GestNews:

Servei de Cerca Automatitzada de Notícies

Estudiant: Carlos E. García Gómez
Enginyeria en Informàtica

Consultor: Jordi Ferrer Duran

15 de Gener del 2012

DEDICATORI I AGRAÏMENTS

El desenvolupament del present projecte i en general els estudis que estic completant, m'han suposat un gran esforç intel·lectual i sobre tot una gran dedicació de temps. Voldria agrair a totes aquelles persones que m'han ajudat d'una forma o altra a aplegar fins a aquest punt i poder finalitzar aquest projecte.

Voldria començar agraint als companys amb qui he tingut el plaer de cursar les diferents assignatures. Per les aportacions realitzades per diferents canals i que m'ha ajudat molt a aconseguir el objectius marcats.

També voldria agrair a tots el professors que he tingut en aquest període de temps i que m'han dirigit i ajudat en la consecució dels objectius definits en les diferents assignatures.

Gràcies a Jordi Ferrer per tot el suport prestat en el desenvolupament del projecte, per la seva velocitat en la resolució dels dubtes i per la orientació prestada en la direcció de la PFC.

Gràcies també a la col·laboració de totes aquelles persones que escriuen i treballen en diferents portals, blogs, fòrums a Internet. Perquè moltes vegades la seves aportacions, codi, explicacions m'han facilitat molt a comprendre les coses i aplegar a una solució millor.

Finalment, voldria fer un agraïment a tota la meua família, per el temps que no els he pogut dedicar en aquests anys d'estudis, i molt especialment a la meua dona Mònica, per la seva comprensió i suport en els moments difícils.

RESUM

La present memòria té com a objectiu descriure i donar a entendre la solució desenvolupada per solucionar el problema plantejat en el Projecte Final de Carrera (d'ara endavant PFC). Identificar els objectiu, explicar les metodologies emprades, enumerar les diferents tecnologies emprades i la justificació de la seva elecció, etc.

L'àrea del coneixement en què es centra la PFC és l'àrea del Compiladors, que dins dels estudis d'Enginyeria Informàtica, es centra en els coneixements desenvolupats per les assignatures de Compiladors I i Compiladors II. Tanmateix, la complexitat de la problemàtica plantejada a la PFC, requereix de l'aplicació de coneixements teòrics i pràctics adquirits en altres assignatures, principalment:"Enginyeria del Programari Orientat a l'Objecte", "Enginyeria del Programari de Components i Sistemes Distribuïts", "Bases de Dades", "Metodologia i Gestió de Projectes Informàtics".

Com es pot intuir pel paràgraf anterior, per poder desenvolupar amb èxit una solució al problema plantejat, és necessari del domini d'un ampli ventall de coneixements adquirits en els estudis d'Enginyeria Informàtica.

L'estructura de la present memòria comença amb un capítol d'introducció, on es presenta el tema del projecte, els objectiu específics i la metodologia que es seguirà per aconseguir els objectius definits per la PFC. També es mostren les tasques que s'ha de dur a terme i la seva planificació per aconseguir l'objectiu final.

La part principal de la memòria la formen els capítols 2, 3, 4, 5, 6, 7 i 8 que detallen diferents aspectes del desenvolupament de l'aplicació. Per acabar hi ha la conclusió treta del desenvolupament, un glossari i la bibliografia emprada.

Per finalitzar el document, s'han afegit uns annexos amb el pla de proves aplicat a l'aplicació desenvolupada.

ÍNDEX DE CONTINGUTS

1. Capítol 1, Introducció. *7*

1.1. Justificació del PFC i context en el qual es desenvolupa: punt de partida i aportació del PFC. **7**

1.2. Objectius del PFC. **8**

1.3. Enfocament i mètode seguit. **9**

1.4. Planificació del projecte. **9**

1.5. Seguiment de la Programació **14**

1.6. Productes obtinguts **16**

1.7. Breu descripció dels altres capítols de la memòria. **17**

2. Capítol 2. Arquitectura de la solució. *18*

2.1. Introducció **18**

2.2. Identificació dels Components **19**

2.3. Relació entre els Components. **20**

2.4. Flux d'execució **21**

2.5. Tecnologies emprades **24**

3. Capítol 3. Accés Dades Canals d'informació. *31*

3.1. Introducció **31**

3.2. Format RSS **31**

3.3. Problemes trobats. **32**

3.4. Mètode per extraure informació del canal RSS. **35**

3.5. Accés a la informació del canal RSS. **37**

3.6. Mètodes per extraure el contingut d'una notícia publicada. **39**

3.7. Implementació per extraure el contingut d'una notícia (canal conegut). **41**

4. Capítol 4. Model de Dades. *46*

4.1. Introducció **46**

4.2. Tractament de la Persistència. **46**

4.3. Diagrama Entitat / Relació. **47**

4.4. Detall de les entitats i les seves relacions.. **47**

5. Capítol 5. Regles de selecció de notícies. *51*

5.1. Introducció **51**

5.2. Compilació de les regeles. **51**

5.3. Operadors i gramàtica. **52**

5.4. Implementació del compilador **54**

5.5. Execució de les regeles. **56**

6. Capítol 6. Creació de la Sortida de l'aplicació. *57*

6.1. Introducció **57**

6.2. Execució de les regles de selecció 57
6.3. La Màquina Virtual. 58
6.4. Procés de generació i enviament d'avisos. 62
6.4.1. Format ICALENDAR 62
6.4.2. Format ODF 64
6.4.3. Procés de generació d'avisos. 64
6.4.4. Procés d'enviament d'avís. 68
7. Capítol 7. Interfície de l'aplicació. 70
7.1. Introducció 70
7.2. Patró MVC 70
7.3. Eina de desenvolupament de la Interfície. 71
7.4. Elements de la Interfície web. 72
7.5. Implementació de la Interfície web. 74
8. Capítol 8. Altres Aspectes del Sistema 79
8.1. Introducció. 79
8.2. Configuració del Sistema. 79
8.3. Activació del Servei de Cerca Automatitzada de Notícies. 79
8.4. Servidor Eixint de Correus. 79
9. Capítol 9. Conclusions. 80
10. Glossari. 81
11. Bibliografia. 83
12. Annexos. 85
12.1. Estructura de directoris del Projecte. 85
12.2. Pla de Proves. 86
12.3. Millores. 89

ÍNDEX DE FIGURES

Figura 01. Diagrama de Gantt 13
Figura 02. Diagrama de Gantt abans replanificació 15
Figura 03. Diagrama de Capes 18
Figura 04. Relació entre el Components. 20
Figura 05. Flux Execució Gestió d'usuaris. 21
Figura 06. Cas d'ús Interfície Web. 22
Figura 07. Flux Execució Servei de Cerca Automatitzat. 22
Figura 08. Diagrama de Cas d'us Servei de Recerca Automatitzat. 23
Figura 09. Diagrama de Blocs APIs JAXP. 28
Figura 10. Diagrama de funcionament API JAXB operació Marshall. 30
Figura 11. Exemple de noticia publicada. 33
Figura 12.Vista Alta d'un Canal. 35
Figura 13. Autòmat extracció informació RSS. 37
Figura 14. Informació treta d'una noticia publicada 40
Figura 15. Contingut de la noticia emmagatzemada a la BBDD 40
Figura 16. Arbre d'una estructura bàsica d'un document HTML 43
Figura 17.Classe Canal amb la propietat CanalBean. 46
Figura 18.Diagrama E/R. 47
Figura 19. Relació Usuaris Canals. 48
Figura 20. Relació Canals - Noticies 48
Figura 21. Relacions Canals – Avisos i Canals - Regles 49
Figura 23.Vista Editar/Afegir Regla de selecció. 51
Figura 24. Vista Regles afegides correctament. 53
Figura 25. Exemple tret de Wikipedia 63
Figura 26.Exemple Avís tipus: Registre en document ODT. 64
Figura 27.Exemple del document xml creat per la generació del fitxer ICS. 66
Figura 28.Flux d'execució patró MVC 71
Figura 29. Vista d'accés al sistema. 75
Figura 30. Vista llista de canals de l'usuari. 75
Figura 31. Vista llista de les regles associades a un Canal. 77
Figura 32. Vista Editar/Afegir Regles de Selecció. 78

1. Capítol 1, Introducció.

1.1. Justificació del PFC i context en el qual es desenvolupa: punt de partida i aportació del PFC.

Un dels grans avantatges, o problemes, segons com es miri, que tenim és la gran quantitat de continguts i serveis als que tenim accés a Internet. Això, per una banda, ens permet tenir accés a una gran quantitat d'informació però, per una altra banda, ens fa perdre molt de temps buscant i seleccionant la informació més rellevant o la que més ens interessa entre tota la que tenim disponible.

De fet, un dels problemes a què ens enfrontem cada dia i al qual li dediquem molt de temps és la recerca d'informació d'interès per a nosaltres a Internet. Diaris, blogs, xarxes socials, correu electrònic, informació meteorològica, i moltes altres més, constitueixen fonts d'informació quotidianes a les quals accedim en repetides ocasions. Per a la major part de nosaltres, el trobar informació d'interès en la gran quantitat de dades publicats, representa un treball i una gran pèrdua de temps.

L'aportació que pot proporcionar el desenvolupament d'aquesta PFC a la societat és un canvi en la forma de treballar a l'hora de poder accedir a informació del nostre interès. Així, l'aplicatiu facilitarà a l'usuari l'accés a informació del seu interès d'una forma ràpida. Realment ens permet un canvi de forma de treballar, ja que envolta de perdre temps buscant informació del nostre interès en els diferents servidor de continguts, l'aplicació, mitjançant unes regles de selecció de notícies, seleccionarà la informació i ens avisa de la seva existència en el moment de la publicació.

Més concretament el projecte consisteix en la construcció d'una aplicació que llegeix de forma automàtica notícies o continguts d'uns determinats canals RSS, concretament els canals d'interès de l'usuari i que aquest es vol subscriure.

Per fer-ho, aquesta aplicació s'ha de subscriure a diferents fonts de notícies web i obtenir d'aquestes informació en un dels formats més usuals basats en XML, el format RSS (Really Simple Syndication). Cada cop que aparegui un nou contingut en els canals en què s'ha subscrit l'usuari, accedeix a la informació més rellevant de la notícia i verifica si és de l'interès de l'usuari. L'aplicació determina l'interès d'una notícia verificant el compliment de regles de selecció.

Un cop s'ha seleccionat una notícia com de l'interès de l'usuari, el sistema avisa a l'usuari de la seva existència, per a la qual cosa l'usuari podrà indicar les següent formes de ser avisat[1]:

- Succés en format ICALENDAR.
- Registres successos en format ICALENDAR.
- Registres successos en format ODT.

[1] Veure Punt 6.4

1.2. Objectius del PFC.

L'objectiu principal de l'aplicació és facilitar a l'usuari l'accés a informació del seu interès, de forma que aquest no perda temps cercant-la, és a dir, envolta de què l'usuari cerqui la informació, el sistema avisa quan es publica informació d'interès de l'usuari.

Per aconseguir aquest objectiu l'aplicació consta de dues parts:

- Una interfície web que permet als diferents usuaris, registrats en la aplicació, subscriure's a diferents canals i establir unes regles de selecció de notícies segon el seu gust.
- Un servei o motor encarregat de cercar entre les notícies publicats als canals, les que compleixen les regles establertes per l'usuari i avisar-li segons el mecanisme d'avís establert.

Amb el desenvolupament d'aquest aplicatiu es demostra, per part de l'estudiant, la consecució dels següents objectius:

- Posar en pràctica coneixements del pla d'estudis, aplicant-los a un cas complex i d'utilitat dins de l'Enginyeria Informàtica.
- Treballar en diferents tècniques de cerca i recollida d'informació per Internet.
- Utilitzar eines i mecanismes de gestió de projectes: planificar amb detall el desenvolupament del projecte, definir fites, lliuraments i realitzar tasques de control i seguiment.
- Desenvolupar una aplicació que automatitzi la selecció de continguts.
- Estudiar llibreries per al tractament de documents XML.
- Conèixer el protocol RSS (que es basa en XML) por poder extraure les notícies dels canals RSS.
- Conèixer i Aprofundir en el funcionament dels servidors d'aplicacions.
- Aprendre tècniques d'extracció d'informació útil de documents HTML segons uns patrons de cerca determinats.
- Fer ús de llenguatges de marcat com XHTML, XML i XSL.
- Conèixer i fer ús de diferents tècniques d'anàlisi i extracció d'informació.

1.3. Enfocament i mètode seguit.

Com es pot extraure de la lectura detinguda dels requeriments de l'aplicació que es demana, la solució a implementar requereix de la utilització de tecnologies molt diferents: interfície web, tractament d'un volum d'informació relativament important, emmagatzemament de dades, accés a diferents fonts de dades a Internet, documents en XML, reconeixements d'expressions, etc.

Aquestes tecnologies requereixen d'una metodologia de programació prou complexa, i segons la plataforma de desenvolupament triada, pot ser inclòs molt diferent, la qual cosa m'ha fet decidir-me per emprar la plataforma J2EE com a tecnologia de desenvolupament. Aquesta plataforma, a més d'estar desenvolupada baix la filosofia del software lliure, cosa imprescindible per l'enunciat de la PFC, ens proporciona moltes facilitats per poder treballar en les diferents tecnologies que requereix el desenvolupament de l'aplicació.

L'elecció de la plataforma de desenvolupament, en aquest cas J2EE, ens va condicionar totalment la forma de tractar i implementar els diferents elements de què es compon la solució, i de com accedir i tractar els recursos que necessita. Per un altre costat, donat el seu abast, aquesta plataforma ens proporciona una solució integral, per poder atacar amb ella els diferents desenvolupaments dels components de l'aplicatiu. L'elecció d'una plataforma diferent, hauria requerit possiblement d'emprar diferents eines de desenvolupament. A més, gràcies a la gran quantitat d'eines disponibles i de la gran varietat de llibreries de llibre distribució que existeixen, m'han facilitat molt el desenvolupament de l'aplicatiu.

1.4. Planificació del projecte.

En la planificació del projecte s'ha tingut en compte la dificultat de l'aplicatiu a desenvolupar i les fites establertes externament. Aleshores s'ha dividit el projecte en tasques i s'ha realitzat una estimació temporal d'aquestes.

Les tasques en què es divideix el projecte les podem veure, junt a una breu descripció, en les següents taules:

1	**Inici del Projecte**
Descripció	Comença el projecte.

2	**Estudi del Projecte**
Descripció	Estudi dels requeriments del projecte i primera aproximació a les solucions tecnològics a aquestos requeriments, cosa que ens permet fer una primera aproximació al projecte i identificar les diferents tasques a realitzar.

3	**Estudi i Selecció Tecnològic.**
Descripció	Estudi de les diferents solucions tecnològiques que es poden aplicar (servidor web, servidor d'aplicacions, etc.) al projecte i selecció de les que es consideren més adient. Selecció de gran part de l'arquitectura tecnològica a utilitzar i determinació dels requeriments de maquinari necessari.

4	**Instal·lació i Configuració del Programari.**
Descripció	Una vegada determinat els requeriments, s'instal·la i configura el programari principal de l'aplicació i del desenvolupament.

5	**Accés dades externs.**
Descripció	Es defineix l'accés al canals d'informació.

5.1	**Estudi de RSS.**
Descripció	Estudi del protocol RSS, en els quals es basa l'intercanvi d'informació en els canals RSS.

5.2	**Estudi accés servidors RSS.**
Descripció	Estudi de les diferents llibreries que ens permetrà accedir a la informació d'un canal al qual s'està subscrit (analitzadors)

5.3	**Estudi API XML.**
Descripció	Estudi de les diferents possibilitats per al tractament de documents XML.

5.4	**Mòdul de lectura.**
Descripció	Implementació del mòdul que s'encarregui de llegir d'un canal d'informació RSS i traure la informació rellevant per ser emmagatzemada.

5.5	**Mòdul d'extracció del contingut d'una notícia.**
Descripció	Implementació del mòdul que s'encarregui de llegir l'enllaç d'una notícia publicada d'un canal conegut i extraure únicament el contingut de la notícia.

6	**Model de BBDD.**
Descripció	Creació del model de BBDD.

6.1	**Estudi del model de BBDD**
Descripció	Estudi de les necessitats d'emmagatzemament del projecte.

6.2	**Creació del model de BBDD**
Descripció	Implementació de les entitats definides i les seves relacions en el gestor de BBDD.

6.3	**Mòdul d'accés al model de BBDD**
Descripció	Creació del mòdul d'accés a les entitats del model de BBDD.

7	**Interfície de l'usuari**
Descripció	Es dissenya la interfície de l'usuari a partir dels requeriments i s'implementa.

8	**Regles de Selecció**
Descripció	Estudi i creació de les regles de Selecció de Notícies.

8.1	**Llenguatge Regles de Selecció.**
Descripció	Estudi de la gramàtica de les regles de selecció.

8.2	**Implementació del Compilador.**
Descripció	Implementació dels analitzadors de les regles de selecció.

8.3	**Mòdul de selecció de Notícies.**
Descripció	Implementació del mòdul encarregat de l'execució de les regles de selecció (màquina virtual)

9	**Creació de la sortida.**
Descripció	Estudi i creació del mòdul encarregat de la sortida del projecte.

9.1	**Construcció de l'XML.**
Descripció	Construcció dels documents XML a partir de la notícia seleccionada.

9.2	**Estudi Model Sortida.**
Descripció	Estudi del model i la llibreria adient per poder realitzar la sortida desitjada.

9.2	**Transformació XSLT.**
Descripció	Construcció del transformador XSLT mitjançant la llibreria seleccionada per aconseguir la sortida.

10	**Integració.**
Descripció	Adequació i integració dels diferents mòduls desenvolupats per al projecte.

11	**Proves.**
Descripció	Tot i que s'han realitzat proves parcials per cada desenvolupament, en aquesta tasca es realitzaran proves en conjunt després de la integració. A més, en aquesta tasca s'inclou les accions necessàries per corregir els errors trobats.

12	**Documentació.**
Descripció	Elaboració de la documentació relativa al projecte.

12.1	**Pla de Treball.**
Descripció	Elaboració del pla de treball.

12.2	**Memòria del Projecte.**
Descripció	La memòria del Projecte es va realitzant a mesura que avança el projecte. A més, es reserva una setmana aproximadament per tancar i revisar la memòria.

12.3	**Elaboració Presentació del Projecte.**
Descripció	Elaboració d'una presentació digital que serà exposada mitjançant un vídeo.

13	**Seguiment del Projecte.**
Descripció	Tasca en la qual s'avalua l'estat del projecte respecte al Pla de Treball definit inicialment. En aquesta tasca hi ha definides unes fites corresponent a les PACs.

14	**Debat Virtual.**
Descripció	Després del lliurament Final del projecte, s'inicia un debat virtual amb els components del tribunal d'avaluació. Aquest debat té una duració de 3 dies.

Les fites definides són:

Data	Fita	Producte a Lliurar
09/10/11	Pac1	Pla de Treball
13/11/11	Pac2	Desenvolupament de la interfície i del model de BBDD
14/12/11	Pac3	Regles de Selecció de Notícies.
11/01/12	PFC	Lliurament Final (Aplicatiu, Memòria, etc.)

En el diagrama de Gantt següent es mostra la planificació de les tasques i el seu grau de consecució. Com es pot veure, en aquest moment el projecte està quasi tancat.

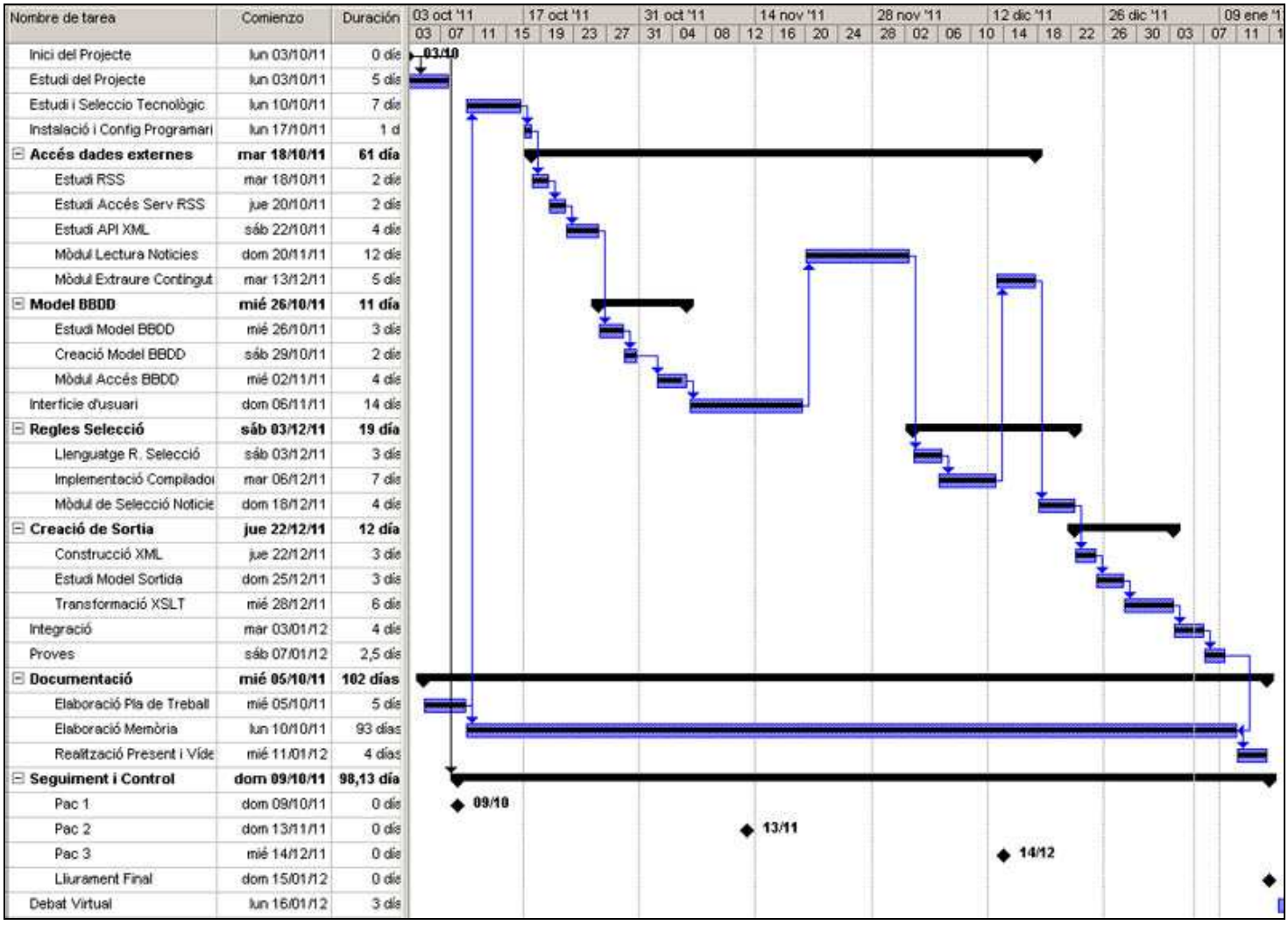


Figura 1. Diagrama de Gantt

1.5. Seguiment de la Programació

A continuació presento el seguiment de la programació en funció de les fites establertes per la PFC. Les desviacions produïdes en cadascuna d'elles i els aspectes que les han motivat, a més, els ajustos realitzats en la programació per poder aconseguir els objectius marcats amb garanties.

PAC1

S'havia de realitzar les tasques 1, 2 i 12.1 (Pla de treball). Es van aconseguir realitzar les tasques en el temps planificat sense massa problemes.

PAC 2

Estava planificat realitzar les tasques 3 (Estudi i Selecció Tecnològica), 4 (Instal·lació i Configuració del Programari), part de la tasca 5 (Accés a dades externes), 6 (Model de BBDD), 7 (Interfície d'usuari) i començament de la 12.2 (memòria).

Pràcticament es va realitzar totes les tasques que hi havia planificades, però en la realització de la tasca 5 es va detectar un problema. Aquest problema està detallat al capítol 3 punt 3. Bàsicament consisteix en la forma de tractar les regles de selecció de notícies per contingut. L'estudi de la seva solució, es va deixar per més endavant, i va produir, en la consecució d'aquesta fita, un endarreriment en l'elaboració de la memòria. Aleshores, com que les desviacions no eren molt implorants, es va mantenir la mateixa planificació.

El diagrama de Gantt següent mostra la primera planificació del projecte. Abans de la detecció de la problemàtica de l'extracció del contingut d'una notícia publicada[2].

[2] Veure capítol 3 punt 3.

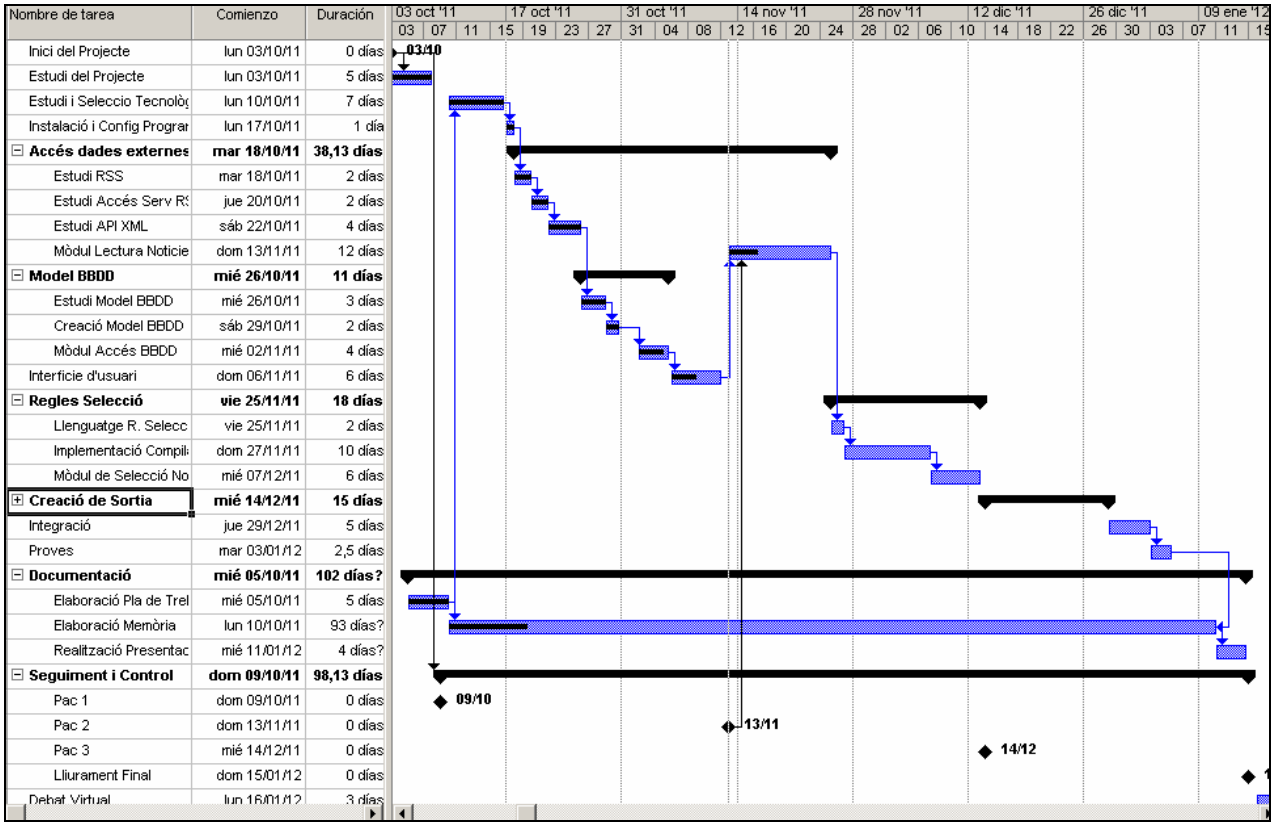


Figura 2. Diagrama de Gantt abans replanificació.

PAC 3

S'havia planificat realitzar les tasques 8 (Regles de selecció) i continuar la 12.2 (memòria), però degut a l'aparició de diferents problemes en el projecte, es va haver de replanificar per evitar desviacions en la consecució de l'objectiu final.

Les problemàtiques trobades van ser:

- Llibreria log4j: En el desenvolupament de l'aplicació es va decidir utilitzar la llibreria log4 per gestionar el log del sistema. Aquesta decisió va produir un endarreriment de uns dies, ja que el log no funcionava correctament. Finalment es va descobrir que la llibreria que havia instal·lada a jboss no funcionava correctament. Per solucionar el problema es va incloure la llibreria que duia ECLIPSE al projecte. Però fins que es va descobrir el problema, es va perdre molt de temps.
- Tractament de les regles de contingut de notícies: Aquesta problemàtica es va detectar abans de l'entrega de la PAC 2 , però l'estudi de la seva problemàtica es va realitzar posteriorment. Finalment, es va decidir, donada la complexitat de la solució, incloure una nova subtasca que tractés aquest assumpte. La subtasca és la 5.5 i s'encarrega de l'extracció del contingut de la notícia d'un canal conegut[3] pel sistema.

