna informació del tipus de dada inclosa a l'entitat.
- **Sessió**: Conjunt de peticions fetes des del mateix client. A vegades interessa fer us d'una sessió per a conèixer en tot moment l' informació associada al client. Per exemple, a una reserva on-line d'un vol existeix informació associada a l'usuari com les dades de cerca o el nombre de la targeta de crèdit que estan associades a ell i que haurien de mantenir-se fins al fi de la sessió.

Una vegada aclarats tots aquests conceptes, fonamentals per a implementar el codi que haurà de connectar-se als servidors web i tenint en compte les possibilitats d'HttpClient, es dóna una llista de, a priori, totes les característiques que seran necessàries d'aquesta API:

- Generació de peticions HTTP als servidors webs, on s'executen tant mètodes GET com mètodes POST. Inclou tot lo necessari per formar les peticions: capçaleres, paràmetres, etc

- Recepció de resposta per part dels servidors web després d'una petició HTTP

- Gestió de connexions HTTP: Definició de les connexions actives, Gestió de connexions concurrents i alliberament de recursos.

- HTTP es un protocol sense estat: Necessitem per tant de qualque mecanisme per a identificar la sessió d'un usuari. Per a això farem us de **cookies**. Les cookies o galetes son petits tokens que els navegadors emmagatzemen físicament a fitxers de l'ordinador dels usuaris que es connecten a un servidor web i que son generades per aquest servidor amb l' intenció de recuperar aquesta informació amb posterioritat. Per exemple, el servidor web d'una plana de venta on-line de llibres pot generar una cookie amb els gustos dels usuaris de tal forma que davant una nova connexió d'un usuari, es llegeix la seva cookie amb el tipus de llibre preferit.

- També es útil la consideració de fer servir HTTP cache, que facilita l'emmagatzemament en cache de les peticions, de tal forma que noves peticions que existeixen a cache es serveixen de cache directament, estalviant temps de processament al cercador.

Els detalls d'implementació en relació a tot lo comentat fins ara en relació a l'API HttpClient estan especificats a l'apartat "Extracció de dades" d'aquest mateix capítol.

1.2.3 Fonts d'informació

Abans d'especificar les planes dels operadors que seran l'objectiu de la tasca de l' scraping, s'explica l'estratègia de cerca que s'ha triat en relació al tipus d'informació que s'haurà de cercar a les planes dels operadors.

Primer de tot, es necessari aclarir un concepte clau en aquest apartat: El concepte de "***operador***". Per al nostre objectiu un operador es equivalent a un servidor web que ofereix informació útil de cara a mostrar els resultats finals. Es a dir, una font d'informació de preus d'estàncies a un o més hotels o de vols entre ciutats. Cada operador serà objecte d'estudi per separat: s'ha de conèixer com es realitzen les connexions a fi d'arribar a l' informació final de disponibilitat i preus.

Com a primera decisió en l'estratègia de cerca d'aquests operadors, s'ha decidit fer una cerca separada d'hotels i de vols per a mostrar l' informació. Existeixen operadors que permeten fer una cerca alhora de vol i hotel i si be es considera una solució vàlida fer una cerca d'aquest tipus i una posterior comparació entre dos o més operadors que permeten fer reserves d'hotel i vol dins el mateix paquet no es aquesta l'alternativa escollida. **Es farà una cerca separada d'hotels i de vols** pero s'haurà de mostrar a l'usuari el preu final de la combinació d'hotel i de vol com un únic paquet.

Es a dir, per al conjunt d'hotels i de vols es faran totes les combinacions necessàries i del conjunt de combinacions s'acabaran mostrant les cinc més econòmiques. Òbviament no fa falta fer aquesta combinació per a tots els hotels i per a tots els vols: per exemple, en cas de que es trobin més de cinc hotels la combinació es fa en base als cinc més econòmics i els altres desapareixen de la llista de resultats.

Aquesta elecció dóna mes flexibilitat dons permet una elecció major d'operadors i per tant facilita una possible ampliació futura del cercador. També dóna més potència al cercador: es coneix el preu (així com altres detalls) per separat de cada component i dóna més robustesa doncs permetria amb poques modificacions al codi del cercador, reservar només l'hotel o només el vol.

Com dèiem abans, no és necessari conèixer per exemple tots els preus d'hotels retornats per un operador: és suficient conèixer els cinc que ofereixen una estància més econòmica. Per als operadors que ofereixen els resultats ordenats només es necessitarà la primera plana de resultats. Per a la resta, si es pot s'intentarà simular la petició a una URL que retorni els resultats ja ordenats:

la majoria d'operadors solen oferir un enllaç o similar amb aquesta possibilitat. Si la consulta de preus ordenats no es trivial, la darrera alternativa es llegir tots els resultats, establint totes les connexions necessàries i finalment fer l'ordenació en base a tots els resultats.

Fetes aquestes consideracions es dóna a continuació la llista d'operadors escollits, segons la separació de la cerca en hotels i vols. Es fa també un seguit de comentaris específics dins cada tipus de cerca.

1.2.3.1 Estudi genèric d'un operador

Tant si es tracta d'un operador que ofereix preus d'estàncies en hotels com si es tracta d'un operador que ofereix vols, a l'hora d'analitzar el codi font, l'estructura general de la pàgina i, fonamentalment les crides HTTP entre planes successives l'estratègia es similar en tot dos casos. Les diferències i peculiaritats de cada tipus d'operador es comentaran als apartats posteriors.

S'ha fet us d'un dels molts plugins o extensions del popular navegador Mozilla Firefox. El plugin escollit ha sigut HttpFox. Aquest plugin facilita molt la tasca d'estudi de cada operador i dóna moltes facilitats a l'hora de conèixer les crides que es realitzen per a consultar els preus a un operador i per tant a l'hora de traduir aquestes crides per HTTP a crides que haurem de programar amb l'API HttpClient.

El plugin es molt senzill d'utilitzar i s'ha fet us principalment per a conèixer el seguit de URLs que s'executen per a obtenir una informació determinada, junt amb el mètode HTTP associat a cada URL (GET, POST) i els paràmetres enviats a cada URL. En molts poc casos s'ha necessitat conèixer les capçaleres o les cookies enviades/rebudes. En resum, aquestes son les funcionalitats del plugin utilitzades:

- Columna *Type*: Ens diu contra quin tipus de recurs es fa una petició HTTP. El tipus de contingut que interessa es el marcat com a "text/html", que es el que conté l' informació dels formularis de cerca, dels resultats, dels passos intermedis, etc. No obstant, a vegades es necessita conèixer el contingut a peticions de codi javascript (application/javascript) doncs per exemple amb codi javascript es poden assignar valors rellevants a paràmetres summitats al formulari de cerca.

- Columnes *Method* i URL: Les més importants: Es diu l' URL exacta que es crida i amb quin mètode HTTP: GET o POST.

- Columna *Started*: Serveix per a conèixer el temps des que s'inicia la primera crida que van tardant en executar-se les peticions HTTP. A vegades ajuda també a conèixer el temps aproximat que ha tardat en executar-se el cercador de l'operador i poder simular aquest comportament, en temps d'espera amb el nostre cercador.

- Part inferior: *Capçaleres i cookies*. Seleccionant una petició HTTP veiem a la finestra inferior altra informació relacionada amb aquesta petició: Les capçaleres que s'envien, les capçaleres que es reben, les cookies que aquesta petició rep i les cookies que genera. Podem conèixer per exemple, a quina petició s'ha generat cada cookie creada per un servidor concret.

- Part inferior: *Paràmetres*. A la pestanya *Query String* es mostren els paràmetres (nom i valor) enviats a una petició Get i a la pestanya *POST Data* els paràmetres enviats a una petició tipus POST. El nom de la URL completa (protocol, servidor, recurs i paràmetre) es

coneix de forma diferent segons el tipus del mètode HTTP. Amb mètodes GET, polsant amb el botó dret del ratolí al camp URL de la finestra superior podem capturar la URL completa. Amb mètodes POST, només capturem la URL sense paràmetres doncs aquests formen part de l'entitat transmesa junt amb la petició. Per a conèixer el llistat complet de paràmetres, a la pestanya *POST Data* podem activar l'opció "Raw" del radio button per a poder visualitzar la concatenació completa del llistat de paràmetres enviat (Per defecte es mostren per separat, cada paràmetre a una línia diferent).

- Part inferior: Contingut. A la pestanya *Content* es veu el contingut enviat a l'entitat de la resposta a una petició HTTP. Per exemple, si s'ha sol·licitat un arxiu d'estils CSS, podem veure el contingut del CSS. El més interessant es poder copiar el contingut de les peticions d'arxius HTML (type=text/html) a fi de poder fer comparacions posteriors entre aquest contingut i el contingut retornat pel servidor executant la mateixa petició amb l' HttpClient.

A la següent figura podem observar HttpFox en execució, ressaltant les àrees d'informació més rellevants, que han estat esmentades anteriorment:

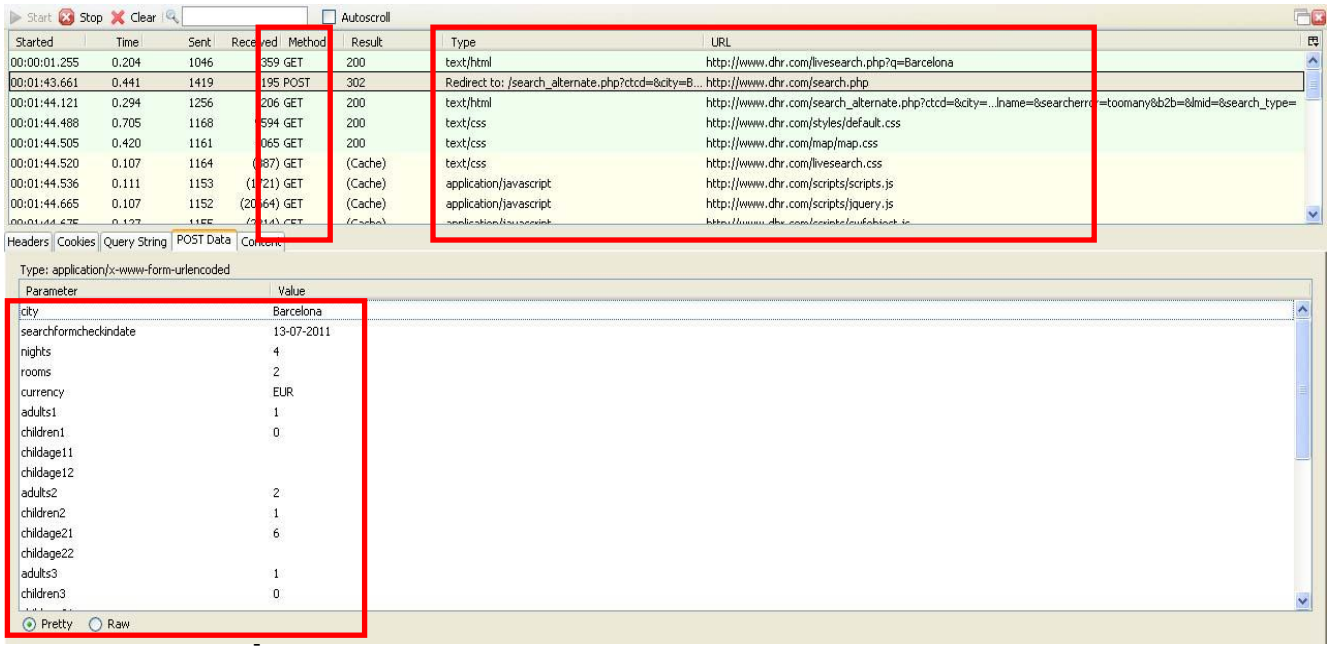


Figura 5. *HttpFox*

L'estudi dels operadors d'hotels té un seguit de particularitats que no es donen amb els operadors de vols. La més important es dóna en l'elecció d'habitacions: mentre que en la reserva d'una estància a un hotel es necessari indicar el nombre d'habitacions i per a cada habitació el nombre d'adults i de nins que aniran, a una reserva d'un vol només es necessari indicar el nombre total de passatgers (adults i nins que viatgen). Això no obstant, no es així amb tots els operadors d'hotels doncs alguns només requereixen el nombre total de persones per cada habitació. I encara es pitjor en altres on s'indica el nombre total d'habitacions, el nombre total d'adults i el nombre total de nins, produint confusió si l'usuari vol indicar qualque distribució específica a una determinada habitació.

S'ha optat pel criteri majoritari a aquests operadors, que també es el mes lògic i senzill: escollir només aquells operadors que permeten indicar el nombre d'habitacions i per a cada habitació el nombre d'adults i el nombre de nins. Per a cada nin també es requerirà indicar la seva edat.

Així doncs els operadors escollits permeten indicar al formulari de cerca els següents camps:

- Data d'inici i data de fi
- Ciutat
- Nombre d'habitacions
- Distribució de persones per habitació:
    - Nombre d'adults
    - Nombre de nins
    - Edat de cada nin de l'habitació

També es molt important dir que:

- **L'operador ha de donar els imports en EUROS** o ha de donar la possibilitat de poder indicar aquesta moneda.
- Si be el camp ciutat pot ser un camp lliure, per a facilitar l' integració de la cerca entre hotels i vols i el mapeig de ciutats (cada ciutat en funció de l'operador tindrà un codi o altre) i donat que els objectius d'aquest treball queden igualment assolits, **la cerca d'hotels es limita a un nombre limitat de ciutats Espanyoles**. El llistat complet apareix a un annex d'aquesta memòria.

Amb totes aquestes consideracions, s'ha cercat per Internet un conjunt d'operadors d'hotels la llista completa dels quals es dóna a la bibliografia del final de la memòria.

A continuació es dóna una fitxa per a cada operador analitzat amb les dades més rellevants de cadascun: els paràmetres del formulari de cerca que serveixen de punt d'entrada a la consulta de disponibilitat que ofereix cada operador i les URLs principals que es segueixen per a consultar la disponibilitat final. Finalment es comenta per a cada operador l' URL que obté els resultats ordenats: en alguns casos ens podem estalviar llegir totes les planes de resultats i fer una posterior comparació de tots els resultats doncs amb la primera plana dels resultats tenim suficient (aquest plantejament ajuda molt a optimitzar l'execució del nostre cercador).

1.2.3.2.1 Operador DHR

http://www.dhr.com/countries/spain.htm

Restriccions	***Màxims***	***mínims***
Habitacions	5	1
Nombre adults per habitació	6	1
Nombre nins per habitació	2	0
Edat nin	17	0
Tipus camp variable	***Nom***	***Exemple***
Desti http://www.dhr.com//livesearch.php?q=Madrid&beach=0	city	Barcelona, Spain
Data inici	searchformcheckindate	13-07-2011

	checkindate	
Data fi	N/A	N/A
Nits	nights	4
Habitacions	rooms	2
Distribució adults	adults1, adults2, adults3, adults4, adults5	1,2,3,4,5,6 Sense: 1
Distribució nins	children1, children2, children3, children4, children5	0,1,2 Sense: 0
Edats nins	childage11, childage12 childage21, childage22 childage31, childage32 childage41, childage42 childage51, childage52	0,1,2,…,17 Sense: blanc
Crida 1: GET (fa un redirect contra altra pàgina)		
http://www.dhr.com/search.php **city=Barcelona, Spain**&search_type=&**searchformcheckindate**=13-07-2011&**nights=4**&**rooms=2**&**adults1=1**&**children1=0**&**adults2=2**&**children2=1**&**adults3=1**&**children3=0**&**adults4=1**&**children4=0**&**adults5=1**&**children5=0**&**childage11**=&**childage12**=&**childage21=6**&**childage22=**&**childage31=**&**childage32=**&**childage41=**&**childage42=**&**childage51=**&**childage52=**&**hotelname=**&**currency=EUR**&radius=0&x=55&y=9&from=/countries/spain.htm&**checkindate=13-07-2011**		
Crida 2 (ORDENACIÓ)		
A partir del resultat retornat per l' URL anterior, es consulta la URL que retorna els mateixos resultats però ordenats per preu. Aquesta URL és la URL de destí del link "**Price**": event que s'agafa mitjançant una expressió regular, simulant el que faria l'usuari si polsés el radio button d'ordenació per preu.		

1.2.3.2.2 Operador Hoteles.com

http://www.hoteles.com/

Restriccions	***Màxims***	***mínims***
Habitacions	8	1
Nombre adults per habitació	8	1
Nombre nins per habitació	3	0
Edat nin	17	0
Tipus camp variable	***Nom***	***Exemple***
Desti http://www.hoteles.com/suggest/atxt_destination/b0/c4/d30/e5/fes_ES/gbarcelona	destination destinationId	Barcelona, España 444495

Data inici	searchParams.arrivalDate	12/05/2011
Data fi	searchParams.departureDate	15/05/2011
Nits	N/A	N/A
Habitacions	rooms	2 Sense: no ve
Distribució adults	searchParams.rooms[0].numberOfAdults searchParams.rooms[1].numberOfAdults	
Distribució nins	children[0] children[1]	0,1,2,3 min: 0 max: 3 Sense: 0
Edats nins	searchParams.rooms[1].childrenAges[0] searchParams.rooms[1].childrenAges[1] searchParams.rooms[2].childrenAges[0]	Sense: no s'envia
Crida 1: GET		
http://www.hoteles.com/search.do? **destination=Barcelona,+España**&**searchParams.arrivalDate=13/07/2011**&**searchParams.departureDate=17/07/2011**&**rooms=2**&**searchParams.rooms[0].numberOfAdults=1**&**children[0]=0**&**searchParams.rooms[1].numberOfAdults=2**&**children[1]=1**&**searchParams.rooms[1].childrenAges[0]=6**&asaReport=HomePage::City&**destinationId=444495**&searchParams.landmark=&hotelId= Hi ha que capturar la url del meta retornat i redirigir després a aquesta url.		
Crides successives per a llegir tots els resultats		
Lamentablement, els resultats no es tornen ordenats. Es fan crides HTTP a totes les planes de resultant, com si l'usuari polsés "Siguiente" fins a no tenir més resultats.		

1.2.3.2.3 Operador skoosh

http://www.skoosh.es/

Restriccions	***Màxims***	***mínims***
Habitacions	5	1
Nombre adults per habitació	6	1
Nombre nins per habitació	2	0
Edat nin	18	0
Tipus camp variable	***Nom***	***Exemple***
Desti http://www.skoosh.es/s/twibo/index.h	city_specified	Barcelona

tml?ac_format=txt&_=1302797083908&limit=10&timestamp=1302797083908&auto_query=barcelona		
Data inici	check_in_date	13/07/11
Data fi	ceck_out_date	17/07/11
Nits	nights	4
Habitacions	rooms	2
Distribució adults	adults_1 adults_2	1,2,…
Distribució nins	child_1 child_2	0,1,2 Sense: 0
Edats nins	child_1_1 child_1_2 child_2_1 child_2_2	1,2,… Sense: blanc

Crida 1: GET

http://www.skoosh.es/s/twibo/check.html?

city_id=&__date_format=euro:check_in_date,check_out_date&filter_an=&attraction_specified=&**city_specified=Barcelona**&**check_in_date=13/07/11**&**check_out_date=17/07/11**&**nights=4**&**rooms=2**&**adults_1=1**&**child_1=0**&currency_name=EUR&**adults_2=2**&**child_2=1**&**adults_3=2**&**child_3=0**&**adults_4=2**&**child_4=0**&**adults_5=2**&**child_5=0**&**child_1_1=none**&**child_1_2=none**&**child_2_1=6**&**child_2_2=none**&**child_3_1=none**&**child_3_2=none**&**child_4_1=none**&**child_4_2=none**&**child_5_1=none**&**child_5_2=none**

Crida 2: GET (ajax d'espera de resultats)

Es fan crides periòdiques fins conèixer que els resultats de preus ja han estat calculats per l'operador.

Es simulen les crides que per AJAX (tècnica de programació que permet de forma asíncrona fer crides contra pàgines de servidor) faria el navegador.

Es finalitza quan s'arriba a un màxim (per evitar bucles infinits) o bé quan la resposta indica que ja s'han trobat els resultats.

http://www.skoosh.es/s/twibo/ajax_action.html?sub_action=results_available&result_id=wlfxHFHV&get_json=1&attempt=28&_=1302797204797

La darrera crida es fa quan ja s'han trobat resultats.

Crida 3: GET (consulta de resultats)

Es retorna html amb el resultat de la cerca, JA ORDENAT PER PREU.

http://www.skoosh.es/s/twibo/results.html?result_id=wlfxHFHV-c30-i110713-n4-r1a1-r2a2-r2c1a6-curEUR

1.2.3.2.4 Operador lastminute.com

http://www.es.lastminute.com/site/viajes/hoteles/

Restriccions	***Màxims***	***mínims***
Habitacions	3	1
Nombre adults per habitació	4	1
Nombre nins per habitació	2	0
Edat nin	12	0
Només es poden fer reserves a un any vista		
Tipus camp variable	***Nom***	***Exemple***
Desti	lmnLocation	barcelona
Data inici	lmnCheckInDay lmnCheckInMonth	14 4
Data fi	lmnCheckOutDisplay lmnCheckOutDay lmnCheckOutMonth	18+Abril 18 4
Nits	lmnLengthOfStay	4
Habitacions	lmnRooms	2
Distribució adults	lmnAdultsRoom1 lmnAdultsRoom2	1,2,…
Distribució nins	lmnChildrenRoom1 lmnChildrenRoom2	1,2,… Sense: 0
Edats nins	lmnRoom1ChildAge1 lmnRoom1ChildAge2 lmnRoom1ChildAge3 lmnRoom1ChildAge4 … lmnRoom2ChildAge1 lmnRoom2ChildAge2 lmnRoom3ChildAge4	1,2,3… Sense: blanc: no s'envia
Crida 1: GET		

http://www.es.lastminute.com/trips/interstitial/searchHotels?

skin=eses.lastminute.com&**configId=S91144128**&filterResultsBy=all&returnURL=http://www.es.lastminute.com/site/travel/hotels/esRebrand_homepage.html&errorURL=http://www.es.lastminute.com/site/viajes/hoteles/error.html&preserveName-skin=eses.lastminute.com&preserveName-CATEGORY=hotels&preserveName-partnerId=0&preserveName-SEARCH=basic&preserveName-requestURL=/site/viajes/hoteles/esRebrand_homepage.html&lmnShowRestaurants=False&searchTypeName=cityName&**lmnLocation=barcelona**&method=hotelsFullSearch&lmnCountry=Any&**lmnCheckInDay=14**&**lmnCheckInMonth=4**&**lmnLengthOfStay=4**&**lmnCheckOutDisplay=18+Abril**&**lmnCheckOutDay=18**&**lmnCheckOutMonth=4**&**lmnRooms=2**&**lmnAdultsRoom1=1**&**lmnChildrenRoom1=0**&**lmnRoom1ChildAge1=**&**lmnRoom1ChildAge2=**&**lmnRoom1ChildAge3=**&**lmnRoom1ChildAge4=**&**lmnAdultsRoom2=2**&**lmnChildrenRoom2=1**&**lmnRoom2ChildAge1=6**&**lmnRoom2ChildAge2=**&**lmnRoom2ChildAge3=**&**lmnRoom2ChildAge4=**&**lmnAdultsRoom3=2**&**lmnChildrenRoom3=0**&**lmnRoom3ChildAge1=**&**lmnRoom3ChildAge2=**&**lmnRoom3ChildAge3=**&**lmnRoom3ChildAge4=**

Internament es fa una redirecció a altra plana fent us de l'etiqueta meta. Aquest valor s'ha de capturar per tal de redirigir al resultat de la cerca.

Crida 1: GET (ordenació)

Per a obtenir els hotels ordenats s'ha de redirigir al link del text "menor precio". Aquest link retorna els hotels ordenats per preu, de menor a major.