Degut als entrebancs trobats, va caldre replanificar les tasques per reflectir la nova situació. La nova i final planificació es pot veure a la Figura 1.

[3] Veure capítol 3 punt 3 per més informació.

LLIURAMENT DEL PROJECTE

La replanificació indicada al punt anterior va requerir d'un augment de recursos humans per dur-la a bon termini. Com que el projecte es desenvolupat per una única persona, es va haver d'augmentar les hores de dedicació.

Sense altres consideracions, es van realitzar les tasques marcades sense massa problemes: Finalitzar la tasca 8.2 (implementació del compilador) i 8.3 (màquina virtual), tasca 5.5 (extracció del contingut d'una notícia publicada), la tasca 9 (creació de la sortida). Les tasques 10 (integració del sistema) i 11 (pla de proves). També es va finalitzar la tasca 12 (documentació) i 13 (seguiment).

Queda pendent la tasca 14 (debat virtual) que es realitzarà posteriorment al lliurament del projecte.

1.6. Productes obtinguts

Els productes que es presenten amb la finalització del projecto són:

- **GestNews.zip:** Fitxer comprimit amb el desenvolupament sencer del projecte GestNews amb l'eina Eclipse. En aquest fitxer estan la interfície web amb la Gestió de Preferències de l'usuari, el Servei de Cerca Automatitzat, etc.

- **GestNews.war:** Empaquetat generat amb Eclipse amb el desplegament en el servidor d'aplicacions de la Gestió de Preferències de l'usuari.

- **sca.zip:** Empaquetat que conté el jar d'execució del component del Servei de Cerca Automatitzat i l'estructura de directoris que necessita (configuració, cache, temporals, etc.)

- **Manual d'usuari:** Document explicatiu del funcionament de la part visible de la aplicació. Amb aquest document l'usuari podrà saber com subscriure's a un canal, afegir regles de selecció de notícies, etc.

- **Manual d'instal·lació:** Document amb la informació tècnica necessària per a ficar en funcionament l'aplicatiu.

- **Memòria:** Document amb la memòria del projecte.

- **Vídeo de Presentació:** Vídeo de presentació que explica i sintetitza el treball realitzat en el projecte**.**

- **Presentació:** Presentació utilitzada al vídeo.

1.7. Breu descripció dels altres capítols de la memòria.

A continuació es comenten breument el capítols que formen la part principal de la memòria:

Capítol 2: Arquitectura de la solució.

Aquest segon Capítol té com objectiu estudiar l'arquitectura de l'aplicatiu desenvolupat, els seus components, la seva relació, el flux d'execució que existeix i la tecnologia emprada.

Capítol 3: Accés dades externs.

Tracta de la forma en què els diferents mòduls accedeixen als servidors de continguts. De l'estàndard RSS i de com s'analitza aquests tipus de documents per extraure la informació.

Capítol 4: Model de dades.

S'analitza el model de persistència de l'aplicació. De les entitats que s'han utilitzat per implementar aquest model i de les seves relacions.

Capítol 5: Regles de selecció de notícies.

Introducció a les regles de selecció de notícies i a la seva utilitat en el sistema. Es mostra la gramàtica que les accepta i la implementació dels analitzadors que verifiquen i tradueixen aquest tipus de regles.

Capítol 6: Creació de la sortida de l'aplicació.

En aquest capítol es tracta el procés d'execució de les regles de selecció i de la generació i enviament dels avisos als usuaris.

Capítol 7: Interfície de l'aplicació.

Introducció i especificació del patró de disseny emprat, de la metodologia i del desenvolupament de la interfície de l'aplicació i del seu funcionament.

Capítol 8: Interfície de l'aplicació.

Es tracten altres aspectes del sistema que es consideren d'interès, però per la seva naturalesa no encaixen en el capítols anteriors.

2. Capítol 2. Arquitectura de la solució.

2.1. Introducció

Aquest segon Capítol té com objectiu estudiar l'arquitectura de l'aplicatiu desenvolupat, els seus components, la seva relació i el flux d'execució que existeix. També s'indica la tecnologia emprada en cada cas i els motius de la seva elecció.

En el disseny arquitectònic de l'aplicació s'ha buscat aconseguir una arquitectura distribuïda i escalable, per la qual cosa s'ha optat per seguir una arquitectura de disseny en n capes. La arquitectura en n capes es basa en separar la complexitat de l'aplicatiu en capes, així apareix la capa de presentació, la capa de negoci i la capa d'emmagatzemament. Això permet la possibilitat de tindre totes les capes juntes executant-se en una mateixa màquina, per a un entorn de producció menut, o separar una o més capes per ser executades en màquines diferents, en el cas de què la solució vaja a tindre més càrrega. Evidentment, aquesta arquitectura també ens permet créixer sense moltes modificacions, és a dir, començar per un entorn de producció menut i anar creixent (separar les capes en nodes diferents d'execució) en funció de la càrrega que haja d'absorbir el sistema.

A més de seguir l'aplicatiu un disseny d'arquitectura en capes, per augmentar el grau de distribució s'ha separat la capa de negoci en dos components diferents, com si foren dos subcapes de negoci. Una d'aquestes capes s'encarrega de tractar amb la gestió de l'usuari (les preferències de l'usuari en subscripció a canals RSS, regles de selecció, etc.) i l'altra és un servei i que implementa el Servei de Cerca Automatitzat (d'ara en davant SCA). Aquestes dues subcapes, evidentment s'han de comunicar entre elles, per a la qual cosa entra en joc un altre component molt important: el Gestor de BBDD. Una descripció Visual la podem veure al següent imatge.

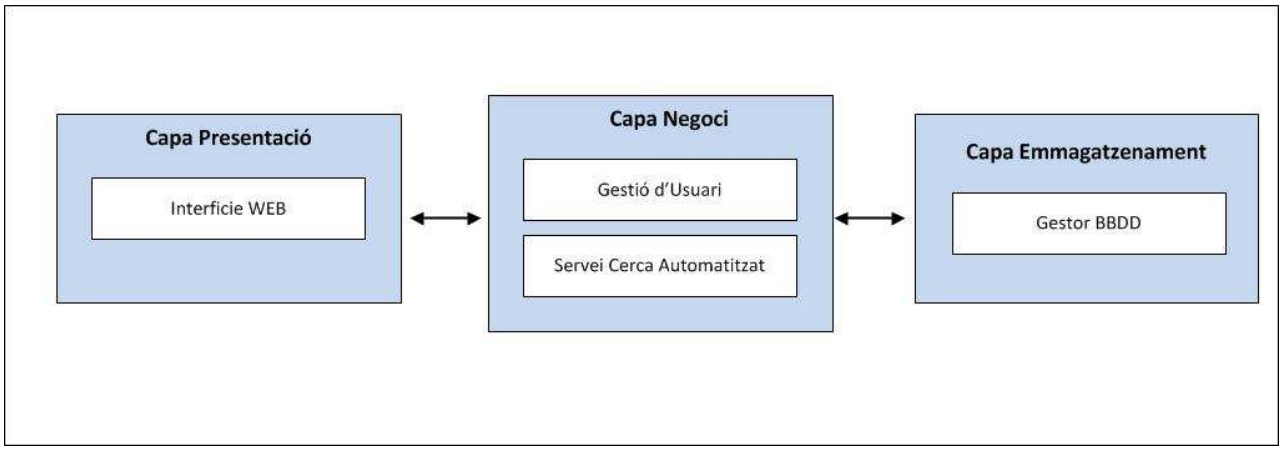


Figura 3. Diagrama de Capes

2.2. Identificació dels Components

Si s'estudia l'arquitectura de l'aplicació, es pot veure que existeixen quatre components o elements molt bé identificats, que interactuen entre ells. A continuació es presenta una breu descripció d'aquests components:

- Gestió de l'usuari i administració de preferències (GUAP). L'aplicació consta d'una part visible i que s'ha dissenyat mitjançant una interfície web. L'objectiu del component de Gestió d'usuari i administració de preferències (GUAP) és identificar a l'usuari i permetre'l definir els seus gustos en quant a la informació que li ha de cercar el sistema, per a la qual cosa l'usuari ha d'indicar els canals d'informació a subscriure's i les regles de cerca d'informació. Com he indicat, aquest component proporciona la part visible de l'aplicatiu, i dintre seu estan el únics components que treballen directament amb el usuari.

- Base de Dades. El Gestor de BBDD forma la pedra angular mitjançant el qual gira tot l'aplicatiu. A més, és el mecanisme de comunicació entre l'interface web i el Servei de Cerca Automatitzat. En general, mitjançant la interfície web l'usuari introdueix les seves necessitats d'informació a la BBDD i estableix les regles de selecció de notícies. Per altre costat el Servei de Carca Automatitzat (SCA), trau aquesta informació i realitza la recerca de les notícies en els diferents canals.

- Servei de Cerca Automatitzat (SCA). S'encarrega de comprovar si hi ha noves notícies en els canals d'informació subscrits (procés de sincronització de notícies amb el servidor de contingut) i estudiar si les notícies que publiquen compleixen les regles establertes pels usuaris (procés d'execució de notícies). També s'encarregarà d'avisar a l'usuari, mitjançant el canal d'avís que aquest ha establert, si hi ha notícies publicades que compleixen les regles (procés de generació d'avisos)

- Canals d'informació. Els canals d'informació són les fonts externes que proveeixen de notícies als usuaris mitjançant un document XML que segueix una estructura RSS. Hi ha gran quantitats de canals que proporcionen diferent tipus de notícies i és el Servei de Cerca Automatitzat (SCA) l'encarregat de triar aquestes notícies a partir de les regles de selecció establertes per l'usuari.

Als dos punts següent es mostrarà la relació existent entre els diferents components i el seu flux d'execució.

2.3. Relació entre els Components.

Dels quatre components identificats al punt anterior, la GUAP (Gestió d'usuari i Administració de preferències), i el SCA (Servei de Cerca Automatitzat) són els dos components que duen més treball, ja que s'han de desenvolupar totalment. Al gestor de Base de Dades, s'ha d'implementar el model de Dades que ens permetrà definir el disseny de persistència. I en quant al component definit pels Canals de Notícies, són recursos disponibles a Internet i únicament s'ha de definir el mòduls per poder accedir a ells per extraure la informació.

La relació entre els diferents components la podem veure al diagrama següent:

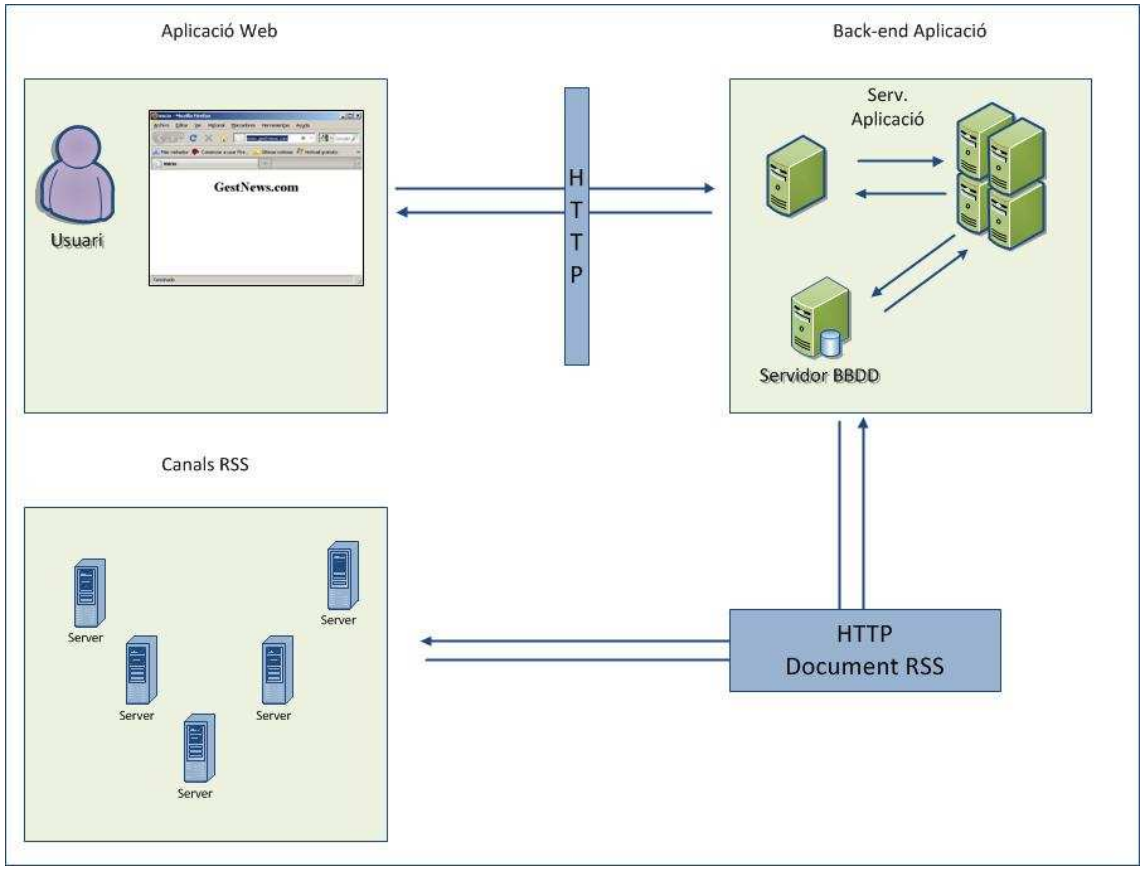


Figura 4. Relació entre el Components.

A la figura anterior podem veure com l'usuari accedeix a l'aplicació mitjançant un navegador. En la seva navegació per l'aplicació, s'aniran generant peticions encapsulades en el protocol http i que aplegaran al servidor d'aplicacions qui s'encarregarà d'executar-les. Posteriorment, des del servidor d'aplicacions s'accedeix als canals RSS en què l'usuari estigui subscrit, per extraure les notícies publicades i determinar quines notícies són del seu interès. Per a les notícies seleccionades es generarà un avís a l'usuari.

2.4. Flux d'execució

Per conèixer millor la relació entre els diferents components de l'aplicació es presenta, a grans trets, el seu flux d'execució dels dos mòduls que requereixen d'implementació.

Gestió d'usuari i Administració de Preferències (GUAP)

A la següent imatge es pot veure com el mòdul GUAP genera els diferents fluxos d'informació quan l'usuari navega per la interfície web.

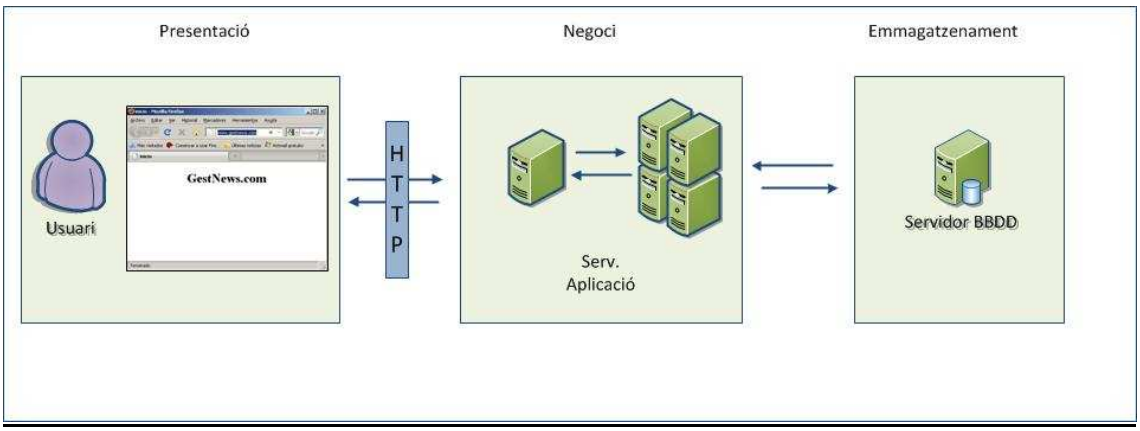


Figura 5. Flux Execució Gestió d'usuaris.

L'usuari accedeix a l'aplicació mitjançant la interfície Web. En la seva navegació produeix peticions que són encapsulades en **http** i dirigides al servidor d'aplicacions. Aquestes són executades pel serv. d'aplicacions i el resultat de les quals són enviades a l'usuari en format html. El component GUAP, s'encarrega de gestionar a l'usuari i d'obtindre les seves necessitats d'informació, podem veure-ho en més detall al diagrama de Cas d'ús que hi ha a la figura 6.

Per al tractament de la persistència, s'ha decidit utilitzar un gestor de BBDD. Aquesta decisió es basa en què un gestor de BBDD proporciona una major independència entre els dos mòduls que hi ha a la capa de negoci (Gestió d'usuari i Adm. Preferències i Servei de Cerca Automatitzat), per un altre costat l'aplicació treballarà amb una quantitat important d'informació i per minimitzar les consultes al servidor RSS externs i estalviar ample de banda, molta d'aquesta informació va ser emmagatzemada, per la qual cosa es requereix d'un mecanisme més sofisticat que per exemple la serialització d'informació en fitxers de dades.

Com a indicació avançar que la interfície web està dissenyada seguint el patró MVC (Model View Controler[1]).

[1] El model MVC serà tractat al capítol dedicat a la Interfície Web (Capítol 7).

A continuació es presenten les tasques que ha de desenvolupar la lògica del mòdul d'interfície Web.

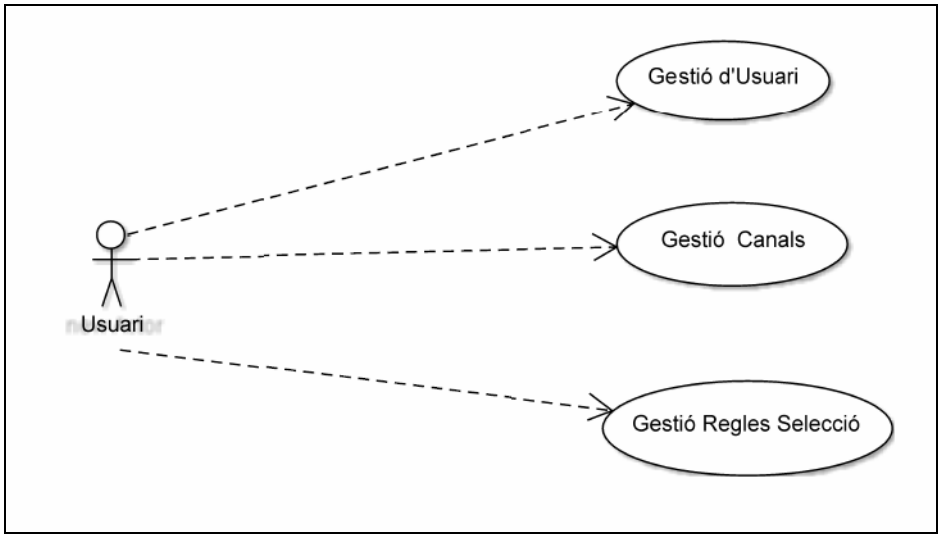


Figura 6. Cas d'ús Interfície Web.

Servei de Cerca Automatitzat (SCA)

El flux d'execució està representat a la següent imatge.

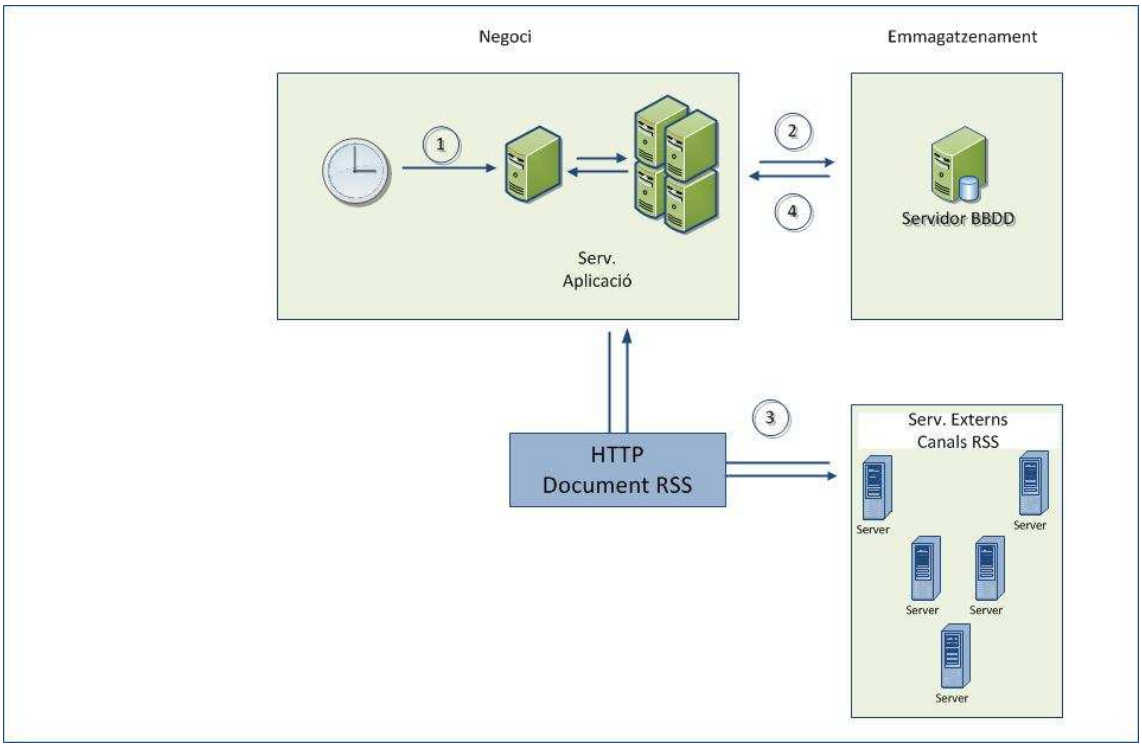


Figura 7. Flux Execució Servei de Cerca Automatitzat.

Com es pot veure a la imatge anterior, el flux d'execució no és iniciat per un usuari. Des del moment que es llança el servei, aquest es queda en estat de

sleep i es fica en funcionament cada cert temps[2]. Com es pot imaginar, aquest servei funciona independent del component GUAP, tot i que s'alimenta de la informació que aquest emmagatzema, per la qual cosa aquest servei pot estar executant-se en el mateix node o en un diferent, sempre i quant tingui accés al gestor de BBDD.

En el moment que es fica en execució (punt 1), recorre la BBDD per trobar tots els usuaris donats d'alta i els canals en què estan subscrits (punt 2). Seguidament accedeix als diferents canals de notícies RSS (punt3) per comprovar si hi ha notícies noves, en cas afirmatiu se les descarrega i les desa a la BBDD (punt 4), també comprova si aquestes noves notícies compleixen les regles de selecció associades al canal i establertes per l'usuari. En cas afirmatiu, avisa a l'usuari corresponent mitjançant el tipus d'avís que haja indicat.

Tot aquest flux d'execució queda resumit al següent diagrama de Cas d' ús i que correspon als tres processos que realitza.

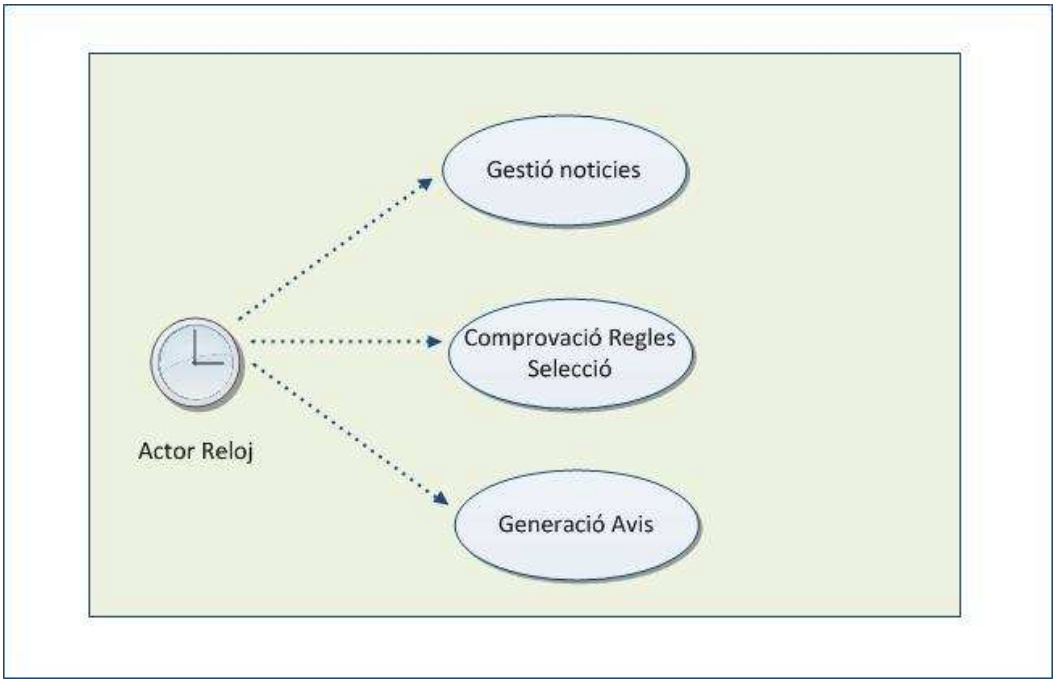


Figura 8. Diagrama de Cas d'us Servei de Recerca Automatitzat.

[2] El mode de treball del mòdul SCA indicat és el mode *ALONE*, també pot funcionar en mode *CRON*. Més informació capítol 8.

2.5. Tecnologies emprades

En aquest apartat es tracta de les diferents tecnologies emprades en el desenvolupament de l'aplicatiu i la justificació de la seva elecció.

▪ *Gestió de l'usuari (GUAP)*

Al requerir l'aplicatiu una interfície d'usuari web, s'ha de triar una eina de desenvolupament que ens permeta crear una interfície d'aquest tipus i d'un servidor web. El servidor lliure per excel·lència és Apache, aleshores la eina de desenvolupament ha de ser compatible amb ell i amb el gestor de BBDD triat (MySQL).

En un primer moment vaig pensar en desenvolupar la interfície en PHP, ja que la complexitat del mòdul no és gran i amb aquesta eina es podria fer perfectament i s'integra molt bé amb Apache i MySQL. A més, les aplicacions web desenvolupades en PHP no produeixen molta càrrega en el servidor Web.

Finalment, vaig triar la plataforma J2EE ja que ens proporciona una solució integral per a tot el projecte, com he indicat ja al punt 1.3. Aleshores, havia de triar entre les dues opcions de servidors d'aplicacions més importants que ens proporciona aquesta plataforma: Tomcat i JBoss. Vaig estar dubtant uns dies quina opció triar, ja que els dos permeten l'execució de servlets. Resumint tenim:

- Tomcat bàsicament és un contenidor de Apache (mòdul) per l'execució de servlets.
- JBoss és un servidors d'aplicacions total, i a més de permetre l'execució de servlets, pot executar EJBs i el clustering.

En definitiva, en qualsevol dels dos serv. d'aplicacions es pot desenvolupar el projecte, per la qual cosa trií JBoss ja que no el conec i tinc curiositat per fer-lo funcionar. No obstant això, l'elecció tampoc és una tasca definitiva ja que una aplicació desenvolupada per a qualsevol dels dos servidors d'aplicacions es pot migrar fàcilment a l'altre.

Per un altre costat, vull remarcar que si l'elecció haguera segut en una situació real, és a dir, que l'aplicació desenvolupada es fiqués en funcionament per a una empresa determinada, haguera tingut en compte altres aspectes com l'eficiència del sistema, per la qual cosa haguera triat Tomcat ja que carrega menys el servidor o l'existència d'altres aplicacions rodant en el servidor d'aplicacions (no instal·laria un segon servidor d'aplicacions, si ja en tenim un funcionant).

▪ *Servei de Cerca Automatitzat (SCA)*

En quant al mòdul del SCA, es podria desenvolupar en qualsevol llenguatge que permeta accedir a la informació emmagatzemada en el gestor de BBDD utilitzat (MySQL).

Com què actualment existeixen pràcticament per a la major part de llenguatges de programació llibreries per accedir a MySQL, l'ús d'aquest gestor de BBDD no representa cap problema a l'hora de triar un llenguatge de programació.

No obstant això, l'elecció normal en aquest cas és JAVA, per diverses raons:

- Java disposa d'unes llibreries molts avançades que es permet utilitzar d'una forma molt senzilla moltes de les tecnologies que requereix el desenvolupament (accés a recursos d'Internet, manipulació de documents XML, accés a BBDD, analitzadors, etc.)
- La interfície web es desenvolupada amb Servlets i moltes de les classes desenvolupades per la interfície web, poden ser reutilitzades per al SCA, la qual cosa simplifica el desenvolupament. (Reutilització de classes)
- Tot el desenvolupament es realitza amb la plataforma J2EE i amb el mateix entorn de programació: ECLIPSE, que proporciona moltes eines i s'integra molt bé amb el servidor d'aplicacions. No obstant, per al desenvolupament de les vistes web he utilitzat altres eines de disseny: Kompozer, Gimp, FireBug, etc.
- Els avantatges i facilitats que proporciona JAVA com a llenguatge de programació i que no cal remarcar: Orientat a Objectes, facilitat de programació, tractament d'excepcions, etc.

Vull remarcar, que tot i que el desenvolupament de l'aplicació està fet en ECLIPSE i les servlets de la interfície web comparteixen classes amb el serveis de Recerca, els dos components funcionen de forma independent. Les servlets es posen en funcionament amb el servidor d'aplicacions i el servei es pot ficar en marxa, per exemple, quan es fica en marxa el servidor on està instal·lat.

▪ *Gestor de Base de Dades*

Per al tractament de la persistència de l'aplicació s'ha decidit utilitzar un Gestor de Base de Dades per les següents raons:

- **Estalvi en requeriments de processament de les notícies**. A l'emmagatzemar les notícies, cada vegada que s'accedeix al Canal RSS per sincronitzar les notícies, únicament es verificarà les regles de selecció amb les noves notícies, ja que les altres han segut verificades anteriorment. Amb aquesta tècnica, ens estalviem processar totes les notícies publicades cada volta. El problema és que es necessita d'un mecanismes d'emmagatzemament més complex que l'ús de fitxers de dades.
- **Estructura de dades relativament complexa**. L'aplicatiu ha de treballar amb usuaris, canals, notícies, regles, etc. Aquest tipus diferents de dades es modelen molt millor en una BBDD que en fitxers de dades, a més, l'accés a una BBDD sempre serà molt més ràpid i flexible.
- **Estalvi d'amplada de banda**. L'aplicatiu ha de treballar amb diferents Canals RSS (als que els usuaris estiguin subscrits) l'accés als Canals requereix un ample de banda per part del servidor. L'emmagatzemament de les notícies ens pot estalviar diferents accessos al servidors RSS i per tant estalvi d'ample de banda. Tant mateix, aquest estalvi requereix d'un augment d'informació per emmagatzemar i que sempre serà tractada molt millor en una BBDD que en fitxer de dades.
- **L'arquitectura establerta**. Separar la interfície gràfica GUAP del SCA té certes avantatges, per exemple que estiguin en màquines diferents, però requereix d'un mecanisme que permeta compartir d'informació. Si la informació s'emmagatzema en fitxers de dades, es podria utilitzar un

Servidor de Fitxers per exemple per compartir la informació entre els dos components. Tanmateix la utilització d'un Servidor de BBDD és més elegant, requereix de menys recursos que un Servidor de fitxers, i ens permet establir un nivell de seguretat més alt (sempre que el servidor de BBDD estigui correctament configurat.)

Per totes aquestes raons s'ha optat en utilitzar un Gestor de Base de Dades per al tractament de la persistència de l'aplicació.

Hi ha diverses opcions en Gestors de Base de Dades lliures, però he triat MySQL per les següents raons:

- **Necessitats funcionals**. L'aplicació necessita únicament d'un Gestor de BBDD relacional. No es necessari d'altres funcions avançades com el tractament transaccional, les Base de Dades distribuïdes, etc.
- **Rapidesa**. Al ser una aplicació web, el sistema requereix d'un Gestor de BBDD ràpid i que admet múltiples connexions simultànies.
- **Càrrega i Recursos**. L'aplicació no necessita d'un tractament molt alt d'informació, aleshores, no necessitem d'un Gestor de BBDD molt potent però que consumeix molt recursos i que carregui molt el servidor.
- **Connectivitat**. Que haja bona connectivitat entre JAVA i el Gestor de BBDD.

MySQL compleix perfectament totes aquestes requisits funcionals i en quant a rapidesa de resposta en aplicacions lleugeres, és un dels millors, per això és el més emprat en aplicacions web a Internet.

- ***Consulta als Canals RSS[3].***

A Internet hi ha multitud de servidors que publiquen informació (Servidor de Contingut), aquest servidors moltes vegades també publiquen un document XML en un format normalitzat que conté certa informació de les notícies que publiquen. Aquest format pot ser el format RSS, tot i que també existeixen altres com ATOM[4].

Un Canal RSS és un servidor de contingut d'informació que a més de publicar la informació (notícies), publiquen aquest document XML en format RSS amb l'objectiu de facilitar la distribució dels seus continguts o notícies a lectors RSS o altres servidors web que tracten aquest fitxer RSS. D'esta forma, aquests clients llegint el fitxer RSS poden conèixer ràpidament si s'ha publicat nou contingut i certa informació d'aquest.

El Sistema GestNews llegeix els documents XML en format RSS publicats pels diferents canals RSS en què estan subscrits els usuaris registrats al sistema, i extraure els diferents ítems de les notícies (títol, descripció, autor, etc.). Per poder extraure aquesta informació des de Java, existeixen diferents opcions:

- Utilitzar un parser d'un determinat proveïdor com pot ser XERCES, Crimpson, Picolo, etc.

[3] Més informació http://en.wikipedia.org/wiki/RSSi http://www.rssboard.org/rss-specification
[4] En aquesta la PFC treballo únicament amb els Canals RSS. Informació http://en.wikipedia.org/wiki/Atom_(standard)

- Utilitzar directament un dels APIs XML en Java: SAX, DOM, JAXP o altres més específics JDOM, dom4j, etc.

Els parsers donen suport a un o més APIs XML, per exemple XERCES suporta SAX, DOM i JAXP. A més, et faciliten el tractament dels documents XML i proporcionen independència dels APIs XML. Tanmateix, he preferit utilitzar directament els APIs de tractament de XML (SAX, DOM, JAXP) entre altres coses per no tindre que instal·lar més llibreries al projecte.

Després de decidir l'ús de les APIs XML s'ha de decidir quina opció triar:

- SAX[5] (Simple API for XML): Aquesta API proporciona un control molt baix sobre el document XML, ja que les classes i interfases de SAX no modelen el document XML. Proporcionen el contingut a l'aplicació client mitjançant mecanismes de excepcions, és a dir, el parser és una interface amb uns mètodes que són cridats quan aquest està llegint el document XML, per exemple quan apareix un nou element o quan acaba. Aquest tractament, permet que el parser sigui molt ràpid i consuma molt pocs recursos del sistema. És l'única opció quan el document XML és molt gran, però no permet realitzar consultes XPATH de diferents elements, cosa que és necessari algunes voltes.
- DOM[6] (*Document Object Model)*: És un API que modela el document XML com si fora un arbre i és representat com un objecte de tipus Document. Totes les accions d'escriure, llegir, modificar, etc. són realitzades mitjançant cridades a mètodes. Evidentment, tractar un document XML mitjançant aquesta API és molt més senzill, però requereix de més recursos per part del sistema.
- JAXP[7] (*Java API for XML Processing)*: És el API estàndard de Java i dóna suport tant a SAX com a DOM. A més també permet realitzar altres accions com processament de XSLT que tractarem al següent punt. La seva funcionalitat la podem veure a la imatge següent:

[5] Més informació http://www.saxproject.org/
[6] Més informació http://en.wikipedia.org/wiki/Document_Object_Model
[7] Més informació http://en.wikipedia.org/wiki/Java_API_for_XML_Processing

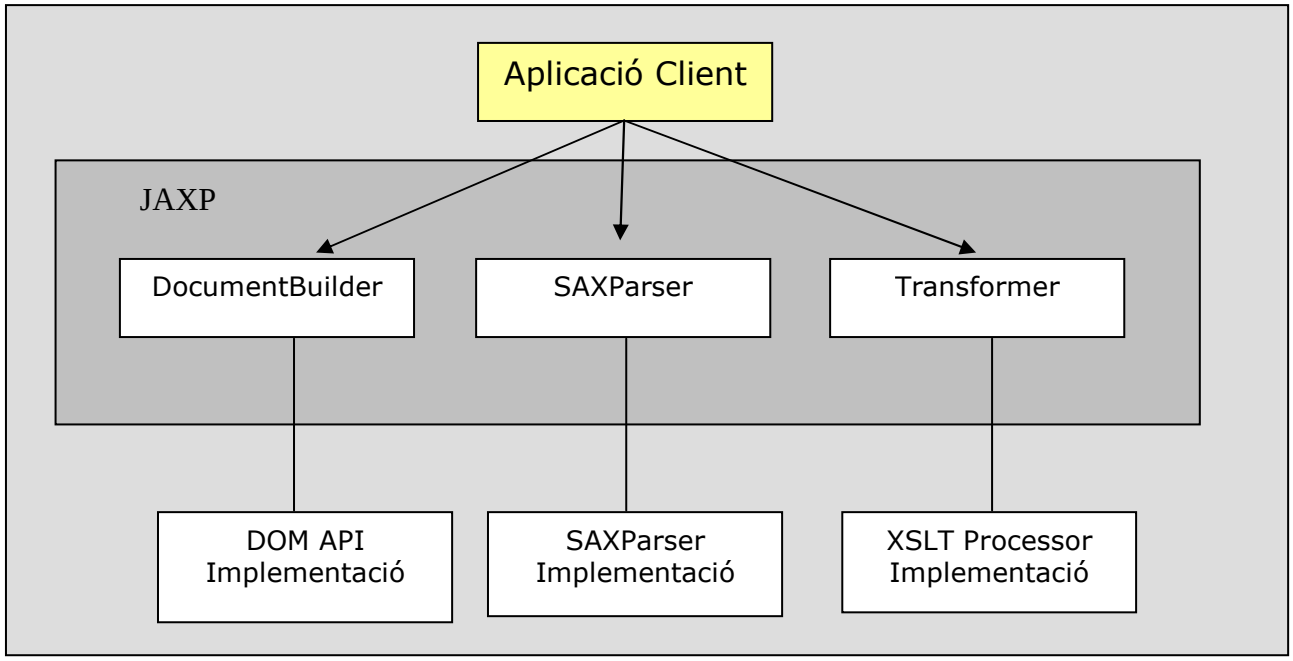


Figura 9. Diagrama de Blocs APIs JAXP.

Per al desenvolupament del mòdul de l'aplicació, que trau la informació de les notícies des del document XML que proveeix el canal RSS, he utilitzat la llibreria JAXP, ja que ens permet treballar tant amb DOM com amb el parser SAX.

Tot i que en capítols posteriors s'anirà concretant més, he creat una classe Canal que s'encarrega del tractament del canal. Entre els mètodes que implementa, hi ha getRSSInfo(), que té com objectiu connectar-se a la URL d'un Canal RSS, per extraure tota la informació del canal en un vector de NotíciaBean.

Per extraure la informació, en un principi vaig pensar crear un document **DOM** mitjançant JAXP i anar fent-li consultes XPATH per extraure els diferents ítems de les notícies. Aquesta metodologia resulta prou senzilla, però després d'implementar-la em vaig donar compte que hi ha canals que publiquen documents RSS prou grans, ja que poden tindre moltes notícies i dins de les notícies alguns fiquen molta informació. Aleshores, per representar aquest document XML en un document **DOM** consumeix molta memòria. Cosa que no es desitjable en un servidor. A més, els clients poden estar subscrits a molts canals, la qual cosa requereix d'un ús intensiu de la memòria en el sistema, i pot repercutir en altres aplicacions que també s'estiguin executant al servidor.

Per solucionar-lo, vaig substituir la creació del document **DOM**, per la utilització d'un parser en **SAX**, ja que realment es pot traure tota la informació del document RSS en una llegida del document XML en qüestió. La metodologia és un poc més complexa i per desenvolupar-lo m'he recolzat en una classe (Automata.java) que implementa un Autòmat d'estats. El resultat és una solució prou elegant, molt més ràpida i amb necessitats menors de recursos del sistema. Una explicació detallada es veurà en el capítol 3 (Accés dades Externes).

- ***Generació d'avisos als usuaris.***

Aquest mòdul està desenvolupat dins del component SCA i per les diferents tecnologies emprades en la seva implementació s'ha volgut presentar en aquest apartat.

Com el seu nom indica, aquest component és l'encarregat de generar els avisos als usuaris del sistema quan aquest detecta que una regla de selecció de notícia s'ha complert. Per la qual cosa ha de realitzar les dues tasques següents:

o **Generació de l'avís**: El sistema permet triar a l'usuari entre tres tipus d'avisos[8]: Succés ICALENDAR, Registro de successos ICALENDAR I Registre de successos en format ODT.

El mecanisme per generar aquest tipus d'avisos és el mateix en cadascú dels casos. El sistema genera un document xml amb la informació adient, i posteriorment aplica una plantilla xslt per generar el fitxer ICS o ODT segons sigui necessari.

Per a la generació del document xml es pot emprar diferents tecnologies:

- Creació manual: Donada la simplicitat dels documents xml a generar, es podria perfectament realitzar-se de forma manual. Tanmateix, sempre he cercat un sistema flexible i fàcilment ampliable. Aleshores, aquest mecanisme no és el més adient.
- Document DOM: Es pot crear un document d'arbre d'elements DOM i posteriorment serialitzar-lo en un fitxer. Aquest mecanisme és molt senzill, fàcil de modificar i d'implementar. No obstant això, l'ús d'aquesta tecnologia requereix de molts recursos en el sistema i s'ha intentat evitar.
- L'API JAXB[9] (Java Arquitecture for XML Binding): És la que s'ha emprat al sistema. Aquesta llibreria està també inclosa en l'API estàndard de JAVA i per la qual cosa no s'han d'incloure més llibreries al projecte. El seu funcionament és semblant als documents DOM, permet crear el document XML oblidant-se dels detalls d'aquest, però requereix de menys recursos. Bàsicament el seu funcionament consisteix en la creació de classes java que representen l'esquema XML. En aquestes es pot modificar propietats, afegir elements, etc. Posteriorment se li aplica el procés Marshal per aconseguir la transformació dels objectes java als elements xml (veure Figura 10) i finalment es desa a un fitxer. Ja es tractarà en més detall aquest procés al capítol 6.

Una volta el sistema ha generat al disc el document xml corresponent, s'aplica una plantilla xslt per aconseguir el tipus d'avís desitjat.

En aquest cas s'ha utilitzat JAXP[10] com a processador XSLT. JAXP s'utilitza també al sistema com a analitzador sintàctic, més concretament s'utilitza l'analitzador sintàctic SAX per analitzar els documents RSS (punt anterior) i

[8] Tot i que de la forma que està desenvolupat l'aplicatiu, fàcilment es pot implementar altres tipus. No han segut desenvolupat més tipus d'avisos per falta de temps.
[9] Referència: http://jaxb.java.net/
[10] Referència: http://xml.apache.org/xalan-j/

als documents HTML, per extraure el contingut d'una notícia publicada. Però també s'utilitza en el sistema com processador de plantilles XSLT. L'API de JAXP ja ha segut tractada al punt anterior.

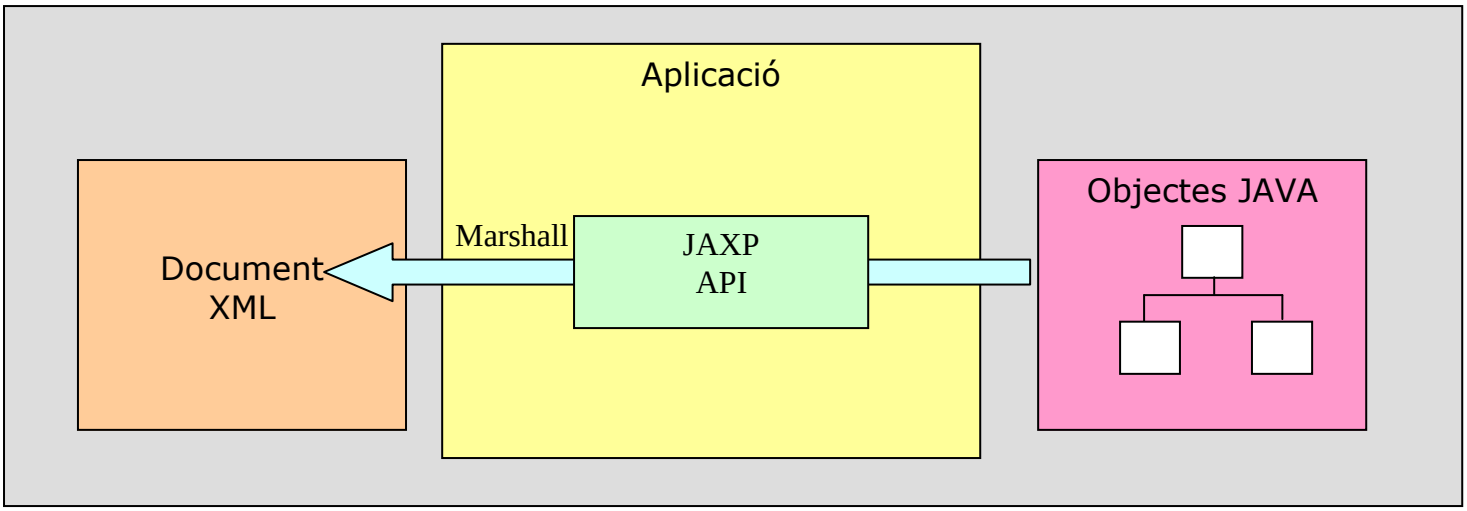


Figura 10. Diagrama de funcionament API JAXB operació Marshall.

- **Enviament del missatge**: Una vegada el sistema ha generat l'avís, aquest s'ha d'enviar a l'usuari. Per a l'enviament de l'avís a l'usuari s'ha decidit fer-ho via correu electrònic. En aquest cas s'ha utilitzat la llibreria JavaMail.

  He estat mirant diferents llibreries que implementin el protocol SMTP en JAVA per poder enviar un missatge via correu electrònic, però la més emprada amb diferència és JavaMail. Aquesta API ens permet enviar d'una forma molt senzilla *e-mails* en diferents formats: text, html i fins i tot amb fitxers adjunts.

  Al mòdul SCA, quan es detecta que una regla de selecció s'ha complit, envia un correu electrònic a l'usuari amb un text informant del nom de la regla que s'ha activat, el títol de la notícia corresponent i el fitxer amb el tipus d'avís especificat.

3. Capítol 3. Accés Dades Canals d'informació.

3.1. Introducció

Com s'ha comentat al capítol anterior, el sistema GestNews ha d'accedir als servidors RSS en què estan subscrits els diferents usuaris del sistema, per poder conèixer les notícies que han publicat. Posteriorment i en funció de les regles que els usuaris han establert, seleccionar les notícies que les compleixen.

Per poder traure aquesta informació, el sistema ha d'accedir a documents XML en format RSS que els servidors publiquen. Aquests documents contenen informació de les seves notícies.

En aquest capítol tractarem com es realitza aquest procés al sistema GestNews, per traure la informació i analitzar-la. També els problemes que han sorgit.

3.2. Format RSS

Els servidor d'informació o Canals RSS publiquen un document XML en un format conegut com RSS. Aquest document XML conté informació del canal i de les notícies que estan en aquests moments publicades. Aquest document XML en la versió 2.0 de RSS (la utilitzada per GestNews) segueix la següent estructura:

- Element **channel** amb els seus subelements. Conté informació relacionada amb el canal.
  - Obligatoris:

Element	Descripció
title	El nom del canal
link	Enllaç a la pàgina html que correspon amb el canal.
description	Paràgraf que descriu el canal.

  - Opcionals: Els més importants són:

Element	Descripció
language	Llengua en què es publica les notícies al canal.
managingEditor	Direcció electrònica de la persona responsable de la publicació.
webMaster	Direcció electrònica de l'administrador (WebMaster)

pubDate	Data de publicació de les notícies del canal.

- Seguidament un conjunt d'elements **item** amb els seus subelements. Cada ítem conté informació de cadascuna de les notícies publicades:
  - Obligatoris:

Element	Descripció
title	Títol de la notícia.
link	Enllaç a la pàgina html que conté tota la notícia publicada.
description	Descripció de la notícia.

  - Opcionals: Els més importants són:

Element	Descripció
author	Autor de la notícia.
category	Una o més categories en què es pot incloure la notícia.
pubDate	Data de la publicació de les notícies.
comments	URL a una pàgina amb comentaris de la notícia.

S'ha marcat en color vermell els subelements de **channel** i **item** que es trauen i s'emmagatzemen a la BBDD per al funcionament de l'aplicació.

Per a més informació del format RSS es pot accedir a les següents direccions:

Especificació RSS 2.0

http://en.wikipedia.org/wiki/RSS

http://cyber.law.harvard.edu/rss/rss.html

3.3. Problemes trobats.

Com s'ha vist al punt anterior, el format RSS estableix informació respecte certs aspectes de la notícia com: autor, títol, descripció, data publicació, etc., i un enllaç al contingut de la notícia, però no estableix l'estructura del format que ha de tindre el contingut de la notícia publicada.

Són els canals que publiquen la notícia, els que tenen llibertat a determinar aquest format. Moltes vegades, a més del contingut de la notícia, inclouen més informació com pot ser: publicitat, imatges, vídeos, comentaris dels lectors, etc. inclús en algunes ocasions enllaços o fragments d'altres notícies. A més, com aquests continguts addicionals es creen moltes vegades dinàmicament, la

informació pot variar de forma aleatòria cada volta que entrem o fins i tot, atenent a altres paràmetres com potser el *marketing* dirigit[1].

A la imatge següent, es pot veure un exemple d'una notícia publicada pel canal RSS de tv3. Es pot veure el contingut de la notícia dins del requadre negre i més continguts, com poden ser altres notícies (requadre vermell).

Figura 11. Exemple de notícia publicada.

Aquesta llibertat que deixa el format RSS, en quant a la publicació de les notícies es lògica. Establir un format fix a les notícies publicades, aniria en contra dels interessos dels servidors de continguts, ja que aquesta llibertat els permet diferenciar-se d'altres i ficar, per exemple, publicitat. Tanmateix, per al nostre sistema de recerca automatitzada de notícies és un problema.

El problema apareix en la forma de tractar les regles de contingut de les notícies. Si l'usuari estableix una d'aquestes regles, com què el sistema GestNews no pot separar el que realment es notícia d'allò que s'ha afegit pel

[1] El sistema emmagatzema informació relacionada amb els interessos del visitant i posteriorment li mostra publicitat, notícies, etc. en funció del seu gust.

canal RSS al publicar la notícia (publicitat, comentaris, altres notícies, etc.), pot produir falsos positius, ja que la regla s'ha de verificar en tot el document publicat.

Un exemple d'un fals positiu es pot veure a la figura anterior. Imaginem que ens interessa conèixer les notícies on apareix la paraula "guerra" i li fiquem aquesta regla al sistema: contains(notícia,"guerrra"). En aquest cas, la notícia anterior verificaria la regla, però erròniament, és a dir, es produeix un fals positiu. Tot i que en el cos de la notícia no apareix aquesta paraula, el canal RSS, com es pot veure a la imatge anterior, afegeix un enllaç a altres notícies publicades. En un d'aquests enllaços, si que apareix aquesta paraula i el sistema marcaria erròniament la notícia. A més, potser que quan li arribi l'avís a l'usuari i aquest decideix entrar en la notícia, no vegi la paraula causant del fals positiu, ja que el servidor de contingut (Canal RSS) decideixi ficar en substitució, un enllaç a una notícia publicada posteriorment.

Una solució a aquest problema seria que els servidors de contingut, a més de publicar el document XML en format RSS, identificaren el contingut de la notícia al document publicat. Per exemple establint en els ***tags*** que delimiten la notícia un determinat **id** o **class.** Aquest ***tag*** podria ser <p>, , etc.

He d'indicar que la major part de servidors RSS que he estudiat, si que delimiten la notícia amb un class determinat, per exemple *class = "contingut*", *class = "cos notícia*", però no és un estàndard i cadascú la identifica segons ha considerat millor.

Com que aquesta solució normalitzada no existeix, s'ha pensat d'extraure la notícia d'alguna forma.

Després de donar-li voltes al problema, s'ha aplegat a la següent solució: ensenyar al sistema l'estructura dels canals RSS que es considerin interessants, perquè puguin extraure el contingut de la notícia del document publicat. En aquestos canals RSS es realitzarà correctament les recerques per contingut de notícies. Aleshores, s'informa a l'usuari a l'hora d'inscriure's en un canal d'aquesta problemàtica. L'usuari té llibertat d'inscriure's tant en un canal conegut com en un de no conegut, però únicament en els canals coneguts, la recerca de la notícia per contingut es realitzarà correctament i en la resta de canals, es realitzarà en tot el fitxer **html** publicat.

Ha de quedar clar, que aquests falsos positius únicament es produeixen en les regles que afecten al contingut de la notícia, altres regles que fan referència a: autor, títol, sinopsis (descripció) no són afectats per aquests fals positius.

A la següent imatge (figura 12) es pot veure la vista de l'aplicació que s'encarrega de donar d'alta als canals (subscripció). En aquesta vista, es pot veure una menuda explicació de la problemàtica i un enllaç que explica més detalladament la situació. Més a baix, apareix una taula amb un llistat dels canals en què el sistema reconeix la seva estructura, i finalment unes caixes de text on l'usuari pot donar d'alta el canal: nom del canal i la url on es troba publicat el document RSS.

També vull indicar, que en la vista on apareixen els canals en què està subscrit l'usuari, a més d'altra informació, apareix una icona *"d'ok"* en els canals que s'ha subscrit l'usuari i que són reconeguts pel sistema. (Ver figura 30).

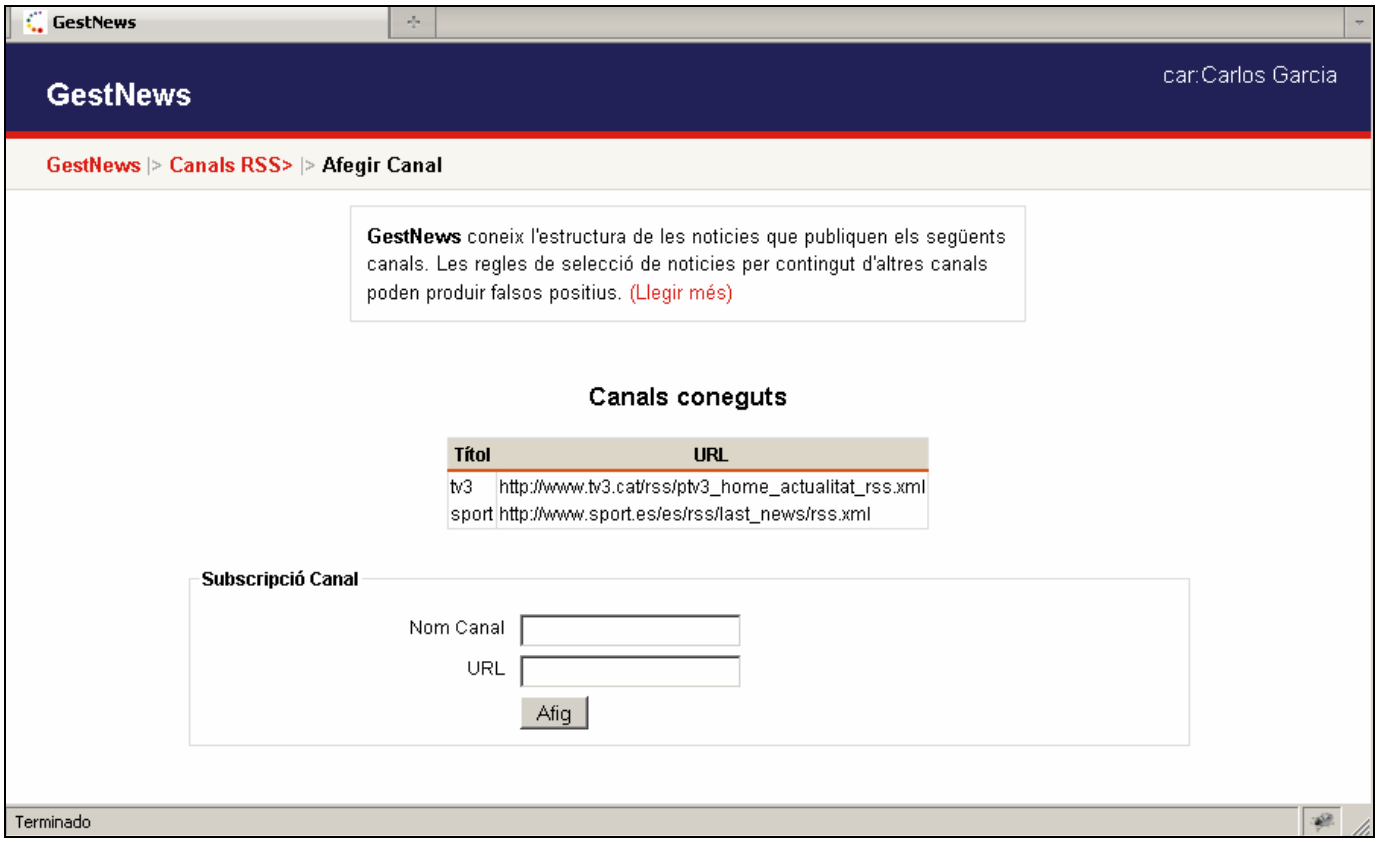


Figura 12.Vista Alta d'un Canal.