1.2.3.2.5 Operador Hotelopia

http://www.hotelopia.es/

Restriccions	***Màxims***	***mínims***
Habitacions	5	1
Nombre adults per habitació	6	1
Nombre nins per habitació	4	0
Edat nin	12	0
Tipus camp variable	***Nom***	***Exemple***
Desti http://www.hotelopia.es/TaskManager/FreeTextHandler.ashx?web=170&nav=170&20110227040610	textDestination IdSearcher	Barcelona 368226
Data inici	date_in DayIn MonthIn YearIn day_in month_year_in	13/07/2011 13 07 2011 13 2011-07
Data fi	date_out DayOut	17/07/2011 17

	YearOut MonthOut day_out month_year_out	2011 07 17 2011-07
Nits	N/A	N/A
Habitacions	RO	2
Distribució adults	HA1,HA2,HA3,HA4,HA5	1,2,3,4,5,6 Sense: 0
Distribució nins	HNA1,HNA2,HNA3,HNA4, HNA5	0,1,2,3,4 Sense: 0
Edats nins	H1AG1,H1AG2,H1AG3,H 1AG4,H2AG1,H2AG2…	0,1,2,… Sense: 0
Crida 1: POST		
http://www.hotelopia.de/TaskManager/resultHandler.ashx? **noCache=1302804779617**&_dest=results.aspx&todayOffset=0&**textDestination=Barcelona**&**IdSearcher=368226**&**date_in=13/07/2011**&**date_out=17/07/2011**&**DayIn=13**&**MonthIn=07**&**YearIn=2011**&**DayOut=17**&**MonthOut=07**&**YearOut=2011**&**RO=2**&advanced=true&free_hotel_name=&RangMin=1&RangMax=30&BookToday=true&quiet=true&**day_in=13**&**month_year_in=2011-07**&**day_out=17**&**month_year_out=2011-07**&HA1=1&HN1=0&H1AG1=0&H1AG2=0&H1AG3=0&H1AG4=0&HA2=2&HN2=1&H2AG1=6&H2AG2=0&H2AG3=0&H2AG4=0&HA3=1&HN3=0&H3AG1=0&H3AG2=0&H3AG3=0&H3AG4=0&HA4=1&HN4=0&H4AG1=0&H4AG2=0&H4AG3=0&H4AG4=0&HA5=1&HN5=0&H5AG1=0&H5AG2=0&H5AG3=0&H5AG4=0&boardtype= Aquesta crida executa el procés de cerca. Després d'esperar a l'execució dels resultats, la url 'http://www.hotelopia.es/results.aspx' retorna els resultats finals.		
Crida 2: ORDENACIÓ		
La URL 'http://www.hotelopia.es/results.aspx?order=price_asc' retorna els resultats ordenats. Es correspon amb l'acció de polsar l'enllaç d'ordenació "Precio".		

1.2.3.2.6 Operador getaroom

http://www.getaroom.com/

Restriccions	***Màxims***	***mínims***
Habitacions	8	1
Nombre adults per habitació	8	1
Nombre nins per habitació	8	0
Edat nin	17	0
Tipus camp variable	***Nom***	***Exemple***
Desti http://www.getaroom.com/locations/list.txt?q=barcelona&limit=50&timestamp=1302640989902	destination	Barcelona, Spain
Data inici	check_in	07/13/2011
Data fi	check_out	07/17/2011
Nits	N/A	N/A
Distribució habitacions	rinfo	[[18],[18,18,6]]
Crida 1: GET		

http://www.getaroom.com/searches/show?

sort_order=price&page=1&amenities[]=&property_name=&lucky=true&destination=**Barcelona,+Spain**&search[destination]=**m19**&check_in=**07/13/2011**&check_out=**07/17/2011**&rinfo=**[[18],[18,18,6]]**&commit=GO

Els resultats ja venen ordenats. Això es fa passant el paràmetre sort_order amb valor price a la consulta anterior.

1.2.3.3 Operadors de vols

Contràriament a lo que inicialment em va parèixer l'estudi dels operadors de vols ha estat més simple i ràpid que l'estudi dels operadors d'hotels. En aquest cas i per a tots els operadors escollits els paràmetres de cerca que es requereixen son els següents:

- Dat d'inici i data de fi
- Ciutat origen i ciutat destí
- Nombre total d'adults
- Nombre total de nins
- Nombre total de bebès (Nins fins a 1 any)

A part de necessitar menys paràmetres que els operadors d'hotels un avantatge molt important es que els resultats ja es donen ordenats per preu. No cal per tant fer cap crida HTTP posterior que retorni els resultats ordenats.

Com es va fer amb els operadors d'hotels, els preus es donen en EUROS.

En alguns operadors com Vueling, s'envia el codi de l'aeroport i no el nom / codi de la ciutat. S'ha tingut que fer un mapeig entre ciutat i codi d'aeroport. El codi d'un aeroport es un codi estandarditzat per l'organització IATA i esta format per tres lletres. Per exemple, el codi de l'aeroport de El Prat, a Barcelona, es "BCN".

A partir d'una ciutat d'origen el nostre cercador utilitza una funció genèrica que retorna el codi de l'aeroport d'aquesta ciutat. Lo mateix fa amb la ciutat destí. Aquesta es altra raó per limitar el nombre de ciutats a un conjunt de ciutats dins l'àmbit estatal (les ciutats configurades al sistema son ciutats amb aeroport). Una millora de l'aplicació consisteix en, donada una ciutat qualsevol mapejar el aeroport més proper i que aquest sigui l'aeroport que li correspon a la ciutat.

El llistat complet de les ciutats per a les quals es poden realitzar cerques es dóna a un annex d'aquest document, on s'indica també l'aeroport que li correspon a cada ciutat.

Com es va fer amb els operadors d'hotels, a continuació es dóna una fitxa per a cada operador de cerca de vols seleccionat, mostrant les dades rellevants i seguint un format molt similar al cas dels operadors d'hotels.

1.2.3.3.1 Operador eDreams

http://www.edreams.es/vuelos/

Restriccions	*Màxims*	*mínims*
Nombre adults (>=12)	9	1
Nombre nins (2-11)	9	0

Nombre bebes (0-1)	9	0
Tipus camp variable	***Nom***	***Exemple***
Origen http://www.edreams.es/engine/AutoComplete/flight?searchKey=Bar	departureLocationGeoNodeId departureLocation	9629 Alicante
Destí	arrivalLocationGeoNodeId arrivalLocation	9646 Barcelona
Data inici	departureDate	13/07/2011
Data fi	returnDate	17/07/2011
Nits	nights	4
Nombre adults	numAdults	2
Nombre nins	numChilds	1
Nombre bebes	numInfants	1
Crida 1: GET		
http://www.edreams.es/engine/ItinerarySearch/search? buyPath=2&auxOrBt=0&searchMainProductTypeName=FLIGHT&hasLiligoPopUnder=true&hasLiligoPopUnder=true&liligoRatio=50&websiteCode=ES&dynArrivalCity=&dynDepartureCity=&dynDepartureDate=&dynReturnDate=&dynDepartureTime=&dynReturnTime=&dynNumOfChildren=&dynOnlyDirectFlight=&dynNumOfAdults=&dynNumOfRooms=&country=ES&language=es&tripTypeName=ROUND_TRIP&**departureLocationGeoNodeId=9629**&**departureLocation=Alicante**&**arrivalLocationGeoNodeId=9646**&**arrivalLocation=Barcelona**&**departureDate=13/07/2011**&departureTime=0000&**returnDate=17/07/2011**&returnTime=0000&**numAdults=3**&**numChilds=1**&**numInfants=0**&filteringCarrier=&fake_filteringCarrier=Todas+las+compañías&cabinClassName=TOURIST&image=		

1.2.3.3.2 Operador lastminute.com

http://www.es.lastminute.com/site/viajes/vuelos/

Restriccions	***Màxims***	***mínims***
Nombre adults (>=12)	9	1
Nombre nins (2-11)	9	0
Nombre bebes (0-1)	9	0
Tipus camp variable	***Nom***	***Exemple***
Origen **http://www.es.lastminute.com/site/viajes/vuelos/predictive-text-backend.html?skin=eses.lastminute.com&LOCALE=es_ES&TYPE=A%2CC&LOCATION=ALI**	DPDAP	Alicante+(ALC),+ES
Destí	DPAAP	Barcelona+(BCN),+ES

Data inici	date_out_day date_out_month	13 7
Data fi	date_in_day date_out_month	17 7
Nits	N/A	N/A
Nombre adults	DPNOA	3
Nombre nins	DPNOC	1
Nombre bebes	DPNOI	0
Crida 1: GET		
http://www.es.lastminute.com/site/viajes/vuelos/interstitial_page.html? skin=eses.lastminute.com&DPCAT=flights&TRANS=ViewTransportationSearchBox,doSearchForTransportation&DPCLS=X&DPSAC=LA2&DPDCC=EUR&DPCID=0&DPFIT=flights&DPSRT=PA&DPHSI=0&DPHEI=9&DPFSI=0&DPFEI=10&DPSFT=&DPLOC=es_ES&DPPID=0&DPTRT=flights&DPTTT=R&DPDES=&DPXIN=T&DPXCH=T&DPXTR=F&DPXCP=F&DPXAL=F&DPXMG=F&mondial_auto=false&failureURL=http://www.es.lastminute.com/site/travel/flights/flights-search-error.html&successURL=/travel/flights/transport-results.html&DPAIR=NONE&isEurostarSearch=False&DP1WF=0&**DPDAP=Alicante+(ALC),+ES**&**DPAAP=Barcelona+(BCN),+ES**&**date_out_day=13**&**date_out_month=7**&DPODT=0001&**date_in_day=17**&**date_in_month=7**&DPIDT=0001&**DPNOA=3**&DPNOS=0&**DPNOC=1**&**DPNOI=0**		

1.2.3.3.3 Operador Rumbo

http://www.rumbo.es/

Restriccions	***Màxims***	***mínims***
Nombre adults (>=12)	9	1
Nombre nins (2-11)	9	0
Nombre bebes (0-1)	9	0
Tipus camp variable	***Nom***	***Exemple***
Origen http://www.rumbo.es/js/JSON/locations/es/B.json	depIata depCity	ALC Alicante+(ALC)
Destí	arrIata arrCity	BCN Barcelona,+El+Prat++(BCN)
Data inici	depDate	13/07/2011
Data fi	retDate	17/07/2011
Nits	N/A	N/A
Nombre adults	paxAdt	3

Nombre nins	paxChd	1
Nombre bebes	paxInf	0
Crida 1: GET		
http://www.rumbo.es/viajes/vuelos/spider.do buscadorWT=Buscador+Home+Vuelos&**arrIata=BCN**&**depIata=ALC**&resultType=P&from=&onlyRenfe=&promCode=&flowType=&it_mtp=A&queryType=R&**depCity=Alicante+(ALC)**&**arrCity=Barcelona,+El+Prat++(BCN)**&**depDate=13/07/2011**&**retDate=17/07/2011**&**paxAdt=3**&**paxChd=1**&paxInf=0&cabin=&train=false&lowCost=true&directOnly=false&largeFamilyDiscountCode=&residentDiscountCode=		

1.2.3.3.4 Operador Spanair

http://www.spanair.com/web/es-es/

Restriccions	***Màxims***	***mínims***
Nombre adults (>=12)	9	1
Nombre nins (2-11)	9	0
Nombre bebes (0-1)	9	0
Tipus camp variable	***Nom***	***Exemple***
Origen *Limitats als valors d'un select d'opcions.*	selOrigenes	ALC
Destí	selDestinos	BCN
Data inici	ctl00$PHBody$MultiBuscador1$BuscadorVuelos1$fecha_ida$bdpDate$textBox	13-07-2011
Data fi	ctl00$PHBody$MultiBuscador1$BuscadorVuelos1$fecha_regreso$bdpDate$textBox	17-07-2011
Nits	N/A	N/A
Nombre adults	dropAdultos	3
Nombre nins	dropNinos	1
Nombre bebes	dropBebes	0
Crida 1: GET		
http://www.spanair.com/web/templates/Amadeus/_FramesetBuscador.aspx? __EVENTTARGET=&__EVENTARGUMENT=&__VIEWSTATE=/wEPDwUJLTI&IdZanox=&IdTradeDoubler=&IdNonstop=&URLReferer=/web/es-es/&PageNameOmniture=Inicio&BotonPulsado=V&language=es-es&configFile=http://www.spanair.com/xml/buscador/buscador-principal.xml&NumBonosGP100=0&tipoTrayecto=RT&**selOrigenes=ALC**&**selDestinos=BCN**&**ctl00$PHBody$MultiBuscador1$BuscadorVuelos1$fecha_ida$bdpDate$textBox=13-07-**		

2011&**ctl00$PHBody$MultiBuscador1$BuscadorVuelos1$fecha_regreso$bdpDate$textBox=17-07-2011**&**dropAdultos=3**&**dropNinos=1**&**dropBebes=0**&ctl00$PHBody$MultiBuscador1$LoginExcesoEquipajes$tbApellido=&ctl00$PHBody$MultiBuscador1$LoginExcesoEquipajes$tbLocalizador=&ORIGEN=&DESTINO=&version=&tipoViaje=&escalas=&maximo=

1.2.3.3.5 Operador Vueling

http://www.vueling.com/?language=ES

Restriccions	***Màxims***	***mínims***
Nombre adults (>=12)	25	1
Nombre nins (2-11)	12	0
Nombre bebes (0-1)	9	0
Tipus camp variable	***Nom***	***Exemple***
Origen	from1 to2	ALC ALC
Destí	to1 from2	BCN BCN
Data inici	departDay1 departMonth1 displayDate1 departDate1	13 201107 Miércoles 13 Julio, 2011 20110713
Data fi	departDay2 departMonth2 displayDate2 departDate2	17 201107 Domingo 17 Julio, 2011 20110717
Nits	N/A	N/A
Nombre adults	ADULT	3
Nombre nins	CHILD	1
Nombre bebes	INFANT	0
Crida 1: GET		

http://www.vueling.com/booking/booking/selecciona-tu-vuelo?

event=search&module=SB&page=SEARCH&language=ES&mode=&sid=&ref=&travel=2&**from1=ALC**&**to1=BCN**&**from2=BCN**&**to2=ALC**&**departDay1=13**&**departMonth1=201107**&**displayDate1=Miércoles+13+Julio,+2011**&depart1FlexBy=0000&**departDay2=17**&**departMonth2=201107**&**displayDate2=Domingo+17+Julio,+2011**&depart2FlexBy=0000&fechas=0000&**ADULT=3**&defaultADULT=-1&**CHILD=1**&defaultCHILD=-1&**INFANT=0**&defaultINFANT=-1&toCity1=Destino&toCity2=??&**departDate1=20110713**&**departDate2=20110717**&numberMarkets=2&cualquier=&nom_cualquier=&m1_cualquier=&m2_cualquier=&frdisc=&mode_orig=&zanpid=&utm_campaign=&tduid=&mode_TESTAB=FxYHEDAeU1Q5Uz5zAAFSNlAyQg==&mode_TESTABClassB=NT8+KC0oU1Q5MSIBYHslNFIqIw==

1.2.4 Extracció de dades

Analitzats els operadors i coneixent per a cadascun les seves particularitats, s'han de traduir les connexions HTTP analitzades amb l' HttpFox de Firefox a crides a un objecte HttpClient. La resposta relativa a la consulta de disponibilitat a un determinat operador s'emmagatzema a una variable tipus String.

El procés general es comenta a continuació:

L'accés i posterior tractament de l' informació proporcionada per cada operador es tracta a la classe *pfc.wrapper.Wrapper* i a les seves classes "filla", que hereten d'aquesta classe. Tenim doncs un wrapper per cada operador: entenem un wrapper com una classe Java que serveix d'interfície entre el cercador i un operador determinat. Es a dir, un wrapper abstreu a la lògica de negoci del model de dades, que en aquest cas no està representat per una base de dades sinó que es troba repartit a diferents servidors externs. L'accés a aquesta informació es fa, com ja s'ha dit, amb crides per HTTP.

Cada classe wrapper estén de la classe Thread del package *java.lang* de tal forma que **s'executen en paral·lel totes les cerques** als diferents operadors, amb l'objectiu de guanyar eficiència doncs no té molt sentit esperar sense fer res la resposta d'un servidor quan queden dades d'altres servidors que encara no coneixem (es com si un usuari executés vàries consultes a la vegada des de diferents navegadors al mateix temps amb el seu PC).

El mètode *run()* implementat a la classe Wrapper s'encarrega inicialment de fer totes les connexions HTTP de l'operador associat (recordem que a cada operador li correspon una instància de la classe pfc.wrapper.Wrapper) i una vegada té l' informació consultada realitza el tractament necessari: filtra l' informació útil, realitza l'ordenació, etc. Aquest procés es veurà a un apartat posterior. En relació a les connexions HTTP, el codi general del mètode run() aplicable a tots els operadors, és el següent:

```
public abstract class Wrapper extends Thread {
      public void run(){
            try {
                  //Comprobació dels paràmetres enviats
                  assertPeticioWrapper();

                  //Lectura de l'informació dels operadors externs
                  this.cookieStore = new BasicCookieStore();
      °           this.httpContext.setAttribute
                  (ClientContext.COOKIE_STORE, cookieStore);
                  getScrapedData();
                  //…
            }
            catch (PeticioWrapperException pwe) {}
            catch (java.lang.IllegalStateException ise) {}
            catch (IOException ioe) {}
            catch (Exception e) {}
      }
}
```

Com es veu, la classe Wrapper es una classe abstracta doncs son les classes filla (les classes dels paquets *pfc.wrapper.hotel.hotels* i *pfc.wrapper.flight.flights* les que particularitzen el comportament

de cada operador. Però es aquesta classe abstracta la que defineix un algorisme comú de treball per a cada operador, el qual personalitza segons les seves característiques pròpies.

Del mètode anterior es comenten les principals accions:

```
assertPeticioWrapper();
```

Verifica que les dades de la petició de cerca (nombre d'habitacions, nombre d'adults a cada habitació, ciutats, etc) son correctes per al wrapper. En cas contrari es llança una excepció controlada. Això es fa per a evitar consultes a wrappers que no suporten els paràmetres de cerca: per exemple, si coneixem que un operador d'hotels no permet la consulta per a més de 3 habitacions, directament es llança una excepció (instància de la classe pfc.exception.PeticioWrapperException) i ens estalviam posteriors consultes. Aquest mètode ho implementa cada wrapper segons les seves necessitats *-alguns son més flexibles que altres-*.

```
this.cookieStore = new BasicCookieStore();
this.httpContext.setAttribute(
      ClientContext.COOKIE_STORE,
      cookieStore);
```

Inicialitza el magatzem de cookies del wrapper en qüestió. Aquesta es una variable que crea i inicialitza cada wrapper. Després de la creació del magatzem es diu a l'objecte que representa el context d'execució quina instància serveix de magatzem de cookies. Donat que el context es únic per a cada wrapper, no existeix cap gestió comuna de cookies per al conjunt de wrappers, lo qual es correcte: El wrapper de l'operador Dhr no té perquè enviar cookies de connexions establertes amb l'operador LastMinute.com, per exemple. Resumint: cada Thread (fil d'execució) relacionat amb un wrapper té un context propi d'execució el qual defineix paràmetres propis que tenen preferència per sobre de paràmetres globals aplicables a totes les connexions que es fan al nivell superior, de la connexió HttpClient.

```
getScrapedData();
```

Realitza la consulta d'informació. Aquest mètode queda implementat per cada wrapper hotel/vol, que executa un seguit de mètodes HTTP. El resultat final s'emmagatzema a la variable privada de tipus String i de nom `resultatScraping`.

Definida l'estructura general del codi, el treball més important consisteix a implementar el mètode getScrapedData a cada wrapper hotel/vol. El conjunt de peticions HTTP es donen al codi font del cercador: per a cada wrapper s'executen de forma seqüencial un o més mètodes HTTP (GET i POST) a fi d'obtenir el resultat de cerca i emmagatzemar-lo a una variable String.

Vistos els wrappers de vols i hotels, el cercador una vegada inicia els fils d'execució, queda en espera a que aquests finalitzen. Executant el mètode join de cada wrapper (mètode heretat de la classe Thread), es provoca una espera del procés que ha llançat els threads (el cercador) fins que aquestos finalitzen. Com es pot veure al fragment de codi següent, l'execució dels wrappers es fa progressivament amb un marge d'espera entre threads a fi de no saturar la màquina virtual de Java i facilitar la visualització de les traces d'execució. A un entorn real s'hauria d'estudiar si aquest marge desapareix, es redueix o s'augmenta lleugerament:

```
//S'inicien els threads que cerquen estancies a hotels
Vector<Wrapper> vWrappersHotelsPeticio = new Vector<Wrapper>();
for (int i=0; i<vWrappersHotels.size(); i++) {
      Wrapper wrapper =
            (Wrapper)Class.forName(vWrappersHotels.get(i)).newInstance();
      wrapper.setBeanPeticioConsulta(beanPeticioConsulta);
      wrapper.setHttpClient(cachingHttpClient);
      vWrappersHotelsPeticio.add(wrapper);
      wrapper.start();
      Thread.sleep(pfc.conf.Constants.TIME_BETWEEN_STARTS);
}

//S'inicien els threads que cerquen vols
Vector<Wrapper> vWrappersVolsPeticio = new Vector<Wrapper>();
for (int i=0; i<vWrappersVols.size(); i++) {
      Wrapper wrapper =
            Wrapper)Class.forName(vWrappersVols.get(i)).newInstance();
      wrapper.setBeanPeticioConsulta(beanPeticioConsulta);
      wrapper.setHttpClient(cachingHttpClient);
      vWrappersVolsPeticio.add(wrapper);
      wrapper.start();
      Thread.sleep(pfc.conf.Constants.TIME_BETWEEN_STARTS);
}

for (Wrapper wrapper: vWrappersHotelsPeticio) wrapper.join();
for (Wrapper wrapper: vWrappersVolsPeticio) wrapper.join();
```

Tots els wrappers reben la mateixa petició de consulta i els mateixos objectes `cachingHttpClient, httpclient`:

`httpclient`

Única instància de la classe DefaultHttpClient. Es l'objecte contra el qual tots els wrappers executen els seus mètodes HTTP (com es veu al codi anterior, tots els wrappers reben el mateix objecte). Això facilita la definició de comportaments comuns a totes les connexions i alhora permet particularitzacions a cada wrapper concret en funció del seu context d'execució.

`cachingHttpClient`

Utilitzant el patró *decorator*, "decora" l'objecte HttpClient de tal forma que amb pocs cavis al codi es pot prescindir del seu us substituint-lo per la instància HttpClient directament. S'utilitza un objecte de la classe CachingHttpClient aprofitant una característica d' HttpClient: la cache de peticions. Es a dir, no es faran peticions noves per la xarxa si aquestes peticions són a cache i la resposta es pot enviar de cache. La inicialització (constructor) de l'objecte rep un objecte tipus *org.apache.http.impl.client.cache.CacheConfig* que conté la configuració bàsica d'aquest comportament: el nombre màxim d'entrades que es permeten emmagatzemar i el tamany màxim de cada entrada. Això té més aplicació a un context real on el cercador queda desplegat a un servidor d'aplicacions i l'objecte HttpClient rep contínuament peticions de cerca, donat que s'optimitza el funcionament si hi ha peticions que es poden retornar de cache.