3.4. Mètode per extraure informació del canal RSS.

Com he indicat al punt 2.4.4 del capítol anterior, per extraure la informació del canal RSS vaig pensar en un principi crear un document **DOM** mitjançant JAXP i anar fent-li consultes XPATH per extraure els diferents ítems de les notícies. Posteriorment, vaig pensar que per aquest context era millor solució la utilització d'un parser en **SAX**, ja que realment es pot traure tota la informació del document RSS amb una única llegida del document en qüestió, i el sistema requereix de menys recursos de processador i memòria[2].

La metodologia emprada és un poc més complexa que mitjançant l'ús d'un document **DOM.** En el seu desenvolupament, l'analitzador utilitza un autòmat d'estat finit (modelat a la classe Automata,java). D'esta forma l'analitzador en tot moment pot conèixer l'element del document RSS que està analitzant.

L'aplicació implementa la classe **Canal,** que modela totes les funcions que es poden realitzar a un canal RSS. Entre aquestes funcions destaquen:

- Crear un canal. Es realitza amb el mètode setRegCanal(..) i té com objectiu emmagatzemar a la BBDD la informació referent a un canal.

[2] Per obtindre una discussió més detallada del tema ver el punt 2.4.4.

- Traure la informació publicada d'un canal. És realitza cridant al mètode getRSSInfo(). Aquest crea un parser SAX, amb un enllaç al fitxer amb el document RSS i un objecte a la classe que implementa el parser (ParserRSS);

La classe ParerRSS és l'encarregada d'analitzar el fitxer XML en format RSS. El seu funcionament és el següent:

- Crea un objecte de la classe Autòmat, on es recolza per fer el *parseig*.
- Rep un objecte de la classe CanalBean on emmagatzema informació referent al canal i un vector d'objectes de la classe NotíciaBean. Aquest vector rebrà, per cada notícia llegida, un d'aquests objectes amb tota la informació de la notícia. El vector serà replegat per la classe canal, quan sigui executat pel mòdul SCA, i el contingut dels objectes NotíciaBean emmagatzemat a la taula notícies de la BBDD.

  Es podria haver emmagatzemat a la BBDD cadascuna de les notícies a mesura s'anaven traient, però per fer el programa més didàctic, s'han tret totes les notícies a un vector NotíciaBean i emmagatzemar-les posteriorment. Aquest procés únicament es realitzat quan és executat pel mòdul SCA.

- A mesura que la classe va realitzant el parseig, l'analitzador SAX crida els següents mètodes:
  - Mètode **startElement**. S'activa quan detecta el començament d'un nou element. En aquest mètode s'identifica l'element trobat i es llança l'acció corresponent a realitzar per l'autòmat. L'autòmat en funció de l'estat en què es trobi i l'acció que se li demana realitzar passarà, si correspon, a un nou estat i si no es possible realitzar l'acció indicada, passarà a l'estat error. En aquest mètode també es comprova si la versió del format RSS és la 2.0.

    Quan es troba l'element **item**, es degut a què comença una nova notícia. Aleshores es crea un nou objecte NotíciaBean. Des d'aquest moment, tota la informació que s'hi vaja trobant, serà referent a aquesta notícia i s'emmagatzemarà en aquest objecte.
  - Mètode **endElement**. S'activa quan detecta la finalització d'un nou element. El procés és el mateix que en el cas anterior. Se li indica a l'autòmat l'acció a realitzar i aquest, si correspon, canviarà d'estat.
  - Mètode **characters**. El mètode és llançant quan dins d'un element hi ha informació. Aleshores, es consulta l'estat i el subestat de l'autòmat i s'actua en conseqüència. Si per exemple, es troba l'autòmat en l'estat **Item** i subestat **títol,** això es degut a que s'ha trobat el títol d'una notícia. Aleshores aquesta informació s'emmagatzemarà en l'últim objecte NotíciaBean emmagatzemant al vector d'objectes de NotíciaBean.

Finalment, comentaré un poc el funcionament de la classe **Automata**. La seva representació la podem veure a la figura 13. En aquesta figura podem veure que l'autòmat consta de quatre estats, i que es passa d'un estat a l'altre cridant al mètode ferAccio(..). Com a paràmetre se li passa l'acció a realitzar. Si se li passa una acció, que pel seu estat no es pot realitzar, passarà a l'estat **error**. Aquest error no s'ha pintat a la figura per simplificar-la.

S'ha establert el patró **AXY** per normalitzar la nomenclatura de les accions. On **X** representa l'estat en què s'entra o des d'on es surt, i la **Y** si ha d'entrar o eixir.

A més de l'estat, l'autòmat manté un subestat. Podem dir que l'estat determina els elements més importants del format RSS i els subestats els que contenen informació: títol, descripció, autor, etc.

A l'autòmat se li pot consultar l'estat i el subestat per conèixer el seu estat total. El seu estat total es consultat pel mètode **characters** por poder emmagatzemar la informació que li està aplegant.

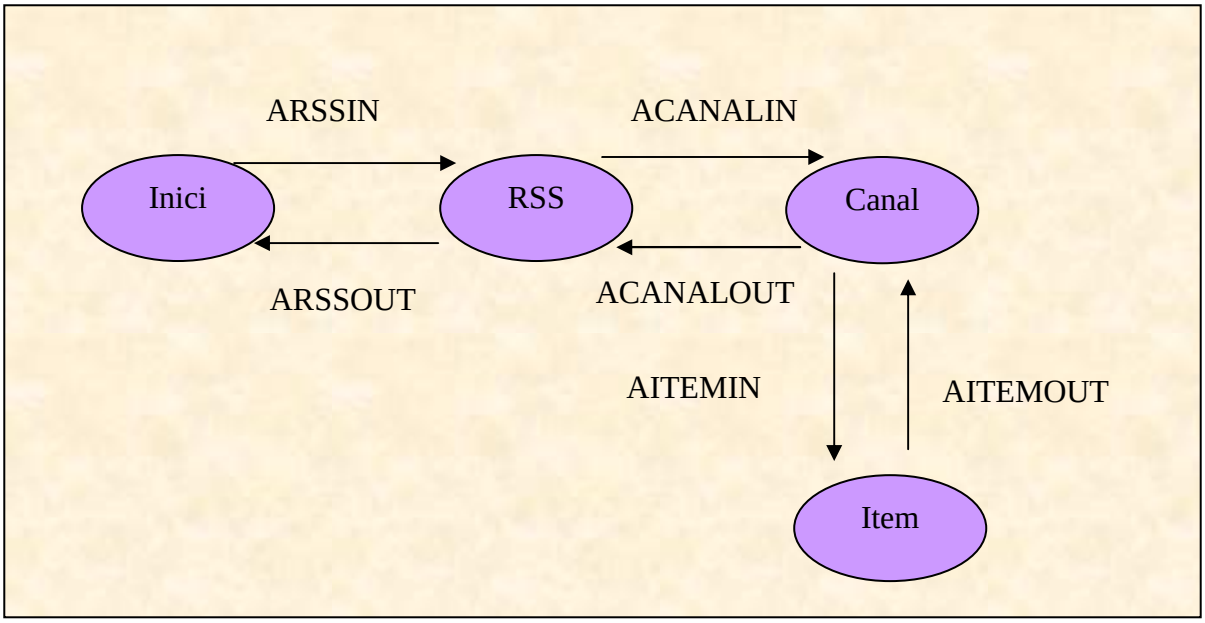


Figura 13. Autòmat extracció informació RSS.

3.5. Accés a la informació del canal RSS.

L'accés a la informació d'un canal RSS es realitza tant des del component GUAP com des del SCA. Tot i que l'objectiu del seu accés és un poc diferent en cada element.

En el primer cas (GUAP), es realitza amb la intenció de traure la informació per crear el canal, i en el segon cas (SCA) per sincronitzar les notícies publicades al canal amb la informació de la BBDD, i poder comprovar si les noves notícies compleixen les regles establertes al canal.

Als punts següent es detalla un poc més aquest procés.

- ***Creació del Canal***

El procés comença des de la vista que es mostra a la Figura 12. El sistema demana a l'usuari un identificatiu per al canal i la URL del canal on es vol subscriure's l'usuari i que conté el document XML en format RSS.

Posteriorment, la servlet executa el mètode intern addCanal(..) que crea un objecte de la classe Canal amb la informació treta del formulari. Des de l'objecte Canal es llança el mètode setRegCanal(..) que té com objectiu traure més informació sobre el canal (document RSS) i deixar aquest preparat perquè quan s'executi el servei SCA, es descarregui les notícies del canal que s'ha afegit (procés de sincronització de notícies).

Tot i que el mètode setRegCanal(..) crida al mètode getRSSInfo() que s'encarrega de tractar el document RSS i extraure informació del canal i un vector d'objectes **NotíciaBean** amb cadascuna de les notícies. Quan s'executa getRSSInfo() des de la servlet, el vector de **NotíciaBean** està buit, ja que en aquest cas el vector amb la informació de les notícies no és tractat pel mètode addCanal(..), però si serà omplert quan sigui executat des del procés de sincronització de notícies del component SCA. S'ha dissenyat així, ja que aquest procés requereix, per als canals coneguts, extraure també el contingut de la notícia i per tant requereix de prou processament[3]. Aleshores, el procés d'extraure les notícies i el seu contingut per als canals coneguts, es realitzat pel Servei de Cerca Automatitzat quan s'activi. Així, estalvia que l'usuari estigui esperant el tractament de les notícies i del seu contingut quan es doni d'alta el canal. És a dir, es un procés que queda pendent per ser realitzat per mòdul SCA.

Llavors, quan s'acaba d'executar el mètode addCanal() tenim a la BBDD la informació del nou canal (taula canals) i aquest preparat perquè posteriorment se carregui amb les notícies (activació SCA). També he d'indicar que abans d'emmagatzemar el canal, es comprova si aquest canal és un canal reconegut pel sistema (ver punt 3.3) i en cas afirmatiu s'emmagatzema també el XPATH amb l'adreça del contingut de la notícia. Aquest procés s'explica en detall al punt 3.6.

- ***Sincronització de les notícies.***

Com s'ha anomenat abans, des del component del Servei de Recerca Automatitzat s'accedeix a la informació publicada pel Canal RSS (document XML en format RSS ver 2.0) amb la intenció de sincronitzar les notícies publicades amb les emmagatzemades a la BBDD. El procés de sincronització té un doble objectiu:

- Eliminar les notícies que ja no són publicades.
- Afegir les noves notícies i marcar-les, per posteriorment comprovar únicament si aquestes compleixen les regles de selecció establertes al canal.

El procediment d'aquest component és el següent:

- Per cadascun dels canals, es crea un objecte canal i es crida al mètode updateNoticiesCanal().
- Des d'aquest mètode es crida al mètode GetRSSInfo(), ja explicat al punt anterior, per traure totes les notícies en un vector d'objectes NotíciaBean i posteriorment al mètode updateNoticies().

[3] Aquest procés es tractat més endavant al punt 3.6

- updateNoticies() desactiva el flag **activa** de cadascuna de les notícies d'aquest canal. Posteriorment es comprova si les notícies emmagatzemades al vector d'objectes NotíciaBean són emmagatzemades a la BBDD. En cas afirmatiu s'activa el flag **activa** i en cas negatiu s'emmagatzema amb aquest flag activat. Finalment, s'eliminen les notícies del canal que no tenen aquest flag actiu, és a dir, només quedaran a la BBDD les notícies publicades. Evidentment, si estem tractant d'un canal conegut, des de updateNoticies() es crea un objecte de la classe NotíciaContingut per extraure el contingut de la notícia i emmagatzemar-lo a la BBDD. Els detalls d'aquest últim procés es tractaran al punt 3.6.
- Amb aquests passos, aconseguim sincronitzar les notícies publicades al canal amb la BBDD.

3.6. Mètodes per extraure el contingut d'una notícia publicada.

En aquest punt, s'introdueixen els dos mètodes que són utilitzats pel sistema a l'hora de tractar el contingut d'una notícia publica per les regles de selecció per contingut, per exemple *contains(notícia,"Nadal").*

Com s'ha tractat al punt 3.3 d'aquest capítol, el sistema coneix l'estructura de determinats canals (els canals coneguts) i en aquests s'utilitzen un mecanisme diferent i més efectiu, que en els canals en què el sistema no coneix la seva estructura.

Canals d'Estructura Coneguda.

Cada canal RSS publica la seva notícia en un document HTML. Tots els documents publicats per un canal RSS segueix la mateixa estructura o plantilla al document HTML, però evidentment, canvia respecte altres canals RSS. Aleshores, s'ha d'implementar un mecanisme genèric, que conegui l'estructura del document HTML i sigui capaç d'extraure una part d'aquest, concretament la part que conté la notícia.

Existeixen diferents mecanismes per aconseguir aquest objectiu, i donades les seves complexitats, podrien ser perfectament l'objectiu d'estudi d'una PFC. Finalment, es va optar per una solució prou genèrica i des del meu punt de vista interessant. Es basa en el fet de què un document HTML és un cas particular d'un document XML[4].

La solució simplement consisteix en determinar l'adreça XPATH del contingut de la notícia en el document HTML publicat (la notícia publicada). L'adreça XPATH s'ha de conèixer en cadascú dels canals RSS on es vol que les regles de selecció de notícies per contingut funcionen rigorosament bé. Aquests canals són els que s'han anomenats canals coneguts en la present Memòria. Posteriorment, el sistema, en funció del canal RSS conegut en què s'estigui treballant, accedeix a l'adreça XPATH corresponent, extrau el contingut de la notícia i l'emmagatzema a la taula **notícies_contingut**.

A la imatge següent, es pot veure una notícia publicada en un canal que reconeix el sistema, és a dir, que el sistema coneix l'adreça XPATH del

[4] Aquesta afirmació de vegades no és certa totalment, ja que m'he trobat alguns canals que publiquen notícies en un document HTML mal forma. Veure més endavat.

contingut de la notícia. En aquest cas, el sistema trauria la informació que està recercada, és a dir, només el contingut de la notícia i l'emmagatzemaria a la BBDD. Aleshores, si l'usuari ha afegit una regla de contingut de notícia en aquest canal, només afectaria al contingut recercat i no a tot el document HTML. Evidentment, en els canals no coneguts s'ha de fer la validació de la regla en tot el document HTML com es comenta posteriorment.

Figura 14. Informació treta d'una notícia publicada.

A la imatge anterior s'ha recercat el contingut de la notícia que el sistema va extraure del document HTML publicat. Aquest contingut és el que s'emmagatzemà a la taula **notícies_conegut**. A la següent imatge, es pot veure el registre corresponent de la taula **notícies_conegut** de la notícia mostrada a la figura anterior. Comentar que s'emmagatzema a la BBDD aplicant la codificació emprada en els documents xml (com es pot veure als accents i als apòstrof).

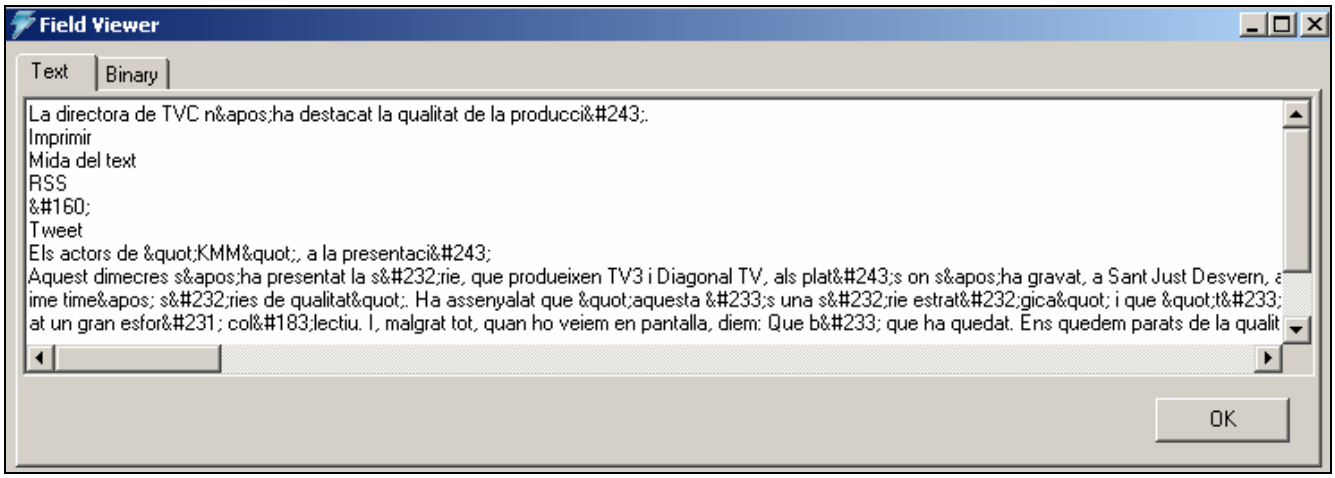


Figura 15. Contingut de la notícia emmagatzemada a la BBDD.

Canals d'Estructura No Coneguda.

En els canals on el sistema no coneix l'estructura del document html publicat, no es pot traure el contingut de la notícia publicada. Aleshores, el sistema, per aquests canals, no trau el contingut de les notícies en el procés de sincronització de notícies. Ho fa la màquina virtual en el moment de l'execució d'una regla de selecció per contingut de notícia. Aquest procés s'explica detalladament al capítol 6.

El motiu de fer-ho així es per aconseguir un sistema més robust. El sistema, quan executa una regla de selecció per contingut de notícia, actua de la següent forma:

- Si és un canal conegut, accedeix a la taula **notices_contingut** per obtindre el contingut de la notícia. Però pot donar-se el cas de què l'analitzador sintàctic SAX detecti un error per una mal formació del document html que conté la notícia[5]. Aleshores, el sistema el tracta com una notícia d'un canal no conegut.
- Si és un canal no conegut (o s'ha produït un error en l'anàlisi) se descarrega la notícia, li lleva certs contingut no vàlids (scripts, comentaris, etc.) i finalment els tags. I en el fitxer resultant s'aplica la regla de selecció corresponent.

Veure capítol 7 per una informació més detallada.

3.7. Implementació per extraure el contingut d'una notícia (canal conegut).

Com s'ha indicat al punt anterior, el sistema accedeix al contingut de la notícia per la seva adreça XPATH. Aleshores, existeix una taula a la BBDD anomenada **canals_coneguts** on s'emmagatzemen els canals que reconeix el sistema, i en un dels camps es desa el XPATH del contingut de les notícies que aquests canals publiquen.

El punt més delicat es com s'ha implementat en el sistema perquè pugui accedir al contingut de la notícia en un document HTML mitjançant el XPATH. La solució més directa és la utilització d'un document DOM. Evidentment, aquesta seria la solució més senzilla, però té un problema molt gran. Els documents html publicats són documents XML molts complexes i grans. Seria totalment impossibles tractar-los amb documents DOM. A més, per complicar-ho tot més, moltes vegades aquests documents XML no són ben formats.

La solució final que es va implementar, després de madurar-la molt, és prou semblant a la utilitzada per tractar els documents RSS. Consisteix en un analitzador SAX recolzat en un autòmat. A continuació es detalla la solució.

[5] Increïblement, en les proves he detectat dos canals que publiquen notícies en documents html normalment mal formats. No són error greus i els analitzadors dels navegadors no tenen problema, però SAX els detecta i genera un error fatal. Són errors del tipus tags no tancats.

- ***Descàrrega del document HTML publicat.***

En el mètode updateNoticies() que hi ha dins de l'objecte canal i del qual ja s'ha parlat anteriorment, quan es detecta que s'està tractant una notícia d'un canal conegut, es crea un objecte NotíciaConegut i aquest s'encarrega de traure el contingut de la notícia i emmagatzemar-la a la BBDD.

Per poder parsetjar d'una forma adient el document HTML que conté la notícia publicada, es descarrega el document HTML al disc dur local mitjançant el seu enllaç. Aquest procés es realitza pel mètode downLoadWithOutScript(...) de la classe **UtilsFile.** Aquest mètode, a mesura que es descarrega la notícia, elimina certs elements que no són necessaris per al tractament posterior i que poden produir que el document HTML no sigui un document XML ben format. Aleshores, per eliminar problemes posteriors en el parseig del document s'eliminen:

- **Scripts:** Poden incloure informació que afecta de forma errònia l'anàlisi del parser i el tractaria com un document XML mal format i no es podria analitzar.
- **URL:** Poden incloure caràcters que no poden formar part d'un document XML ben format.

A mesura l'analitzador va trobant els diferents elements del document HTML analitzat, va llançant el mètode ferAccio(). Aquest mètode té un paràmetre acció, que li indica l'acció a realitzar:

- ***Funcionament de l'autòmat finit.***

A la classe **Automata2** està modelat l'autòmat finit que es utilitzat per l'analitzador per extraure el contingut d'una notícia del document HTML publicat.

La funció d'aquest autòmat és crear un arbre amb els elements (**tags**) que composen el document HTML a mesura l'analitzador els va trobant. Així, imaginem un document HTML amb la següent estructura mínima:

```
<html>
<head>
<meta content="text/html; charset=ISO-8859-1" http-equiv="content-type">
<title>Title</title>
</head>
<body>
<p>hola</p>
</body>
</html>
```

L'arbre que es crearia a la memòria de l'autòmat es mostra a la següent figura:

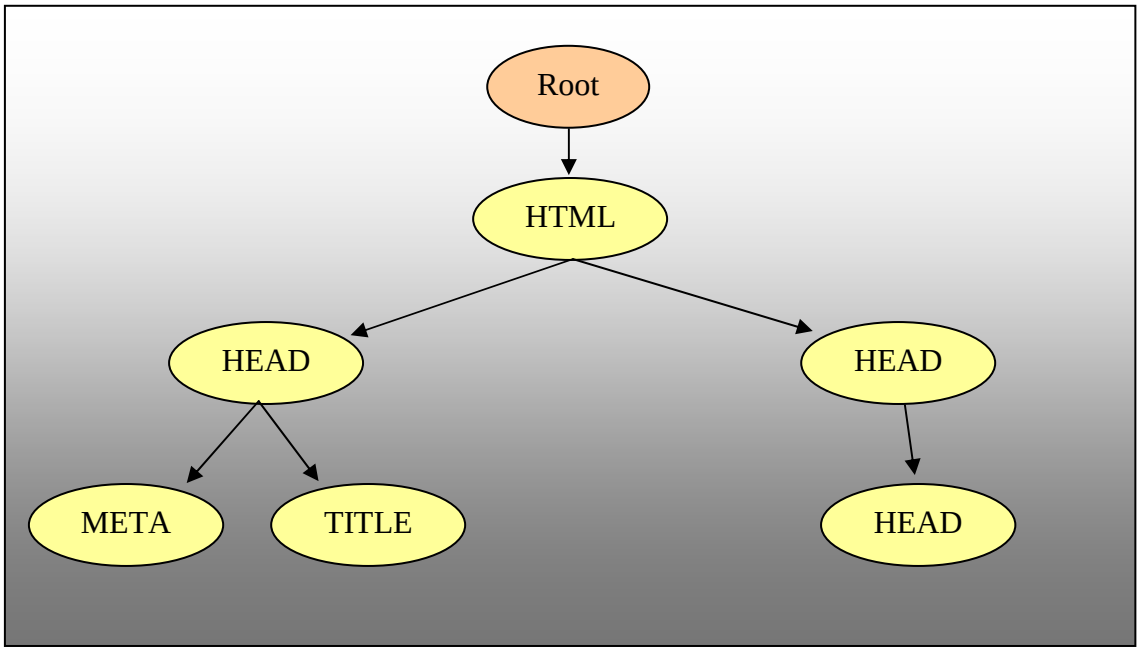


Figura 16. Arbre d'una estructura bàsica d'un document HTML .

A mesura l'analitzador va trobant els diferents elements del document HTML analitzat, va llançant el mètode ferAccio(). Aquest mètode té un paràmetre acció, que li indica l'acció a realitzar:

- **IN:** És llançat quan l'analitzador troba un nou element. Aleshores, es crea un nou node amb el nou element trobat i es penja del node actual. En un document HTML és possible trobar en un mateix nivell diferents elements (tags) amb el mateix nom, per exemple diferents <**div**> en el mateix nivell. Aleshores, tots els elements penjants tenen un índex dins del seu pare. Seguint amb el mateix exemple, el primer **div** dins d'un determinat pare, serà emmagatzemat amb **div**[0], si es troba un altre **div** dins d'aquest, serà emmagatzemat amb **div**[1] i així successivament.

  Així, cada vegada que es llança el mètode ferAccio() amb el paràmetre **in**, es comprova entre tots els elements trobats anteriorment en el pare (els seus fills) si existeix aquest element. En cas afirmatiu s'extrau l'índex del més gran i s'incrementa en 1, i en cas negatiu se li fica l'índex 0. Finalment, es crea el node amb aquest element i el seu índex. També el deixa com estat actual.

  Com que el codi es prou aclaridor el mostre a continuació:

```
if (accio.equals("in")){
    Enumeration e = nodeAct.children();
    max=0;
    //Busque en tots els fills del pare si exiteix l'element
    while (e.hasMoreElements()) {
            DefaultMutableTreeNode n = (DefaultMutableTreeNode)e.nextElement();
            cad= (String) n.getUserObject();

            // Obtenim un Pattern amb l'expresión regular, i d'aquest
            // un Matcher, per extraure l'index
            Pattern patron = Pattern.compile(name+"\\[(\\d+)\\]");
            Matcher matcher = patron.matcher(cad);
```

```
            // Si troba l'expresio
            if (matcher.find()){
                  int val=Integer.parseInt(matcher.group(1));
                  if (val>max)
                        max=val;
            }
      }
      max++;
      //Es crea el nou node
      DefaultMutableTreeNode node= new DefaultMutableTreeNode(name+"["+max+"]");
      //Se li penja al node actual.
      nodeAct.add(node);

      nodeAct=node;

}
```

- **OUT:** És llançat quan l'analitzador ix d'un nou element. L'autòmat simplement comprova que l'element del qual ha d'eixir és l'estat actual i si és correcte puja un nivell en l'arbre, és a dir, deixa al seu pare com estat actual.

```
    cad= (String) nodeAct.getUserObject();
    cad=cad.replaceFirst("\\[\\d+\\]","");
    if (!name.equals(cad)){
          Stat="Error";
    } else{
          if (nodeAct!=root)
                nodeAct=(DefaultMutableTreeNode) nodeAct.getParent();
    }
```

L'autòmat disposa del mètode getStat(). El seu objectiu es retornar l'adreça XPATH de l'estat (node) actual. Aquest estat es utilitzat per l'analitzador per comprovar si hem aplegat al XPATH on està el contingut de la notícia.

- ***Funcionament de l'analitzador.***

A la classe **ParserNoticiHTML** s'implementa l'analitzador dels documents HTML.

La classe crea un objecte autòmat2 (ja explicat al punt anterior) i rep l'adreça XPATH del contingut de la notícia.

Cal destacar la sobreescriptura del mètode startElement () que es llançant quan l'analitzador detecta un nou element en el document XML (HTML). Aquest nou element es enviat a l'autòmat perquè reflecteixi el seu nou estat (afija el nou element a l'estructura d'arbre). Posteriorment, se li pregunta a l'autòmat pel seu estat. Si aquest coincideix amb l'adreça XPATH de la notícia, ja tenim la notícia i activem el flag **Trobat**. Aquest flag és utilitzat pel mètode characters(), quan està actiu, per emmagatzemar el contingut de la notícia.

El mètode endElment() simplement informa a l'autòmat que ha d'eixir de l'element corresponent.

Remarcar que després de treballar amb diferents canals RSS i analitzar moltes notícies que aquests canals han publicat, he comprovat que es publiquen documents HTML mal formats. He trobat errors de diferent tipus: com incloure caràcters no admesos en documents XML que no han segut correctament escapats, tag amb atributs mal formats i fins i tot, tags oberts que no s'han tancat.

Després d'aquest estudi i tractar amb aquest tipus d'errors, m'ha sorprès la gran versatilitat i flexibilitat que tenen els analitzador que incorporen els navegadors actuals, ja que treballen correctament amb documents HTML mal formats.

En el cas concret de la PFC, s'ha hagut de lluitar molt en aquest tipus d'errors per aconseguir que l'analitzador sigui el més robust possible i li afecten el mínim possible d'aquest tipus d'errors. La lluita amb l'anàlisi dels documents XML mal formats s'ha realitzat en dues direccions:

- Eliminar al màxim els errors del document HTML o possibles errors en la descàrrega. Aquest punt ja ha segut tractat al primer punt de l'apartat 3.7.
- Sobreescriure els mètodes que tracten els errors de l'analitzador per poder-los tractar des de dins del sistema i evitar que afecten al seu funcionament.

Des del meu punt de vista, el procés d'extracció d'informació d'un document HTML té una complexitat pròpia per ser tractada des d'una PFC. He intentat aprofundir en aquesta problemàtica al màxim possible, dins de les limitacions de temps establert per aquest projecte. Però considero que en quant a la flexibilitat del tractament de documents HTML / XML mal formats es podrien aconseguir millors resultats.

4. Capítol 4. Model de Dades.

4.1. Introducció

Com s'ha explicat en diferents punts del capítol 2, el tractament de la persistència de les dades en GestNews es realitza mitjançant un gestor de Base de Dades. Concretament s'ha emprat MySQL. La justificació de la seva elecció es detalla dins del punt 2.2 del mateix capítol.

En aquest capítol es mostra el model de dades mitjançant un diagrama Entitat / Relació, les classes java que s'encarreguin de la seva persistència i les propietats més importants d'aquestes.

4.2. Tractament de la Persistència.

Per al tractament de la Persistència de l'aplicació, s'han creat objectes en java per modelar els diferents elements existents en l'univers en què es mou l'aplicació: usuaris, canals, etc. Tots aquests objectes contenen una propietat amb el seu corresponent objecte Bean: userBean, canalBean, etc i mètodes encarregats de la seva persistència com: setRegXXX() i getXXX per a desar i recuperar els Bean a la BBDD. A més, d'altres mètodes per modelar el seu comportament.

Les classes Bean representen objectes amb l'objectiu de ser seralitzats o desats. Només conten propietats referents als objectes que representen i els mètodes setters i getters per poder treballar amb elles. Aquest model es pot veure a la següent imatge.

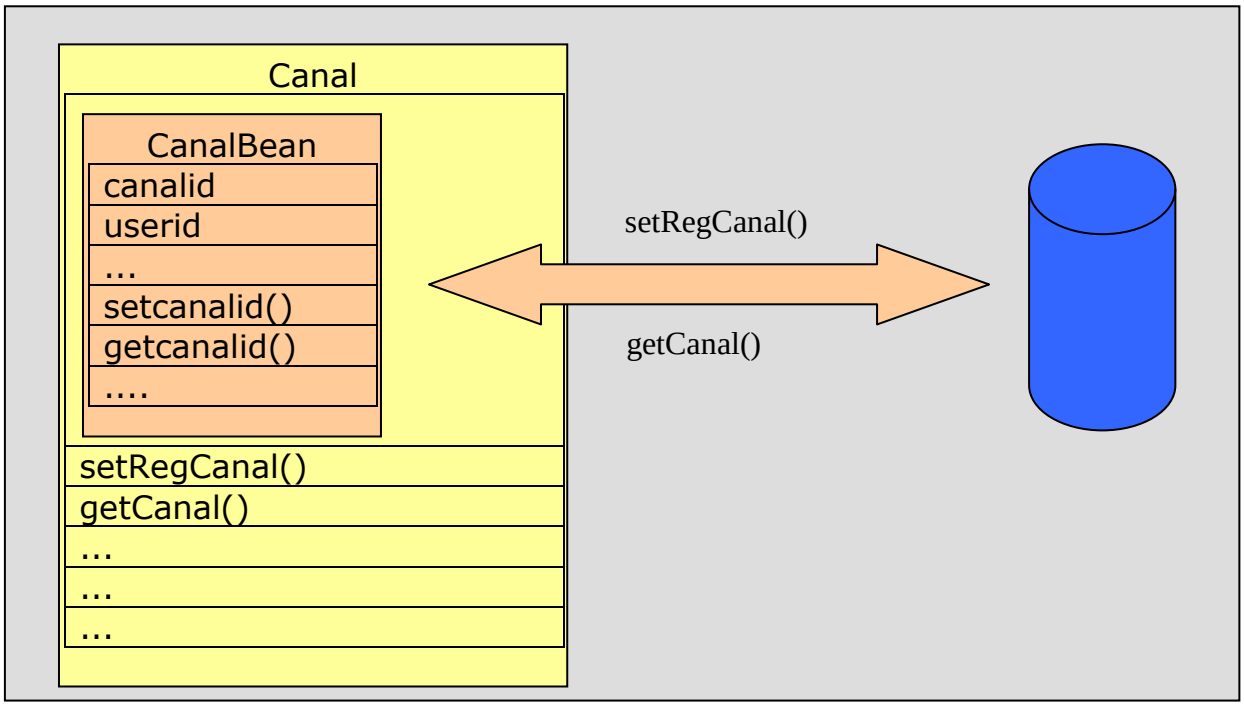


Figura 17.Classe Canal amb la propietat CanalBean.

4.3. Diagrama Entitat / Relació.

El model de dades de l'aplicació es prou senzill. Com es pot comprovar al següent diagrama, implementa bàsicament la persistència dels objectes Bean. També s'han implementat altres entitats amb l'objectiu d'ajudar i facilitar el desenvolupament de l'aplicació: canals_coneguts, notícies_conegudes.

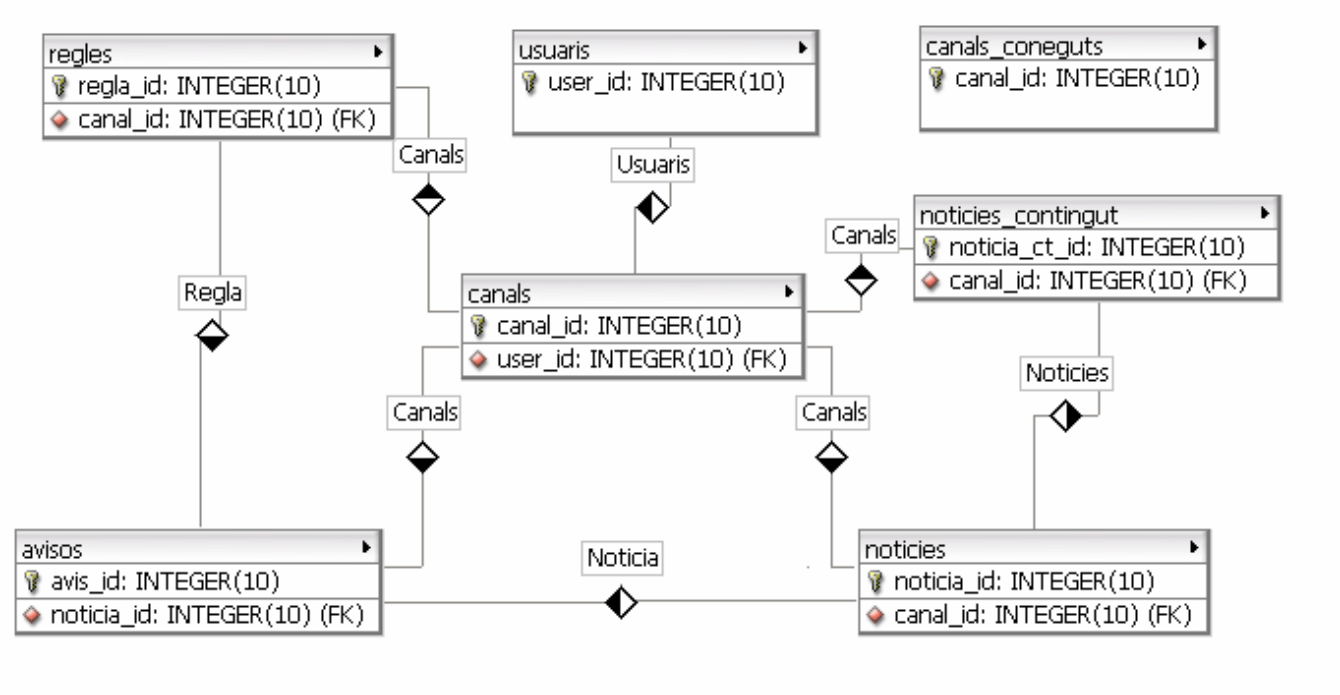


Figura 18.Diagrama E/R.

4.4. Detall de les entitats i les seves relacions..

Una vegada vista la persistència de les classes de l'aplicació i la seva relació amb les entitats implementades a la Base de Dades, es va a detallar les propietats de les entitats més significatives i les relacions més importats:

Relació usuaris - canals.

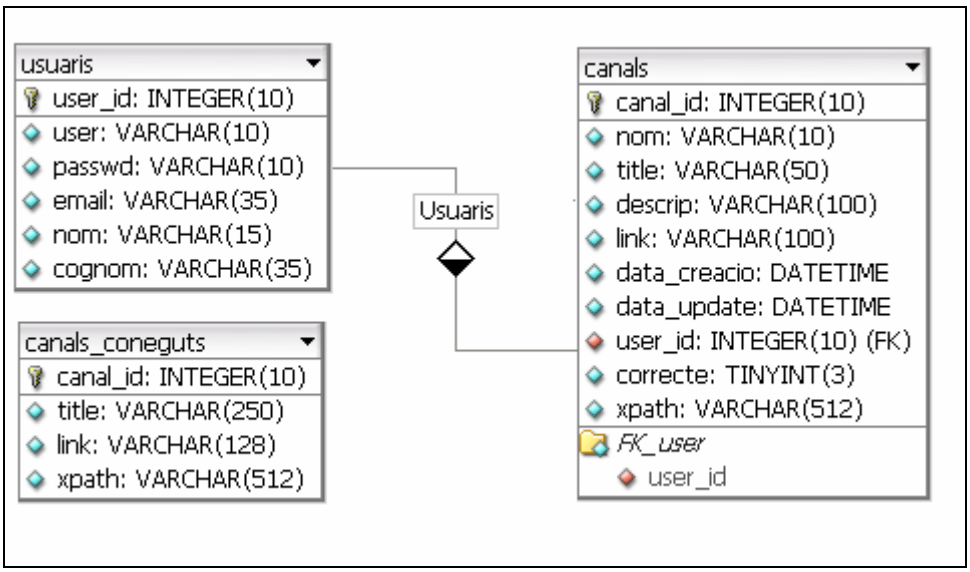


Figura 19. Relació Usuaris Canals.

És una relació prou evident. Únicament destacar el camp **XPATH** de la taula Canals. Aquest camp serà **null** en els canals no coneguts pel sistema, però tindrà l'adreça XPATH del contingut de la notícia per als documents HTML que publica aquest canal. Aquesta informació es trau de la taula **Canals_coneguts** quan es doni d'alta el canal.

Aleshores, per crear un nou canal conegut, simplement s'ha de crear una entrada a la taula **Canals_conegut** amb la XPATH del contingut de la notícia publicada al canal, el seu nom i la seva url.

El camp **correcte** és activat quan s'ha donat d'alta correctament el canal.

Relació canals - notícies

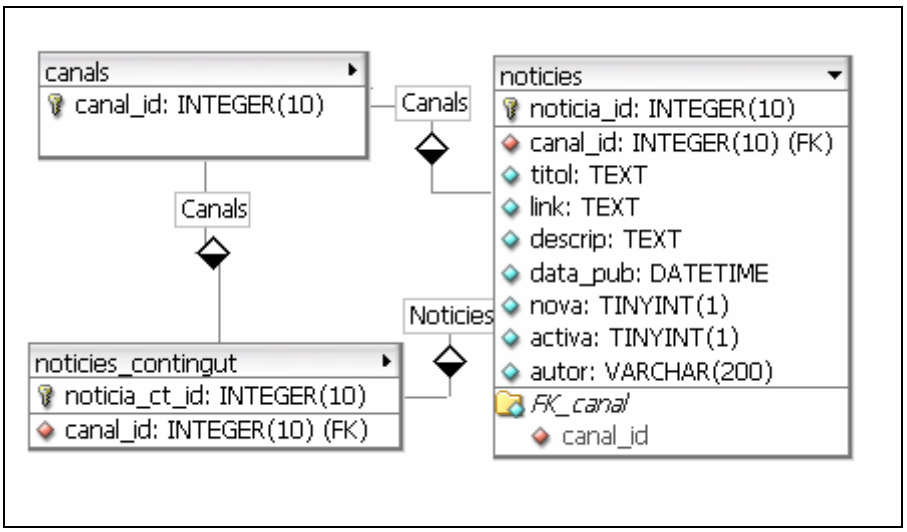


Figura 20. Relació Canals - Notícies

Es pot veure, que referent a les notícies hi ha dues taules. La taula **Notícies**, que conté informació de les notícies tretes del document RSS i la taula **Notices_contingut** que conté el contingut de les notícies tretes de la notícia publicada, però únicament dels canals coneguts pel sistema. De la resta, el contingut de la notícia no està emmagatzemat en aquesta taula. Per més informació veure el punt 3.6 capítol 3.

El camp **activa** de ambdues taules és utilitzat pel sistema per eliminar les notícies que ja no es publiquen en el canal. En el procés de sincronització, primerament es desactiva aquest flag en totes les notícies del canal. A mesura que es van traient notícies del document RSS, es van identificant les existents a la taula **Notícies** i es va activant aquest flag. Al finalitzar el procés, s'eliminen les notícies que no tenen aquest flag actiu, és a dir, les notícies que ja no són publicades. Entre la taula **Notícies** i **Notícies_contingut** hi ha establerta una Integritat Referencial en cascada. Aleshores quan s'elimina de la taula **Notícia** una notícia per no estar publicada, s'elimina automàticament el seu contingut de la taula **Notícies_contingut**.

El camp **nova** de la taula **Notícies**, es activat quan una notícia es donada d'alta en el sistema. Aleshores, quan s'executen les regles de selecció en el canal al qual correspon aquesta notícia, únicament s'executaran sobre les notícies marcades com a noves, ja que les altres ja han segut verificades anteriorment.

Relació canals – regles

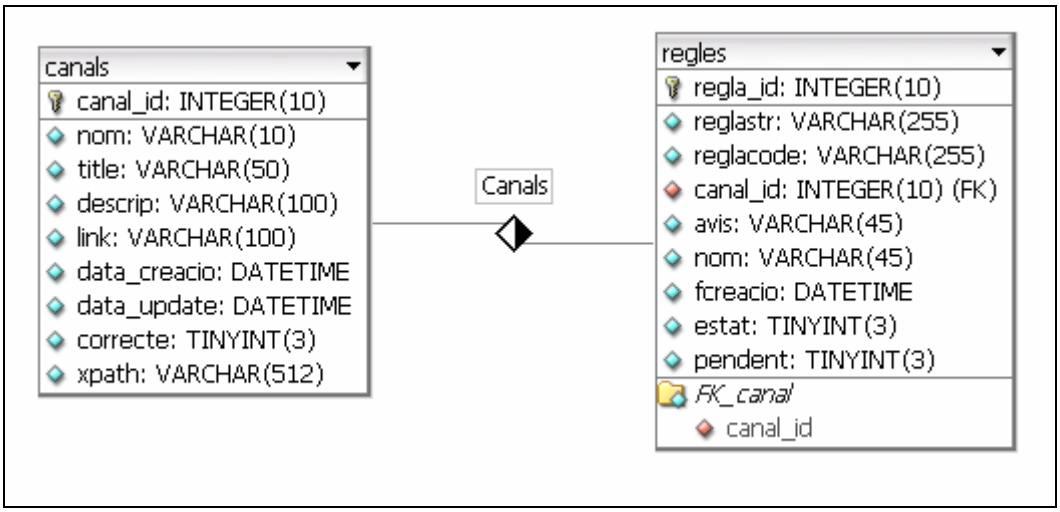


Figura 21. Relacions Canals – Avisos i Canals - Regles

Amb aquesta relació s'estableix les regles de selecció que afecten a un determinat canal. Quan l'usuari crea una regla, l'expressió introduïda per l'usuari és emmagatzemada a la propietat **reglastr**, a més, el sistema compila aquesta regla mitjançant la classe **Analitzador.** Si és correcta, genera l'expressió compilada i l'emmagatzema a la propietat **reglacode**. Activa el flag d'**estat** i el de **pendent**. Si l'expressió no és gramaticalment correcta, aquests dos flag no són activats.

El flag **estat** indica que l'expressió és gramaticalment correcta i que el sistema ha generat una expressió compilada equivalent (reglacode). Aquesta expressió compilada és la que executa el sistema.

El flag **pendent** indica que l'expressió està pendent de processar por el sistema. Aleshores, quan s'activi el component SCA cercarà les regles pendents de processament i les executarà en totes les notícies del canal corresponent. Per més informació veure el capítol 5 (regles de selecció) i capítol 6 en el punt 6.2 (execució de les regles de selecció).

Relació canals – Avisos

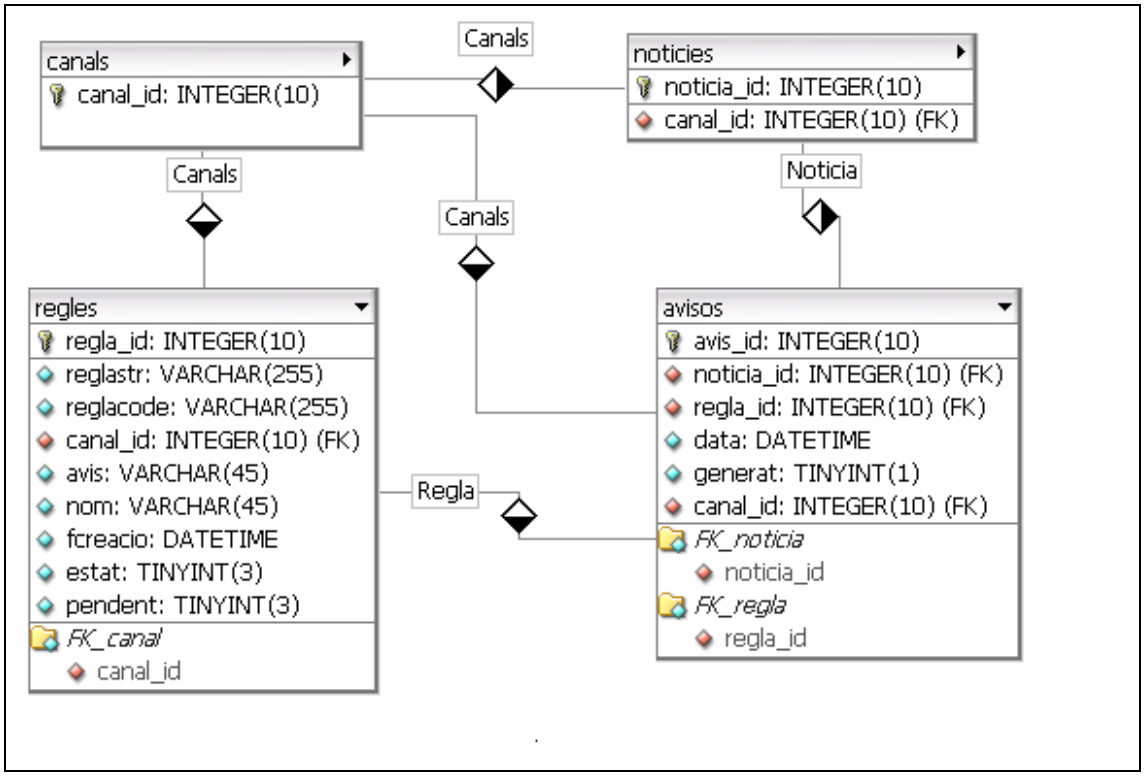


Figura 22. Relacions Canals – Avisos

L'entitat **Avisos** està relacionada amb les entitats Canals, Regles i Notícies. Com es pot veure, la relació Canals – Avisos sembla redundant, ja que es pot traure per la relació Canals – Regles o Canals – Notícies, però he preferit crear-la per simplificar el codi de l'aplicació.

Quan la Màquina Virtual executa una regla i aquesta té un resultat positiu, es genera un registre a la taula **avís** amb el flag **generat** desactivat. Aquest flag indica al sistema si l'avís s'ha generat, és a dir, s'ha enviat a l'usuari. Així, el component SCA busca els avisos amb el flag **generat** desactivat i el tipus d'avís que ha establert l'usuari per aquesta regla (camp **avís** de la taula **Regles**). Si l'avís es genera correctament (s'envia a l'usuari), s'activa aquest flag. Per més informació veure el capítol 6 en el punt 6.4 (generació d'avisos).

5. Capítol 5. Regles de selecció de notícies.

5.1. Introducció

Una regla de selecció és una expressió lògica que els usuaris introdueixen al sistema per determinar el tipus de notícies que les interessa. Posteriorment, el sistema avalua l'expressió amb cadascuna de les notícies per determinar aquelles que la compleixen. Les notícies que compleixen l'expressió seran notificades a l'usuari pel canal que aquest haja determinat.

En aquest capítol s'estudia com tracta el sistema la gestió de les notícies i també la seva implementació.

El capítol es divideix en dues parts:

- **Compilació**: S'explica com GestNews analitza i verifica la correcció de les expressions introduïdes per l'usuari i com converteix aquesta expressió en una expressió més fàcil d'executar pel sistema.

- **Execució**: En aquest apartat es detalla com el sistema executa les expressions prèviament traduïdes pel sistema.

5.2. Compilació de les regeles.

Com s'ha introduït al punt anterior, aquest procés comença quan el sistema rep l'expressió lògica introduïda per l'usuari en la vista Editar/Afegir Regla. Aquesta vista la podem veure a la figura següent:

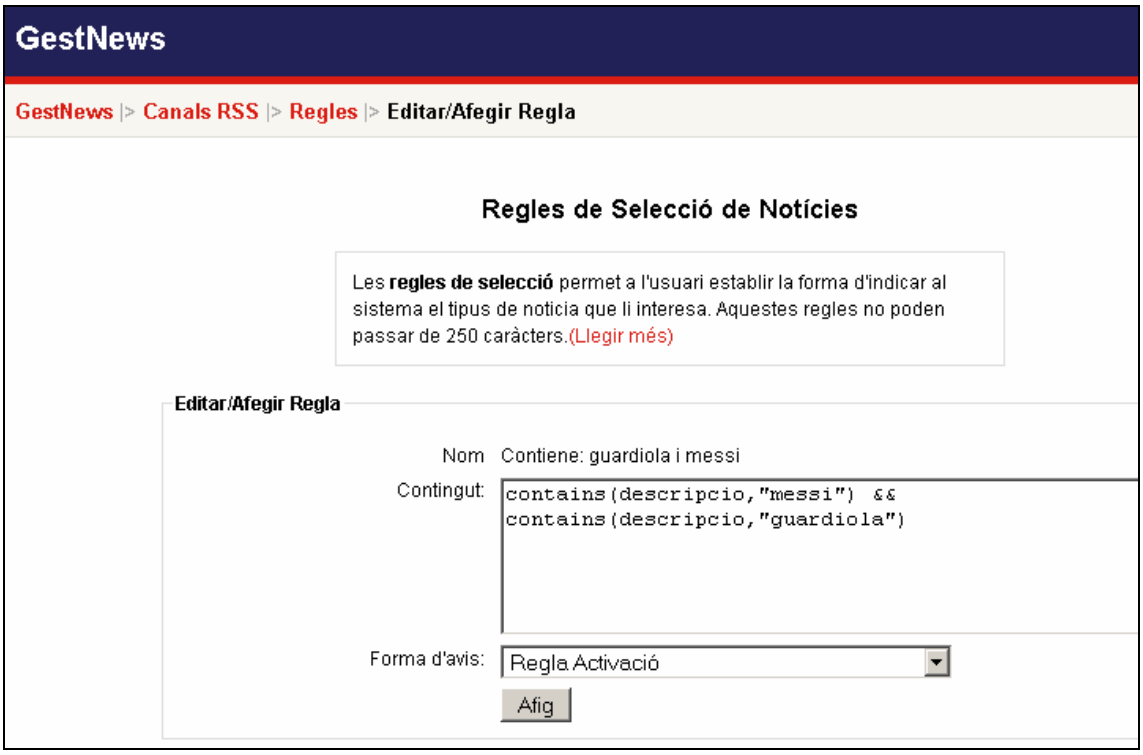


Figura 23.Vista Editar/Afegir Regla de selecció.

L'usuari ha d'introduir el nom amb el qual el sistema ha de reconèixer la regla, la expressió lògica de la regla i el tipus d'avís que el sistema enviarà a l'usuari quan es compleixi la regla.

Una vegada l'usuari polsa el botó **Afig**, el sistema realitza un procés complet de compilació d'aquesta regla, és a dir, realitza un anàlisis sintàctic, lèxic i finalment una traducció d'aquesta regla a una expressió més senzilla per ser executada. Tota aquesta informació és emmagatzemada pel sistema a la taula **regles**.

5.3. Operadors i gramàtica.

Tot i que el compilador no ha de reconèixer una expressió gramatical massa complexa, he decidit utilitzar **jlex** i **cups** per implementar el compilador. En un primer moment vaig pensar en desenvolupar-lo tot des de zero (autòmat d'estat finit per realitzar l'anàlisi lèxic, etc.) però posteriorment vaig decidir recolzar-me en aquestes eines per facilitar-me les coses.

El primer que em vaig plantejar va ser els operadors de cadenes que volia implementar en les regles i posteriorment els operador lògics. Una vegada determinat aquests aspectes, vaig dissenyar la gramàtica que definia el compilador:

- Els operadors de cadenes tenen dos paràmetres: **origen** que determina el origen de dades: títol, autor, descripció i notícia, i el paràmetre **cadena** que determina la cadena a cercar en el títol, autor, etc. Finalment, els operadors que reconeix el compilador són:
  - **begin(origen, cadena):** Vertader si es troba la cadena al començament de l'origen.
  - **end(origen, cadena):** Vertader si es troba la cadena al final de l'origen.
  - **contains(origen, cadena):** Vertader si es troba la cadena en l'origen.
  - **matches(origen, cadena):** Vertader si l'expressió regular que hi ha a la cadena encaixa en l'origen.
- Els operadors lògics: Són els operadors lògics de l'àlgebra de BOOLE: && (AND), || (OR) i ! (NOT)
- A més dels operadors indicats anteriorment, també s'han inclòs els parèntesis.

Exemples d'expressions que ha d'acceptar la gramàtica:

- begin(títol, "amèrica")
- begin(títol, "amèrica") && begin(descripció, "amèrica")
- (begin(títol, "amèrica") && begin(descripció, "amèrica")) || contains(notícia, "amèrica")
- contains(notícia, "picasso") && !contains(notícia, "dalí")

Finalment la gramàtica que defineix el compilador és la següent:

regla:=	ecuacio
ecuacio:=	expr AND expr | expr OR expr | expr
expr:=	terme | NOT expr
terme:=	(ecuacio) | operacio
operacio:=	BEGIN(oper1,oper2) | END(oper1,oper2) | CONTAIN(oper1,oper2) | MATCHES(oper1,oper2)
oper1:=	TITOL | AUTOR | DESCRIPCIO | NOTÍCIA
oper2:=	CADENA

En la gramàtica anterior, els terminals estan en majúscules.

El compilador tradueix les expressions de la gramàtica anterior, en expressions postoperacionals, en les quals primerament apareixen els paràmetres i posteriorment l'operador. Així, les expressions anteriors donades com a exemples es traduïren per:

- _titol__;_amèrica_;__beg_
- _titol__;_amèrica_;__beg__;__descrip__;_amèrica_;__beg__;__&&_
- _titol__;_amèrica_;__beg__;_descrip_;_amèrica_;__beg__;__&&__;_notícia _;_Amèrica_;__cont__;__||__
- _notícia__;_picasso_;__cont__;__notícia__;_dalí_;__cont__;__!__;__&&_

Destacar que tos els operands i operadors, amb excepció de les cadenes van dins de dos caràcters barra baixa i els diferents elements estan separats per la cadena "_;_". El afegir aquests caràcters facilita la separació dels elements en el procés d'execució.

També s'ha d'indicar, que aquest tipus d'expressions postoperacionals faciliten el procés de la seva execució com es podrà veure a l'apartat posterior 4.5.

Quan el compilador analitza correctament l'expressió lògica introduïda per l'usuari, genera el codi postoperacional i emmagatzema els dos a la taula **regles**. Finalment envia a l'usuari a la vista de regles i marca la regla com a pendent. Es pot veure a la imatge següent:

Figura 24. Vista Regles afegides correctament.

A la imatge de la Figura 24, es pot veure que la regla segona s'ha donada d'alta correctament i que està pendent de ser executada. Tanmateix, la regla primera no és correcta i és marcada amb roig. Amb l'enllaç que hi ha al seu nom es pot modificar.

Les regles donades d'alta de forma errònia, tot i que estan marcades com a pendent, no seran executades pel sistema fins que no canvien a l'estat verd.

5.4. Implementació del compilador

Dins del paquet **gestnews.core.analitzador** està el codi de les classes que implementen l'analitzador.