Comentat el que seria el "cor" de tota aquesta part encarregada d'obtenir l' informació dels operadors externs, es pot pensar que un error de codi (potser de programació però fundamentalment degut a canvis als servidors dels operadors, a canvis en el codi HTML, etc) pugui aturar o fer que el

cercador no retorni cap resultat. Això no es així si l'errada es específica només a un operador. Per exemple, si un operador no respon o si degut a un canvi en el codi HTML es llança una excepció doncs es cerca el valor d'un camp que s'ha de transmetre a la següent petició HTTP i que ja no existeix el error queda controlat i simplement no retorna cap resultat el wrapper afectat. L'execució global continua si be, com es obvi, el conjunt de combinacions possibles de preus queda reduït.

En aquest sentit, un dels aspectes més importants a la construcció de qualsevol aplicació es l' informació de debug, es a dir el **log de l'aplicació**. En aquest cas s'utilitza el conegut framework d' Apache log4j. A un fitxer de propietats (fitxer de text amb extensió .properties que en aquest cas s'anomena log4j.properties) s'indica el comportament general del log: es a dir, cada logger on imprimeix les seves traces i amb quin nivell es parteix: DEBUG,INFO,ERROR, etc. Per exemple es pot parametritzar un logger a nivell d'ERROR de tal forma que no imprimeixi traces d'un nivell inferior, com poden ser les traces de DEBUG. Un exemple de configuració d'un logger a aquest fitxer es la configuració del logger HTTP, que imprimeix les traces a un fitxer anomenat "trace.log" i utilitzant un patró d'escriptura determinat:

```
log4j.logger.http=DEBUG,FILE_HTTP
log4j.appender.FILE_HTTP=org.apache.log4j.FileAppender
log4j.appender.FILE_HTTP.File=${JBOSS_HOME}\\server\\default\\log\\trace.log
log4j.appender.FILE_HTTP.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE_HTTP.layout.ConversionPattern=[%d] [%F] [%t] %-5p %c %x - %m%n
```

Com es veu a la configuració anterior, es llegeix una propietat de sistema (la propietat JBOSS_HOME) que s'utilitza com a part del directori on es generen els arxius de log.

Dins el codi de l'aplicació, el funcionament es simple: Es crea una instància d'un logger i posteriorment es crida el mètode "log" indicant el missatge que es vol imprimir com a traça i el nivell de la traça (com es deia abans: DEBUG,INFO,ERROR,…):

```
private final static Logger loger = Logger.getLogger("http");
loger.log(Level.DEBUG, "Inici de l'aplicació");
```

L' utilitat del log ve sobre tot a l'hora de conèixer les peticions HTTP que realitza el cercador i sobre tot les errades d'execució, facilitant la lectura dels passos que segueix el cercador durant tot el flux d'execució.

1.3 Capítol 3. Captura de dades.

1.3.1 Introducció

Una vegada tenim tota l' informació obtinguda dels servidors web per connexions HTTP necessitem filtrar l' informació útil: totes aquelles dades que poden ser d'interès a l'usuari que visualitza els resultats als seu navegador.

Per a aquest propòsit, a aquest projecte s'ha optat per fer servir expressions regulars. Existeixen però altres solucions, amb els seus avantatges i inconvenients. Per exemple, es pot transformar el codi HTML a codi XHTML ben format, es a dir, a codi XML i extreure l' informació amb Xpath, un llenguatge que de forma similar a lo que farien unes expressions regulars a un text pla extreu l' informació útil de l' XML basant-se en les característiques d'aquests tipus de documents. Per exemple, no es complicat expressar lo següent amb sintaxi pròpia de Xpath: "*Llegeix la segona taula que penja del tag body amb un atribut id determinat i d'aquesta extreu el contingut de la tercera columna a la primera fila*".

Com dèiem, per a capturar les dades s'ha decidit utilitzar expressions regulars. Les expressions regulars es poden considerar com a un llenguatge amb les seves pròpies regles i que té la finalitat de cercar patrons de caràcters a cadenes de text. Aquests patrons segueixen una sintaxi determinada que, fent us de caràcters i metacaràcters, especifica l'estructura del text que estem cercant. Per exemple, el preu d'una estància a un hotel es podria considerar com a un o més dígits numèrics seguit d'un separador de decimals i altres dígits numèrics. Es clar doncs, que les expressions regulars seran una clau important a aquest projecte doncs una expressió mal descrita pot extreure informació incorrecta o simplement no extreure res i provocar una errada d'execució.

Formalment, i relacionant el concepte d'expressió regular amb la teoria de compiladors estudiada al pla d'estudis d'Enginyeria Informàtica, les expressions regulars defineixen un tipus de llenguatge que pot ser reconegut i analitzat per un autòmat i son una eina fonamental a l'hora de definir els testimonis d'un determinat llenguatge de programació.
Finalment, i com a curiositat, dir que aquest llenguatge té correspondència amb el patró de disseny "Interpreter".

1.3.2 Expressions regulars

Qualsevol sistema compatible amb expressions regulars permet la cerca d'un patró de diferents formes. Per exemple, si volem cercar determinats noms de colors a un arxiu podem utilitzar l'expressió "blanc|negre|blau".

La potència d'una expressió regular radica en el fet de que amb caràcters especials (metacaràcters) es poden representar llargues cadenes de text amb expressions no massa llargues i relativament senzilles d'entendre. Per exemple, es pot realitzar la cerca de les cadenes constants de text a una classe Java utilitzant l'expressió: "[^"]+", que vol dir "*qualsevol nombre de caràcters exceptuant la doble cometa i entre dobles cometes*".

Però hom pot pensar que allò que podem analitzar amb expressions regulars igualment es pot analitzar amb altres funcions com per exemple les funcions de la classe String: `startsWith` i `charAt`. En cert mode es així, però lo cert es que amb aquestes funcions el codi que s'aconsegueix

es un codi molt més complex i més especialitzat doncs exigeix un tractament manual caràcter a caràcter. D'altra banda, mentre que amb un canvi a la cadena de text només tindrem que modificar una expressió regular i amb pocs minuts estarà llest, no es el cas d'altres solucions que no utilitzin expressions regulars.

Amb Java, es poden utilitzar expressions regulars des de la versió 1.4. Amb anterioritat a aquesta versió, Java no incloïa cap funció per a descriure expressions regulars, fet llamatiu donada la potència de les expressions regulars i el seu grau de presència a moltes de les comandes de Unix, el sistema operatiu al qual es va utilitzar Java per primera vegada. El package ***java.util.regex*** conté les principals classes que s'utilitzaran a l'hora de definir tot el tractament que utilitza expressions regulars. A banda d'aquest paquet, que no necessita de cap configuració ni instal·lació especial per a poder-lo utilitzar, existeixen altres llibreries d'expressions regulars que es poden utilitzar amb el llenguatge Java. Es el cas de per exemple, la llibreria *regexp* del projecte Apache Jakarta o del paquet ORO, també del projecte Apache Jakarta. L'avantatge del paquet java.util.regex per sobre d'altres com els citats es clar: no només es el natiu del l'API de Java i ve integrat amb el temps d'execució, també es una de les opcions més òptimes en quant a temps d'execució i l'opció més coneguda i utilitzada, lo que facilita la cerca de documentació i també la millora de la llibreria amb les noves versions de Java.

La sintaxi, les regles de construcció i el funcionament de les expressions regulars difereixen en funció del sistema. Per exemple, la notació que utilitza el servidor Web Apache no es la mateixa que la que utilitza el Sistema Operatiu Unix a la comanda *grep* ni la mateixa que utilitza el llenguatge Javascript. Existeixen sintaxis més potents que altres i sistemes més optimitzats que d'altres de tal forma que un determinat patró de cerca pot necessitar una expressió més complexa a un sistema que a altre. No obstant, els fonaments i els conceptes més generals i operadors més freqüents i senzills com l'operador "*" son comuns a bona part dels sistemes que admeten expressions regulars.

Abans d'entrar en detall en la sintaxi de les expressions regulars a Java, convé tenir clar un par de detalls:

- La majoria dels metacaràcters de les expressions regulars (caràcters especials que tenen un significat especial) es poden desactivar anteposant-los el caràcter d'escapament,que en les expressions regulars es la barra invertida ("\"). Per exemple, anteposar un caràcter de escapament a un metacaràcter com "." anul·la el seu significat especial i només es correspon al caràcter punt. D'altra banda, i al contrari, anteposar un caràcter d'escapament a uns pocs caràcters alfabètics com n,r,t,s i w els converteix a metacaràcters i prenen un significat diferent. Entrant en el detall del codi del projecte, es important dir que el caràcter d'escapament s'ha d'escapar dues vegades per tal que el compilador de Java ho interpreti correctament. Així, enlloc de posar "\d" , s'haurà de posar "\\d". Aquesta notació però només s'utilitza al codi font i no té cap efecte a els explicacions posteriors.

- El caràcter "separador de línia" és un caràcter el format del qual depèn del sistema operatiu. Per exemple, a sistemes Unix coincideix amb l'expressió '\n' i a sistemes Windows amb '\r\n'. Es important tenir-ho en compte doncs per exemple, si es carrega un fitxer a una variable de l'aplicació i aquesta variable es valida després segon una expressió regular aquesta expressió regular pot ser diferent en un sistema o altre. A una plataforma Windows la següent expressió s'avalua com a certa: System.getProperty("line.separator").equals("\r\n").

La sintaxi que s'explicarà a aquest apartat es òbviament la sintaxi de Java i es la que es dóna a la taula següent (per claredat, es donen les expressions més comunes incloent totes les que s'utilitzen a aquest projecte).

GENERALS

Expressió	*Significat*	*Exemples*
^	Inici de línia/cadena Si NO està activat el mode multilínia, només coincideix amb l' inici de cadena (el mode multilínia es tracta a l'apartat de flags).	***^Hola***
$	Fi de línia/cadena Al igual que el símbol anterior, no representa cap caràcter especial, només una posició. Té el significat oposat: fi de línia/cadena. Si NO està activat el mode multilínia només coincideix amb el fi de cadena, sense tenir en compte els salts de línia.	***Hola***
\b	Límit de paraula Representa una posició: el límit de paraula. A l'exemple, es cerquen mots "Joan" si van precedits per un separador de paraula.	***\bJoan***
\B	No es límit de paraula Cerca la cadena on no hi han límits de paraula. A l'exemple, la cadena Joan no pot anar precedida per un límit de paraula. Per exemple, 'EJoan' coincideix.	***\BJoan***
\A	Inici de cadena Similar a ^ però mai té en compte els salts de línia.	***\Ahola***
\z	Fi de cadena Posició de fi de cadena. Similar a $ però mai té en compte els salts de línia.	***Adeu\z***
\Z	Fi de cadena Posició de fi de cadena. A diferència de l'anterior, encaixa amb el fi de cadena i amb el salt de línia sempre i quan aquest salt de línia sigui el darrer caràcter de la cadena.	***Adeu\Z***
.	Qualsevol caràcter *Important: EXCEPTUANT EL SALT DE LÍNIA (\r, \n)*	***Hola J.an***
[...]	Classe de caràcters Valida qualsevol dels caràcters enumerats dins els claudàtors. Dins una classe de caràcters poden definir-se rangs. Per exemple, la classe [a-z] representa tots els caràcters entre la 'a' i la 'z'.	***Diptong [aeiou][iu]***
[^...]	Negació d'una classe de caràcters Qualsevol caràcter que no sigui a la classe (dins els claudàtors). S'utilitza quan es vol excloure uns determinats caràcters dins una cerca determinada.	***[^aeiou]\z***
|	Alternació entre diferents ítems La correspondència del text és amb un dels operands afectats pel símbol d'alternança.	***true|false***
(...)	Agrupació de subexpressions Agrupa un conjunt de subexpressions a una única expressió. A l'exemple serveix per a agrupar dos punts cardinals: "Al nord de Barcelona" i "Al sud de Barcelona". Sense el parèntesi les expressions que es validarien serien "Al nord" i "sud de Barcelona" respectivament. Com es pot observar, a l'avaluació de l'expressió té major preferència aquest operador que l'anterior operador d'alternança.	***Al (nord|sud) de Barcelona***

ABREVIATURES

Expressió	*Significat*	*Exemples*
\Q	Inici de marcat. Els caràcters situats entre aquesta subexpressió i la del final de marcat son considerats com a caràcters alfabètics.	***\Q(1+3)=4?\E***
\E	Fi de marcat. Finalitza el marcat iniciat per la subexpressió anterior.	***\Q(1+3)=4?\E***
\t	Caràcter de tabulació. Es correspon amb el caràcter no visible de tabulació.	***Temps = \tnublat***
\n	Caràcter de salt de línia Caràcter invisible "Line Feed"	***companyia=\nRumbo***
\r	Caràcter de retorn de carro Caràcter invisible "Carrier Return". A plataformes Windows s'utilitza juntament amb l'anterior per a formar un salt de línia: \r\n	***companyia=\r\nRumbo***
\d	Dígit numèric Es equivalent a la subexpressió [0-9]. Un operador molt utilitzat, fonamentalment per a capturar informació de preus.	***Preu=\d+***
\s	**Espai en blanc** **Encaixa amb els caràcters espaiadors: blanc,separador de línia,retorn de carro i tabulador principalment.**	***Data inici\s+2011***

MULTIPLICADORS

Un multiplicador s'aplica a una subexpressió per a indicar el nombre total d'ocurrències. Podem utilitzar una expressió per a cercar per exemple "mots amb 6 caràcters". Son caràcters que proporcionen la possibilitat que una expressió regular relativament curta validi un conjunt llarg de caràcters.

Expressió	*Significat*	*Exemples*
{n,m}	Entre n i m repeticions	***(Dia=\d){1,30}***
{m,}	m o més repeticions	***\d(,\d){1,9}***
{m}	m repeticions	***#\d{6}***
{,m}	Entre 0 i m repeticions	***0,\d{,2}***
*****	0 o més repeticions -Equivalent a {0,} - Un operador bastant útil i utilitzat a aquest projecte. Valida elements opcionals que es poden repetir un nombre indeterminat de vegades.	***\d****
+	1 o més repeticions -Equivalent a {1,} – Un operador bastant útil i utilitzat a aquest projecte. Valida elements obligatoris que poden aparèixer un nombre indeterminat de vegades. A l'exemple, "\d+" és equivalent a "\d\d*".	***\d+***
?	**0 o 1 repeticions - Equivalent a {0,1} -**	***Manresa(,Barcelona)?***

TIPUS DE MULTIPLICADORS

A aquest projecte farem servir els dos tipus principals de multiplicadors: els egoistes (***greedy multipliers***) i els mandrosos (***non-greedy/lazy/reluctant multipliers***). Els multiplicadors "normals", aquells que s'han explicat a la darrera taula, son els multiplicadors egoistes. Les diferències no son molt grans i de fet tot dos tipus de multiplicadors realitzen la mateixa validació davant moltes cadenes diferents. No obstant existeixen diferències tant en la seva execució com en la seva validació a segons quines cadenes.

Els multiplicadors egoistes consumeixen tant com sigui possible sense comprometre cap de les subexpressions que venen després. Per contra, els multiplicadors mandrosos consumeixen el mínim nombre de caràcters possible. Vegem ara, amb un parell d'exemples la diferència entre aquests dos tipus d'operadors.

Tenim per entrada la següent cadena: *"<td>Barcelona</td>100€</td>"*

L'expressió regular *<td>.*</td>*, que utilitza un multiplicador egoista, encaixa amb la següent text: *<td>Barcelona</td>100€</td>*. Es a dir, llegeix tota la cadena i com aquesta ja coincideix (comença per <td>, conté un nombre indeterminat de caràcters i finalitza amb </td>) retorna la cadena entera com a coincident.

L'expressió regular *<td>.*?</td>*, que utilitza un multiplicador mandrós, encaixa amb el següent text: *<td>Barcelona</td>*. Es a dir, llegeix la cadena caràcter a caràcter: primer troba <td>, després un nombre indeterminat de caràcters i després troba la cadena </td> que coincideix amb la subexpressió regular que segueix. Aquest tipus d'operador s'utilitza amb freqüència a la solució donada, donat que no son poques les vegades en les quals cal un comportament similar a l' explicat: "llegeix un conjunt de caràcters (bàsicament els multiplicadors .*?, .+?) fins a trobar el primer caràcter que coincideix amb el següent caràcter definit a l'expressió regular.

A la sintaxi, els operadors mandrosos es diferencien dels egoistes en que venen precedits pel símbol d'interrogació '?'. Tenim doncs, els següents operadors mandrosos:

{n,m}?
{m,}?
{m}?
{,m}?
*?
+?
??

CLASSES

Com s'ha dit, el package d'expressions regulars de Java es el *java.util.regex*. Aquest package consta de dues classes: ***Pattern*** i ***Matcher***, que seran les que s'utilitzaran per a cercar i validar patrons a les cadenes de text. Per a cada text i expressió regular que vulguem tractar, el mètode general serà el següent:

- Definir un objecte de la classe Pattern (aquest objecte representa l'expressió regular) cridant el mètode estàtic: *Pattern.compile()* i indicant l'expressió regular
- Crear un objecte Matcher partint del patró creat indicant la cadena de text
- Finalment cridar als mètodes de la classe Matcher per a conèixer el nombre de coincidències, els grups que coincideixen, etc.

La creació d'un objecte patró rep com a paràmetres l'expressió regular que es vol validar i de manera opcional un conjunt de flags que especifiquen part del comportament del patró (Això s'explica al següent apartat).

A partir d'un patró s'obté una instància de la classe Matcher especificant la cadena de text. No tots els mètodes d'aquesta classe s'utilitzen a aquesta solució, però es comenten els més importants a excepció d'aquells que per la seva importància dins la solució finalment implementada es desenvolupen a l'apartat "Agrupació". Son els següents:

- `matches`
  Valida tota la seqüència contra el patró. Retorna true si hi ha coincidència.
- `lookingAt`
  Valida la seqüència contra el patró però aquest es verifica contra l' inici de la cadena.
- `find`
  El mètode més important i utilitzat a aquest projecte. Cerca el patró a la cadena, sense tenir en compte la posició dins la cadena. Comença a l' inici de la cadena o, si s'ha cridat anteriorment, al caràcter següent a l'últim de l'anterior coincidència. Com els dos anteriors, retorna true si s'ha trobat coincidència.
- `replaceAll`
  No cerca coincidències com els anteriors però es també molt útil. Reemplaça totes les coincidències del patró per una cadena que rep com a paràmetre. El funcionament es idèntic al de la funció replaceAll de la classe String pero no rep l'expressió regular com a paràmetre doncs esta ja ha estat indicada a la creació del patró.

Es dóna un exemple de codi que cerca el patró "\d+" a una cadena ja inicialitzada i el substitueix pel nombre 0:

```
Pattern patro = Pattern.compile("\\d+");
Matcher matcher = patro.matcher(txt);
int coincidencies = 0;
while (matcher.find()){
      coincidencies = coincidencies + 1;
}

System.out.println("S'han trobat " + coincidencies + " coincidencies");
System.out.println("Nova cadena: " + matcher.replaceAll("0"));
```

El package *java.util.regex* defineix l'excepció ***PatternSyntaxException***. Aquesta excepció apareix quan s'utilitza una expressió regular mal formada (per exemple, amb un parèntesi sense tancar o un multiplicador aplicat a un metacaràcter) i per tant impideix la creació d'un objecte patró que s'inicialitzi amb aquesta expressió.

Altra excepció que, malgrat no està definida dins el package *java.util.regex*, està relacionada amb la cerca de patrons es la classe ***java.lang.IllegalStateException***. Aquesta excepció es llança quan per exemple es vol conèixer els caràcters de la coincidència si no s'ha trobat cap coincidència. Per exemple, el següent codi llança aquesta excepció en cas no es trobi cap coincidència:

```
Matcher matcher = Pattern.compile("AB").matcher(txt);
matcher.find();
System.out.println("match:"+ matcher.group());
```

AGRUPACIÓ

Tots els mètodes explicats abans no son útils per si mateixos si no podem conèixer quin fragment de la cadena coincideix amb el patró. Això ho podem conèixer de dues formes:

1. Mitjançant els mètodes *start()* i *end()* de la classe Matcher. Aquests mètodes retornen la posició dins la cadena en la que comença i acaba respectivament la coincidència de l'expressió. S'executen després de que el Matcher hagi trobat una coincidència, en cas contrari es llança una excepció java.lang.IllegalStateException. Amb les posicions i la cadena es fàcil obtenir la cadena que coincideix amb, per exemple, el mètode substring de la classe String.
2. Mitjançant la definició de grups. Es una solució més flexible i còmoda que l'anterior. Es també la que s'ha utilitzat a aquest projecte.

A una expressió regular, els metacaràcters '(' i ')' no només s'utilitzen per a definir conjunts de subexpressions**: també serveixen per a definir grups de captura**. Els grups de captura son fragments entre parèntesi d'una expressió regular que representen una informació diferenciada dins el conjunt de la subexpressió regular. Per exemple, a una expressió aritmètica els operands d'una operació de resta es poden correspondre amb un grup de captura cadascun: *"(\d+) - (\d+)"*.

Amb els grups de captura la solució d'obtenir els diferents fragments que encaixen amb part de la expressió regular es simplifica molt. La potència dels grups de captura es gran doncs poden anidar-se. Els mètodes de la classe Matcher relacionats amb els grups de captura son els següents:

- `group`
  Retorna tota la cadena que coincideix amb l'expressió. Existeix una variant del mètode que rep com a paràmetre un integer: l'ordre del grup. Així, group(1) retorna la cadena que coincideix amb el primer grup, group(2) la cadena que coincideix amb el segon grup i així successivament. L'índex dels grups de captura comença en 1 i no en 0, essent el resultat de group(0) el mateix que group().
- `groupCount`
  Retorna el nombre total de grups de captura definits a l'expressió regular.
- `start` i `end`
  Aquests mètodes poden no rebre cap paràmetre (s'ha explicat abans) o rebre un paràmetre indicant l'ordre del grup de captura. En aquest darrer cas, retornarien la posició inicial i final de la coincidència del grup de captura.

Es dóna un exemple de codi que extreu els operadors d'una suma entre dos nombres i fa la suma total:

```
Matcher matcher = Pattern.compile("(\\d)\\+(\\d)").matcher(txt);
int total = 0;
while (matcher.find()){
      for (int index=1; index<=matcher.groupCount(); index++){
            total += Integer.parseInt(matcher.group(index));
      }
}
System.out.println("total:"+ total);
```

Un darrer comentari en relació a l'ús de parèntesis per a definir grups de captura dins les expressions regulars: Existeixen casos en els quals es volen utilitzar parèntesis com a metacaràcters però no es vol que la subexpressió que contenen sigui considerada com a grup capturador. En

aquest cas es pot utilitzar el metacaràcter '?'. Per exemple, a l'expressió `<span class=\"(?:discounted-)?price\">(.+?)</span>`, s'utilitza el metacaràcter ? per a indicar que el text "discounted-" és opcional, el qual es troba entre parèntesi per a evitar que l' opcionalitat enlloc d'aplicar-se al text complet s'apliqui només al darrer caràcter ("-"). S'ha utilitzat un parèntesi NO CAPTURADOR "(?:)", per tal de que el text no sigui retornat per les crides als mètodes group(i) de la classe Matcher. Es a dir, que per a aquesta expressió, si existeixen coincidències l'operació group(1) retornaria el contingut de les etiquetes span i no el text "discounted-", que si es retornaria si el parèntesi fos capturador.

FLAGS

Finalment, comentar l' utilitat de passar-li un seguit de flags a una nova instància de la classe Pattern. Això afecta a la cerca i validació de patrons i els resultats poden variar en funció de la utilització o no d'un flag. Un flag es de tipus enter i per tant en cas de voler passar més d'un flag, s'haurà d'aplicar l'operador '|' (or entre bits). El següent es un exemple de creació d'un patró amb dos flags:

```
Pattern r = Pattern.compile(".*", Pattern.DOTALL | Pattern.MULTILINE);
```

A la solució s'utilitza el flag DOTALL. Els possibles flags que admet un patró son els següents:

- CANON_EQ: Habilita l'equivalència canònica, de tal forma que els caràcters compostos es poden cercar segons el seu caràcter base. Per exemple, el caràcter "a" seguit de l'accent "`" coincideix amb el caràcter "é" i també amb la combinació "e´".