A continuació es comenta a grans trets el seu contingut i els aspectes que es considera de més interès:

- **a_lex.lex** És el fitxer de jlex que proporciona el codi per implementar l'analitzador lèxic. Aquest fitxer disposa del codi per generar la classe Alex.java (Analitzador lèxic) compatible amb **cups**. D'aquest fitxer es considera interessant la secció de les regles lèxiques:

```
%state STRING_LITERAL

blank=[\n\r\t ]

%%

<YYINITIAL>{blank}+              {}

<YYINITIAL>begin                 { return new Symbol(sym.TK_BEG); }
<YYINITIAL>end                   { return new Symbol(sym.TK_END); }
<YYINITIAL>contains              { return new Symbol(sym.TK_CONT); }
<YYINITIAL>matches               { return new Symbol(sym.TK_MATCH); }
<YYINITIAL>t[ií]tol              { return new Symbol(sym.TK_TITOL); }
<YYINITIAL>autor                 { return new Symbol(sym.TK_AUTOR); }
<YYINITIAL>descripci[oó]         { return new Symbol(sym.TK_DESCRIP); }
<YYINITIAL>not[ií]cia            { return new Symbol(sym.TK_NOTÍCIA); }

<YYINITIAL>"&&"                  { return new Symbol(sym.TK_AND); }
<YYINITIAL>"||"                  { return new Symbol(sym.TK_OR); }
<YYINITIAL>!                     { return new Symbol(sym.TK_NOT); }
<YYINITIAL>"("                   { return new Symbol(sym.TK_LEFT_BRACKET); }
<YYINITIAL>")"                   { return new Symbol(sym.TK_RIGHT_BRACKET); }
<YYINITIAL>","                   { return new Symbol(sym.TK_COMMA); }
<YYINITIAL>\"                    { yybegin(STRING_LITERAL); }

<STRING_LITERAL>[\n\r]  { throw new ErrorException("Error Sintàctic: símbol no admés");}
<STRING_LITERAL>\"               { yybegin(YYINITIAL); }
<STRING_LITERAL>[^\n\r\"]+       { return new Symbol(sym.TK_STRING,new String(yytext())); }
<STRING_LITERAL>.                { throw new ErrorException("Error Sintàctic: símbol no admés"); }

<YYINITIAL>.                     { throw new ErrorException("Error Sintàctic: símbol no admés"); }
```

Les regles lèxiques mostrades són prou intuïtives. Únicament destacar que s'ha definit l'estat <STRING_LITERAL> per tractar les cadenes de text que se li passen a les funcions de cadenes. També indicar que es llança una excepció per tractar les errades dels símbols no admesos.

- **a_sintac.cup:** És el fitxer amb l'estructura de CUPS per generar el analitzador sintàctic, semàntic. A més, també inclou el codi per la traducció d'expressions. Comentar la secció de codi d'usuari on s'ha rescrit les funcions de captura d'errors de l'analitzador i que generen excepcions que capturarà la classe Analitzador (es comenta posteriorment).

  Evidentment, destacar la secció de la gramàtica amb les seves regles de traducció.

```
//terminals
terminal TK_BEG,TK_END,TK_CONT,TK_MATCH;
terminal TK_AND,TK_OR,TK_NOT;
terminal TK_TITOL,TK_AUTOR,TK_DESCRIP,TK_NOTÍCIA;
terminal TK_LEFT_BRACKET,TK_RIGHT_BRACKET,TK_COMMA;

terminal String TK_STRING;

//no terminals
non terminal expr, terme, operacio, oper1, oper2, regla, ecuacio;

//Gramàtica
regla::=        ecuacio:e {: parser.setResultat(e.toString()); :};

ecuacio::=      expr:e1 TK_AND expr:e2
                        {: RESULT=new String(e1+"_;_"+e2+"_;__&&_"); :}
                | expr:e1 TK_OR expr:e2
                        {: RESULT=new String(e1+"_;_"+e2+"_;__||_"); :}
                | expr:e
                        {: RESULT=e.toString(); :};

expr::=         terme:e {: RESULT=e.toString(); :}
                | TK_NOT expr:e
                        {: RESULT=new String(e+"_;__!_"); :};

terme::=        TK_LEFT_BRACKET ecuacio:e TK_RIGHT_BRACKET
                        {: RESULT=e.toString(); :}
                | operacio:e
                        {: RESULT=e.toString(); :};

operacio::=     TK_BEG TK_LEFT_BRACKET oper1:e1 TK_COMMA oper2:e2 TK_RIGHT_BRACKET
                        {: RESULT=new String(e1+"_;_"+((String) e2).toLowerCase()+"_;__beg_"); :}
                | TK_END TK_LEFT_BRACKET oper1:e1 TK_COMMA oper2:e2 TK_RIGHT_BRACKET
                        {: RESULT=new String(e1+"_;_"+((String) e2).toLowerCase()+"_;__end_"); :}
                | TK_CONT TK_LEFT_BRACKET oper1:e1 TK_COMMA oper2:e2 TK_RIGHT_BRACKET
                        {: RESULT=new String(e1+"_;_"+((String) e2).toLowerCase()+"_;__cont_"); :}
                | TK_MATCH TK_LEFT_BRACKET oper1:e1 TK_COMMA oper2:e2 TK_RIGHT_BRACKET
                        {: RESULT=new String(e1+"_;_"+e2+"_;__match_"); :};

oper1::=        TK_TITOL            {: RESULT=new String("_titol_"); :}
                | TK_AUTOR          {: RESULT=new String("_autor_"); :}
                | TK_DESCRIP        {: RESULT=new String("_descrip_"); :}
                | TK_NOTÍCIA        {: RESULT=new String("_notícia_"); :};

oper2::=        TK_STRING:e  {: RESULT=StringEscapeUtils.escapeXml(e); :};
```

La implementació de la gramàtica és prou evident. Tanmateix es pot destacar:

- La definició del símbol operació. Es pot veure com es canvia l'orde: funcio(param1,param2) per _param1__;_param2_;__operador_.
- La cadena introduïda per l'usuari es passa a minúscula. Això, com es podrà veure a la implementació de la màquina virtual, té com objectiu realitzar una comparació CaseInsensitive.
- Finalment, indicar que la cadena introduïda per l'usuari es escapada al format XML. L'usuari pot introduir caràcters no vàlids a la sentència d'inserció de SQL (com pot ser cometes, etc.). Així, ens estalviem aquest tipus de problemes.

o **J.bat:** És un fitxer per "lots" que s'encarrega d'executar jlex i cups partint dels fitxers indicats anteriorment i genera la classe **Alex**, **parser** i **sym**. Aquestes classes són las encarregades d'implementar l'analitzador lèxic (Alex) i l'analitzador sintàctic (parser). Aleshores, si es modifiquen **a_lex.lex** o **a_sintac.cup** s'ha de tornar a executar el bach **j.bat** perquè generi les noves classes associades.

o **Analitzador.java:** Un objecte d'aquesta classe és creat i utilitzat des de la servlet.

Com s'ha indicat anteriorment, la servlet replega des del formulari de la vista "Editar/Afegir regla de selecció" (Figura 23) l'expressió introduïda per l'usuari. Aleshores, crea un objecte Analitzador i li passa l'expressió. L'objecte Analitzador, crea un objecte **Alex** (analitzador lèxic) i un objecte **parser** que els enllaça per realitzar l'anàlisi i la traducció de l'expressió que ha rebut.

Al finalitzar l'anàlisi, li passa el resultat a la servlet qui, si no ha rebut cap excepció d'error en la compilació, emmagatzema la regla amb l'expressió compilada.

5.5. Execució de les regeles.

Per qüestions d'afinitat, s'ha deixat el procés d'Execució de les Regles de Selecció al punt 6.2 del proper capítol.

6. Capítol 6. Creació de la Sortida de l'aplicació.

6.1. Introducció

Per aquest capítol, s'ha deixat l'explicació de tot el procés pel qual el component SCA genera la sortida de l'aplicació, els avisos als usuaris. Per la qual cosa, s'ha inclòs també en aquest tema la forma en què aquest servei executa les regles de selecció.

Als punts següents s'estudia la metodologia i implementació que segueix el component SCA per gestionar el procés de la generació d'avisos: l'execució de les regles de selecció, la generació dels diferents tipus d'avisos i l'enviament d'aquests a l'usuari.

6.2. Execució de les regles de selecció

Quan s'activa el servei el component SCA, el primer procés que realitzar és la sincronització de notícies i posteriorment el procés d'execució de les regles de selecció. Aquest procés és llançant mitjançant el mètode execRegles() que es defineix dins de la mateixa classe Servei (la que modela el Servi de Recerca de Notícies).

Les accions que realitza aquest mètode són les següent:

- Traure un vector d'objectes CanalsBean. Amb la informació de tots els canals donats d'alta al sistema.

  Seguidament es crea un directori, dins del directori temporal del sistema (veure classe Constants) per a cadascun dels canals, i si existeix es buida. Aquest directori representa una cache de les notícies publicades pel canal corresponent. S'omplirà de notícies si és un canal no conegut i existeix una regla de selecció per contingut de notícia, però aquest procés està dins de les accions realitzades per la maquina virtual.

- Per cadascú dels canals, es crea un vector de NoticiesBean i un altre de ReglesBean amb totes les notícies i totes les regles del canal corresponent.

  Amb el vector de regles, es recorren totes les regles del canal.

- Per cadascuna de les regles, es tracten cadascuna de les notícies de la següent manera:
  - Si la regla està pendent de processar, la màquina virtual executarà la regla amb la notícia tractada.
  - Si la notícia és nova, també la màquina virtual executarà la regla amb aquesta notícia.
  - En altra situació, no s'executa res.

- Una vegada tractades totes les regles pendents de processar i les noves notícies. Si el procés ha segut correcte, les regles passaran a l'estat de no pendents i les noves notícies se li desactivarà aquest flag.

Al següent codi, es presenta la part més destacable d'aquest codi. Per cada canal s'executa:

```
//Trac totes les regles del canal.
vregles=r.getRegles(cb.getCanalid().toString());
//Trac totes les notícies del canal.
vnoticies=n.getVectorNotícia(cb.getCanalid());

//Recorro totes les regles del canal.
for (int i=0;i<vregles.size();i++){
   ReglaBean rb=vregles.get(i);
   ConObj.debugLog("execRegles: Processant la regla"
   +rb.getReglacode());
   //Recorro totes les notícies del canal.
   for (int z=0;z<vnoticies.size();z++){
         NotíciaBean nb=vnoticies.get(z);
         if (rb.getPendent()){
               //Es processaran totes les notícies del canal
               vm.run(nb, rb, cb);
         }else{
               //Es processaran només les notícies noves.
               if (nb.isNova())
                     vm.run(nb, rb, cb);
         }
   }
   //Estableixo la reglaBean en l'objecte regla.
   r.setRegla(rb);
   //Desactiva la regla com a pendent.
   r.pendent(false);

}
//Com que ja han segut tractades les notícies ja no són noves per
//al sistema
for (int z=0;z<vnoticies.size();z++){
   NotíciaBean nb=vnoticies.get(z);
   n.setNotícia(nb);
   //Fique la notícia com que ja no es nova.
   n.nova(false);

}
```

6.3. La Màquina Virtual.

En aquest apartat es tracta la descripció i implementació de la màquina virtual.

▪ *Descripció de la Màquina Virtual.*

La màquina virtual està modelada dins de la classe VMachine dins del paquet getnews.core.servei. Com s'ha indicat al punt 5.3 (operadors i gramàtica) del capítol 5. La màquina virtual executa expressions postoperacionals, és a dir, expressions on primerament són els operands i posteriorment els operadors. A continuació, presento diversos exemples d'expressions postoperacionals generades pel compilador.

- _titol__;_America_;__beg_
- _titol__;_America_;__beg__;__descrip__;_America_;__beg__;__&&_

- _titol__;_America_;__beg__;_descrip_;America_;__beg__;__&&__;_notícia _;_America_;__cont__;__||__

Al punt 5.3 del capítol 5 s'explica els operands i operadors de la gramàtica.

A les expressions anteriors, es pot veure com la cadena "_;_" separa els diferents elements de les expressions postoperacionals.

La màquina virtual divideix la cadena amb l'expressió postoperacional, en un *array* de cadenes composades per cadascú d'aquests elements.

La metodologia que segueix consisteix en traure els diferents elements de *l'array* de cadenes:

- Si no és un operador emmagatzemar-lo en una pila auxiliar.
- Si és un operador, traure de la pila tants operands com necessita aquest operador i realitzar l'operació corresponent. El resultat es tornat a la pila.

Implementació de la Màquina Virtual.

La màquina virtual únicament implementa un mètode públic. Aquet mètode és el mètode run(..) i rep com a paràmetres un objecte NoticaBean, ReglaBean i CanalBean. La funció d'aquest mètode és executar la regla amb les dades d'aquesta notícia. Per a la qual cosa realitza les següents accions:

- Dividir l'expressió postoperacional en un array de cadenes:

```
//Creo un vector amb els diferents elements que composen l'expressió.
String[] code=rb.getReglacode().split("_;_");
```

- Realitzar un preprocés de la notícia. Aquest preprocés consisteix en descarregar la notícia publicada a la cache de notícies del canal, si es tracta d'un canal no conegut i la regla és una regla de selecció de notícies per contingut.

  La descàrrega es realitzarà si el fitxer no existeix, és a dir, si la notícia ha segut tractada per una regla anterior, aleshores ja la tenim desada a la cache i no cal descarregar-la. En aquesta descàrrega s'eliminen certs elements del document HTML innecessaris: scripts, propietats, etc. (mètode UtilsFile.downLoadWithOutScript(..)) i posteriorment a aquest fitxer descarregat se li lleven els tags (UtilsFile.eliminaTagFile(..)), per deixar únicament el contingut textual del document HTML.

  Recordar la problemàtica plantejada al punt 3.3 i 3.6 del capítol 3 amb els canals no coneguts.

- Tractar cadascú dels components de l'expressió postoperacionals i executar les operacions corresponents:

```
for (int i=0;i<code.length;i++){
    inst=code[i];
    ConObj.debugLog("Processant instrucció: "+inst);
    if (inst.equals("_beg_"))           instBegin();
    else if (inst.equals("_end_"))      instEnd();
    else if (inst.equals("_cont_"))     instCont();
    else if (inst.equals("_match_"))    instMatch();
    else if (inst.equals("_&&_"))       instAnd();
    else if (inst.equals("_||_"))       instOr();
    else if (inst.equals("_!_"))        instNot();
    else {
        //No és una operació. Afegeixo l'operand a la pila.
        Pila.push(inst);
    }
}
inst=Pila.pop();
if (inst.equals("_true_"))
    crearAvis();
```

Al codi anterior es pot veure com per cada element de *l'array*, es comprova si és un operador: en cas afirmatiu s'executa l'operació corresponent, i en cas negatiu, no és un operand i s'emmagatzema a la pila.

- Al finalitzar el procés, es mira el resultat que ha quedat a la pila. Si aquest valor és ***true***, la notícia compleix l'expressió i s'ha de preparar per avisar a l'usuari (mètode crear avís)

▪ *Implementació de les operacions.*

Bàsicament la maquina virtual treballa amb dos tipus d'operadors: els operadors relacionals: AND, OR, NOT i les funcions de comprovació de contingut: begin, end, contains i match. Més informació al punt 5.3 del capítol 5.

A continuació es tracten els dos tipus d'operadors:

- Funcions de comprovació. Les funcions de comprovació tenen dos paràmetres: param1: origen de la informació (títol, descripció, autor, notícia) i param2: cadena de caràcters.

  Quan s'executa una d'aquestes funcions, si l'expressió postoperacional és correcta, a la pila ha d'haver-hi, almenys, els dos operands corresponents i s'extreuen. Posteriorment es comprova el valor del paràmetre 1, per determinar amb quina informació ha de tractar (títol, descripció, autor, notícia). Seguidament, es passa a minúscules el paràmetre 2 (ja que es realitza una comprovació *CaseInsensitive,* veure el punt 5.4 capítol 5) i s'aplica la instrucció corresponent de cadenes: startsWith, endsWith, contains o verificar si l'expressió regular es compleix amb la classe Matcher.

  El resultat d'aquesta operació s'afegeix a la pila. La cadena "*_true_"* si es compleix o "_false_" si no es compleix.

  A continuació es pot veure el codi per la instrucció **end**:

```
String oper2=Pila.pop();
String oper1=Pila.pop();
if (oper1.equals("_titol_")){
    String cad=Noticiab.getTitol().toLowerCase();
    if (cad.endsWith(oper2))
        Pila.push("_true_");
    else
        Pila.push("_false_");
} else if (oper1.equals("_autor_")){
    String cad=Noticiab.getAutor().toLowerCase();
    if (cad.endsWith(oper2))
        Pila.push("_true_");
    else
        Pila.push("_false_");
} else if (oper1.equals("_descrip_")){
    String cad=Noticiab.getDescrip().toLowerCase();
    if (cad.endsWith(oper2))
        Pila.push("_true_");
    else
        Pila.push("_false_");
```

En el cas de què l'origen de la informació (paràmetre 1) sigui notícia, és a dir, s'ha de comprovar amb el contingut de la notícia, la metodologia és un poc diferent. Els passos són els següent:

- Si és un canal conegut. Crea un objecte de la classe NotíciaContingut i extrau de la BBDD el contingut de la notícia. Posteriorment converteix aquest contingut a minúscules i li aplica l'operació corresponent.
- Si és un canal no conegut, es crida al mètode cercarEnNotícia() amb únicament l'operador **cond** o **match**. Als canals no coneguts no té sentit cercar una cadena al principi o al final de la notícia, ja que estrictament el contingut no es pot extraure. En aquest casos només té sentit aplicar un **cond** o **match** a tot el document publicat. Aleshores, si l'usuari intenta aplicar un d'aquest paràmetres són transformats pel sistema a l'operand **cond**.

  El mètode cercarEnNotícia() aplica l'operador **cond** o **match** sobre el fitxer desat en el preprocés (mètode preProcess()). Recordar que aquest fitxer és la notícia publicada en HTML però se li han llevat els scripts, els comentaris, els tags, etc. Únicament queda el text visible, però s'ha de recordar que té més informació que la notícia. Veure punt 3.3 capítol 3.

o Operadors relacionals: Els operadors relacionals són els operadors booleans. Els operadors **AND** i **OR** tenen dos paràmetres i l'operador **NOT** en té un. La seva implementació és molt senzilla: traure els operands que necessita l'operador corresponent de la pila i aplicar l'operador booleà adient. Finalment s'apila el seu resultat.

Al següent codi es pot veure la implementació de l'operador AND:

```
//Mètode que executa la instrucció _AND_
private void instAnd()throws ErrorException{
    //trac els operadors
    ConObj.debugLog("Executant operació and");
    try{
        String oper1=Pila.pop();
        String oper2=Pila.pop();
        if ((oper1.equals("_true_")) && (oper2.equals("_true_"))){
            Pila.push("_true_");
        } else
            Pila.push("_false_");
    } catch(EmptyStackException e){
        throw new ErrorException("instAnd: Error execució expressió "+Reglab.getReglacode());
    }
}
```

- ***Creació d'un avís.***

  La màquina virtual, després d'executar una expressió comprova el resultat de la pila. Si el resultat és "_false_" no fa res, però si es "_true_" aleshores crida al mètode crearAvis(), definit en la mateixa classe.

  Aquest mètode simplement crea una entrada a la taula **avís** perquè en un procés posterior (generació d'avisos) el sistema pugui generar i enviar un avís a l'usuari corresponent.

6.4. Procés de generació i enviament d'avisos.

El Servei de Recerca Automatitzat de Notícies (SCA), després de realitzar el procés d'execució de les regles de selecció, executa el procés de generació d'avisos. Aquest procés consisteix en dues parts: el procés de generació del tipus d'avís que l'usuari ha establert a la regla i el procés d'enviament de l'avís.

6.4.1. Format ICALENDAR[1]

ICALENDAR és un tipus de format que permet als usuaris enviar convocatòries de reunió i successos a altres usuaris via correu electrònic o compartir arxius de calendaris entre usuaris que utilitzen una aplicació compatible amb el format ICS. El format *ICALENDAR* és suportat per multitud d'aplicacions actualment entre les que destaquen *Google* Calendar, *Appel ICal*, *ThunderBerd*, etc.

Un fitxer en format *ICALENDAR* és un fitxer amb un format semblant a un fitxer xml, on hi ha elements i els elements poden tindre propietats, tanmateix els elements no estan determinants estrictament per tags.

L'element de nivell superior és el iCalendar i pot englobar una col·lecció de calendaris. Normalment, el fitxer en format ICS es composa d'un objecte *iCalendar* senzill. No obstant això, diversos objectes *d'iCalendar* es poden agrupar.

La primera línia ha de ser BEGIN:VCALENDAR i l'ultima línia ha de ser END:VCALENDAR. El contingut entre aquestes línies s'anomena el *icalbody*.

[1] Referència: http://en.wikipedia.org/wiki/ICalendar.

El cos de l'objecte *d'iCalendar* (el *icalbody*) està format per una llista de propietats del calendari i un o més components inclosos en el calendari. Les propietats del calendari s'apliquen al calendari complet. Els components de calendari són diverses elements del calendari, per exemple, un esdeveniment, una llista de tasques, una alarma, etc.

A continuació es presenta un exemple d'un fitxer ICS:

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//hacksw/handcal//NONSGML v1.0//EN

BEGIN:VEVENT
UID:uid1@example.com
DTSTAMP:19970714T170000Z
DTSTART:19970714T170000Z
DTEND:19970715T035959Z
SUMMARY:Bastille Day Party
END:VEVENT

END:VCALENDAR
```

Figura 25. Exemple tret de Wikipedia.

En l'exemple es pot veure que l'objecte *iCalendar* comença en la primera línia i acaba en l'última. Dins d'aquest objecte hi ha dos propietats: la versió i l'eina que la va generar. També hi ha un element Succés, que es tracta a continuació.

L'element VEVENT descriu un esdeveniment, que té una duració de temps en un calendari. Normalment, quan un usuari accepta un esdeveniment, això farà que es marqui com ocupat aquest temps en el seu calendari. Un VEVENT pot incloure un VALARM que defineix una alarma. Aquests esdeveniments tenen un DTSTART que estableix una hora d'inici i un DTEND que estableix una hora d'acabament. Si l'esdeveniment de calendari és diari no hi ha element DTEND inclòs.

A més, ha d'incloure un element UID. Aquest element és un identificador únic de l'esdeveniment, i serà utilitzat per fer-li referències posterios..

Tots aquests elements es poden veure a l'exemple anterior (fig. 25).

En el cas concret de l'aplicació GestNews, l'avís estarà compost per un calendari amb un succés (avís tipus succés) o un calendari compost per un conjunt de successos (avís tipus registro ICALENDAR).

6.4.2. Format ODF[2]

El format ODF (*Open Document Format for Office Applications*) és un format obert basat en XML i que conté informació de diferents tipus d'aplicacions incloses en el paquet d'ofimàtica OpenOffice. En funció del tipus d'informació que tingui l'arxiu: fulla de càlcul, processador de textos, presentacions digitals, etc., el fitxer tindrà una extensió determinada: ods, odt, odp, etc.

Les regles de selecció en què s'especifica un avís del tipus: registre en document ODT, produeix que el SCA generi un avís amb el registre de totes les notícies publicades que han activat aquesta regla. L'avís que es genera és un document de text (fitxer odt) amb una taula, on hi ha una entrada per cadascuna de les notícies que han activat la regla. En aquesta entrada hi ha un enllaç a la notícia, la data en què es va activar i el títol de la notícia.

Regla: noticia

Noticia	Enllaç	Data
Tienen una opción para que Weiss siga un año más	Accés:	2011-12-26
Premio a Messi, triunfo del Barça	Accés:	2011-12-26
Abraham se acerca a la MLS	Accés:	2011-12-26
El sevillismo analiza si da su apoyo a Del Nido	Accés:	2011-12-26
Piti: "Le pediré a los Reyes un Sant Andreu en Segunda A"	Accés:	2011-12-26
Mourinho ya planifica la próxima temporada	Accés:	2011-12-26

Figura 26.Exemple Avís tipus: Registre en document ODT.

Un exemple d'avís de registre de document ODT generat pel sistema GestNews es pot veure a la imatge anterior.

Com que l'estructura d'un document ODT es prou complexa i queda fora del abast de la memòria[3], únicament indicar que el fitxer ODT és un fitxer comprimit en format *zip*. Dins d'aquest fitxer hi ha diferents components, concretament el fitxer content.xml. Aquest fitxer és el més importat, ja que és un document xml que conté la informació real del document ODT, exceptuant la informació binària com les imatges, etc.

6.4.3. Procés de generació d'avisos.

En aquest procés, primerament es trau un vector de **CanalBean** amb tots els canals del sistema, seguidament per cada canal es trau un vector de **ReglaBean** amb totes les regles del canal. Finalment es crida al mètode generar(...) de l'objecte Avís i se li passa un objecte **ReglaBean**. Aquest

[2] Referència: http://en.wikipedia.org/wiki/Odf i http://es.wikipedia.org/wiki/OpenDocument
[3] Més informació http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office#technical

mètode s'encarrega de generar els avisos pendents de les regles que se li ha passat com a paràmetre. Posteriorment l'envia al seu usuari.

En funció del tipus d'avís que requereix la regla (succés, registre ICALENDAR, registre fitxer ODT) es llança el mètode corresponent.

El sistema permet triar a l'usuari entre tres tipus d'avisos:

- Succés ICALENDAR: L'usuari rep un fitxer ICALENDAR amb un succés, per cadascuna de les noticies que compleixen la regla.
- Registre de successos ICALENDAR: L'usuari rep un únic fitxer ICALENDAR amb tants successos com noticies compleixen la regla.
- Registre de successos en format ODT: Semblant al tipus anterior, però l'usuari rep un fitxer de text ODT amb una taula. On apareix una fila per cadascuna de les noticies que compleixen la regla.

▪ *Mètode generarAvisEvent()*

Es llança quan el tipus d'avís és el succés. Aquest mètode s'encarrega:

o Cercar tots els avisos de la regla de selecció tractada i que no han segut generats prèviament. Per aconseguir aquest objectiu s'executa la següent consulta:

```
ResultSet rs =
    stm.executeQuery("SELECT * FROM avisos where regla_id="+
            reglaid+" and generat=0");
```

o Per cadascú dels registres que trau la consulta, es genera un fitxer ICS (veure punt 6.4.1) amb la informació de l'avís.

La construcció del fitxer ICS es realitza en dues parts: primer es genera un fitxer xml i posteriorment, una plantilla xslt per transformar el fitxer xml en el fitxer ICS.

Realitzar aquest procés en dues parts té com objectiu aconseguir un sistema flexible. Així, es pot generar altres tipus d'avisos simplement creant una nova plantilla **xslt** per al nou tipus d'avís.

Generació del fitxer xml.

Per generar el fitxer xml s'utilitza l'API JAXP (com es va indicar al punt 2.5).

Per cada succés (VEVENT) que s'ha d'incloure en l'element *ICALENDAR* es crea un objecte EventBean i se li carrega amb la informació corresponent. Posteriorment s'afegeix l'objecte EventBean a un vector. En aquest cas el vector únicament tindrà un element (l'avís únicament ha d'incloure un succés). Després s'envia el vector al mètode generarXMLEvents(..) que genera el document xml en un *string*.

Un exemple del document xml generat es pot veure a la imatge següent:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VCALENDAR version="2.0" prodid="GestNews">
    <VEVENT>
        <DTSTAMP>201204T180000</DTSTAMP>
        <UID>403090a0-6ac9-4a4c-a97c-68209d51f406</UID>
        <SUMMARY>Avis Regla: titol_barça</SUMMARY>
        <DTSTART>TZID=Europe/Paris:20120104T184318</DTSTART>
        <DESCRIPTION>Noticia: L'Espanyol-Barça, diumenge, a través de l'especial "Hat-trick derbi"
        Accés http://www.tv3.cat/actualitat/382820/LEspanyol-Barca-diumenge-a-traves-de-lespecial-Hat-
    </VEVENT>
</VCALENDAR>
```

Figura 27.Exemple del document xml creat per la generació del fitxer ICS.

Com s'ha indicat anteriorment, el mètode generarXMLEvents(..) genera el document xml a memòria mitjançant l'API JAXB. El seu codi és el següent:

```java
private String generarXMLEvents(Vector<EventBean> vEvent) throws ErrorException{
    CalendarBean ical = new CalendarBean();
    //A l'objecte CalendarBean li passo el vecotr d'events perquè
    //els incloga dins de l'element VEVENT.
    ical.setvEvent(vEvent);
    StringWriter oWriter = new StringWriter();
    try{
        JAXBContext contextObj = JAXBContext.newInstance(gestnews.core.outbean.CalendarBean.class,
                gestnews.core.outbean.EventBean.class);
        Marshaller marshallerObj = contextObj.createMarshaller();
        //Volem mantindre identacions al fitxer per poder-lo veure nosaltres millor. (debug)
        marshallerObj.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        //Aplicar el procés marshal per generar el document xml a memòria
        //a partir dels objectes CalendarBean i EventBean.
        marshallerObj.marshal(ical, oWriter);
        String output = oWriter.toString();
        return output;
    } catch (Exception e) {
        throw new ErrorException("generarXMLEvents: Error generació icalendar.");
    }
}
```

El codi és simple i amb la ajuda dels comentaris es pot veure clarament els passos que va realitzant.

- Crear un objecte CalendarBean i li passa el Vector de successos perquè generi els elements VEVENT.
- Crear un objecte JAXBContext indicant-li els objectes involucrats en la transformació: CalendarBean i EventBean.
- Aplicar el mètode marshal(..) per generar el document xml en un objecte StringWriter que posteriorment es transformat a string i es retornat.

Generació del fitxer ICS.

La generació del fitxer ICS és realitzada per la crida al mètode transformacioXSLT(..). Aquet mètode rep la cadena amb el document xml i la ubicació de la plantilla xslt. Posteriorment realitza la transformació i

retorna una cadena amb el seu resultat. Aquest *string* es passa a fitxer amb el mètode UtilsFile.writeToFile(..).

A continuació es mostra el codi del mètode transformacioXSLT(..). És un codi simple i amb la ajuda dels comentaris es veu clarament les accions que realitza.

- Finalment es desa al directori temporal per ser enviat a l'usuari.

```
private String transformacioXSLT(String xml,String source) throws ErrorException {
    //Creació del canal per llegir el document xml
    StringReader outReader = new StringReader(xml);
    StreamSource streamSource = new StreamSource(outReader);
    //Creació del canal per escriure el resultat de la transformació.
    StringWriter outWriter = new StringWriter();
    StreamResult streamResult = new StreamResult(outWriter);
    try{
        //Creem el canal per llegir la plantilla xslt
        StreamSource streamSourceXSLT =new StreamSource(new File(source));
        //Crem un TransformFactory per després crear un objecte transformer
        //amb la plantilla xslt.
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer(streamSourceXSLT);
        //Apliquem la transformacío
        transformer.transform(streamSource, streamResult);
        //Retornem la transformació en una cadena.
        return outWriter.toString();
    } catch (Exception e) {
        throw new ErrorException("transformacioXSLT: Error en la transformació XSLT."+e.getMessage());
    }
}
```

- ***Mètode generarAvisCalendar()***

El mètode és llançat quan el tipus d'avís de la regla és registre ICalendar. Té com objectiu cercar un avís pendent de tractar d'aquest tipus i si el troba ha de crear un fitxer ICS amb el registre de totes les notícies publicades en aquest moment que han activat la regla tractada. Totes aquestes notícies seran tractades com successos (VEVENT) dins del fitxer ICS.

La seva implementació és molt semblant al mètode generarAvisEvant() mostrada al punt anterior:

- Cercar tots els avisos d'aquest tipus no generats de la regla tractada i si en troba un, aleshores cercar tots els avisos d'aquesta regla. Per cadascú dels avisos es crea un objecte EventBean i tots ells s'afegeixen a un vector.
- Generar el fitxer xml mitjançant la cridada al mètode generarXMLEvents(..). A diferència del cas anterior, se li passa al mètode un vector amb els diferents objectes EventBean i tornarà un document xml com el mostrat a la figura 27, però amb tants elements VEVENT com avisos hi ha al vector enviat.
- Generar el fitxer ICS cridant al mètode transformacioXSLT(..) com es feia en el mètode generarAvisEvant() (cas anterior).
- Com en el cas anterior, es desa al directori temporal per ser enviat a l'usuari.

- ***Mètode generarAvisODT()***

El mètode és llançat quan el tipus d'avís de la regla és registre en format ODT. L'objectiu és el mateix que generarAvisCalendar() però en aquest cas ha de ser un document ODT i no un fitxer ICS.

La implementació és prou semblant al mètode generarAvisCalendar() però aplicada al format ODT:

- o Cercar tots els avisos no generats de la regla tractada i si en troba un, aleshores cercar tots els avisos d'aquesta regla. Per cadascú dels aviso es crea un objecte ItemODTBean i tots ells s'afegeixen a un vector.
- o Generar el fitxer xml però en aquest cas es crida al mètode generarXMLODT(..). La seva metodologia és molt semblant a generarXMLEvents(..).
- o Generar el fitxer content.xml[4] cridant al mètode transformacioXSLT(..) com en els casos anterior.
- o Finalment i a deferència dels casos anteriors, es crida al mètode UtilsFile.generarODT(..) per generar el document del registre en DOT i que serà enviat a l'usuari.

El mètode generarODT(..) de la classe UtilsFile afegeix a un document ODT plantilla, que hi ha al directori template, el fitxer content.xml generat.

Nota:

S'han creat dos documents xml diferents per generar els tres tipus d'avisos. Un per als avisos que utilitzen el format ICALENDAR i l'altre per als que utilitzen el format ODT. Realment, amb el fitxer xml mostrat a la figura 27 i les corresponents plantilles **xslt** és suficient per a generar els tres tipus d'avisos. En aquest cas, ens haguéssim estalviat també les classes **ItemODTBean** i **ODTBean** emprades per l'api JAXB i que són necessàries per generar el corresponent document xml. Tanmateix, s'ha decidit fer-ho així per proporcionar una major claredat al sistema.

6.4.4. Procés d'enviament d'avís.

Realment és un subprocés dins del procés de generació d'avisos i és activat en els mètodes anterior, després de generar el fitxer amb l'avís. S'encarrega d'enviar l'avís a l'usuari corresponent.

Aquest procés comença creant un objecte EmailBean i carregant-lo amb les dades de les regles, usuari, assumpte, etc. Posteriorment es crida al mètode UtilsOther.enviarCorreo(..) que s'encarrega de l'enviament del missatge.

Per l'enviament s'utilitza l'API JavaMail i els passos realitzats són els següent:

- o Crear un objecte Properties i especificar les propietats del servidor smpt (servidor eixint).

[4] Veure el punt 6.4.2

```
Properties props = new Properties();
// Nom del host de correo
props.setProperty("mail.smtp.host", conobj.getPropietat("SMTP_SERVER"));
// Port d'enviament
props.setProperty("mail.smtp.port",conobj.getPropietat("SMTP_PORT"));
// Nom usuari
props.setProperty("mail.smtp.user",conobj.getPropietat("SMTP_USER"));
// Si requiere o no usuario y password para conectarse.
props.setProperty("mail.smtp.auth",conobj.getPropietat("SMTP_AUTH"));
Session session = Session.getDefaultInstance(props);
```

- Crear un objecte MimeBodyPart amb el contingut textual del correu. En aquest contingut s'indica: la regla que s'ha activat i la notícia que l'ha activada.
- Crear un objecte MimeBodyPart per afegir el fitxer amb l'avís.
- Crear un objecte MimeBodyPart per incloure el contingut textual i el fitxer adjunt.
- Crear un objecte MimeMessage especificat a qui va dirigit, qui l'envia, el contingut, etc.
- Finalment es crea un objecte Transport per enviar el missatge.

Al següent codi es pot veure clarament els passos indicats:

```
//Es composa la parte del text
BodyPart texto = new MimeBodyPart();
String cad="S'ha complit la regla '"+emb.getReglaNom()+ "': " +emb.getReglaCos()+"\n"+
        "Títol de la notícia: "+StringEscapeUtils.unescapeXml(emb.getNoticiaTitol());
texto.setText(cad);

//Es crea el fitxer adjunt.
BodyPart adjunto = new MimeBodyPart();
System.out.println("-->"+emb.getPath_fich()+"/"+emb.getNom_fich());
adjunto.setDataHandler(new DataHandler(new FileDataSource(emb.getPath_fich()+"/"+emb.getNom_fich())));
adjunto.setFileName(emb.getNom_fich());

//Un objecte multiparte inclou el text i el fitxer que s'adjunta.
MimeMultipart multiParte = new MimeMultipart();
multiParte.addBodyPart(texto);
multiParte.addBodyPart(adjunto);

//Es crea la sessió, especificant to, from, subject i contingut
MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress(conobj.getPropietat("SMTP_EMAIL_SENDER")));
message.addRecipient(Message.RecipientType.TO, new InternetAddress(emb.getUserEmail()));
message.setSubject(emb.getAssumpte());
message.setContent(multiParte);
```

7. Capítol 7. Interfície de l'aplicació.

7.1. Introducció

En aquest capítol es tracta l'arquitectura, metodologia i desenvolupament de la interfície de l'aplicació i també a grans trets el seu funcionament.

Com a punt de partida, vull recordar que en el desenvolupament del sistema GestNews gran part del treball ha recaigut en el disseny i creació dels components GUAP i SCA. El component GUAP és la part visible del sistema GestNews, ja que el SCA funciona de forma independent i nodrint-se de la informació extreta per l'altre component.

La interfície de l'aplicació utilitza la funcionalitat de la subcapa de Gestió i Administració de Preferències. Aquesta interfície, s'ha dissenyat com una interfície web que s'executa dins del Servidor d'Aplicacions. El seu disseny segueix el patró MVC (*Model View Controler*).

7.2. Patró MVC

L'ús del patró de disseny MVC (*Model View Controler*) està actualment molt estès a l'àmbit de les aplicacions web. De fet, existeixen multituds de *frameworks* que faciliten la seva implementació (*Hibernate, Struts*, etc.). Tanmateix, el disseny de la interfície web no està basada en cap frameworks, sinó que s'ha emprat, per la seva implementació, objectes de tipus *Beans, JSP (Java Server Pages)* i JSLT (*Java Server Pages Standard Tag Library*) per la creació de les vistes web. Aquest tema ja es tractarà més endavant.

El principal avantatge de l'ús del patró de disseny MVC és que facilita la creació d'aplicacions multicapa, que com ja es va parlar al capítol 2, és el tipus d'arquitectura que es busca en el desenvolupament de l'aplicació. Seguint l'ús d'aquest patró, s'aconsegueix la separació de la capa de presentació, de la capa que conté la lògica de l'aplicació. Així, s'han desenvolupat les classes que duen la lògica de l'aplicatiu per un costat i les vistes amb JSP per un altre. D'esta forma, es separa totalment el codi de l'aplicació de la presentació (vista).

Altre avantatge del patró MVC és que com es separa totalment la capa de la lògica, de la capa de presentació, es pot canviar l'aparença de la interfície sense tocar pràcticament res de l'aplicació.

En el desenvolupament del sistema, he pogut comprovar que l'afirmació del paràgraf anterior es totalment correcta. De fet, al seguir la programació de les tasques establertes a la memòria, vaig desenvolupar el mòdul de gestió d'usuaris, lectura de notícies i accés a dades, seguint el patró MVC amb una interfície web molt "rudimentària" i poc acurada. Posteriorment, en la tasca de disseny de la interfície, vaig redissenyar les vistes web amb més temps i dedicació. Aleshores, vaig canviar la cara a l'aplicació pràcticament sense tocar res de la lògica. Vull remarcar que gran part de la càrrega del treball de la PFC ha recaigut en el disseny de les vistes i més concretament en el disseny de les fulles d'estil CSS.

El flux d'informació que segueix el sistema amb l'ús del patró de disseny MVC es pot veure a la Figura 28. Tot i que s'explica en més detall al punt 7.4, en aquesta imatge es pot veure clarament la separació dels diferents elements: servlet, la capa de negoci i les vistes.

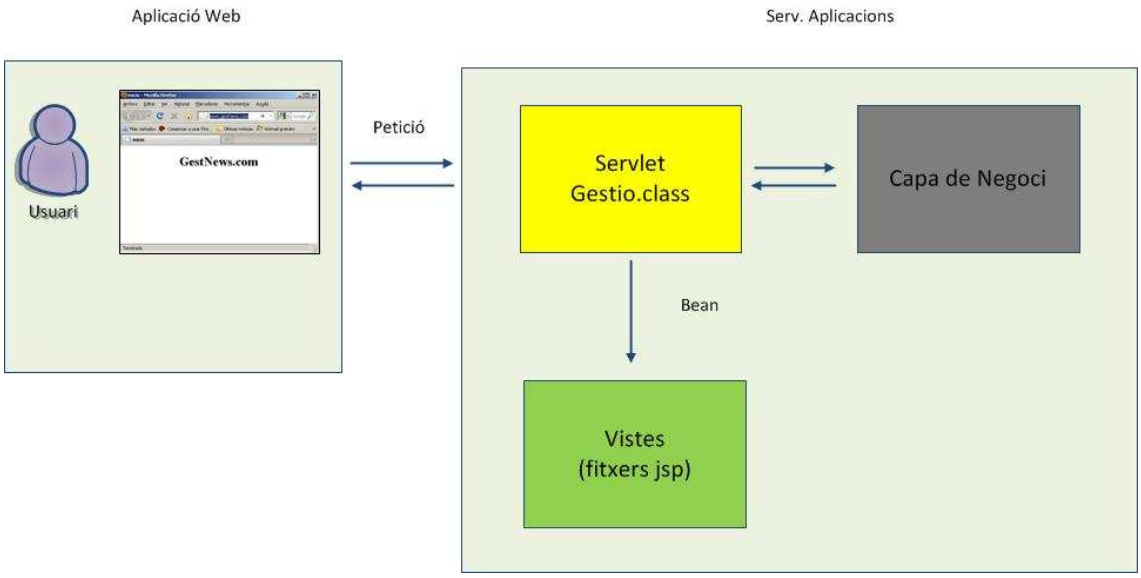


Figura 28.Flux d'execució patró MVC

7.3. Eina de desenvolupament de la Interfície.

Com he indicat anteriorment, el desenvolupament de la interfície web s'ha realitzat amb ECLIPSE Helios. Aquesta eina té molts avantatges per realitzar aquest tipus de desenvolupament, entre aquests avantatges podríem assenyalar les següent:

- **Facilitat en la programació**. ECLIPSE incorpora moltes facilitats per als programadors en Java. Queda fora de la memòria nombrar-les.
- **Avantatges en quant al desenvolupament de les tecnologies** que utilitza el servidor d'Aplicacions. Facilitat en la creació i tractament de servlets, tractament de documents XML, reconeixements de formats jsp, html, etc.
- **Alt grau d'integració amb el servidor d'aplicacions JBoss**. Incorpora moltes facilitats: botons que interactuen amb el servidor JBoss (engegar-lo, apagar-lo, etc.), creació de l'estructura de directoris de l'aplicació, incorporació de llibreries de JBoss, etc.
- **Facilitar les tasques de creació del fitxer de desplegament**. Està molt relacionat amb el punt anterior. ECLIPSE s'encarrega de dur la fastigosa feina de la creació i manteniment del fitxer *buil.xml*. Aquest fitxer conté totes les instruccions que defineixen les tasques necessàries per a construir l'arxiu de desplegament. En aquest cas GestNews.war.
- **Sincronització amb el servidor per al desplegament**. D'una forma transparent per al desenvolupador, ECLIPSE es pot ficar en contacte amb el Servidor d'Aplicacions per passar-li el fitxer de desplegament. Aquest executa l'eina **ant,** que agafa el fitxer de desplegament i "prepara" l'aplicació en el servidor d'aplicacions perquè estigui activa.

7.4. Elements de la Interfície web.

En aquest punt, es descriuen els diferents elements que composen la interfície web:

- **index.jsp**: La JSP (*Java Server Page*) index.jsp és el punt d'inici del component GUAP. Com es veurà més endavant, presenta la vista del començament on es demana que l'usuari s'identifiqui al sistema. A partir d'aquest punt, totes les peticions (interacció) de l'usuari serà gestionada per la servlet **Gestio**.
- **Gestio.java:** És la servlet de l'aplicació. En el model MVC, podem veure a la servlet com un component que actua de controlador, rep les peticions que realitza l'usuari des del navegador i determina l'acció a realitzar. Aquestes accions són llançades a la capa de negoci. Una volta realitzades, genera el contingut de la resposta i selecciona la vista web que l'ha de mostrar a l'usuari.
- **Vistes Web**. Seguint el patró de disseny MVC s'han dissenyat les vistes web mitjançant fitxers JSP (*Java Server Page*). Aquest fitxers JSP personalitzen l'eixida de les vistes (informació dels canals, notícies, etc.) mitjançant Bean que reben de la servlet. Així, se separa les vistes de l'aplicació de la capa de negoci. Gràficament es pot veure a la figura 28.
- **Capa de Negoci**. Són classes desenvolupades en JAVA i que contenen la lògica de l'aplicació. Aquestes classes també són utilitzades per modelar la solució del component SCA.
- **Fulls d'estil**. En el disseny de la interfície web s'han utilitzat fulls d'estil **css**, concretament dos fitxers d'estils: styles.css i styles2.css. Al primer es defineixen els estils generals i al segon els estils concrets.
- **Fitxers html**. Hi ha també fitxers d'ajuda que no executen cap acció, i en aquest casos s'ha utilitzat el format **html.**

▪ *La Servlet.*

Donat que la servlet representa una part molt important en el disseny, s'ha volgut aprofundir un poc més en la seva descripció.

En un primer moment es va desenvolupar el component GUAP amb diverses servlets. Cadascuna s'encarregava de diferents aspectes de la interfície (gestió d'usuaris, gestió de canals, etc.), però després d'analitzar millor aquest component, es va determinar que la lògica de control que requereix no és molt complexa i es podia simplificar certs aspectes (connexió BBDD, log, etc.) fent una única servlet.

Aspectes destacats de la servlet:

- Mètode **init**. Quan la servlet rep una consulta, el servidor d'aplicacions comprova si la servlet la pot resoldre. Si no s'està executant o està ocupada en una consulta anterior, el servidor crearà un fill amb la servlet i executarà aquest mètode.

  En GestNews, el mètode init s'encarrega de crear un objecte global *Connexio* que té com a missió establir una connexió amb la BBDD i obri el log. Així, altres mètodes poden utilitzar la BBDD i escriure al log.

Posteriorment, passarà el control a la servlet segons sigui la consulta feta amb el mètode **POST** o **GET**.

- Mètode **doGet**. En el cas que la servlet pugui atendre la consulta, si s'ha realitzat mitjançant el mètode GET s'executarà aquest mètode, que simplement és un embolcall del mètode **doPost**.
- Mètode **doPost**. Aquest mètode s'executa quan la servlet ha rebut una consulta pel mètode **POST** o també pel mètode **GET** (hauria seguit reenviat). La funció que realitza és traure el paràmetre **op**, enviat des d'una consulta del navegador, i en funció d'aquest, determinar l'acció a realitzat. Aquesta acció sempre finalitza retornat una **url** i a més, en diverses ocasions, establint paràmetres en la sessió de l'usuari (informació, Bean, etc.). Aquesta **url** indica la situació de la vista a mostrar, la qual serà reenviada a un objecte *RequestDispatcher*. Aquest s'encarrega d'executar el fitxer jsp i extraure, si es necessari, la informació de la sessió de l'usuari.

La imatge de la figura 28 pot ajudar a visualitzar tot aquest procés. A més, s'adjunta el codi del mètode **doPost** perquè resulti més comprensible.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        //Trac la operacio demanda
        String operacio= request.getParameter("op");
        String address;
        //Per defecte redirigir a la pàgina inicial
        address= "index.jsp";
        if (operacio==null){
                //Accés sense operació a realitzar.
                address= "/resultats/error_greu.jsp";
                request.setAttribute("error","L'operació no és possible.");
                ConObj.errorLog("No s'ha indicat operacio.");
        } else if (operacio.equals("enter")){
                //Accés al sistema.
                address=accesSistema(request);
        } else if (operacio.equals("mostrar_altausuari")){
                //Mostrar la pantalla de inscriure's
                address="/resultats/alta_usuari.jsp";
        } else if (operacio.equals("altausuari")){
                //Registar usuari al sistema.
                address=altaUser(request);
                //----------------------Tractament de Canals RSS----------------------
        } else if (operacio.equals("listcanal")){
                //Llistat dels canals del usuari.
                address=listCanal(request);
        } else if (operacio.equals("mostrar_addcanal")){
                //mostrar la finestra d'afegir canal.
                address=mostrarAddCanal(request);
        } else if (operacio.equals("addcanal")){
                //Afegeix i realitza les operacions de comprovació del canal.
                address=addCanal(request);
        //----------------------Tractament de Regles de Selecció------------------
        } else if (operacio.equals("listregles")){
                //Llistar les regles del canal.
                address=listRegles(request);
        } else if (operacio.equals("mostrar_addregla")){
                //Preparo el sistema per mostrar la finestra de add_regla.
```