- CASE_INSENSITIVE: No es distingeix entre majúscules i minúscules.

- COMMENTS: S'ignoren els espais en blanc i els comentaris inclosos al patró. Els comentaris comencen amb el caràcter '#' i finalitzen al primer salt de línia. A partir del salt de línia es torna a tenir en compte els caràcters de l'expressió regular.

- DOTALL: Amb aquest flag el metacaràcter "." coincideix amb qualsevol caràcter INCLOGUENT el salt de línia. A bona part del codi HTML llegit a la pràctica existeixen salts de línia que formen part del text i per tant s'han d'incloure a l'expressió regular. Amb aquest flag ens estalviem molts problemes.

- LITERAL: Habilita el parseig literal del patró. Els metacaràcters de l'expressió queden desactivats (per exemple el caràcter '.' no coincideix amb qualsevol caràcter sinó només amb el punt.

- MULTILINE: Habilita el mode multilínea de tal forma que els metacaràcters "^" i "$" no només depenen de que sigui el principi o final de la cadena però també de que hi hagi un salt de línia abans o després. Per exemple, "ABC$" coincideix amb les cadenes "ABC" i "ABC\nD". Sense aquest flag només coincidiria amb "ABC".

- UNICODE_CASE: Ignora la distinció entre majúscules i minúscules que utilitza Unicode.

- UNIX_LINES: Fa que el caràcter "\n" sigui l'únic caràcter que coincideix amb el caràcter de "salt de línia". En cas contrari, coincideix amb tots els possibles caràcters que Java reconeix com a salts de línia com '\r' o '\r\n'.

1.3.3 Aplicació de les expressions regulars

Coneguts els fonaments de les expressions regulars i les eines de que disposem per a, donada una cadena de text, extreure tota l' informació que encaixi amb una determinada expressió regular, es tracta ara d'analitzar el codi HTML extret dels diferents operadors i veure quines expressions podem construir per tal d'identificar els diferents fragments d'informació: nom d'hotel, preu d'estància, preu de vol, etc. En la mesura de lo possible, aquestes expressions haurien de tenir les següents característiques:

1. Facilitat de lectura. No han de tenir subexpressions massa complexes que puguin dificultar sense necessitat la lectura i posterior manteniment del codi, així com la detecció d'errades. En aquest sentit, es important conèixer els avantatges que proporcionen els diferents metacaràcters, abreviatures i multiplicadors del llenguatge. Per exemple, es recomanable utilitzar l'expressió "\d+" enlloc de "[0-9][0-9]*".
2. Poc sensibles a canvis. El codi HTML que s'ha d'analitzar canvia amb freqüència, doncs es produeixen canvis de disseny, incorporació de nova informació, reestructuració d'elements i tot tipus de canvis al codi que l'operador retorna al navegador de l'usuari. Lògicament aquests canvis venen donats també pel fet de que els continguts que es mostren son continguts dinàmics generats segons una petició determinada feta a un moment concret. Per aquest motiu, es important definir expressions lo més generalitzables possibles. Per exemple, si un preu es precedit pels tags " " i "", perquè no utilitzar l'expressió "<.*?>" que reconeixeria aquests dos elements junt amb altres com un salt de línia "<hr />"? S'ha de cercar un equilibri entre un extrem i altre doncs segons quines generalitzacions poden fer que una expressió tengui coincidència amb moltes parts del text i per tant no s'extregui l' informació necessària. En aquest sentit, es important definir varis nivells de detall: no fer cerques d'elements al conjunt del text sinó a les parts concretes i de petita mida que sabem contenen l' informació: No cal trencar-se el cap cercant una expressió que cerqui un preu a tota una cadena que conté tota una plana si podem fer-ho a una part més petita (per exemple, una taula amb un identificador determinat). Això evita errades doncs la variabilitat que es pot produir a un petit fragment de codi HTML es menor i facilita el manteniment del codi font doncs les expressions regulars i la forma en com es va accedint a les diferents dades, es fan més simples.

L'arquitectura bàsica del cercador segueix essent la mateixa: cada wrapper (es a dir, les classes que representen els operadors i que contenen tots els detalls propis i adequats a les particularitats de cada operador) després de llegir l' HTML dels operadors i si no s'ha produït cap error important, crida al mètode abstracte `getScrapedItems` que analitza l' informació de l' HTML que es troba a la variable privada `resultatScraping`. Com a resultat, aquesta operació desa a la variable local `resultatItemsScraping` el conjunt de resultats. Aquesta variable es un vector d'instàncies de la classe ItemBean, cadascun dels quals conté els atributs necessaris per al cercador així com altres atributs amb informació pròpia de l'operador com la direcció de l'hotel, dada no proporcionada per tots els operadors.

Tant per a hotels com per a vols definim el concepte d'**item**: un resultat amb un preu determinat que pot ser una estància a un hotel o un vol entre dues ciutats. Consequentment definim una classe abstracta **ItemBean** la qual té dues classes filles: les classes **HotelBean** i **FlightBean**. L'estructura de classes es similar a la que s'utilitza amb els wrappers i es pot consultar a la secció d'annexos d'aquesta memòria. La classe ItemBean dóna accés a un seguit de mètodes comuns a tots els items de resultats:

- `getPreu`: El mètode més important. Indica el preu (en euros) de l'item. Es un mètode que implementarà cada classe filla no abstracta de la classe ItemBean i que retorna un nombre real calculat segons els atributs interns d'aquesta classe.
- `getNomOperador`: El nom descriptiu de l'operador en el que apareix i per tant es pot reservar el item. Es una informació addicional que es mostrarà al resultat de disponibilitat.
- `toDebugHTML`: Representació de l'item com a format HTML. Utilitzat en tasques de debug, quan es volen crear fitxers HTML dels resultats intermedis dels items que es van processant (al igual que es fa amb les crides HTTP on els resultats es poden en cas de proves emmagazemar a un fitxer HTML).
- `getHTML`: Representació dem part de l' informació de l'ítem com a format HTML, informació que es mostra al llistat resposta de la cerca que visualitza l'usuari.
- `getKey`: Retorna un String que identifica cada ítem. Cada ítem de cada hotel es identificat pel seu nom.

Tots els ítems tenen també una atribut local que conté les dades de la petició, doncs a vegades es necessita conèixer el nombre de nits o el nombre d'habitacions per a formar el preu final i un mètode local (`compareTo`) que compara l'ítem amb altre ítem passat com a paràmetre en funció dels seus preus. Aquest mètode s'explica a l'apartat "Ordenació de preus".

El concepte d'ítem serveix per a definir el mètode general d'obtenció de l' informació:

- S'analitza el contingut de la variable `resultatScraping` i s'iteren tots els ítems
- Per cada ítem es crea una instància de la classe filla d'HotelBean per a hotels i FlightBean per a vols (un ItemBean). Es donen valors a l'objecte analitzant cada ítem amb expressions regulars.

Aquest es un mètode general que potser no s'utilitzi a qualque wrapper, però s'intentarà seguir aquesta estructura.

1.3.3.1 Operadors d'hotels

La classe HotelBean conté els atributs i mètodes propis dels ítems d'hotels: el nom de l'hotel i el preu. El preu es un atribut que s'ha baixat de la classe ItemBean a la classe HotelBean doncs la majoria d'ítems de vols tenen maneres particulars de cercar el preu i no cal que utilitzin aquest atribut. A continuació es comenten les expressions regulars emprades i les particularitats de les classes filles d'HotelBean (els ítems de cada operador d'hotels).

1.3.3.1.1 Operador DHR

Amb l'anàlisi d'aquest operador, **s'ha decidit descartar-lo** i prescindir d' ell a la consulta de preus. El motiu està a la forma en com aquest operador mostra els preus. A diferència de la resta d'operadors el preu no consisteix a una seqüència de dígits numèrics sinó que es tracta d'una imatge.

L'element *src* de la imatge que representa el preu es una crida a una plana PHP que rep com a paràmetre una cadena alfanumèrica que consisteix a la representació en Base 64 del contingut binari de l' imatge. Es a dir, podríem convertir el contingut binari una imatge, per exemple del preu "50", i obtindríem el paràmetre que utilitza l'atribut SRC de l'element img. Aquest atribut s'envia a una funció PHP que internament (el codi està ocultat de l' HTML doncs lògicament al ser codi PHP es

tracta al servidor web de l'operador) retorna el contingut real de l' imatge a partir de la seva representació de text en Base 64.

Per exemple, a continuació tenim el nombre 36 i com apareix a l' HTML:

```
<img
src="/lib/getEncodedImage.php?image/gif;base64,R0lGODdhEAAMAOMAAMTf8gNyvJPD5DONyR
t/wkua0KvR62Oo13u23QAAAAAAAAAAAAAAAAAAAAAAAAAAAACwAAAAAEAAMAAAERxDISau9x0wxAimaZBA
Bsh3IERRTJ1iGEBwSUl6B1+bBEEqCwkqSQ9hYlQ5xIFEmAxICE+AsFFJDgOqgQh5IBNqkQEJezoAIADs=
">
```

Tenim l'opció de mapejar cada atribut SRC de cada preu a un preu diferent. Això es una tasca força tediosa doncs obliga a tractar de forma individual tots els possibles preus. No es gaire recomanat tampoc lligar un preu a una cadena alfanumèrica com aquesta, doncs un petit canvi a les imatges dels preus invalidaria aquest operador.

Aquests sons els motius principals per a descartar-lo. Amb aquesta decisió, el nombre total d'operadors consultats son el mateix tant per a hotels com per a vols: 5.

1.3.3.1.2 Operador Hoteles.com

Els items d'aquest operador tenen la següent forma gràfica:

Figura 6 *Item Hoteles.com*

L'expressió regular per a obtenir els ítems es la següent: *"<li class=\"hotel(?: top_offer)?\">.+?
"*. Amb una crida al mètode estàtic getValues de la classe *pfc.util.Er* obtenim un vector amb tots els ítems de l' HTML.

Per cada item obtenim la següent informació:

- Nom de l'hotel (*<a.*?>(.+?)*). Contingut del primer enllaç de l'ítem
- Zona (*<div class=\"desc area_data\">(.+?))*
- Categoria ((\\d+) estrellas). Dígit numèric que indica el nombre d'estrelles de l'hotel - Es un camp opcional-
- Preu. Si no es troba amb l'expressió *"<ins>(.*?)</ins>"* es mira l'expressió *"class=\"price\">\\s*(.*?)"*. Això es fa així doncs alguns ítems tenen el preu taxat.

1.3.3.1.3 Operador skoosh

Els ítems d'aquest operador tenen la següent forma:

Figura 7 *Item Skoosh*

L'expressió regular *"class=\"(table double_col detail width530|table_full_width)\" style=\"\">.*?<div class=\"spacer2\">"* obté tots els items: tots els divs amb la classe indicada i que estan succeïts pel div amb classe "spacer2".

Per cada item obtenim la següent informació:

- Nom de l'hotel
- Categoria
- Descripció. La descripció conté enllaços html. Al target de l'enllaç se li ha d'afegir al principi el domini de l'operador (http://www.skoosh.com) per a que funcionin. S'acaba fent un replaceAll del contingut dels href pel nou: Primer s'obtenen els continguts dels href i després es fan servir com a expressio regular substituint-los per ells mateixos més el domini de l'operador.
- Descripció de les habitacions: Continguts dels paràgrafs que venen precedits pel text "1 de habitación", "2 de habitación", etc.
- Preu: Es llegeix el preu total en negreta, no el preu desglossat per habitació

1.3.3.1.4 Operador lastminute.com

Els ítems d'aquest operador tenen la següent forma:

Figura 8 *Item lastminute.com*

L'expressió regular *"<div class=\"product available \">.*?(<div class=\"product available \">|<div class=\"end-of-list\">)"* obté els diferents ítems de l'operador. Els divs que s'estan cercant tenen com a element delimitador els div's amb classe "product available" o "end-of-list". Com aquests elements formen part del grup de captura hi ha que reiniciar el matcher i retrocedir-ho unes posicions, de tal forma que la cerca del següent ítem comenci de nou amb un element de classe "product available", si no estem en el darrer ítem. Si no es fa així, com el matcher comença a cercar després de la coincidència i el començament del següent ítem forma part de la coincidència actual, no apareixerien tots els ítems al resultat de coincidències.
Per cada ítem obtenim la següent informació:

- Nom de l'hotel
- Zona
- Categoria
- Direcció
- Descripció
- Preu. L'expressió més complexa de l'operador *–"(?:class=\"number (?:\\s+highlight)?\">\\s*([\\d,.]+))|(?:strike\">\\s*[\\d,.]+)"*. S'utilitza un operador d'alternança per a tenir en compte les diferents possibilitats en la que es mostra el preu: taxat o no taxat. També s'utilitza el multiplicador per a ocurrències opcionals doncs hi ha una classe que pot no aparèixer. Destacar també l' utilització del metacaràcter '\s', per a admetre un nombre indeterminat de separadors.

1.3.3.1.5 Operador Hotelopia

Els ítems d'aquest operador tenen la següent forma:

Figura 9.a *Item Hotelopia*

Figura 9.b *Item Hotelopia*

L'expressió regular *"<li class=\"hotel[^\\n]*?\"[^\\n]*?id=\"result.*?\">.*?(<li>.*?.*?)*
"* obté els diferents ítems de l'operador. Es permeten qualsevol nombre de <li> dins els <li> amb identificador que comença per la paraula "result".

Per cada ítem obtenim la següent informació:

- Nom de l'hotel
- Categoria: El nombre d'estrelles forma part del nom de l' imatge que es mostra per a indicar la categoria de l'hotel (*"(?:/(\\d+)EST\\.gif)|(?:/HS(\\d+)\\.gif)"*)
- Descripció
- Descripció d'habitació/règim: Per a la **primera taula** de les diferents taules de possibles preus dins el mateix hotel, es llegueixen les columnes que contenen el tipus d'habitació i el règim. Es fan dues distincions doncs el codi HTML es diferent en funció de si es mostra una habitació o vàries. Es fa aquesta distinció segons el nombre de resultats retornats per l'expressió *"<tbody class=\"multi\">.*?Reservar"*.
- Preu: Es llegeix el preu de la primera taula de resultats dins l'hotel, doncs es el preu més econòmic. El preu es troba al codi HTML que conté les habitacions.

1.3.3.1.6 Operador getaroom

Els ítems d'aquest operador tenen la següent forma:

Figura 10 *Item getaroom*

L'expressió regular *"<li class=\"searchresult\">.*?(<li.*?>.*?</li.*?>.*?)*
"* obté els diferents ítems de l'operador i es molt similar a la que utilitza amb l'operador Hotelopia per a obtenir els seus ítems. Si no es troba cap resultat s'utilitza altra expressió regular més simple per a trobar el llistat d'ítems: *"<li id=\"property.*?>.*?
"*.

Per cada ítem obtenim els següents atributs:

- Nom de l'hotel
- Descripció
- Preu

1.3.3.2 Operadors de vols

La classe FlightBean conté els mètodes comuns a les classes filla que definiran l' informació relativa als ítems dels diferents operadors de vols. A continuació es comenten les expressions regulars emprades i les particularitats de les classes filles de FlightBean (els ítems de cada operador).

Especialment a l'obtenció de dades dels ítems dels operadors de vols, més que als operadors d'hotels, es llegeixen dades que son redundants doncs son conegudes ja que coincideixen amb el filtre de cerca. Per exemple, els ítems mostren la data de sortida, que coincideix amb la data de sortida del formulari de cerca del cercador de VTC, les ciutats origen i destí, etc. Aquesta informació es llegeix i es tracta com a atributs dels diferents ítems, doncs es considera interessant conèixer aquestes dades davant possibles canvis no detectats o no tractats. Per exemple, pot passar que es configuri una nova ciutat al cercador que no disposi d'aeroport i que l'operador et retorni la ciutat més propera amb aeroport. O que hi hagi un error i que es cerquin dades o ciutats incorrectes. Sempre es útil i recomanable mostrar junt amb el preu d'una estància en un hotel o d'un vol les dades exactes que apareixen a la web de l'operador i per a les que s'ha realitzat la cerca. Per seguretat i també per que es una ajuda visual més a l'usuari que consulta la disponibilitat i els preus. Al cap i a la fi, i com es va fer amb els operadors d'hotels es tracta de llegir les diferents dades de cada ítem i no només les dades estrictament identificatives com el preu o el nom de l'hotel.

1.3.3.2.1 Operador eDreams

Els ítems d'aquest operador tenen la següent forma:

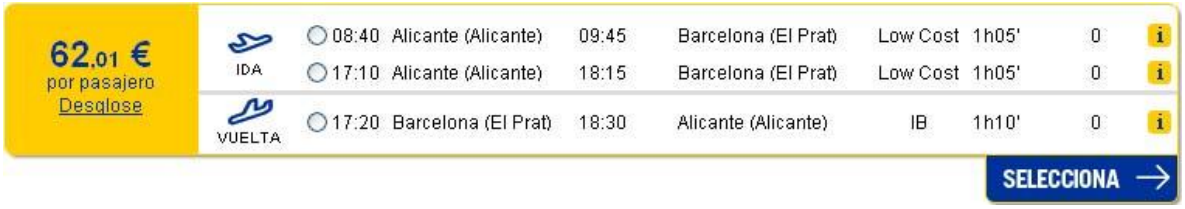


Figura 11 *Item eDreams*

L'expressió *"<form[^\n]*?id=\"itinerary.*?</form>"* s'utilitza per a obtenir els diferents ítems de l'operador, tots els formularis amb l' id especificat. Per tal de no obtenir dades d'altres formularis s'inclou la condició [^\n] entre el començament de l'etiqueta <form> i el seu id. Així, s'obté l'etiqueta "<form" més propera a l'identificador dels formularis dels items. Es va detectar que entre la primera coincidència de "<form" i la part "id=\"itinerary" es llegien dades que no eren les correctes, i es soluciona indicant que no hi hagi cap salt de línia enmig (com s'utilitza el flag DOTALL a l'objecte Pattern, s'ha d'explicitar que no es tingui en compte el salt de línia).

Per cada ítem obtenim la següent informació:

- Preu: L'expressió *"singleItinerayPrice.*?(\\d+)<.*?>(,\\d+)<"* té en compte que entre la part entera i la part decimal del preu poden existir tags HTML.
- Data d'anada
- Dates d'anada (ciutat i hora de sortida i companyia): Es crea un vector on cada element del vector es un array amb una possible combinació d'anada diferent. Com s'ha comentat, existeixen dades que poden parèixer redundants però donen seguretat si l'usuari que tria un ítem basant-se en el preu pot consultar les dades que realment es van retornar. Dins el codi HTML de l'operador, les diferents combinacions de les anades es troben com a columnes a una taula HTML.
- Data de tornada
- Dates de tornada: Es fa un tractament similar al que es fa amb les dades d'anada, amb el mateix tipus d'informació.

1.3.3.2.2 Operador lastminute.com

Els items d'aquest operador tenen la següent forma:

Figura 12 *Item lastminute.com*

L'expressió *"<form[^\n]*?id=\"fare.*?</form>"* s'utilitza per a obtenir els diferents ítems de l'operador, i es pareguda a la que s'utilitzava per a obtenir els ítems de l'operador eDreams per motius similars.

Per cada item obtenim la següent informació:

- Preu per adult. L'expressió "*por adulto.*?strong>(\\d+\\.\\d+)*" escapa el metacaràcter ".", que en aquest cas s'utilitza com a separador de la part entera i la part decimal.
- Preu per nin
- Preu per bebè
- Dates d'anada. Cada combinació possible d'anada té les dades: nom de la companyia aèria, identificador del vol, data de sortida i data d'arribada
- Dates de sortida. Cada combinació possible de sortida té les dades: nom de la companyia aèria, identificador del vol, data de sortida i data d'arribada

1.3.3.2.3 Operador Rumbo

Els ítems d'aquest operador tenen la següent forma:

Figura 13 *Item Rumbo*

En aquest cas, l'expressió regular que retorna els diferents ítems es molt simple: *"IDA:.+?boton_reservar.gif",* es a dir entre el text "IDA" i el botó de reserva.

Per cada ítem obtenim la següent informació:

- Preu: El preu total no es el preu que es mostra a l' imatge de l'item, aquest es només el preu per adult. El preu final i total es pot consultar a una capa flotant que es visualitza quan el ratolí es posa al text "Desglose del precio". A partir de l' informació d'aquesta capa s'extreu el preu final. Primer s'obté el contingut d'aquesta capa que es el paràmetre de la funció javascript *overlib* i després d'aquest contingut es llegeix el preu total:

```
String overlib=
    pfc.util.Er.getValue("overlib(.*?)Desglose del precio<",item);
String preu=
    pfc.util.Er.getValue("TOTAL.*?>(\\d+(?:,\\d+)?)&euro",overlib);
```

- Preu adult: Es el preu visualitzat i que es llegeix per a tasques de debug, però no es té en compte al calcular el preu final, que es l'atribut anterior.
- Dates d'anada. Cada combinació possible d'anada té l'hora i ciutat de sortida, l'hora i ciutat d'arribada i la companyia aèria.
- Dates de sortida. Cada combinació possible d'anada té l'hora i ciutat de sortida, l'hora i ciutat d'arribada i la companyia aèria.

1.3.3.2.4 Operador Spanair

Els ítems d'aquest operador tenen la següent forma:

Figura 14 *Item Spanair*

Aquest operador es una mica més especial que els anteriors doncs no es mostra cada "ítem" per separat sinó que primer tenim les possibles combinacions d'anada, cadascuna amb el seu preu, i després les possibles combinacions de tornada, cadascuna també amb el seu preu. Per a obtenir les dades reals dels ítems s'han de combinar totes les possibles anades amb totes les possibles tornades essent el preu final de cada ítem la suma de l'anada més la suma de la tornada.
L'expressió "*class=\"flights\">.*?</table>*" obté dues taules: la de les anades i la de les tornades. A cada taula es llegueixen les diferents files que contenen les dades individuals de les anades o les tornades. Per cada anada o tornada les dades son:

- Hora
- Ruta (ciutat origen i ciutat destí)
- Duració del viatge
- Escales
- Preus (preu visualitzat, preu mitjà per adult, preu mitjà per nin i preu mitjà per infant): El preu mostrat a la secció HTML de cada ítem no es el preu final sinó el preu mitjà per adult. Es necessita lleguir el preu mitjà d'adult, nin i infant per calcular el preu final i real del vol. Aquesta informació es troba dins el codi javascript de la plana, doncs l' informació dels preus es carrega als atributs d'uns objectes tipus Recommandation. Per exemple,

```
var theRecommandation = new Recommandation(2, '135.00');
theRecommandation.addPassenger('ADT', '45.00');
theRecommandation.addPassenger('CHD', '45.00');
theRecommandation.addPassenger('INF', '15.00');
```

Es veu que a aquest codi s'indica el preu de cada adult (ADT), de cada nin (CHD) i de cada infant (INF). Cada objecte Recommandation té un id (El **2** en negreta) que es pot obtenir de l'

HTML de cada ítem, en la crida a la funció executada al seleccionar el radio button de cada anada/tornada: “onclick="javascrip:theFFCOForm.selectFlight(0,0,**2**, this);”.

Es té un vector d’anades i un altre de tornades:la solució final passa per fer totes les combinacions possibles. Es mostra el codi que s’encarrega d’aquesta tasca concreta:

```
for (String[] anada: vItemsAnades) {
   for (String[] tornada: vItemsTornades) {
      SpanairBean spanairBean = new SpanairBean(beanPeticioConsulta);
      spanairBean.setAnada(anada);
      spanairBean.setTornada(tornada);
      vVolBean.add(spanairBean);
   }
}
```

1.3.3.2.5 Operador Vueling

Els gitams d’aquest operador tenen la següent forma:

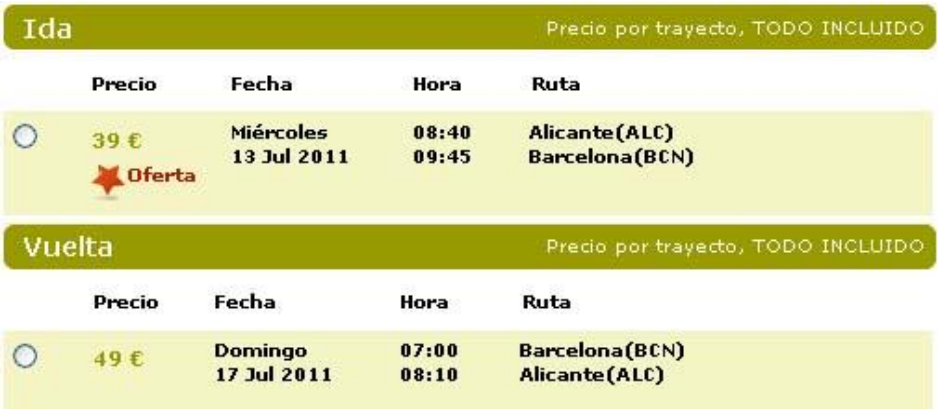


Figura 15 *Item Vueling*

El tractament de l’ informació que es fa amb aquest operador es molt similar al que es fa amb l’operador Spanair: primer es llegeix ’informació de les anades, després la de les tornades i es fan totes les combinacions entre les anades i les tornades.

L’expressió *“>Ida(?:</span|</div)(.*?)Vuelta(?:</span|</div)(.*)*” obté l’ informació de tots els items cercats a l’operador. Amb aquesta expressió es consulta d’una banda l’ informació de les anades (primer grup de captura) i de l’altra l’ informació de les tornades (segon grup de captura)

Dins l’ informació retornada per cada grup de captura es consulten les diferents anades o tornades amb l’expressió “*class=\"destacado.*?\".*?(<table.*?</table>.*?)*</tr*”. Com es dedueix de l’expressió, aquestes es troben a una fila HTML amb classe “destacado”. Com dins la fila poden existir altres files, s’ha d’incloure els elements <table></table> dins l’expressió per a excloure les files internes –amb el multiplicador * per a permetre un nombre indeterminat de taules-. El contingut acaba amb el tancament de la fila (</tr).

Per cada anada/tornada es llegeixen les següents dades:

- Preu. L'expressió per a obtenir els preus *"*(\\d+(\\.\\d+)?)(<![^>]*>)?\\s*€*"* fa us de l'operador "?" per a indicar que el segon grup no es un grup de captura. Aquest segon grup es un element opcional (ho indica el multiplicador '?') i es un comentari HTML (<!-- -->). S'utilitza la mateixa expressió per a cercar el preu de l'adult i el preu del bebè, que es opcional.
- Preu per bebè (es una dada apart, mostrada baix el preu)
- Data del vol
- Ruta (ciutat origen-ciutat destí)

1.3.4 Ordenació de preus

Dins l' informació proporcionada per cada ítem retornat per cada operador tenim l'atribut més important i fonamental: el preu. Cada operador retorna preus com a resultat d'una cerca de preus i disponibilitat, però aquests preus no son fàcilment comparables entre operadors doncs no sempre representen el preu final. Per exemple, a una consulta de disponibilitat a un hotel el preu mostrat pot representar el preu mitjà per habitació mentre que a altre el preu pot representar el preu mitjà per nit. Similarment, a un operador de vols el preu pot representar el preu mitjà per passatger i no el preu total del vol. Resumint, seria incorrecte extreure la conclusió de que un preu de "200 €" es superior a un preu de "150€" si el preu de 200€ aplica a tota la estància (preu total) i el preu de 150€ es el preu mitjà per nit a una estància de dues nits.

Tota aquesta qüestió de tractament de preus es tracta als següents mètodes:

Mètode getPreu

Mètode abstracte inclòs a la superclasse abstracta de totes les classes que representen les particularitats dels ítems dels diferents operadors: la classe *pfc.bean.ItemBean*. Aquest mètode es implementat per totes les classes concretes que tenen l' informació de cada ítem dels diferents operadors doncs cada classe té la seva forma d'implementar com calcula el preu final. Internament cada classe concreta i filla de *pfc.bean.ItemBean* té un o més atributs amb l' informació de preus que utilitza juntament amb les dades de la petició (com el nombre de nits o el nombre de persones) per a calcular el preu final. Per exemple, la lògica relacionada amb el càlcul del preu de cada ítem a l'operador LastMinute és la següent:

```
public float getPreu() {
	return Float.valueOf(this.preu.replace(",","."));
}
```

Mètode getFormatPreu

Retorna el preu formatat, amb un format comú de presentació compartit per tots els preus. Es mostra el preu com a un nombre real amb dos decimals essent la coma el separador dels decimals. Lògicament, aquest mètode es troba implementat a la superclasse *pfc.bean.ItemBean*, doncs l' implementació es la mateixa per a tots els ítems.

La forma en la qual es calcula el preu final per ítem es la següent

- Hotels
  - ***Hoteles.com***

    El preu directament extret de cada ítem es el preu total, impostos inclosos. No fa falta cap tractament especial. El mètode `getPreu` és el següent:

```
return Float.valueOf(this.preu.replace(",","."));
```

- ***Skoosh***
  El preu directament extret de cada ítem es el preu total, impostos inclosos. No fa falta cap tractament especial. El mètode getPreu és el següent:

```
return Float.valueOf(this.preu.replace(",","."));
```

- ***lastminute.com***
  El preu directament extret de cada ítem es el preu total, impostos inclosos. No fa falta cap tractament especial. El mètode `getPreu` és el següent:

```
return Float.valueOf(this.preu.replace(",","."));
```

- ***Hotelopia***
  El preu directament extret de cada ítem es el preu total, impostos inclosos. No fa falta cap tractament especial. El mètode `getPreu` és el següent:

```
return Float.valueOf(this.preu.replace(",","."));
```

- ***getaroom***
  El preu directament extret de cada ítem es el preu mitjà per nit i per ocupant. S'ha de multiplicar pel nombre d'ocupants i pel nombre de nits. El preu final normalment no inclou impostos, sinó que es una dada que depèn de cada hotel. Es decideix mostrar una dada informativa dins l' informació addicional llegida de cada ítem. El mètode `getPreu` és el següent:

```
return Float.valueOf(preu.replace(",","."))*
   beanPeticioConsulta.getNits()*
   beanPeticioConsulta.getNumHabitacions();
```

- Vols
  - ***eDreams***
    El preu directament extret de cada ítem es el preu mitjà per passatger: hi ha que multiplicar-ho pel nombre total de passatgers. El mètode `getPreu` és el següent:

```
return Float.valueOf(preu.replace(",",".")) *
beanPeticioConsulta.getNumTotalPax();
```

  - ***lastminute.com***
    El preu que s'extreu de l'ítem es el preu mitjà de cada adult/nin/bebè. El preu mitjà del nin o del bebè pot no aparèixer si no existeixen passatgers d'aquest tipus. També comentar que l'operador pot incloure despeses d'emissió amb posterioritat, típicament 8'5€ per passatger. El mètode `getPreu` és el següent:

```
float preuAdult = Float.valueOf(this.preuAdult.replace(",","."));
float preuNin =
     this.preuNin==null?0:Float.valueOf(this.preuNin.replace(",","."));
float preuBebe =
     this.preuBebe==null?0:Float.valueOf(this.preuBebe.replace(",","."));
return
     preuAdult * beanPeticioConsulta.getNumTotalAdults(12) +
     preuNin * beanPeticioConsulta.getNumTotalPax(2,11) +
     preuBebe * beanPeticioConsulta.getNumTotalBebes(1);
```

  - ***Rumbo***
    El preu directament extret de cada item es el preu total, impostos inclosos. No fa falta cap tractament especial. El mètode `getPreu` és el següent:

```
return Float.valueOf(preu.replace(",","."));
```

  - ***Spanair***
    Es l'operador amb els preus més complicats de calcular.
    El preu directament extret de cada ítem es el preu per adult, el preu per nin i el preu per infant que pot ser diferent per a l'anada i per a la tornada. Tenint en compte el preu de

cada tipus de passatger (adult,nin,infant) es sumen els preus per a l'anada i per a la tornada. Es sumen també els impostos tant de l'anada com de la tornada (recordar també que tota aquesta informació s'extreu del codi Javascript de la pàgina retornada pel operador, doncs no es troba directament al fragment HTML de l'ítem). El mètode `getPreu` és el següent:

```
float preuAdultAnada = anada[5] == null ? 0 :
   Float.valueOf(anada[5].toString().replace(",","."));
float preuAdultTornada = tornada[5] == null ? 0 :
   Float.valueOf(tornada[5].toString().replace(",","."));
float preuNinAnada = anada[6] == null ? 0 :
   Float.valueOf(anada[6].toString().replace(",","."));
float preuNinTornada = anada[6] == null ? 0 :
   Float.valueOf(anada[6].toString().replace(",","."));
float preuBebeAnada = anada[7] == null ? 0 :
   Float.valueOf(anada[7].toString().replace(",","."));
float preuBebeTornada = anada[7] == null ? 0 :
   Float.valueOf(anada[7].toString().replace(",","."));
float preuTasesAnada = anada[8] == null ? 0 :
   Float.valueOf(anada[8].toString().replace(",","."));
float preuTasesTornada = tornada[8] == null ? 0 :
   Float.valueOf(tornada[8].toString().replace(",","."));

int numAdults = this.beanPeticioConsulta.getNumTotalAdults(12);
int numNins = this.beanPeticioConsulta.getNumTotalPax(2,11);
int numBebes = this.beanPeticioConsulta.getNumTotalBebes(1);
return
   (preuAdultAnada + preuAdultTornada) * numAdults +
   (preuNinAnada + preuNinTornada) * numNins +
   (preuBebeAnada + preuBebeTornada) * numBebes +
   preuTasesAnada + preuTasesTornada;
```

- ***Vueling***

  El preu directament extret de cada ítem es el preu mitjà per adult i opcionalment, el preu mitjà de cada bebè, impostos inclosos. Recordar també que els preus per adult i per nin s'extreuen del codi Javascript, doncs el preu mostrat a la secció HTML de cada item es correspon amb el preu d'un adult. El mètode `getPreu` és el següent:

```
int numPax = this.beanPeticioConsulta.getNumTotalAdults(12) +
      this.beanPeticioConsulta.getNumTotalPax(2,11);
int numBebes = this.beanPeticioConsulta.getNumTotalBebes(1);
return
      Float.valueOf(anada[0].toString().replace(",",".")*numPax +
      (anada[1]==null?0:Float.valueOf(anada[1].toString().replace(","."
))*numBebes) +
      Float.valueOf(tornada[0].toString().replace(",",".")* numPax +
      (tornada[1]==null?0:Float.valueOf(tornada[1].toString().replace(","
,".",))*numBebes);
```

Una vegada definida l'operació "getPreu" que retorna un nombre real que representa el preu total per ítem i que facilita la comparació de preus entre ítems, hi ha que fer l'ordenació i escollir finalment les cinc combinacions més econòmiques. L'algorisme associat a aquest tractament és el següent:

```
Vector vFinalHotelBean  = {tots els items dels diferents operadors
  d'hotels}
Vector vFinalFlightBean = {tots els items dels diferents operadors de
```

```
  vols}

vFinalHotelBean  = ordenació(vFinalHotelBean);
vFinalHotelBean  = Obtenir_5_items_mes_economics(vFinalHotelBean);

vFinalFlightBean = ordenació(vFinalFlightBean)
vFinalFlightBean = Obtenir_5_items_mes_economics(vFinalFlightBean);
```

S'ha definit la classe **CombinacioBean** que representa una única combinació d'hotel i vol. Els seus atributs son l'ítem de l'hotel i l'ítem del vol. Té un mètode que retorna el preu de la combinació: la suma del preu de l'hotel més la suma del preu del vol:

```
public float getPreu() {
      return hotelBean.getPreu() + flightBean.getPreu();
}
```

Definida la classe CombinacioBean, la següent acció es muntar les diferents combinacions en base als 5 ítems més econòmics d'hotels i als 5 items més econòmics de vols. El codi es el següent:

```
Vector<CombinacioBean> vCombinacions = new Vector<CombinacioBean>();
for (ItemBean hotelBean: vFinalHotelBean){
      for (ItemBean volBean: vFinalFlightBean){
            vCombinacions.add(new CombinacioBean(hotelBean, volBean));
      }
}
```

Finalment, ordenant el vector vCombinacions, obtenim el resultat de la cerca: les combinacions de vol i hotel més econòmiques per a tots els operadors:

```
vCombinacions = ordenació (vCombinacions);
vCombinacions = Obtenir_5_items_mes_economics(vCombinacions);
```

Per a realitzar l'ordenació d'un vector d'ítems, la solució es la mateixa tant per als items de vols, els items d'hotels com les combinacions d'hotel i vol. S'ha implementat la interfície ***java.lang.Comparable*** a les classes els objectes de les quals hem d'ordenar (*pfc.bean.ItemBean* i *pfc.bean.CombinacioBean*). Aquesta interfície serveix per a definir un ordre entre els objectes de la classe que l'implementa. L' implementació d'aquesta interfície obliga a implementar el mètode `compareTo`, que defineix les regles d'ordenació entre dos objectes que es volen ordenar. Aquest mètode retorna un nombre enter amb el següent significat:

- Valor positiu, major que zero: Si l'objecte actual es "major" que l'objecte passat com a paràmetre (té un preu major)
- Valor negatiu, major que 0: Si l'objecte actual es "menor" que l'objecte passat com a paràmetre (té un preu menor)
- Valor zero: Si els objectes son iguals dins l'ordenació: tenen el mateix preu.

Els mètodes compareTo son els mateixos a les dues classes ItemBean i CombinacioBean amb la diferència del tipus d'objecte que s'ordena. El codi es el següent:

```
      public int compareTo(ItemBean itemBean) {
            if (this.getPreu() > itemBean.getPreu()) return 1;
            if (this.getPreu() < itemBean.getPreu()) return -1;
            return 0;
      }

      public int compareTo(CombinacioBean combinacioBean) {
            if (this.getPreu() > combinacioBean.getPreu()) return 1;
```

```
            if (this.getPreu() < combinacioBean.getPreu()) return -1;
            return 0;
        }
```

L'interfície `java.lang.Collections`, proporciona el mètode estàtic `sort(List<T> list)` que rep per paràmetre una llista –en aquest cas un vector- i l'ordena ascendentment. Els elements de la llista (vector) implementen la interfície Comparable (son comparables uns amb els altres), i per tant el mètode compareTo ens ajuda a conèixer quin es el criteri d'ordenació. En aquest cas, els elements s'ordenen per preu de menor a major. Per exemple, la següent línia de codi fa la darrera ordenació de totes: l'ordenació de les combinacions de vol+hotel:

```
java.util.Collections.sort(vCombinacions);
```

1.4 Capítol 4. Creació de l' XML.

1.4.1 Introducció

Un dels requisits del cercador és la generació d'un arxiu XML de sortida del procés de cerca amb les dades necessàries per a poder formalitzar amb posterioritat una reserva a través de l'aplicació de reserves de que disposa l'organització VTC. La proposta inicial del projecte dóna a entendre que l'arxiu ha de contenir les dades *mínimes*, es a dir, les dades que identifiquen una possible reserva com per exemple les dades inicial i final, la ciutat de destí o el nom de l'hotel que es decideix reservar. Inevitablement, per a l'ús que se li dóna a aquest arxiu XML això ha de ser així però no es exigència de l'enunciat que no apareguin certes dades addicionals. S'incorporaran dades addicionals del resultat de cerca (com per exemple la descripció d'un hotel a un operador determinat) a l' XML, que aporten informació addicional i útil del resultat de cerca i que finalment es mostraran dins el codi HTML que visualitzarà l'usuari com a resultat de la cerca. Aquesta darrera part però queda detallada al següent capítol.
Així doncs, ara l'objectiu és transformar els resultats de la cerca, que tenim representats als objectes creats amb l' informació extreta mitjançant expressions regulars a format XML. Aquesta informació XML s'haurà d'emmagatzemar físicament a un arxiu XML per tal de permetre accessos posteriors.

Amb la plataforma Java disposem de moltes formes de serialitzar o convertir els objectes Java a format XML:

- Es pot fer manualment, sense cap funció ni API especial. Simplement es tracta d'anar concatenant a una variable (String), els elements XML que vulguem definir assignant-los contingut en funció dels valors retornats pels *getters* dels objectes que contenen les dades. Això dificulta el manteniment del codi de serialització i no dóna cap valor afegit a la solució a aquest projecte. L' utilització dels mètodes de la classe XMLStreamWriter del package *java.xml.stream* és una solució més correcta però també bastant manual.
- Es pot fer amb la classe XMLEncoder del package *java.beans* que obté una representació XML d'un determinat JavaBean (objecte Java serializable i amb getter's i setters que faciliten l'accés als valors dels diferents atributs). Es senzill d'utilitzar.
- Existeixen moltes APIs de diferents proveïdors que ajuden a la creació de representacions XML d'objectes Java. El projecte XMLBeans, d'Apache- es sens dubte una de les millors. Presenta per cert, moltes similituds amb l'API que finalment utilitzarem. Entre altres llibreries es poden mencionar *Apache Betwixt (http://commons.apache.org/betwixt/)* -també d'Apache-, *XStream* (http://xstream.codehaus.org/) o Simple (http://simple.sourceforge.net/).
- L'API JAXB, que es l'opció contemplada a l'apartat de planificació, és la llibrera escollida.

1.4.2 JAXB

JAXB i JAXP son les dues principals APIs de Java que faciliten el tractament de documents XML, i seran les dues que s'utilitzaran a aquest projecte (L'API JAXP es tracta al capítol posterior). Es dedueix doncs que uns dels avantatges de l' utilització de JAXB es que no requereix de l' instal·lació de cap programari addicional.

L'API JAXB (*Java Arquitecture for XML Binding –Arquitectura Java para uniones XML-* proporciona una llibreria que bàsicament ofereix la generació de classes Java partint d'un esquema XML i que permeten al desenvolupador incorporar dades de documents XML a les seves aplicacions i fer ús d'elles sense tenir la necessitat de conèixer els fonaments de XML. JAXB també permet manipular els valors dels objectes instanciats a partir de les classes generades i generar de nou un document XML partint d'aquestes dades.

El principal avantatge de JAXB es que permet al desenvolupador oblidar-se dels detalls dels documents XML: proporciona una capa que abstreu tots els detalls inherents als documents XML, de tal forma que aquests poden llegir-se i crear-se cridant funcions d'aquest API que internament ja es carreguen d'aquest tipus de detalls. Altres avantatges son els següents: permet la validació de les dades (no es permet la creació d'objectes Java partint de XMLs que no compleixen estrictament amb el seu esquema), es ràpid (més ràpid encara que l'analitzador SAX –Simple API for XML-, analitzador dirigit per esdeveniments), és fàcil d'utilitzar, facilita la personalització del procés de generació dels objectes Java partint d' XMLs (per exemple es pot configurar l'esquema d'unió -el document que especifica els detalls de la generació dels objectes Java a partir dels esquemes- especificant el tipus que tindrà al llenguatge Java un atribut generat partint d'un element de l' XML), i facilita l'extensió de les classes generades per tal de que proporcionin noves funcionalitats.

JAXB defineix dues operacions bàsiques i fonamentals:

- *Unmarshal*: Procés de creació d'un conjunt de classes i interfícies Java partint d'un document (DTD o esquema XML) que especifica les regles de construcció del document XML que serveix de base per a la creació de les classes i interfícies.
- *Marshal*: Procés de conversió d'un objecte Java a format XML (serialització d'objectes).

Si be una aplicació JAXB completa farà ús normalment de bona part de les operacions disponibles a l'API JAXB, a aquest projecte només cal utilitzar l'operació de marshalling o serialització dels objectes. Típicament, una aplicació JAXB completa que construeix objectes a partir d'un document XML, els modifica i els serialitza de nou a un document XML realitzarà les següents operacions:

- Definició de l'esquema. Es construeix l'especificació XML que defineix l'estructura i organització del document XML en el qual es basarà l'estructura de classes que posteriorment es construirà. Aquesta especificació, i en funció de la versió de JAXB que utilitzem, pot es un esquema XML (amb la notació estàndard especificada pel W3C) o un DTD. Les versions inicials de JAXB treballaven amb DTD's exclusivament.
- Generació de classes. Es genera l'estructura de classes i les interfícies que representen l'esquema XML. Les implementacions de JAXB disposen d'un compilador d'enllaçat que construeix aquest conjunt de classes partint de l'esquema XML. Existeix un comportament per defecte en quant a com es creen les classes a partir de l'esquema. Aquest comportament pot personalitzar-se (per exemple, es pot dir el nom que tindran les classes generades) amb anotacions a l'esquema XML mateix o a un document diferent que el compilador haurà de llegir.

- Generació d'objectes. L'operació d'unmarshal s'utilitza a l'hora de construir l'arbre (arbre doncs un XML segueix un tipus d'estructura jeràrquica) d'objectes que representa el contingut d'un document XML. Es creen doncs les instàncies de les classes definides anteriorment i que representen els elements de l' XML. Llògicament, es necessita un document XML d'entrada que serà la font d'informació a l'hora de construir els objectes. Donat que l'estructura de classes ja existeix, es possible també crear objectes des de zero sense necessitat de tenir que processar cap XML per a la creació dels objectes.
- Generació de l'XML: Una vegada hem fet les operacions necessàries amb els objectes, podem crear un document XML partint d'aquests arbre d'objectes: és l'operació d'unmarshal, l'inversa de l'operació marshal. Aquesta es l'operació que s'utilitzarà al cercador una vegada tenim l' informació final de disponibilitat i preus.
- Validació de les dades: Es possible validar si els objectes creats a partir d'un XML compleixen (son vàlids) les regles de l'esquema definit. En funció dels tipus d'errors trobats en la validació de l' XML, durant la creació dels objectes poden reportar-se els errors o fins i tot pot fallar la creació dels mateixos. També és possible validar un arbre d'objectes abans de serialitzar-lo a un fitxer XML. No obstant, a diferència de l'objecte *Unmarshaller* creat a la construcció de l'arbre d'objectes, l'objecte *Marshaller* no té cap mètode "*setValidating*" per a indicar-ho: s'ha de crear un nou objecte (*Validator*) que amb l'operació "validate" valida l'arbre d'objectes. En tots dos casos (marshal i unmarshal) l'operació de validació es una operació opcional.

Totes aquestes operacions es poden veure gràficament a la següent figura, on només la fletxa etiquetada amb l'operació "marshal" representa l'operació que s'utilitzarà al projecte.

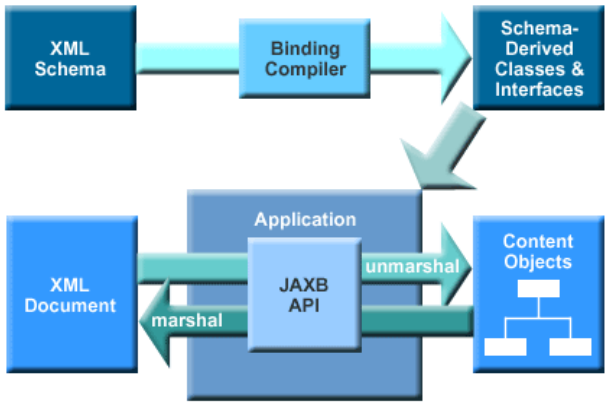


Figura 16 *JAXB*

L'API JAXB està formada per un conjunt de classes que son les que s'utilitzen per a les operacions anteriors. Totes aquestes classes es troben al package ***javax.xml.bind***.