```
                address=mostrarAddRegla(request);
        } else if (operacio.equals("addregla")){
                //Afegeix una regla al sistema.
                address=addRegla(request);
        } else if (operacio.equals("editregla")){
                //Permet que una regla pugui ser editada.
                address=editRegla(request);
        } else {
                address= "/resultats/error.jsp";
                request.setAttribute("error","L'operació no existeix.");
                ConObj.errorLog("L'operació no existeix.");
        }
        //Reenvio l'execució al fitxer jsp indicat per la url de address.
        //Aquest fitxer mostra la vista.
        RequestDispatcher dispatcher = request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
}
```

7.5. Implementació de la Interfície web.

Als següents punts es comentarà els aspectes tècnics més destacats de la implementació de la interfície web.

- ***Accés al Sistema.***

Per accedir al component GUAP ho em de fer des del navegador. Si el servidor d'aplicacions s'està executant al mateix ordinador i sobre el port 8080, es pot utilitzar l'adreça http://127.0.0.1:8080/GestNews. En el cas que estigues en altre ordinador o en un domini determinat, ficaríem la seva direcció o el domini corresponent. Evidentment, també podríem fer-ho amb un enllaç o amb un redireccionant des d'un servidor web com Apache.

Aquesta url produeix l'execució de la vista **index.jsp**, que ens mostra la pàgina d'accés al sistema (figura 29).

Si el client no està registrat al sistema, ha de polsar el botó d'**alta** per donar-se'n. Aquesta funció la realitza el mètode altaUser(..). En el cas de què estigui registrat, mitjançant el seu usuari i contrasenya es pot autentificar en el sistema. En aquest cas, el sistema comprova si l'usuari està donat d'alta i en cas afirmatiu, si la contrasenya és correcta. Tot aquest procés es realitzat pel mètode accesSistema(..)

.

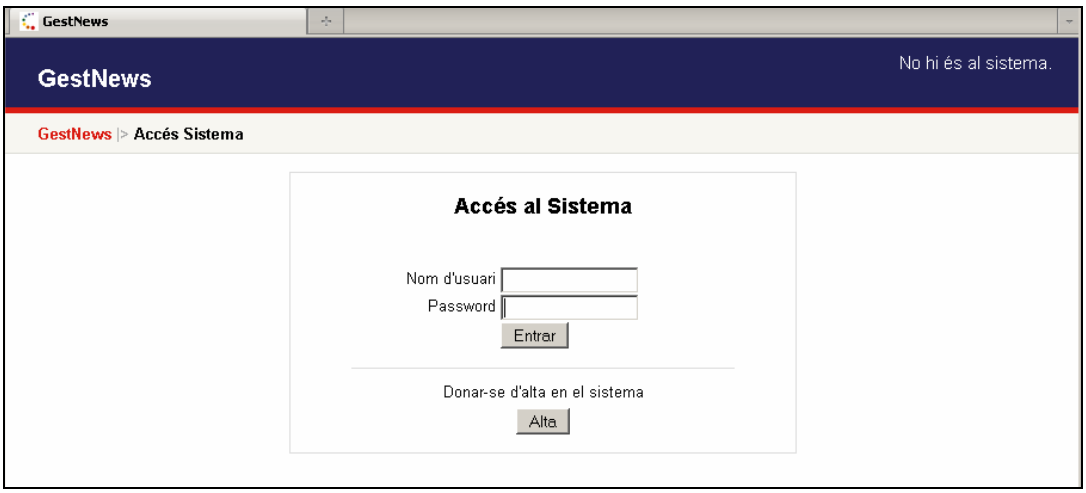


Figura 29. Vista d'accés al sistema.

- ***Implementació de la gestió de Canals.***

Una vegada l'usuari s'ha autenticat, li apareix una vista amb tots els canals en què està subscrit (figura 30) . El mètode que s'encarrega d'aquesta funció és listCanal(..). El procés que realitza és el següent:

- Trau informació de l'usuari de la seva sessió.
- Crea un objecte de la classe **Canal** i crida al mètode getCanals(..). Aquest mètode trau un vector d'objectes de la classe **CanalBean** amb la informació de cadascú dels canals en què està subscrit l'usuari.
- Afegeix el vector d'objecte **CanalBean** a la sessió de l'usuari, per a ser visualitzats per la vista **canals.jsp**.
- La vista **canals.jsp** trau de la sessió aquest vector i el tracta amb jstl. Així per cada objecte **CanalBean** afegeix una fila a la taula amb informació del canal.

La vista resultant la podem veure a la imatge de la figura 30.

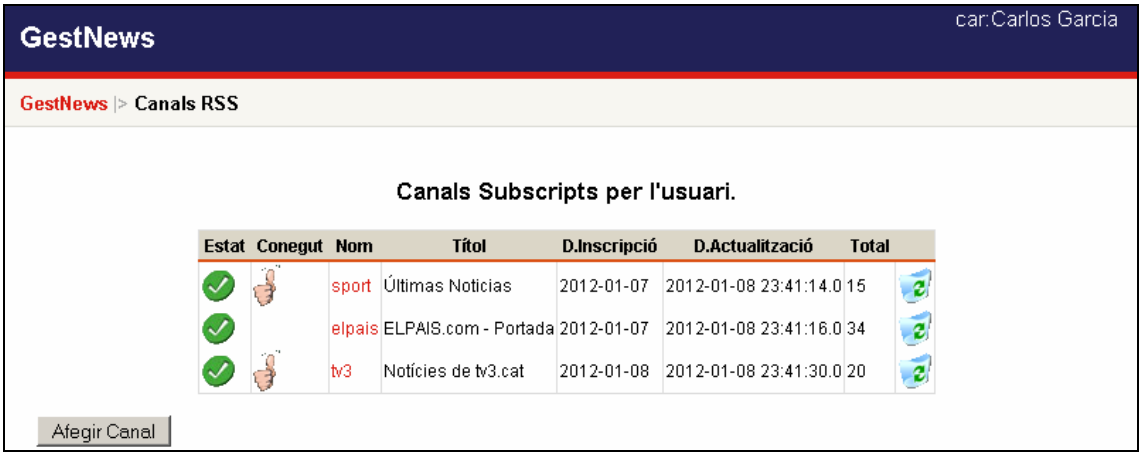


Figura 30. Vista llista de canals de l'usuari.

Com es pot veure a la imatge anterior, a la part superior apareix l'usuari, el seu nom i el seu cognom. Una barra de navegació per indicar la ubicació del sistema on es troba. Una llista dels canals subscrits per l'usuari i al peu un botó que li permet subscriure's a més canals.

De la taula dels canals subscrits, es va a comentar els camps que apareixen:

- **Estat**: Indica si en el procés d'afegir el canal, s'han processat correctament el document RSS que publica.
- **Conegut**: Indica si el canal es conegut pel sistema. (Més informació punt 3.3)
- **Nom**: És el nom que ha establert l'usuari al canal. Té un enllaç per poder accedir al manteniment de les regles de selecció.
- **Data d'inscripció i data d'actualització**. El primer indica la data en què l'usuari s'ha subscrit al canal i la segona l'última vegada que el canal s'ha actualitzat (procés de sincronització de notícies)
- **Total**: El nombre total de notícies publicades al canal.
- **Esborrar**: Icona que permet esborrar el canal.

Respecte al botó d'afegir un canal, ja s'ha parlat d'aquest tema als punts 3.3 i 3.4

▪ *Implementació de la gestió de Regles.*

Quan l'usuari selecciona un canal, vista figura 30, li apareix una vista amb les regles de selecció associades al canal (figura 16). El mètode que s'encarrega d'aquesta funció és listRegles(..). El procés que realitza aquest mètode, és molt semblant al procés descrit per a listCanals() al punt anterior, i és el següent:

- Trau informació de la sessió de l'usuari (l'identificatiu del canal a tractar).
- Crea un objecte de la classe **Regla** i crida al mètode getRegles(..). Aquest mètode trau un vector d'objectes de la classe **ReglaBean** amb la informació de cadascú de les regles que té associades aquest canal.
- Afegeix el vector d'objectes **ReglaBean** a la sessió de l'usuari, per a ser visualitzats per la vista **regles.jsp**.
- La vista **regles.jsp** trau de la sessió aquest vector i el tracta amb jstl. Així per cada objecte **ReglaBean** afegeix una fila a la taula amb informació de la regla corresponent.

La vista resultant es pot veure a la imatge de la figura 31.

Figura 31. Vista llista de les regles associades a un Canal.

Com es pot veure, la vista és molt semblant a la vista de la gestió de Canals. A la part superior apareix l'usuari, el seu nom i el cognom. Una barra de navegació per indicar la situació del sistema web on es troba i la llista de les regles de selecció de notícies associades al canal. Al peu, un botó per poder afegir més regles.

Comentar un poc les columnes que apareixen a la llista:

- **Estat**: Indica si la regla és correcta lèxicament. Quan s'afegeix una regla al sistema, es comprova si la regla és una expressió correcta mitjançant un compilador.

  En cas positiu, el compilador la tradueix a una expressió més fàcil d'executar pel sistema, i les dos expressions (la regla introduïda per l'usuari i resultat de la compilació) són emmagatzemades a la taula **regles** de la BBDD. Concretament la regla al camp **reglastr** i la traducció al camp **reglacode**.[1]

  En el cas que la compilació de la regla no sigui possible, s'avisa a l'usuari i la regla es emmagatzemada a la taula regles, però es marcada amb el flag d'estat a 0, la qual cosa és indicada per la columna estat a la llista amb un color roig (ver figura 31).

  Aquests aspectes, ja s'han tractat més detingudament al capítol 5.

- **Pendent**: Quan la regla s'ha introduït al sistema, es marcada com a regla pendent de processar. El mòdul SCA cada cert temps mira les regles pendents de processar i les processa (ver capítol 5).
- **Nom**: És l'identificatiu que ha establert l'usuari a la regla. Té un enllaç per poder modificar la regla en qüestió. En aquest cas apareix la vista mostrada a la imatge de la figura 32.
- **Avís**. Indica el tipus d'avís que l'usuari vol rebre quan una notícia valida la regla.
- **Data Creació**. Mostra la data en què l'usuari va crear la regla.

[1] Més informació punt 6.2 i següents.

o **Esborrar**: Icona que permet esborrar la regla.

Tant si l'usuari polsa el botó Afegir Regla com si polsa en l'enllaç que hi ha al nom de la regla. Apareix la vista **d'Editar/ Afegir Regla**. Figura 32.

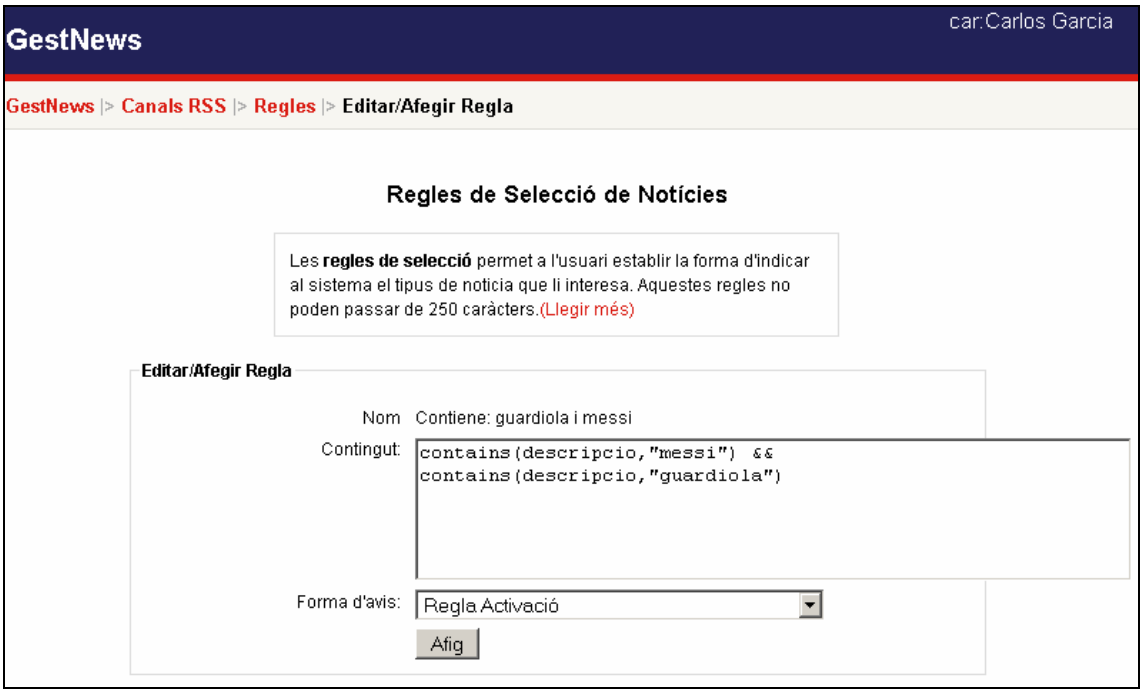


Figura 32. Vista Editar/Afegir Regles de Selecció.

A la figura anterior, es pot veure la vista **Editar/Afegir Regla**. En aquest cas es tracta d'editar una regla existent, ja que el nom de la regla no es pot modificar en aquest cas. Quan es pitja el botó Afegir Regla, la vista que apareix és molt semblant, però s'ha d'indicar el nom de la regla. Quan una regla s'afegeix o es modifica, la regla queda pendent de processar.

Tant la gestió d'afegir una regla com d'editar-la és realitzada pel mètode addRegla(..). Una altra cosa, són els mètodes encarregats de mostrar la vista Editar/Afegir Regla. En el cas d'afegir una regla es realitzat pel mètode mostrarAddRegla(..) i en el cas de l'edició, pel mètode editRegla(..). Aquestos dues mètodes són prou evidents pel que no vaig a fer cap comentari.

Al capítol 5 ja s'han tractat les regles de selecció, concretament al punt 4 es parla com s'ha implementat el mètode addRegla(..).

8. Capítol 8. Altres Aspectes del Sistema

8.1. Introducció.

En aquest capítol es tracten altres aspectes del sistema que es consideren d'interès, però per la seva naturalesa no encaixen en el capítols anteriors.

8.2. Configuració del Sistema.

Dins del projecte, en la carpeta WebContent/WEB-INF/conf hi ha el fitxer **propierties.cfg**. Aquest és un fitxer de propietats que conté diferents aspectes de configuració del sistema. Tant del component GUAP com del SCA. Ambdues components utilitzen un objecte de la classe Connexió, que és la encarregada de gestionar aquest paràmetres de configuració.

Hi ha paràmetres com l'usuari i contrasenya del servidor de Base de Dades, el servidor SMTP, etc.

També, en el mateix directori hi és el fitxer **log4j.propierties**. És un fitxer de propietats i conté las propietats de configuració del sistema de log. Per aquest objectiu s'ha utilitzat la llibreria log4j d'Apache, que com no pertany a les llibreries Standard de JAVA, s'ha afegit al projecte. La gestió de log es tractada per l'objecte Connexió.

8.3. Activació del Servei de Cerca Automatitzada de Notícies.

Com ja s'ha comentat en diverses ocasions, el component SCA és un component independent del Sistema[1]. Aquest component té dues formes de treballar, i la forma de fer-ho ho determina la propietat del sistema SERVEI_MODE:

- **CRON:** En aquest mode, el component funciona pensant que cada cert temps és activat mitjançant un agent extern: tipus el CRON del sistema en plataformes UNIX o AT en plataformes WINDOWS. Quan és activat, realitza els tres processos que té programats (sincronització de notícies, execució de regles de selecció i enviament d'avisos) i finalitza la seva execució.
- **ALONE:** En aquest mode, el component quan es fica en funcionament realitza el seu procés i es desactiva en SERVEI_SINC milisegons. Passat aquest temps, torna a realitzar tot el procés i es torna a desactivar. Aquest cicle terminarà quan s'aturi el servei.

8.4. Servidor Eixint de Correus.

Com s'ha comentat al punt 6.4.4 (capítol 6), per a l'enviament de missatges s'utilitza la llibreria **JavaMai**l. Aquesta llibreria necessita d'un servidor eixit per a l'enviament de correus, a més d'especificar totes les seves propietats que estan

[1] Més informació veure el capítol 2.

definides al fitxer de propietats del sistema[2]. Durant el procés de proves s'ha configurat un servidor de correu intern en una plataforma LINUX com a servidor smtp. Concretament un servidor **Debian** amb **Exim4**[3].

Les proves han segut positives. El sistema GestNews ha funcionat correctament amb aquest servidor de correus.

Com que GestNews necessita d'un servidor de correus i el servidor de correus que he utilitzat és a una xarxa interna (no té ip vàlida a Internet), he creat un compte a gmail.com (gestnews1@gmail.com) i he configurat el sistema perquè utilitzi el servidor smtp de yahoo (smtp.gmail.com).

Després de varies proves, he comprovat que aquest servidor no realitza reley extern, i per tant les proves les he hagut de realitzar amb comptes de gmail.com, és a dir, l'usuari quan s'inscriu al sistema, ha de tindre un compte en gmail.com perquè li puguin aplegar correus de GestNews.

9. Capítol 9. Conclusions.

Vivim actualment a la societat de la informació, que es caracteritza per tindre al nostre abast gran quantitat d'informació. Internet representa el recurs més important, al existir multitud de servidors de contingut que publiquen gran quantitat d'informació. Aquesta gran quantitat de recursos disponibles representa un gran avantatge, ja que sempre que cerquem algun tipus d'informació segur que la podem trobar, però aquesta cerca quasi sempre es tradueix en una pèrdua de temps en cercar justament allò que necessitem.

Per facilitar la recerca d'informació es va crear el format RSS i els lectors RSS. Així, els servidors de continguts o canals RSS, mitjançant aquest format, faciliten l'accés als seus recursos als usuaris subscrits. Aquesta facilitat, permet als usuaris conèixer les novetats sense la pèrdua de temps que requereix accedir als diferents canals d'informació. El lector RSS realitza aquest treball per ells i els informa de les últimes notícies publicades.

El sistema GestNews dóna un pas més en les facilitats que proporcionen als usuaris en la cerca d'informació d'interès. Així, l'usuari se subscriu als canals que poden publicar informació del seu interès i estableix una regles de selecció de notícies. Quan el sistema detecta que hi ha una notícia que compleix aquesta regla, és una notícia potencialment d'interès, l'avisa mitjançant diferent tipus d'informació.

Per poder facilitar informació d'interès a l'usuari, el sistema GestNews implementa dos components: el component GUAP que s'encarrega de replegar les preferències de l'usuari i el component SCA que és un motor encarregat de cercar les notícies i avisar a l'usuari quan troba una notícia potencialment d'interès per l'usuari.

Durant el desenvolupament ens hem trobat problemàtiques pròpies de l'assignatura de Compiladors com l'anàlisi de documents RSS, l'anàlisi de

[2] Més informació veure el punt 8.2.
[3] Referències: http://www.exim.org/

documents HTML, l'anàlisi i la compilació de les regles de selecció, etc. que ens ha permet aprofundir molt en aquesta àrea del coneixement: cercant i coneixent llibreries i eines existents, implementant soluciones específiques, etc., però a més també s'han hagut de tractar problemàtiques pròpies d'altres assignatures: tractament amb diferents components i la seva comunicació, ús d'un servidor d'aplicacions (assignatura: Programari de Components i Sistemes Distribuïts), l'ús d'una Base de Dades (assignatura:Base de Dades), etc.

Al capdavall, la problemàtica plantejada en el present TFC m'ha semblat molt interessant i completa, i m'ha permès aprofundir en diferents àrees del coneixement pròpies dels estudis d'Enginyeria Informàtica. Per un altre costat, amb el seu desenvolupament s'ha complit els objectius marcats al començament de la memòria.

L'ús d'eines com GestNews poden estalviar molt de temps als usuaris en la cerca d'informació sense pràcticament cap esforç, ja que és el sistema el que avisa als usuaris de les novetats relacionades amb el gust. Un exemple molt pràctic seria cercar diversos canals d'informació relacionats amb la publicació de convocatòria d'oposicions. Si l'usuari fica en cada canal per exemple la regla: contains(descipcio,"informàtica"). L'usuari estaria informat automàticament de totes les publicacions d'oposicions d'informàtica. Si pensem un poquet podem traure infinites possibilitats al sistema.

10. Glossari.

API (Application Programming Interface): Conjunt de classes que proporcionen una funcionalitat.

Aplicació web: Aplicacions que els usuaris poden accedir mitjançant un navegador web.

Arquitectura J2EE (Java 2 Enterprise Edition): És una plataforma de programació per desenvolupar aplicacions en JAVA, en un context distribuït.

BBDD: Són les sigles de Base de Dades.

Base de Dades: Una base de dades és un conjunt que pertanyen a un mateix context, emmagatzemades per a un ús posterior.

Bean: Component de software reutilitzable. Disposen de una interfície pública per tal de instanciar-los i uns mètodes per accedir als atributs i insertar valors als atributs (mètodes setter i getters).

Document XML: Document escrit utilitzant el format XML. Consta d'una capçalera i de la instància del document. Tot document XML ha de seguir la sintaxi del llenguatge XML així ha de poder ser representat en forma d'arbre amb una única arrel.

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.

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

Framework: Un marc de treball que disposa d'un conjunt de tècniques i eines estàndards per a resoldre un problema particular, orientat a facilitar el desenvolupament.

HTML: Llenguatge de marques per crear pàgines web.

JAXB: API que facilita la creació de documents XML. Proporciona una manera senzilla i transparent per a mapejar documents XML i crear objectes Java.

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

JDBC: Manejador per connectar java a una base de dades

JSP (Java Server Pages): Una tecnologia Java que permet generar contingut dinàmic per una web, usant documents HTML, XML o altres.

JSTL (JavaServer Page Standard Tag Library): Component de J2EE que estén les pagines JSP proporcionant quatre biblioteques de tags amb utilitats àmpliament usades en el desenvolupament de pagines web dinàmiques.

MVC: sigles que determinen el model de disseny Model Vieu Controler. És un patró de disseny que separa les dades d'una aplicació, la interfície d'usuari i la lògica de control en tres components diferents.

Patró de disseny: Plantilla de disseny que es fa servir com a solució a un problema conegut.

Registrar-se: Complimentar un formulari per ser introduir les dades a un sistema.

SAX (*Simple API for XML*): API que implementa un analitzador sintàctic XML. Es llegeix el document XML de manera seqüencial generant esdeveniments cada cop que s'identifica un element significatiu.

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 i determina quina és la següent vista que s'ha d'executar.

SGBD: sigles que identifiquen a un Sistema Gestor de Bases de Dades. És un tipus de programari molt específic encarregat de gestionar una Base de Dades.

Tag: Etiqueta o marca d'un llenguatge basat en XML que defineix algun element.

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.

XSLT (*eXtensible Stylesheet Language Transformations*): Llenguatge basat en XML que permet definir un conjunt de regles per transformar els elements d'un document XML.

11. Bibliografia.

API JAVA

http://download.oracle.com/javase/6/docs/api/

Expressions Regulars:

http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular

http://www.regular-expressions.info/reference.html

http://iie.fing.edu.uy/~vagonbar/unixbas/expreg.htm

ICALENDAR

http://en.wikipedia.org/wiki/ICalendar

http://tools.ietf.org/html/rfc5545

ODT

http://en.wikipedia.org/wiki/Odf

http://es.wikipedia.org/wiki/OpenDocument

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office#technical

JavaMail

http://www.oracle.com/technetwork/java/javamail/index.html

http://www.chuidiang.com/java/herramientas/javamail/enviar-correo-javamail.php

http://www.chuidiang.com/java/herramientas/javamail/empezar-javamail.php

JAXP

http://en.wikipedia.org/wiki/Java_API_for_XML_Processing

http://jaxp.java.net/1.4/1.4.5/ReleaseNotes.html

http://www.programacion.com/articulo/el_api_jaxp_113

JBOSS

http://en.wikipedia.org/wiki/JBoss_application_server

http://www.jboss.org/jbossas/docs/6-x

JDBC

http://www.oracle.com/technetwork/java/javase/jdbc/index.html

http://www.developer.com/java/data/article.php/3417381

JSTL

http://www.oracle.com/technetwork/java/index-jsp-135995.html

http://www.cs.wcupa.edu/docs/tld/

RFC822

http://www.faqs.org/rfcs/rfc822.html

http://www.rfc-editor.org/info/rfc822

RSS

http://en.wikipedia.org/wiki/RSS

http://cyber.law.harvard.edu/rss/rss.html

SERVLETS

http://www.java-tips.org/java-tutorials/tutorials/introduction-to-java-servlets-with-eclipse-1.html

http://www.javaworld.com/javaworld/jw-11-2002/jw-1129-jsf.html

http://www.javaworld.com/javaworld/jw-12-2002/jw-1227-jsf2.html

http://www.oracle.com/technetwork/articles/javase/javaserverfaces-135231.html

12. Annexos.

12.1. Estructura de directoris del Projecte.

Tot i que a la memòria ja s'han comentat alguns detalls de l'estructura de directori del projecte. A continuació es mostra detalladament, indicant la funcionalitat de les parts més rellevants:

- **build:** Classes compilades del projecte**.**
- **src:** Fonts de les classe del projecte. Dins d'aquest directoris es troben els següent paquets:
  - **gestnews.core.analitzador**: Classes dels analitzadors de les regles de selecció.
  - **gestnews.core.bean**: Classes bean del model de persistència de dades.
  - **gestnews.core.exception**: Classes que modelen les excepcions definides al projecte.
  - **gestnews.core.main**: Classes que modelen els objectes principals del projecte: Canals, Notícies, Connexió, Avís, etc.
  - **gestnews.core.outbean**: Classes bean utilitzades per la generació dels avisos.
  - **gestnews.core.servei**: La classe principal i altres relacionades únicament amb el component SCA.
  - **gestnews.core.servlet**: Classes que implementa la servlet.
- **temp:** Directori utilitzat pel sistema per emmagatzemar fitxers temporals (cache de canals, etc.)
- **Template:** Directori que conté les plantilles xslt i plantilla per generar l'avís del registre en format ODT.
- **WebContent:** Directori d'ús exclusiu del servidor d'aplicacions i conté els següent subdirectoris:
  - **css**: Plantilles css utilitzades en les vistes web.
  - **html**: Pàgines web en format html pur.
  - **imatges**: Imatges utilitzades per les vistes web.
  - **resultats**: Fitxers jsp que creen les vistes web de la interfície.
  - **WEB-INF:** Conté el directori amb els fitxers de configuració de diferents aspectes del sistema. I el fitxer web.xml amb la informació del desplegament del sistema en el servidor d'aplicacions.

12.2. Pla de Proves.

Al present annex es presenta els aspectes més significatius de les proves realitzades al sistema i les seves valoracions.

Dins del directori **query,** que s'adjunta amb la memòria, he deixat un fitxer sql. Aquests fitxers possibiliten la càrrega de diferents dades al sistema (partint de la BBDD buida), per no tindre-les que introduir per la interfície web, la qual cosa facilita les diferents execucions indicades al pla de proves.

El mode de funcionament del Servei de Cerca automatitzat és ALONE[4]. D'esta forma es pot llançar als moments desitjats per veure els resultats.

Creació d'un usuari al sistema: query/alta_user1.sql

Es crea l'usuari carlos amb el e-mail gestnews1@gmail.com. Els usuaris del sistema han de tindre una direcció de correu en el domini de gmail.com, ja que el servidor smtp configurat no admet relay extern[5].

Des d'aquest moment ja es pot accedir al sistema via web amb aquest usuari.

Subscripció a Canals RSS coneguts.

Subscripció a un canal conegut pel sistema: Canal Sport. Adreça: http:// www.sport.es/es/rss/last_news/rss.xml. El canal es donat d'alta correctament. Es pot veure a la vista de canals, que és un canal conegut. El nombre de notícies 0, ja que no s'ha activat el Servi de Cerca Automatitzada.

S'activa manualment el mòdul SCA. Es pot veure (mirant al log) com comença el procés de sincronització de notícies, accedeix a les diferents notícies i troba el contingut de la notícia publicada.

Si s'actualitza la vista dels canals subscrits en aquest moment, es pot veure que hi ha 15 notícies. Si es mira a la taula de **Notícies** es pot veure com se les ha descarregades i a la taula de **Notícies_contingut** com també s'ha descarregat el contingut de les notícies.

Subscripcions a Canals RSS no conegut[6]. query/alta_canal2.sql (alta d'aquest dues canals)

Subscripció al canal RSS de C9. Adreça: http://www.rtvv.es/rss/. El canal s'afegeix correctament. En la vista de canals subscrits, es pot veure que el procés ha segut correcte, però que no és un canal conegut pel sistema. Com en el cas anterior, apareix que el canal té 0 notícies.

S'activa manualment el mòdul SCA. Es veu al log com realitza la sincronització amb el canal Sport i posteriorment amb C9. Si s'actualitza la vista dels canals

[4]Més informació veure el punt 8.3.
[5]Més informació veure el punt 8.4.
[6] Més informació veure el punt 3.3.

subscrits, es veu que al C9 apareixen 10 notícies. Es pot mirar a la taula **Notícies** per veure les notícies, però com que no és un canal conegut no apareixen a la taula **Notícies_conegut**.

Nota:

S'ha provat la subscripció a altres servidors de contingut no coneguts: ElPAIS[7], Antena3[8], UOC, Telecinco, etc. A tots ells, el sistema ha permès la subscripció correctament. Al fitxer query/alta_canal4.sql. Està la subscripció del canals: Sport, C9, ELPAIS i A3.

Eliminar la Subscripció a un Canal.

S'han eliminat la subscripció als canal C9 i A3 mitjançant la interfície web sense cap problema.

Afegir regla de selecció per títol.

S'ha volgut seleccionar les notícies que el títol comenci amb la paraula "villa"[9]: begin(titol,"villa).

El sistema ens diu que s'ha afegit correctament al sistema i es pot veure a la vista de regles del canal, que la regla està pendent de ser processada. S'activa el Servei de Cerca i es pot veure com una notícia activa correctament la regla i el sistema envia el succés.

Afegir regla de selecció per descripció.

S'ha volgut seleccionar les notícies que aparegui la paraula "messi" i "guardiola". contains(descripcio,"messi") && contains(descripcio,"guardiola") [10]

El sistema ens diu que s'ha afegit correctament al sistema i es pot veure a la vista de regles del canal, que la regla està pendent de ser processada (i la regla que s'ha afegit anteriorment ja s'ha processat). S'activa el Servei de Cerca i es pot veure com una notícia activa correctament la regla i el sistema envia el succés.

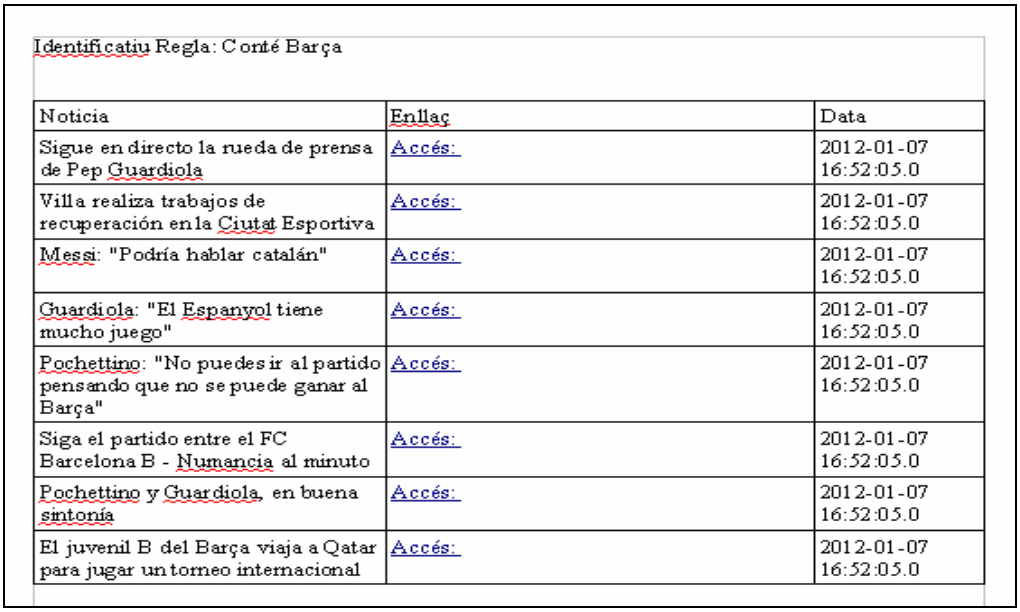
Afegir regla de selecció per contingut de notícia (canal conegut)

Per a l'usuari afegir una regla de selecció de notícia per contingut a un canal conegut o no conegut pel sistema li és transparent. És el sistema qui actua d'una forma u altra en funció de si és un canal conegut o no conegut[11].

S'ha volgut seleccionar les notícies on aparegui la paraula "barça" al contingut de la notícia publicada. Com és molt provable que apareguin més d'una (aquesta regla és aplicada al canal Sport), s'utilitza l'avís registre en format ODT.

[7] http://www.elpais.com/rss/feed.html?feedId=1022
[8] http://www.antena3.com/rss/1.xml
[9] Evidentment, s'ha mirat al canal que hi hagués notícies que compleixen aquest criteri per poder comprovar que els sistema les selecciona. Si es tornen a aplicar aquestes regles (més endavant en el temps), potser que es no seleccioni cap, per no estar publicades les mateixes notícies.
[10] Recordem que la cerca es CaseInsensitive
[11] Veure Punt 3.3.

S’activa el Servei de Cerca i com era d’esperar hi ha 8 notícies publicades que al seu contingut apareix la paraula “barça”. Es pot veure a la següent imatge.



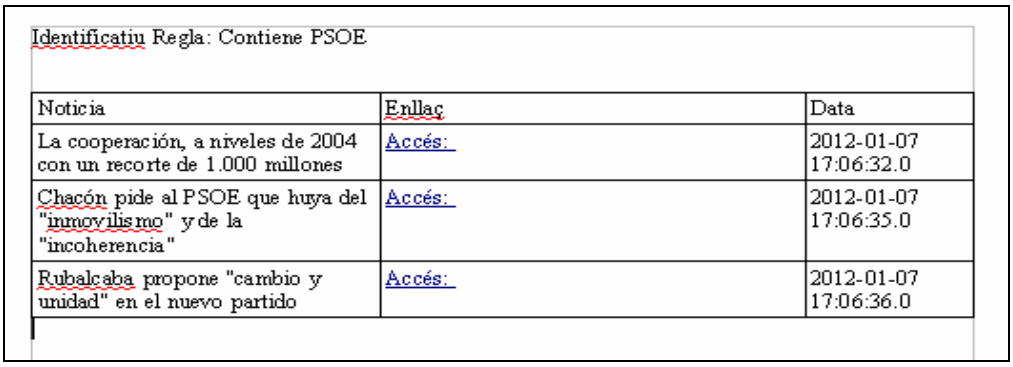
Identificatiu Regla: Conté Barça

Noticia	Enllaç	Data
Sigue en directo la rueda de prensa de Pep Guardiola	Accés:	2012-01-07 16:52:05.0
Villa realiza trabajos de recuperación en la Ciutat Esportiva	Accés:	2012-01-07 16:52:05.0
Messi: "Podría hablar catalán"	Accés:	2012-01-07 16:52:05.0
Guardiola: "El Espanyol tiene mucho juego"	Accés:	2012-01-07 16:52:05.0
Pochettino: "No puedes ir al partido pensando que no se puede ganar al Barça"	Accés:	2012-01-07 16:52:05.0
Siga el partido entre el FC Barcelona B - Numancia al minuto	Accés:	2012-01-07 16:52:05.0
Pochettino y Guardiola, en buena sintonía	Accés:	2012-01-07 16:52:05.0
El juvenil B del Barça viaja a Qatar para jugar un torneo internacional	Accés:	2012-01-07 16:52:05.0

Afegir regla de selecció per contingut de notícia (canal no conegut)

Ara es vol treballar amb el canal ELPAIS. Com que hi ha tantes notícies relacionades amb el PSOE aquestos dies, volem saber quantes notícies publicades parlen en el seu cos del PSOE. Aleshores establim la següent regla: contains(notícia,"psoe").

Quan s’activa el Servei de Cerca, es pot veure que hi ha tres notícies que activen la regla. A la bústia apareix un missatge amb el registre en format ODT següent:



Identificatiu Regla: Contiene PSOE

Noticia	Enllaç	Data
La cooperación, a niveles de 2004 con un recorte de 1.000 millones	Accés:	2012-01-07 17:06:32.0
Chacón pide al PSOE que huya del "inmovilismo" y de la "incoherencia"	Accés:	2012-01-07 17:06:35.0
Rubalcaba propone "cambio y unidad" en el nuevo partido	Accés:	2012-01-07 17:06:36.0

Al sistema s’han aplicat molts altres regles, tant en canals coneguts com no coneguts i els resultats han segut correctes. Per no fer massa extens l’annex no es mostren els resultats.

12.3. Millores.

Al següent apartat es vol indicar alguns aspectes per millorar el sistema:

- Als canals coneguts es realitza una extracció de la notícia per una adreça XPATH. L'analitzador SAX és un analitzador molt potent però no crec que sigui el més adient per analitzar documents HTML, ja que en les proves que he realitzat m'he trobat amb canals de contingut que publiquen documents HTML mal formats. Aleshores, una possible millora seria realitzar un preprocés més robust per eliminar aquestes errades.

  També vull valorar el gran treball dels desenvolupadors dels navegadors actuals al realitzar un anàlisi d'un document HTML mal format i no ser sensibles a moltes errades.

- Implementar més tipus d'avisos del sistema. Per exemple, la generació d'un registre d'avisos en HTML. De la forma que està desenvolupat el sistema, introduir nous tipus d'avisos no representa molt de treball, simplement implementar noves plantilles **xslt**.

- Donar la possibilitat a l'usuari de veure el contingut de les notícies des del sistema. Implementar aquesta millora no és complex, ja que totes aquestes dades estan emmagatzemades a la BBDD del sistema. Simplement s'haurien de mostrar mitjançant una transformació xslt.

- Millorar la seguritat de la interface web. La interface web no ha segut desenvolupada pensant en aspectes de seguretat i càrrega. Són uns aspectes molts importants actualment i que per qüestions de temps no s'han tractat en profunditat.