1.4.3 Mapeig de les classes a l' XML

Amb JAXB, amb poques línies de codi es pot generar un fitxer XML partint d'un objecte Java. S'ha creat una nova classe que s'utilitzarà per a instanciar aquest objecte. L'objecte serveix d'arrel a la creació de l'arbre XML i conté totes les dades útils que s'han trobat als operadors. Com l'objectiu fonamental de l' XML es tenir un document físic que permeti posteriorment fer una reserva a partir de una de les combinacions d'hotel+vol seleccionades, es importantíssim emmagatzemar al fitxer les dades que es varen utilitzar per a cercar l' informació de disponibilitat i preus. Per a aquest propòsit s'ha afegit la propietat "*beanPeticioConsulta*" a aquest objecte que conté l' informació del filtre de cerca de disponibilitat i preus. La classe definida s'anomena "ResultBean" i el codi es el següent:

```
package pfc.bean;

import java.util.Vector;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;

import pfc.BeanPeticioConsulta;

@XmlRootElement(name="resultat")
public class ResultatBean {

	@XmlElement(name="peticio")
	private BeanPeticioConsulta beanPeticioConsulta;

	@XmlElementWrapper(name = "combinacions")
	@XmlElement(name="combinacio")
	private Vector<CombinacioBean> vCombinacions;

	public ResultatBean() { }

	public void setBeanPeticioConsulta(
	BeanPeticioConsulta beanPeticioConsulta) {
		this.beanPeticioConsulta = beanPeticioConsulta;
	}

	public void setvCombinacions(Vector<CombinacioBean> vCombinacions){
		this.vCombinacions = vCombinacions;
	}
}
```

S'han utilitzat **anotacions** (es poden veure varis exemples d'anotacions al codi anterior de la classe ResultBean) dins el codi de les classes dels objectes utilitzats a la generació de l' XML que permeten personalitzar certs comportaments per defecte del marshalling, és a dir, customitzar l'estructura de l' XML generat per defecte. Totes les anotacions formen part de l'API JAXB i més concretament del package *javax.xml.bind.annotation*. El conjunt d'anotacions que s'han utilitzat son les següents:

`@XmlRootElement`

L'anotació més senzilla i l'única que és obligatòria. Especifica la classe que serà l'element arrel del document XML. En aquest cas, i com es veu al codi anterior, l'objecte ResultBean es convertirà a l'arrel del document XML. Es fa servir l'atribut *name* de l'anotació per a especificar el nom del tag XML que es farà servir. Amb això sabem que l'XML generat tindrà una forma similar a la següent:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resultat>
</resultat>
```

`@XmlElementWrapper(name = "combinacions")`

Defineix un wrapper o embolcall. En aquest cas, s'utilitza per a que el tag "combinacions" serveixi d'envoltori dels diferents tags "combinacio". Es a dir, obtenim "`<combinacions><combinacio /><combinacio /></combinacions>`" enlloc de "`<combinacio /><combinacio />`"

`@XmlElement`

S'utilitza per a indicar que un atribut o un mètode d'un bean es transformarà a un element d'un XML. L'atribut *name* d'aquesta anotació indica el nom de l'element de l' XML. Per exemple,

```
@XmlElement(name= "preu")
public String getFormatPreu() {
```

Es converteix a l'element "<preu>0</preu>". Si no s'especifica cap anotació d'aquest tipus, els atributs públics dels objectes inclosos a la serialització s'inclouen com a elements de l' XML sempre i quan tinguin valors no nuls i, o be siguin públics, o be siguin privats i tinguin els getters/setters respectius.

`@XmlTransient`

S'utilitza aquesta anotació aplicada a la classe abstracta ItemBean, classe pare d'HotelBean i FlightBean. No es vol tenir aquesta classe abstracta en compte a l'hora de crear l' XML. Es volen tenir en compte tractaments d'aquesta classe per defecte que estan definits a les classes HotelBean i FlightBean com per exemple, l'ordenació d'elements (veure anotació XmlType)

`@XmlType`

S'utilitza a l'hora de definir l'ordre entre els diferents atributs interns d'un objecte. L'ordre ve definit per la propietat "propOrder". Per exemple,

```
@XmlType(propOrder = {"formatPreu","hotelBean","flightBean"})
```

ens diu que dins l'element "combinacio" (l'anotació està definida dins la classe CombinacioBean, que té el nom "combinacio" a l'XML doncs s'ha utilitzat l'anotació XmlElement), el primer element es "formatPreu", el segon "hotelBean" i el tercer "flightBean". Amb això es substitueix l'ordre per defecte.

A part de definir les anotacions (l'única realment obligatòria es `@XmlRootElement`), JAXB exigeix que les classes dels objectes involucrats a la serialització tinguin definit obligatòriament un constructor per defecte (sense cap argument).

Tenint clar l'estructura d'objectes que volem mapejar a l'XML i una vegada s'han aplicat les anotacions als elements (classes/atributs/mètodes) i s'han definit els constructors per defecte a totes les classes involucrades al procés de serialització, només cal executar l'operació marshal del nostre objecte Marshaller. A continuació hi ha el codi que a la solució realitza aquesta tasca:

```
JAXBContext contextObj = JAXBContext.newInstance(
   pfc.bean.ResultatBean.class,pfc.bean.hotel.HotelBean.class,
   pfc.bean.flight.FlightBean.class);

Marshaller marshallerObj = contextObj.createMarshaller();
```

```
marshallerObj.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

StringWriter oWriter = new StringWriter();
marshallerObj.marshal(resultatBean, oWriter);

return oWriter.toString();
```

Com es veu, primer es crea un objecte JAXBContext on s'especifica el conjunt de classes que es mapejen a l' XML. Les classes son ResultBean, HotelBean i FlightBean. Com s'ha aplicat l'anotació XmlTransient a ItemBean, s'han d'especificar les dues classes filles HotelBean i FlightBean, no així totes les filles d'aquestes dues que ja estan representades per les classes pare. El no tenir que afegir les diferents classes relatives als ítems dels diferents operadors (pfc.bean.hotel.HotelopiaBean,pfc.bean.flight.RumboBean,etc) facilita l'adaptació de nous operadors al projecte.

Seguidament es crea un objecte Marshaller i se li dóna el valor de "true" a la seva propietat "Marshaller.JAXB_FORMATTED_OUTPUT" que indica que es mantinguin identacions al fitxer XML. Aquest objecte té altres propietats, com per exemple la codificació del fitxer XML generat, que es mantenen als seus valors per defecte. En el cas de la codificació del fitxer, s'utilitza codificació UTF-8.

Al codi, a continuació es crea l'objecte ResultBean que conté les diferents combinacions d'hotel+vol i la petició que fa al cercador. Amb aquest objecte, que representa l'arrel de l' XML i un segon paràmetre que indica el destí de la sortida (on s'emmagatzema l' XML) es crida l'operació marshal de l'objecte Marshaller que transforma l'arbre d'objectes a un XML que en aquest cas s'escriu al Writer StringWriter (flux de caràcters que amb facilitat es pot convertir a una variable String).

A l'exemple el resultat s'emmagatzema a un objecte de tipus Writer, però existeixen vàries signatures del mètode que permeten escriure la sortida a altres objectes, com per exemple nodes DOM o fitxers de text. Com es necessita el contingut de l' XML per a un processament posterior (explicat al capítol posterior), l' XML queda emmagatzemat a una variable que, posteriorment s'escriu també a un fitxer físic de cara a la possibilitat de realitzar posteriors formalitzacions de reserves, i complint amb els requisits de les especificacions de VTC.

1.5 Capítol 5. Creació de l' HTML.

1.5.1 Introducció

Necessitem mostrar a l'usuari l' informació resultat de la consulta de preus i disponibilitat i aquesta informació s'ha de mostrar en format HTML doncs l'accés a l'aplicació es fa amb un navegador web. Es a dir, l'usuari feta la consulta de disponibilitat i preus, visualitzarà una plana HTML on sortirà el resultat de la consulta demanda, que poden ser les combinacions trobades de vol + hotel (recordem que el màxim de combinacions que es mostren son cinc) o be un missatge d'error relacionat amb els paràmetres de cerca (per exemple, la data de fi es anterior a la data inicial), un error fatal en l'execució del procés de cerca o simplement un missatge de que no s'han trobat resultats per al filtre de cerca seleccionat.

L' HTML que finalment mostrarà el navegador de l'usuari que ha fet la cerca es pot generat de diverses maneres, entre d'altres:

- Dins l'estructura de classes amb arrel la classe pfc.bean.ItemBean, es poden crear mètodes que retornin l' HTML que mostra cada ítem d'informació: l' HTML per a un vol i l' HTML per a una estància determinada en un hotel. Per a les diferents combinacions (instàncies de pfc.bean.CombinacioBean) s'hauria de fer lo mateix, retornant l' HTML unificat dels ítems vol i hotel. Finalment, cridant a aquests mètodes per a totes les combinacions i concatenant els valor retornats a una variable que emmagatzemés altra informació addicional es pot generar l' HTML final. Aquesta es una solució bastant manual i bastant simple, donat que ja existeixen mètodes utilitzat a tasques de debug que retornen informació en format HTML per a cada ítem.
- L' HTML es pot generar un llenguatge de servidor com JSP. Es construeix una plana JSP que genera una part d' HTML estàtic i altra d' HTML dinàmic generat amb l' informació de les diferents combinacions. Per exemple, es pot tenir una instància de la classe ResultBean (la classe del projecte pfc.bean.ResultBean) i, amb un simple bucle Java iterar el seu vector de resultats (cada resultat es una combinació d'hotel i vol) i anar generant l' HTML de sortida amb l' informació de cada resultat. Això es una solució possible, doncs òbviament aquesta solució admet moltes variacions depenent de frameworks, decisions de disseny, etc. L'objecte iterat amb els resultats pot ser un atribut enviat al request i que es captura a la JSP.
- L' HTML es pot generar amb un llenguatge de programació de la part client. Com a exemple d'aquesta solució, el navegador carrega la plana HTML la qual executa una crida AJAX al servidor web que retorna l' informació de preus i disponibilitat en un format de dades definit (pot generar l' HTML directament o retornar l' informació en format XML, Json, text pla, etc). El client rep la resposta a la crida Ajax, la llegueix i la mostra mitjançant la modificació de l' HTML amb un llenguatge de programació com pot ser Javascript. Aquesta solució requereix de l'activació de javascript al navegador de l'usuari i és un poc més lenta doncs deixa la plana en "stand by" durant els temps de presentació del resultat i requereix de dues crides –la primera per a fer la cerca i la segona per a obtenir els resultats de la cerca en un format analitzable amb javascript- per a obtenir un codi HTML l' informació del qual ja es té amb la primera crida.
- L' HTML es pot generar com a resultat de l'aplicació d'una transformació XSLT al fitxer XML generat amb anterioritat. Aquest HTML es retorna al navegador de l'usuari que ho visualitza sense fer cap tipus de tractament. Aquesta és una solució que abstreu el codi de la part de presentació de l' informació i ens permet introduir a aquest projecte una nova API que es tracta a continuació: JAXP.

1.5.2 JAXP

JAXP (*Java API for XML Processing*) es una API de Java que facilita el tractament i manipulació d'arxius XML. Aquesta API serveix d'interfície per a un conjunt d'operacions freqüents de tractament de documents XML:

- Accés a analitzadors sintàctics XML. Un analitzador sintàctic de XML és un programa que aplica l'anàlisi lèxica i sintàctica a un document XML i que ens permet, entre d'altres, comprovar si un document XML està ben format (és un document XML), és vàlid (compleix amb la seva especificació DTD/schema) o consultar els valors de determinats elements. Existeixen dos tipus principals d'analitzadors sintàctics XML: Els basats en SAX (Simple API for Xml) i els basats en DOM (Document Object Model), cadascun amb les seves pròpies característiques. Dins l'API de Java, el package *javax.xml.parsers* conté les classes relacionades amb l'instanciació i utilització d'analitzadors sintàctics.
- Processadors XSLT. Un processador XSLT (*eXtensible Stylesheet Language for Transformations*) és un programa que rep com a entrada un document XML i una plantilla XSL i, aplicant una transformació XSL al document XML genera un nou document. XSL es una família de llenguatges que inclou tres especificacions diferents i alhora relacionades: XSLT, XSL-FO i Xpath. Dins l'API de Java, el package *javax.xml.transform* conté les classes relacionades amb l'instanciació i utilització de processadors XSLT.

A aquest projecte no s'instancia explícitament cap analitzador sintàctic, però si que s'utilitza un processador XSLT per a generar un document HTML de sortida (el qual internament si que utilitza un analitzador sintàctic).

En resum, JAXP serveix d'interfície amb qualsevol analitzador XML i processador XSLT, permetent que sigui l'usuari el que triï el fabricant de l' implementació per a cada cas concret. JAXP facilita el canvi de proveïdor doncs es pot susbstituir l'implementació d'un analitzador sintàctic XML per altre sense afectar el codi font de l'aplicació. Una altre avantatge de JAXP es que per a la seva utilització no requereix de l' instal·lació ni configuració especial de cap element addicional, ve completament integrada amb la versió de Java instal·lada al projecte.

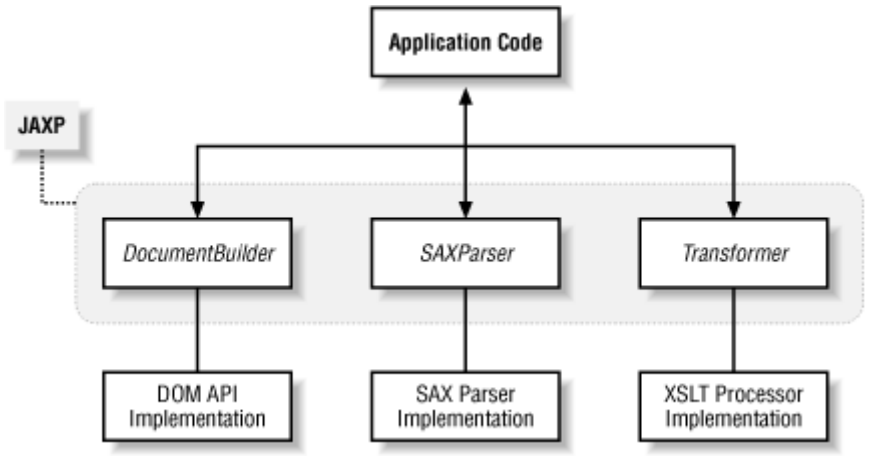


Figura 17 *JAXP*

1.5.3 Transformació de l' XML

Si volem transformar un document XML d'entrada en un altre tipus document de sortida amb l'API JAXP, ens haurem d'ajudar de l'especificació XSL. XSL (eXtensible Stylesheet Language) és una família d'especificacions que permeten descriure les transformacions que s'han d'aplicar a un document XML per a generar altre document de sortida, que pot ser un document de text, altre document XML, un document HTML, etc. XSL està format per tres especificacions que a vegades també reben el nom de XSL. Son les següents:

- XSL-FO (*XSL Formatting Objects*): Especificació basada en XML que permet descriure el format visual amb el que es vol representar un document XML. A un document XSL-FO l'unitat bàsica per a presentar l'informació es el "formatting object": planes, taules, regions, paràgrafs, etc. L' utilització mes comuna de XSL-FO es dóna en la creació de documents PDF.
- XSLT (*XSL Transformations*): Defineix una sintaxi per a transformar un document XML d'entrada en un altre document de sortida: és un llenguatge de transformació basat en XML que ajuda a especificar les transformacions que s'han de donar al document XML. Una fulla XSLT és un document de text escrit en el llenguatge XSLT que conté les transformacions que s'han d'aplicar al document XML d'entrada per a generar el document de sortida.
- Xpath (*XML Path Language*) és l'especificació que defineix la sintaxi per a accedir al document XML. XPath es un llenguatge no basat en XML i que utilitzarem conjuntament amb XSLT primer per a accedir als elements de l' XML i després per a aplicar les transformacions.

De les tres especificacions anteriors, farem servir les dues darreres: XSLT i Xpath. El mode de funcionament es el següent: es llegeixen i es carreguen en memòria els documents (dades d'entrada) XML i XSLT. La fulla XSLT serà única per a tots els XML doncs el document de sortida tindrà una estructura similar en tots els casos i el document XML contindrà les dades relatives a la cerca i resultat de disponibilitat i preus. Tindrem un programa (processador XSLT) encarregat d'aplicar les transformacions XSL definides a la plantilla XSLT al document XML i generar un document de sortida, que en aquest cas serà el document HTML. Tot aquest procés es pot veure gràficament a la següent figura:

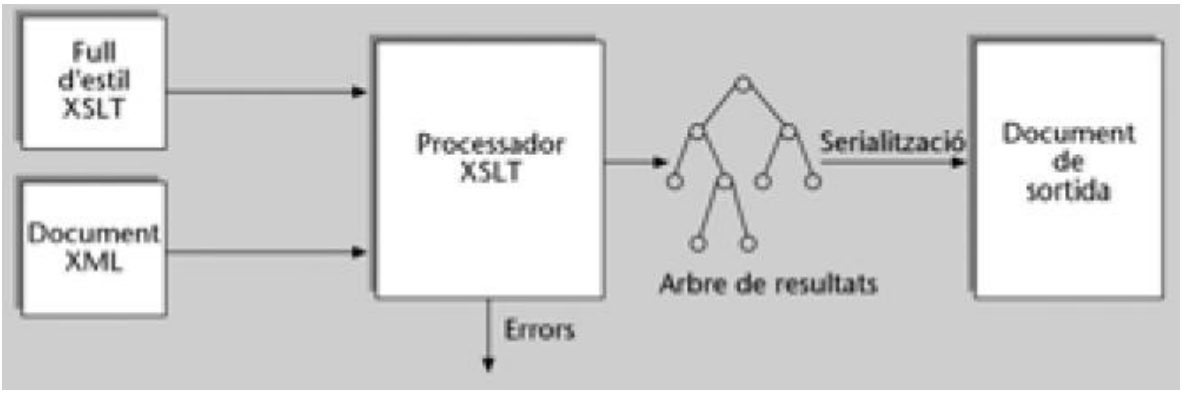


Figura 18 *Processador XSLT*

Amb JAXP podem escollir el processador XSLT que utilitzarem, i això serà independent del nostre codi, que serà el mateix si decidim canviar de processador XSLT. Dos dels processadors més coneguts dins el llenguatge Java son Xalan (del projecte Apache) i Saxon (de la companyia Saxonica). Xalan té l'avantatge de que ja ve integrat amb l'API de Java i no requereix de cap configuració especial. D'altra banda, s'han realitzat proves amb els dos processadors i no s'han detectat diferències importants en quan a temps d'execució, que en aquest cas no es massa llarg donat que els arxius XMLs que es processen i que contenen les dades no son massa grans.

El codi que al projecte executa el processador XSLT es el següent:

```
private String generarHTML(String xml)
throws TransformerFactoryConfigurationError, TransformerException,
FileNotFoundException
{
      StringReader oReader = new StringReader(xml);
      StringWriter oWriter = new StringWriter();
      StreamSource streamSource = new StreamSource(oReader);
      StreamResult streamResult = new StreamResult(oWriter);

      TransformerFactory transformerFactory =
            TransformerFactory.newInstance();
      Transformer transformer = transformerFactory.newTransformer(
            new StreamSource(new File("plantilla.xslt")));
      transformer.transform(streamSource, streamResult);

      return oWriter.toString();
}
```

Es fan unes observacions del codi anterior:

- El document (contingut) XML es rep a l'entrada del mètode dins una variable String
- El resultat de la transformació s'escriu a una variable tipus StringWriter que implementa un flux de caràcters que finalment es converteix a una variable tipus String (*oWriter.toString()*)
- S'utilitza un objecte TransformerFactory per a obtenir l' implementació factoria del processador XSLT. La crida *TransformerFactory.newInstance()* obté una instància que s'utilitza per a crear posteriorment l'objecte Transformer (la classe TrasnformerFactory es una classe abstracta que impedeix instanciar-la. El mètode newInstance crea l'instancia de l' implementació del processador XSLT que volem utilitzar).
- Es crea l'objecte Transformer que rep com a paràmetre la plantilla XSLT que s'utilitza en la transformació (`transformerFactory.newTransformer(new StreamSource(new File("plantilla.xslt")));`)
- Finalment, es realitza la transformació XSLT indicant el document XML i el fluxe de sortida en el qual s'escriu el resultat de la transformació (`transformer.transform(streamSource, streamResult);`)

En relació a l' implementació del processador XSLT, no fa falta cap configuració especial per a utilitzar aquest processador XSLT doncs el jar que conté les classes d'aquest processador ja està inclòs al path del sistema i es per tant l' implementació per defecte que s'utilitza. Si es volgués utilitzar el processador XSLT Saxon (o altre), es tenen vàries opcions:

- No modificar el nostre codi que crida al processador XSLT
    - L'opció més comuna es modificar la propietat de sistema *javax.xml.transform.TransformerFactory*, indicant el path de la classe que implementa l'interfície *javax.xml.transform*. Aquesta propietat es pot indicar de vàries maneres: Pot ser un paràmetre opcional més a la crida de la màquina virtual, pot modificar-se en temps d'execució a la nostra aplicació (*System.setProperty*), pot indicar-se al fitxer de configuració jaxp.properties dins la carpeta lib del directori arrel de la màquina virtual, un fitxer de configuració al directori META-INF/services, etc.
    - Existeixen altres alternatives que no requereixen de la modificació de cap propietat de sistema. Per exemple, a l'execució d'una aplicació Java arrencada directament amb l'IDE Eclipse, els JARs importats a nivell de projecte tenen preferència per sobre dels

inclosos al path del sistema. Es a dir, que si existeix un processador la llibreria (jar) del qual s'ha importat al nostre projecte, aquest processador serà el processador XSLT que s'utilitzarà per a transformar els documents XML. Als servidors d'aplicacions es té una situació molt similar: les llibreries definides a una aplicació tenen preferència en relació a les llibreries que afecten a la totalitat d'aplicacions desplegades al servidor. En resum, si no existeix cap configuració per software es té en compte l'ordre en que s'han carregat les classes de les implementacions dels processadors XSLT (els jar's). Aquesta es l'opció utilitzada a aquest projecte (utilitzar l' implementació el jar de la qual s'ha carregat en primer lloc).

- Modificar el codi que crida al processador XSLT
    - Aquesta opció té l'inconvenient d'anular el principal avantatge de JAXP: ara el codi no es independent del proveïdor del processador XSLT. Per exemple, podem fer us del processador XSLT Saxon i, per qualque motiu, volem utilitzar el processador Xalan a qualque part de la nostra aplicació. Si la màquina virtual s'ha executat amb la variable de sistema *javax.xml.transform.TransformerFactory* amb valor l' implementació del processador Saxon (java -Djavax.xml.transform.TransformerFactory=net.sf.saxon.TransformerFactoryImpl) per a utilitzar l' implementació proporcionada per Xalan, podem fer la següent modificació al codi:

```
TransformerFactory transformerFactory =
com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl.newIn
stance();
```

Dèiem abans que a la lògica que realitza la transformació XSLT es carrega en memòria el full XSLT el qual es analitzat amb una analitzador sintàctic XML que construeix la seva representació DOM (Document Object Model) que facilita el posterior accés a les instruccions de transformació. Hom pot pensar que aquesta acció introdueix un overhead innecessari a l'aplicació, i es així. Mentre el document XML varia per cada cerca realitzada, la plantilla XSLT es la mateixa sempre, per a totes les cerques de disponibilitat i preus. Es pot fer el parseig de l' XSLT una única vegada i aprofitar la representació en memòria de l' XSLT per a fer totes les transformacions XSLT que es fan a les successives peticions del cercador. Per fer això, JAXP defineix l' interfície *java.xml.transform.Templates*. La solució consisteix a crear un únic objecte templates, que a efectes d'implementació es una variable estàtica de la classe pfc.Cercador, que s'utilitza cada vegada que es vol crear una nova instància de la classe *java.xml.transform.Transformer*, ara amb l'avantatge que el constructor d'aquesta classe Transformer no construeix l'estructura DOM de la fulla XSLT doncs esta ja ha estat creada amb la creació de l'objecte templates.

El codi es el següent. Primer, es crea l'objecte templates:

```
templatesXSLT = TransformerFactory.newInstance().newTemplates(new
StreamSource(new File("sol.xslt")));
```

Després, a cada execució de la transformació XSLT s'utilitza aquest objecte templates:

```
private String generarHTML(String xml)
throws TransformerFactoryConfigurationError, TransformerException,
FileNotFoundException
{
	StringReader oReader = new StringReader(xml);
	StringWriter oWriter = new StringWriter();
	StreamSource streamSource = new StreamSource(oReader);
	StreamResult streamResult = new StreamResult(oWriter);

	Transformer transformer = templatesXSLT.newTransformer();
	transformer.transform(streamSource, streamResult);

	return oWriter.toString();
}
```

L'avantatge es més clar i mes profitós a aplicacions web on existeixen moltes peticions web, i per tant moltes transformacions XSLT al llarg del temps. A una aplicació web es donen accessos concurrents i és en aquest context on la seguretat en l'accés concurrent als recursos té força importància. Tenim una instància templates compartida per tots els "threads" de l'aplicació i moltes instàncies "transformers", concretament una per cada thread o petició al cercador. El processador XSLT manté certa informació d'estat específica a cada transformació i, de fet, ja l'API de Java ens informa de la no recomanació d'utilitzar objectes Transformers amb accesos concurrents (*An object of this class may not be used in multiple threads running concurrently. Different Transformers may be used concurrently by different threads*). D'altra banda, l'API ens informa que la classe Templates ha de ser segura davant múltiples accesos (*Templates must be threadsafe for a given instance over multiple threads running concurrently, and may be used multiple times in a given session*). Això queda garantit pel fet que la creació de l'objectes templates només es accessible per un thread, doncs el mètode que l'instancia (el constructor de la classe pfc.Cercador) té el modificador *synchronized* (només accedeix un thread).

1.5.4 Plantilla XSLT

S'ha d'escriure la plantilla XSLT (serà un arxiu de nom resultat.xslt) amb el codi de les transformacions necessàries per a generar l' HTML de presentació.

Es defineix un template (plantilla) per l'arrel del document HTML, que és la primera regla que executarà el processador XSLT:

```
<xsl:template match="/">
</xsl:template>
```

Dins aquest template hi ha que destacar dos elements: l' iterador que itera les descripcions d'habitacions, dada que es mostra dins el resum dels paràmetres enviats al cercador (dades, ciutats, habitacions, etc) i la crida (inclusió) del template que itera les diferents combinacions. En relació a l' iterador, hi ha que destacar la funció "current()", que retorna l'element (node) actual de l' iteració, essent `select="current()"` equivalent a `current="."`:

```
<xsl:for-each
	select="//resultat/peticio/txtDistribucions/txtDistribucio">
	<div><xsl:value-of select="current()" /></div>
</xsl:for-each>
```

Com dèiem, s'inclou un template que s'activa amb tots els elements "combinacio":

```
<xsl:template match="/">
```

```
        <!—(...) -->
        <xsl:apply-templates
    select="//resultat/combinacions/combinacio"/>
    </xsl:template>
    <!—(...) -->
    <xsl:template match="combinacio">
    </xsl:template>
```

Totes les combinacions es mostren de la mateixa manera, on les dades de cada combinació es troben agrupades dins un element HTML fieldset. Una combinació es dibuixa com una taula on tenim columnes principals amb dades: la primera conté el preu de la combinació, la segona el preu de l'hotel, la tercera el preu del vol i la darrera un botó que simula la continuació del procés de reserva. Junt amb els preus de l'hotel i el vol tenim certa informació addicional dins una capa oculta (element div d' HTML) que es visualitza per javascript. S'utilitzen per això identificadors que depenen de la posició iterada, dins el conjunt de les combinacions. Per a obtenir l'índex d'iteració actual s'utilitza la funció position(). Per exemple:

```
<xsl:attribute  name="id">dDetailHotel_<xsl:value-of  select="position()"
/></xsl:attribute>
```

Aquesta instrucció assigna a l'atribut **id** de l'element HTML precedent el valor resultant d'avaluar el contingut del tag `xsl:attribute`. Com es pot veure, s'utilitza la funció `position()` que retorna l'índex de l'iteració.

El segment d' HTML amb informació addicional de l'estància a l'hotel o del vol s'inclou amb les següents instruccions:

```
<xsl:value-of select="hotel/html" disable-output-escaping="yes" />
<xsl:value-of select="vol/html" disable-output-escaping="yes" />
```

L'atribut disable-output-escaping permet codificar els elements '<' i '>' com a '<' i '>'. Amb valor "no", al document HTML s'imprimirien els literals exactes '<' i '>' i l' HTML resultat no es visualitzaria correctament.

Un dels darrers elements que s'inclouen a la plantilla XSLT és una estructura condicional:

```
<xsl:if test="position() &lt; last()"><br /></xsl:if>
```

Amb aquesta instrucció, es retorna l'element "
" (salt de línia HTML) si es compleix la condició de l'atribut "test", es a dir, si l' iteració actual no es la darrera: "position() < last()". Com es veu, el caràcter especial "<" s'escapa a la condició (test) per la seva representació "&t;".

1.6 Capítol 6. Interfície web.

1.6.1 Introducció

El darrer pas de tots, una vegada implementats els diferents mòduls del cercador, és la construcció de l'aplicació web. El desenvolupament dels passos anteriors (connexions HTTP, aplicació d'expressions regulars, construcció de l'arxiu XML, etc) és pot testejar a una aplicació Java executant-se a un entorn local (com una aplicació d'escriptori). Amb aquesta aplicació el desenvolupament es més ràpid doncs no requereix de l'execució de cap servidor d'aplicacions, ni del desplegament dels nous arxius al servidor (hi ha que tenir en compte també que a JBoss, el servidor d'aplicacions escollit, nous canvis a les classes java requereixen desplegar de nou l'aplicació).

El pas d'aplicació d'escriptori a aplicació web no es massa traumàtic però tampoc és un simple "copy & paste" d'arxius entre directoris: S'ha de definir l'arquitectura de l'aplicació web, conèixer l'estructura d'arxius que formaran part de l'aplicació, crear aquesta estructura i, finalment, provar el correcte flux d'execució de l'aplicació per a diferents dades d'entrada. Tot això queda detallat als següents subapartats.

1.6.2 Arquitectura

S'ha decidit utilitzat el patró de disseny MVC (*Model View Controller*) per a dissenyar l'aplicació web. Aquest patró és àmpliament utilitzat em l'àmbit de les aplicacions web i està recolzat per multitud de frameworks que donen suport a les diferents capes que defineix aquests patró (Hibernate i Struts en son un exemple d'aquests frameworks). Aquest model ens assegura una separació entre el model (les dades), la vista (la presentació) i el control de l'aplicació. L'arquitectura de l'aplicació web està basada en XML i segueix el patró MVC esmentat anteriorment. Gràficament podem veure aquesta arquitectura a la figura següent:

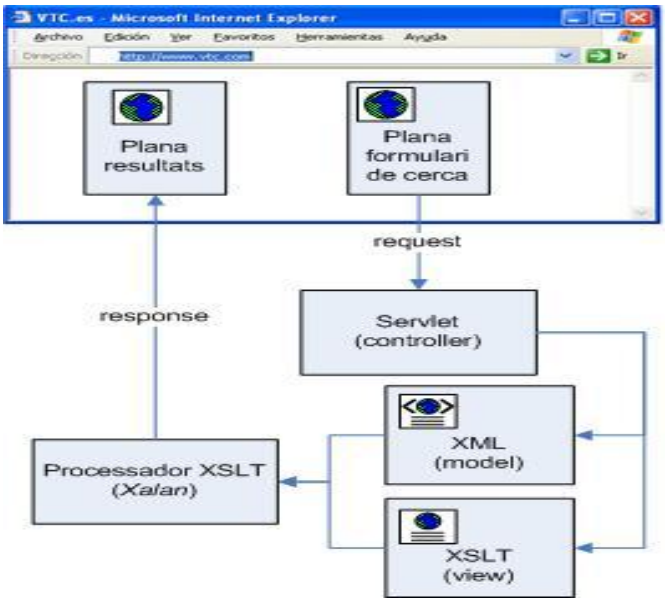


Figura 19 *Arquitectura web*

A aquesta arquitectura, el document XML representa el model (les dades del resultat de la cerca) i el full XSLT la vista doncs conté el codi HTML (la presentació) i les transformacions que s'apliquen a les dades.

Fetes aquestes consideracions, definim dos components web:

- Servlet (*pfc.Pfc*). Un servlet es un component web que actua de controlador dins l'arquitectura MVC (s'encarrega del flux de l'aplicació: rep totes les peticions i determina quina acció s'ha de realitzar a continuació. Finalment determina quina es la següent vista). Ho podem veure com una miniaplicació de servidor que rep les peticions HTTP i genera un contingut com a resposta a les peticions. Al nostre cas, el servlet dóna resposta a les peticions POST: rep els paràmetres del formulari de cerca, fa les crides a la capa de negoci i genera el contingut de la resposta.
- JSP *index.jsp*. La JSP (Java Server Page) index.jsp implementa la vista d'entrada a l'aplicació. Es un arxiu amb extensió JSP i escrit amb codi Java i HTML i que inclou l' informació d'entrada a l'aplicació. La plana d'entrada es un formulari amb els diferents camps que permeten fer la cerca de disponibilitat i preus: ciutat origen, ciutat destí, data d'inici, data de fi, nombre d'habitacions i, per cada habitació, nombre d'ocupants i edats dels nins. El servlet redirigeix a aquesta vista en cas de que es produeixi qualque error (per exemple, si no es selecciona el nombre d'habitacions). El codi Java que inclou aquesta JSP en forma d' scriptlet (fragment de codi Java entre etiquetes <% i %>, incrustat a la pàgina JSP i entremesclat amb el codi HTML) realitza les següents funcions:
  - Recupera un error d'execució en cas de venir d'una consulta de preus (El servlet retorna l'error dins l'objecte request (`-request.setAttribute("info_error", error);-`). Aquest error, si existeix, es visualitza per pantalla.
  - Inicialitza els camps del formulari amb els valors d'una consulta anterior. En cas d'error, a part de mostrar el missatge d'error, es preseleccionen els camps del formulari amb els valors seleccionats al moment de la cerca, per evitar la nova selecció de tots els camps.
  - Dóna valors als selectors de dia i mes

Definits tots els components web de l'aplicació, aquests s'assemblen junt a la resta d'arxius a un únic arxiu comprimit que es desplega (s'instal·la) al servidor d'aplicacions (El servidor d'aplicacions J2EE escollit es JBoss). Aquest arxiu comprimit es un fitxer .war (Web ARchive) que conté tots els arxius necessaris per l'aplicació, entre ells el fitxer web.xml que configura el servlet. El fet de tenir un únic arxiu amb tots els fitxers facilita la portabilitat de l'aplicació entre servidors d'aplicacions sense requerir cap modificació especial. L'estructura d'arxius i directoris d'aquest war segueix una especificació estàndard.

1.6.3 Construcció de l' interfície

S'ha creat un fitxer XML, build.xml, que conté totes les instruccions que defineixen les tasques necessàries per a construir l'arxiu **vtc.war,** que es l'arxiu de l'aplicació web. Aquest arxiu build.xml es l'arxiu que Ant llegueix i executa produint com a resultat el fitxer war. Ant es una eina de codi obert basada en XML i desenvolupada per Apache i que automatitza la construcció i compilació d'aplicacions Java.

Ant ve integrat amb l'IDE Eclipse, des del qual pot executar-se l'arxiu XML sense necessitat d'iniciar cap consola de línia de comandes i executar ant des de la consola. El fitxer build.xml defineix les tasques que Ant haurà d'executar amb l'objectiu final de crear el fitxer vtc.war. Una vegada compilades les classes, generats els jar's, i construït el war (que no deixa de ser un arxiu comprimit com ho es el .jar però amb una utilitat diferent), aquest es copia al directori deploy del servidor JBoss. A mode d'exemple, la tasca que dins l' XML realitza aquesta acció es la següent:

```
<target name="deploy" depends="war">
      <copy file="${build}/vtc.war" todir="${deploy}"/>
</target>
```

Una vegada creat, l'arxiu **vtc.war** té la següent estructura:
/index.jsp: Jsp del formulari de cerca (Home de l'aplicatiu)
/css: Fitxers CSS -style.css-
/js: Fitxers JS –hideView.js-
/META-INF: Directori creat per Ant durant la compilació i que inclou el manifest
/WEB-INF

/WEB-INF/lib: Llibreries que formen part de l'aplicació. Les llibreries externes son HttpClient i Log4j. També tenim la llibreria més important: vtc.jar, que inclou els compilats de les classes desenvolupades.
/WEB-INF/classes: Conté els arxius interns utilitzats per l'aplicació. En aquest cas, unicament la plantilla XSLT –*resultat.xslt*-. També pot contenir classes Java (.class), que en aquest cas son dins el jar vtc.jar
/WEB-INF/web.xml: Arxiu bàsic de configuració de l'aplicació (configura el servlet).
/WEB-INF/jboss-web.xml: Descriptor XML propi de JBoss.

Aquest arxiu s'instal·la copiant-lo al directori deploy del nostre servidor JBoss. Automàticament, JBoss ho detecta i ho desplega. Es pot veure a la sortida estàndard de l'aplicació (que s'imprimeix a l'arxiu de log server.log) el resultat del desplegament. Existeixen dos mètodes bàsics de desplegament d'aplicacions a JBoss: comprimit i desplegat, i en aquest cas s'utilitza el primer on tenim tota l'aplicació a un únic arxiu (vtc.war).

Finalitzant, els passos / detalls mes importants en la construcció de l'aplicació web (del conjunt d'arxius detallats a l'estructura anterior) son els següents:

Configuració del servlet. El principal (i únic) servlet de l'aplicació es declara i configura a l'arxiu web.xml. S'indica l' URL per a la qual el servlet rep peticions i el nom complet de la classe que estén la classe HttpServlet (pfc.Pfc).

Implementació del servlet pfc.Pfc. Aquest servlet executa la crida al cercador quan rep peticions HTTP de tipus POST (el mètode *doPost* rep els paràmetres de la cerca que s'envien al request –objecte de tipus *HttpServletRequest*-) i executa el cercador. En funció del resultat redirigeix a una vista o a l'altra. Si l'execució és correcta (no és produeix cap errada als paràmetres, a l'execució, es troben dades, etc), es retorna l' HTML resultat de la transformació XSLT que rep com a entrada la plantilla XSLT i el document XML generat amb les dades de preus i disponibilitat retornades pels operadors. Si l'execució finalitza amb qualque tipus d'error, el servlet (controlador) retorna a la vista home de l'aplicació (index.jsp), on es mostra un missatge de l'error produït.

Programació de les vistes. En aquesta aplicació el flux d'execució és molt senzill: només existeixen dos vistes: la que inclou el formulari de cerca (index.jsp) amb els camps necessaris per a iniciar la cerca i la que mostra el resultat de la cerca, que es mostra una vegada s'ha executat el cercador i ha retornat informació útil. Donat que aquesta darrera vista es genera en temps d'execució mitjançant la transformació XSLT, només tindrem un arxiu JSP: index.jsp. Aquest arxiu està format per codi HTML i codi Java incrustat (scriptlet). S'inclou codi en llenguatge de servidor per a certes tasques com son la generació de les dades HTML dels selectors de dates (partint de la data del sistema), la visualització del missatge d'error en cas de venir d'una execució anterior o la preselecció dels camps del formulari (dades, ciutats, nombre d'habitacions,...). Aquestes darreres dues operacions es

fan a partir d'atributs inclosos al request rebut per la plana. Una JSP es compila i s'executa al servidor com si fos un servlet i per tant podem enviar atributs al request des del servlet que la JSP rep i llegueix abans d'enviar l' HTML resultant al client (navegador) que va fer la petició. Tot aquest flux s'executa seqüencialment al servidor.

Adaptar les rutes dels fitxers interns de l'aplicació. Internament l'aplicació llegueix dos arxius de configuració (log4j.properties i config.properties). log4j.properties configura els logs en els quals escriu l'aplicació mitjançant la llibreria log4j d'Apache- i config.properties conté paràmetres de configuració com l'activació/desactivació de la cerca d'informació a determinats operadors. Dins el war generat, s'inclouen a l'arrel del jar que conté les classes del cercador: vtc.jar. Amb les següents tres línies es carrega un fitxer a memòria i es fa accessible amb un objecte tipus Properties:

```
InputStream inputStreamCONFIG = Thread.currentThread().
  getContextClassLoader().getResourceAsStream("fitxer.properties");
Properties perfilBundle = new Properties();
perfilBundle.load(inputStreamCONFIG);
```

Altre arxiu al qual s'accedeix internament és la plantilla XSLT (resultat.xslt). Aquest arxiu, al no tractar-se d'un fitxer properties es localitza a altre path dins el war: /WEB-INF/classes/resultat.xslt. El seu accés es fa al iniciar el cercador, a la creació de l'objecte *templatesXSLT* de tipus Templates. Finalment, durant l'execució del cercador es generen els arxius XML i, opcionalment (segons la configuració del fitxer config.properties), es generen arxius de debug amb dades de l'execució i es llegeixen arxius amb contingut HTML retornat per operadors a cerques anteriors – veure manuals d'instal·lació i execució. L'accés a aquests arxius es fa mitjançant un objecte java.io.File el qual rep una ruta absoluta de l'arxiu. El format d'aquestes rutes depèn de l'estructura de directoris de la màquina en la qual s'executa el servidor d'aplicacions –per exemple, pel cas del directori de localització dels fitxers XML generats pot ser "C:\vtc\fitxers\xml\" o "/home/adminuser/vtc/xml".

Amb totes aquestes modificacions, es pot generar i desplegar l'aplicació web al servidor JBoss.

Abans però de finalitzar aquest apartat, convé ressaltar una de les diferències més importants entre la programació d'una aplicació local d'escriptori (no es requisit de VTC, però també se'ns podria haver demanat crear un executable que s'instal·lés localment. En lloc d'interfície web es podria implementar una interfície gràfica amb l'API Swing nativa de Java) i la programació d'una aplicació web: a un entorn web podem tenir molts usuaris accedint concurrentment a l'aplicació, amb un entorn que es veu clarament afectat pel nombre d'accessos. Les necessitats dels usuaris des del punt de vista de la usabilitat del sistema son també diferents. En aquest sentit, s'ha volgut minimitzar el nombre d'accessos a fitxers que s'utilitzen a l'execució del cercador i s'han realitzat les inicialitzacions més importants a l' inici del servlet. Es disposa d'un únic objecte global a tota l'aplicació (s'utilitza el patró de disseny Singleton a fi de controlar que només hi hagi una única instància) el qual s'inicialitza al mètode *init* del Servlet. A aquest mètode init es realitzen altres accions com la inicialització del log4j:

```
public void init(ServletConfig config) throws ServletException {
  //...
  Cercador.getInstance();
}
```

També es realitzen certes tasques a la destrucció del servlet (quan per exemple, el servidor JBoss es para). Bàsicament s'alliberen els recursos i connexions HTTP establertes:

```
public void destroy() {
      Cercador.shutdown();
```

```
}
```

Tots els canvis introduïts i explicats a aquest apartat es refereixen fonamentalment a la capa de presentació i a la configuració de l'aplicació web al servidor d'aplicacions. La resta del codi (la part fonamental: les connexions HTTP, la transformació XSLT, l'aplicació d'expressions regulars, etc) és la mateixa i no té cap diferència en relació a si no s'hagués optat per l'opció de construir una aplicació web.

1.7 Conclusions

A Internet es troben disperses quantitats ingents de dades l'accés a la totalitat de les quals es fa impossible per a qualsevol ésser humà. Amb llibreries de Web Scraping es poden construir robots informàtics que de forma automàtica recopilen certa informació de determinats servidors web i l'analitzen segons uns determinats objectius. A aquest projecte s'ha construït un sistema que, utilitzant tècniques de Web Scraping, permet consultar informació de preus de vols i estàncies en hotels i presentar-la al usuari que ha fet la cerca.

A aquest document s'ha detallat la seqüència d'activitats que formen part del procés que dona solució a l'enunciat plantejat i que permeten finalment mostrar els resultats: s'inicien connexions HTTP contra els servidors web dels diferents operadors d'hotels i vols establint un seguit de crides fins a obtenir les dades de la consulta, s'extreuen les dades de preus (i altres) amb expressions regulars, es formen els objectes de dades definits i s'ordenen segons el seu preu. Finalment, es combinen els resultats d'hotels i vol fins a mostrar les cinc combinacions d'hotel i vol més econòmiques. També es crea un document XML amb les dades que identifiquen tant la petició de cerca (dades, ciutats, nombre d'habitacions, etc) com els resultats trobats. Junt amb una plantilla XSLT, aquest document serveix d'entrada a una transformació XSLT que genera l' HTML amb el resultat visual de la cerca que finalment es retorna al client que ha fet la petició.

Dins l'estructuració del codi del cercador, es distingeixen dues parts: una -la part més complexa i a la que s'ha dedicat més temps- depenent de cada operador escollit i que té codi molt especialitzat i molt subjecte a possibles variacions al codi HTML retornat pels servidors web dels operadors (les connexions HTTP junt amb les expressions regulars que analitzen el codi HTML concret de cada operador), i una altra que no depèn de cap operador en concret i que no es veu afectada pel fet de voler afegir, modificar o prescindir del codi i per tant de l' informació que aporten els operadors. Ens referim a la part que inicia els *threads* o fils d'execució encarregats d'iniciar la cerca als operadors i a la que posteriorment fa l'ordenació de preus, crea el document XML i realitza la transformació XSLT. Per a fomentar l' independència del codi d'uns operadors en relació als altres i per a facilitar una estructura comú de funcionament s'ha fet servir determinades característiques pròpies dels llenguatges orientats a objectes com per exemple l'herència de classes. En aquest sentit, el llenguatge de programació escollit (Java) ha facilitat molt aquesta tasca.

Tot el desenvolupament s'ha fet amb tecnologia J2EE, i amb ajuda de llibreries externes com HttpClient, log4j o Ant i altres APIs ja incloses a l'API de Java com JAXB o JAXP. El producte resultant es un fitxer .war que conté l'aplicació web amb totes les classes i arxius necessaris per a l'execució del cercador a un entorn web.

Aquest projecte ha sigut un bon exemple de l'ús de llibreries de Web Scraping que inclou en detall totes les operacions fins a obtenir per pantalla els resultats desitjats i que no s'ha centrat massa en la construcció de certs elements que només donen cert valor afegit al cercador com el disseny de l' interfície web.

GLOSSARI

Analitzador sintàctic XML: Programa que llegeix dades XML d'una font d'entrada i en fa l'anàlisi lèxica i sintàctica

API (*Application Programming Interface*): Conjunt de mètodes que serveixen d'interfície entre diferents productes software. Dóna accés a l'aplicació a un conjunt independent de serveis facilitant la modularitat del sistema.

Cookie: Petit arxiu de text emmagatzemat al node client i que conté informació de navegació a un determinat servidor. Es la solució més comuna al manteniment de l'estat de navegació (sessió) d'un usuari a un servidor web, fonamentalment degut a que HTTP és un protocol sense estat.

Document XML: Document escrit utilitzant el format XML. Consta d'una capçalera i de la instància del document (el contingut real del document, constituït per elements, atributs i dades). Tot document XML ha de seguir la sintaxi del llenguatge XML –per exemple, ha de poder ser representat en forma d'arbre amb una única arrel- (en cas contrari no és un document XML).

DOM (*Document Object Model*): API que implementa un analitzador sintàctic XML. Els analitzadors sintàctics basats en DOM retornen un arbre DOM a les aplicacions que els utilitzen. L'ús de DOM requereix una quantitat de memòria proporcional a la mida i a la complexitat del document XML.

DTD (*Document Type Definition*): Representació formal de les restriccions que ha de satisfer un document XML. Un document XML es vàlid si compleix amb les restriccions que marca el seu DTD associat. Un DTD és el conjunt de normes que s'han de seguir per a crear una representació d'un document XML d'un tipus determinat. Aquestes normes s'expressen mitjançant una notació anomenada declaració de marcatge.

Expressió regular: cadena de text que, fent ús de caràcters i metacaràcters, descriu un conjunt de cadenes a les quals representa. Diem que una expressió regular encaixa amb una cadena de text si aquesta forma part del llenguatge que defineix l'expressió.

HTML (*HyperText Markup Language*): Llenguatge de marques que descriu l'estructura i contingut d'una plana web. Es requisit bàsic i imprescindible de qualsevol navegador interpretar i visualitzar codi HTML.

HTTP (*Hypertext Transfer Protocol*): Protocol de xarxa emprat a la comunicació entre els diferents elements d'una arquitectura web com son els servidors, els clients, els proxies, els routers, etc. Es un protocol de petició/resposta que defineix la sintaxi i semàntica dels elements que actuen de client i envien les peticions i dels elements que actuen de servidors i reben les respostes.

Ítem: Dins aquest treball, ens referim per ítem a qualsevol resultat individual i que pot ser objecte de reserva dins un operador. Així, un ítem d'hotel es refereix a un resultat individual de la consulta feta a un operador d'hotels, amb els seus atributs: Nom de l'hotel, preu de l'estància, etc. D'altra banda, un ítem de vol es refereix a un resultat individual de la consulta feta a un operador de vols, amb els seus atributs: Aeroports d'origen/destí, hora d'anada, hora de tornada, etc. Existeixen molts pocs atributs compartits per al conjunt d'ítems, i la majoria son específics per cada operador (per exemple, no tots els ítems d'hotel tenen l'atribut "descripció").

JavaBean: Classes escrites amb Java i que segueixen una convenció concreta. Per exemple, un JavaBean és un objecte serialitzable (és pot convertir a una sèrie de bytes i emmagatzemar-me a un fitxer o també es pot transmetre per la xarxa) i que dona accés als seus atributs (propietats) a través de *getters* i *setters* d'aquests mateixos atributs.

JAXB: Una de les principals APIs que dins Java faciliten l'accés i tractament de documents XML. Proporciona una manera senzilla i transparent per a mapejar documents XML i objectes Java. Amb JAXB es pot crear una estructura de classes partint de l'esquema d'un document XML, es poden crear els objectes d'aquestes classes partint de documents XML i es poden generar documents XML a partir d'objectes Java.

JAXP: Una de les principals APIs que dins Java faciliten l'accés i tractament de documents XML. Ofereix una única manera de connectar una aplicació Java amb un analitzador sintàctic XML o amb un processador XSLT. Amb JAXP el programador es lliure d'escollir el proveïdor (l' implementació) de l'analitzador XML i del processador XSLT amb total transparència per al codi font de l'aplicació.

JSP (*Java Server Pages*): Document de text (amb extensió jsp) que s'executa com si fos un servlet però que facilita el disseny de l'aplicació doncs poden incloure codi HTML al seu contingut. Té el paper de component web a una aplicació web que empra el model MVC (és una vista).

MVC (*Model View Controller*): El model MVC és un patró de disseny àmpliament emprat a aplicacions web i permet estructurar-les en tres capes (model, vista i controlador) facilitant el desenvolupament i posterior extensió de l'aplicació. El model conté la part central de la funcionalitat de l'aplicació (la lògica de negoci) i la seva missió consisteix a mantenir les dades de l'aplicació. La vista ofereix la presentació del model, és a dir, decideix la forma en com es presenten les dades a l'usuari. La vista pot accedir a les dades del model pero no pot modificar-les. Finalment, el controlador és el component que reacciona a les entrades introduïdes per l'usuari.

Operador: Dins aquest treball, ens referim per operador a qualsevol plana web que dona serveis de consulta de preus d'estàncies a hotels o de vols. Un operador representa per al sistema una font de dades per a l'obtenció de preus. El nombre mínim d'operadors requerits per al funcioment del cercador són dos: un per a cerques d'hotels i altre per a cerques de vols.

Patró de disseny: Dins el desenvolupament de software, i més concretament els llenguatges de programació orientats a objectes, un patró descriu una solució comuna a un problema de disseny i que per tant pot ser reaprofitada en moltes situacions.

RFC (*Request For Comments*): Document el contingut del qual suposa la proposta d'un protocol de Internet. Les RFCs són publicades per l'organització IETF (*Internet Engineering Task Force*).

SAX (*Simple API for XML*): API que implementa un analitzador sintàctic XML. Es llegueix el document XML de manera seqüencial generant esdeveniments cada cop que s'identifica un element significatiu. SAX és un analitzador bastant ràpid i que a penes gasta memòria.

Scriptlet: Fragment de codi java encastat a una pàgina JSP i entremesclat amb codi HTML. Aquests fragments de codi solen anar dins els tags '<%' i '%>'

Servlet: Classe Java que estén la funcionalitat d'un servidor web. Dins el model MVC un servlet té el paper de controlador: rep paràmetres davant peticions HTTP (majoritàriament Get i Post) i determina quina és la següent vista que s'ha d'executar. Al servidor d'aplicacions és un component web que s'executa dins un contenidor de servlets.

URI (*Uniform Resource Identifier*): Cadena de caràcters que identifica un recurs a Internet. Facilita l' identificació i accés a aquests recursos amb els protocols que operen a Internet. Per exemple, el valor de l'atribut img d'una imatge HTML proveeix un URI.

Web Scraping: Tècnica utilitzada en l'extracció d'informació de llocs web. Bàsicament, un software que empra tècniques de Web Scraping realitza tècniques d'enginyeria inversa de l' informació de varis servidors web amb l'objectiu de filtrar certa informació útil per als seus objectius. Una eina que empra Web Scraping simula la navegació en determinats llocs web que faria un usuari obtenint informació desestructurada i donant-li format i significat a les dades llegides.

Web Semàntica: Web estesa on és possible compartir i processar informació de manera sencilla. L'ús de llenguatges ontològics i altres tecnologies que faciliten la definició de les dades de la web, faciliten l'accés a la mateixa per part d'eines automàtiques que tenen com un dels seus objectius la cerca d'informació.

W3C (*World Wide Web Consortium*): Organització (consorci) internacional que vetlla per l'estandardització dels formats de la xarxa. Una de les seves recomanacions més importants és HTML, de fet el creador original d'aquest llenguatge, Tim Berners-Lee és qui dirigeix aquest consorci.

XHTML (*eXtensible HyperText Markup Language*): Llenguatge de marques que estén HTML. Un document XHTML és un document XML ben format i per tant pot ser analitzat i tractat per eines de processament i tractament XML.

XML (*eXtensible Markup Language*): Llenguatge de marques que dota els documents d'una estructura lògica que facilita la seva manipulació i transmissió per la xarxa.

XML schema: Notació per a descriure la estructura i les restriccions dels continguts dels documents XML. Es notació XML i suporta diferents tipus de dades. Dona major potència expressiva que els DTD.

XSL (*eXtensible Stylesheet Language*): Especificació del W3C que té com a objectiu proporcionar un conjunt de regles i elements que permetin definir diferents presentacions per als documents XML. Un document XSL defineix la presentació que es vol donar a un document XML. El terme XSL es pot utilitzar per a referir-se a tres especificacions diferents: XSL-FO (XSL Formatting Objects), XSLT (XSL Transformations) i Xpath.

XSLT (*eXtensible Stylesheet Language Transformations*): Llenguatge basat en XML que permet definir un conjunt de regles per transformar els elements d'un document XML. Les eines que implementen aquest algorisme de transformació s'anomenen processadors XSLT. Per exemple, es pot generar un document HTML partint d'un document XML i de la plantilla XSLT que descriu les transformacions que s'han d'aplicar al document XML que conté les dades.

BIBLIOGRAFIA

Web Scraping

http://en.wikipedia.org/wiki/Web_scraping
http://rosettacode.org/wiki/Web_scraping
http://www.readwriteweb.com/archives/web_30_when_web_sites_become_web_services.php
http://www.timedicer.co.uk/web-scraping
http://www.codediesel.com/php/web-scraping-in-php-tutorial/
http://www.slideshare.net/sumant.kumar.raja/java-web-scraping-presentation-854085
http://mindprod.com/jgloss/screenscraping.html
http://datatoolbar.com
http://www.mozenda.com/web-scraping.aspx
http://web-harvest.sourceforge.net/download.php
http://www.manageability.org/blog/stuff/screen-scraping-tools-written-in-java/view

HTTP / HttpClient

http://www.ietf.org/rfc/rfc2616.txt
https://addons.mozilla.org/es-eS/firefox/addon/httpfox/
http://es.wikipedia.org/wiki/Java.net
http://hc.apache.org/httpcomponents-client-ga/
http://hc.apache.org/httpcomponents-client-ga/primer.html
http://hc.apache.org/httpcomponents-client-ga/tutorial/pdf/httpclient-tutorial.pdf
http://hc.apache.org/httpcomponents-client-ga/examples.html
http://ep.cs.nthu.edu.tw/usr.doc/commons-httpclient-2.0.2-r1/html/logging.html

Expresions regulars

http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular
http://download.oracle.com/javase/tutorial/essential/regex/
http://jakarta.apache.org/oro/
http://jakarta.apache.org/regexp/
http://www.regular-expressions.info/reference.html
http://www.regular-expressions.info/possessive.html

JAXB

http://jaxb.java.net/
http://jaxb.java.net/nonav/2.2.3u2/docs/api/
http://www.programacion.com/articulo/el_api_jaxb_115/1
http://java.sun.com/developer/technicalArticles/xml/jaxb/
http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/JAXBWorks2.html
http://download.oracle.com/docs/cd/E12840_01/wls/docs103/webserv/data_types.html
http://ws.apache.org/jaxme/release-0.4/manual/ch02s02.html

http://www.devx.com/Java/Article/34069/1954
http://java.ociweb.com/mark/JavaUserGroup/JAXB.pdf
http://www.rossbeazley.co.uk/easy-xml-parsing-using-jaxb-annotations/java/
http://java-source.net/open-source/xml-parsers
http://www.probandocodigo.com/2009/05/para-qu-sirve-la-api-jaxb.html
http://www.rossbeazley.co.uk/easy-xml-parsing-using-jaxb-annotations/java/
http://technology.amis.nl/blog/2046/using-the-javaxxmlbind-annotations-to-convert-java-objects-to-xml-and-xsd

JAXP

http://xml.apache.org/xalan-j/
http://saxon.sourceforge.net/
http://www.programacion.com/articulo/el_api_jaxp_113/3
http://download.oracle.com/javase/tutorial/jaxp/index.html
http://java.sun.com/webservices/reference/tutorials/jaxp/html/intro.html
http://download.oracle.com/javase/1.4.2/docs/api/javax/xml/transform/TransformerFactory.html#newInstance%28%29
http://www.w3schools.com/xsl/default.asp
http://ca.wikipedia.org/wiki/XSL
http://www.javaworld.com/javaworld/jw-05-2003/jw-0502-xsl.html
http://www.ling.helsinki.fi/kit/2004k/ctl257/JavaXSLT/Ch05.html

INTERFÍCIE WEB

http://download.oracle.com/otndocs/jcp/servlet-3.0-public-oth-JSpec/
http://oreilly.com/catalog/jservlet/chapter/ch03.html
http://www.slideshare.net/setoide/administrando-jboss-3316944
http://www.cse.iitb.ac.in/dbms/Data/Courses/DBIS/Software/servlets/servlet_tutorial.html
http://www.adrformacion.com/cursos/j2ee/leccion2/tutorial1.html
http://javaweb.osmosislatina.com/curso/wars.htm
http://www.proactiva-calidad.com/java/herramientas/log4j/index.html
http://nileshbansal.blogspot.com/2006/08/java-exception-unknown-source.html

API Java

http://sites.google.com/site/apuntesdejava/Home/comparator-y-comparable
http://javacook.darwinsys.com/download.html
http://download.oracle.com/javase/6/docs/api/

Webs dels operadors seleccionats

http://www.dhr.com/countries/spain.htm
http://www.hoteles.com/
http://www.skoosh.es/
http://www.es.lastminute.com/site/viajes/hoteles/
http://www.hotelopia.es/
http://www.getaroom.com/

http://www.edreams.es/vuelos/
http://www.es.lastminute.com/site/viajes/vuelos/
http://www.rumbo.es/
http://www.spanair.com/web/es-es/
http://www.vueling.com/?language=ES

Altres operadors

http://www.booking.com
http://www.expedia.es
http://www.compararhotel.com/
http://www.easytobook.com/
http://www.venere.com/es/
http://www.ichotelsgroup.com/intercontinental/es/us/home
http://www.hotelbook.com/en
http://www.laterooms.com/es/
http://www.reservetravel.com
http://www.asiarooms.com
http://www.hotelschart.com/
http://www.hotel-board.com/
http://www.e-bookinghotel.com/
http://www.octopustravel.com/es/Home.jsp
http://www.ratestogo.com/default.asp
http://www.reserveahotelonline.com/
http://www.asiativ.com/Thailand.aspx
http://www.hotelclub.com/
http://www.hotel.info/
http://www.superbreak.com/
http://www.priceline.com/
http://www.atrapalo.com/
http://es.accommodationz.com/
http://www.travelocity.com/Hotels
http://www.orbitz.com/
http://english.ctrip.com/
http://www.trivago.es
http://www1.hilton.com/es/hi/index.do
http://www.hotels4u.com/
http://www.ebookers.com
http://www.accorhotels.com/es/espana/index.shtml
http://www.bestwestern.com/
http://www.bookdirectrooms.com/
http://www.carlson.com/
http://www.easyclicktravel.es/
http://www.iberia.com/
http://www.viajar.com
http://www.airberlin.com/site/start.php?LANG=spa
http://www.aireuropa.com
http://www.logitravel.com/logitravel/home/home.aspx
http://www.ryanair.com/es
http://web.tubillete.com/

http://www.cheapandgo.com
http://www.easyjet.com/asp/es/Reserve/index.asp?lang=ES
http://www.mirayvuela.com/vuelos-lowcost/clickair.html
http://www.travelgenio.com/

Altres

http://www.motobit.com/util/base64-decoder-encoder.asp
http://validator.w3.org/
http://docstore.mik.ua/orelly/xml/xslt/index/index.htm

Temari assignatures "Compiladors I" i "Compiladors II" (UOC)

Temari assignatura "Enginyeria del programari de components i sistemes distribuïts" (UOC)

Temari assignatura "Enginyeria del Programari Orientat a l'Objecte" (UOC)

Temari assignatura "Metodologia i Gestió de Projectes Informàtics"

Ian F. Darwin, Curso de Java (O'Reilly, 2005)

Patrones de Diseño (Addison Wesley, 2011)

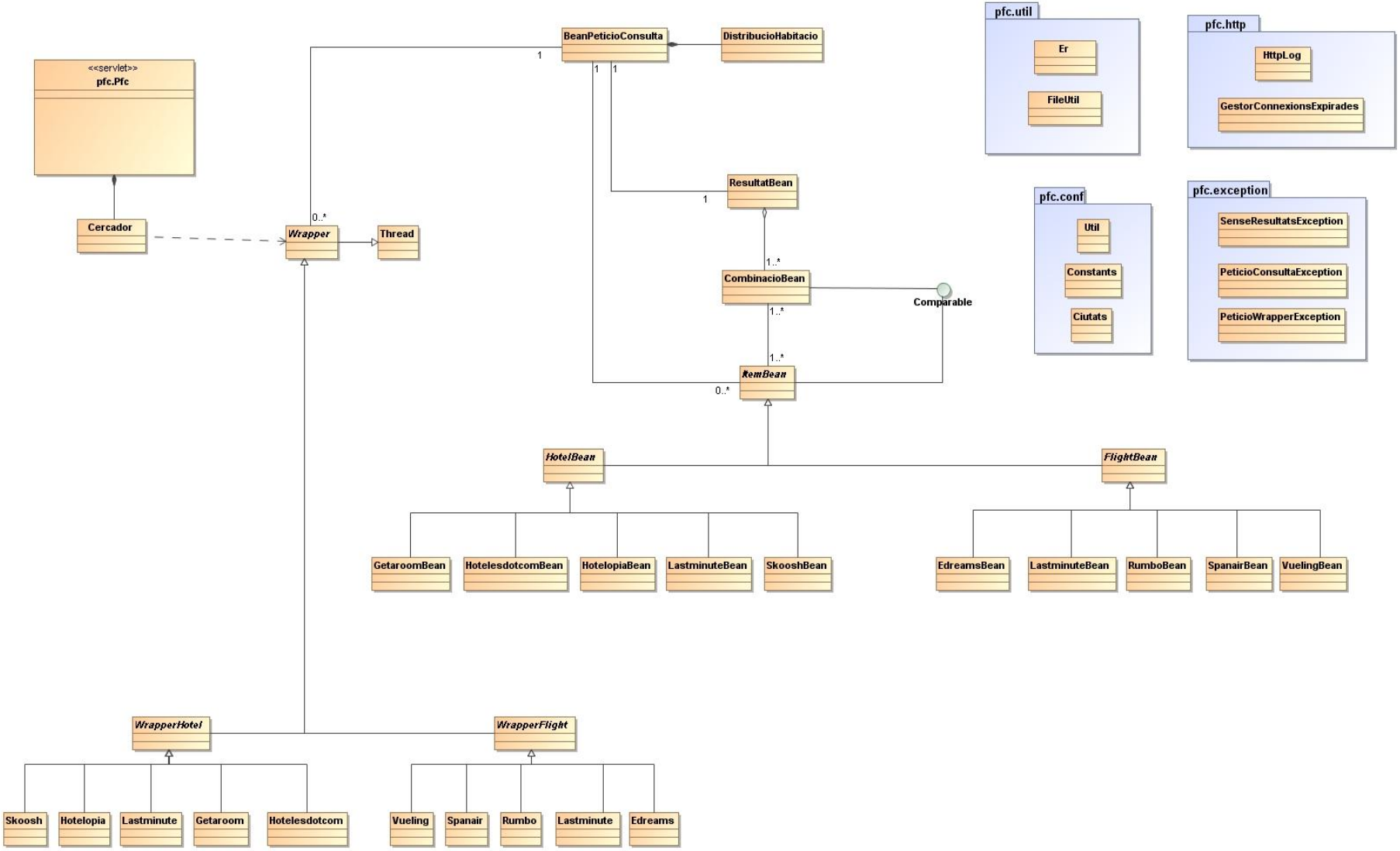
XML Los mejores trucos, Michael Fitzgerald (Anaya O'Reilly, 2005)

Desarrollo Web con JSP (Anaya, 2002)

ANNEXOS

A Diagrama de classes

A la plana següent es dóna el diagrama de les classes del sistema. Per comoditat de lectura i de visualització no es dóna uns llistat dels atributs i mètodes de cada classe - Aquests es poden consultar directament al codi font-


pfc.util
Er
FileUtil
pfc.http
HttpLog
GestorConnexionsExpirades
pfc.conf
Util
Constants
Ciutats
pfc.exception
SenseResultatsException
PeticioConsultaException
PeticioWrapperException
<<servlet>>
pfc.Pfc
Cercador
Wrapper
Thread
BeanPeticioConsulta
DistribucioHabitacio
ResultatBean
CombinacioBean
Comparable
RemBean
HotelBean
FlightBean
GetaroomBean
HotelesdotcomBean
HotelopiaBean
LastminuteBean
SkooshBean
EdreamsBean
LastminuteBean
RumboBean
SpanairBean
VuelingBean
WrapperHotel
WrapperFlight
Skoosh
Hotelopia
Lastminute
Getaroom
Hotelesdotcom
Vueling
Spanair
Rumbo
Lastminute
Edreams

B Ciutats configurades

Es dóna el llistat de ciutats configurades, ordenades per comunitat autònoma. Per a cadascuna s'indica l'aeroport que li correspon:

- Andalucia
  - Almería (LEI)
  - Córdoba (ODB)
  - Granada (GRX)
  - Málaga (AGP)
  - Sevilla (SVQ)
- Aragón
  - Huesca (HSK)
  - Zaragoza (ZAZ)
- Asturias
  - Oviedo (OVD)
- Balears
  - Palma de Mallorca (PMI)
  - Ibiza (IBZ)
  - Mahó (MAH)
- Canarias
  - Las Palmas de Gran Canaria (LPA)
  - Santa Cruz de Tenerife (TCI)
- Cantabria
  - Santander (SDR)
- Castilla la mancha
  - Albacete (ABC)
- Castilla y león
  - Burgos (RGS)
  - León (LEN)
  - Salamanca (SLM)
  - Valladolid (VLL)
- Catalunya
  - Barcelona (BCN)
  - Girona (GRO)
- Comunitat Valenciana
  - Valencia (VLC)
  - Alicante (ALC)
- Extremadura
  - Badajoz (BJZ)
- Galicia
  - La Coruña (LCG)
  - Vigo (VGO)
- La Rioja
  - Logroño (RJL)
- Madrid
  - Madrid (MAD)
- Murcia
  - Murcia (MJV)

- Navarra
    - Pamplona (PNA)
- Pais Basc
    - Vitoria (VIT)
    - San Sebastián (EAS)
    - Bilbao (BIO)

C Millores

A continuació es dóna una llista genèrica amb canvis i millores aplicables a la solució donada. Es útil per a conèixer les limitacions de la solució tècnica lliurada però també per a conèixer el tipus de canvis que sense modificar massa l'estructura de l'aplicació ajudarien millorar-la:

- Realitzar la cerca a un nombre major d'operadors

- Analitzar el temps d'execució dels diferents threads que fan la cerca als operadors. Amb una bona quantitat d'execucions del cercador, es pot conèixer mirant els logs els threads que triguen més temps a executar-se. Aquests threads es poden iniciar abans que la resta.

- Incorporar més ciutats al cercador. A les ciutats sense aeroport se lis pot assignar el aeroport més proper.

- Algunes dades com les ciutats d'origen/destí poden ser generades dinàmicament segons els valors continguts a una base de dades

- Millores de presentació al formulari de cerca inicial: camps més petits, camps d'ajuda, incorporar un calendari per a escollir les dades d'inici/fi, donar la possibilitat d'indicar el nombre de nits i no només la data de fi, etc

- Millores de presentació al formulari de presentació dels resultats: canvis al CSS, presentar un logotip de cada operador enlloc del seu nom, etc.

- Camp nou indicant si es té descompte de resident

- Utilitzar un framework a la part de presentació com pot ser Struts.

- ....