MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DOMENIUL CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI
SPECIALIZAREA CALCULATOARE

TEHNOLOGIA OPEN SERVICES GATEWAY INITIATIVE FOLOSITA IN DEZVOLTAREA UNUI SISTEM SMART HOME

PROIECT DE DIPLOMĂ

Autor: **Oxana Maria HOTEA**

Coordonator: **Şef lucrări ing. Cosmina IVAN**

2010

MINISTERUL EDUCAŢIEI ŞI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DOMENIUL CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI
SPECIALIZAREA CALCULATOARE

VIZAT,

DECAN,
Prof. dr. ing. Sergiu NEDEVSCHI

ŞEF CATEDRĂ,
Prof. dr. ing. Rodica POTOLEA

Autor: **Oxana Maria HOTEA**

TEHNOLOGIA OPEN SERVICES GATEWAY INITIATIVE FOLOSITA IN DEZVOLTAREA UNUI SISTEM SMART HOME

1. **Enunţul temei:** Proiectul îşi propune dezvoltarea unui sistem *smart home* bazat pe aplicaţia open source HomeRun, aplicaţie care permite gestionarea interacţiunii cu mediul fizic înconjurător. Constatând lipsa unor module orientate pe parte software, au fost dezvoltate câteva module în scopul extinderii funcţionalităţilor aplicaţiei care nu necesită echipamente tehnice suplimentare, doar o conexiune la internet şi care îşi dovedesc utilitatea în diverse situaţii.

2. **Conţinutul proiectului:** Pagină de prezentare, Sinteză, Introducere, Fundamentare teoretică, Concepte generale ale aplicaţiei HomeRun, Arhitectura şi proiectarea aplicaţiei, Utilizarea aplicaţiei, Testarea şi evaluarea performanţelor aplicaţiei, Concuzii şi dezvoltări ulterioare, Bibliografie, Acronime, Anexe.

3. **Locul documentaţiei:** Universitatea Tehnică din Cluj-Napoca, Facultatea de Automatică şi Calculatoare

4. **Consultanţi:**

5. **Data emiterii temei:** 1 noiembrie 2009

6. **Data predării:** 25 iunie 2010

Semnătura autorului _____

Semnătura coordonatorului _____

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DOMENIUL CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI
SPECIALIZAREA CALCULATOARE

Declaraţia autorului,

Subsemnata Oxana Maria HOTEA, studentă a Facultăţii de Automatică şi Calculatoare, Universitatea Tehnică din Cluj-Napoca, declar că ideile, analiza, proiectarea, implementarea, rezultatele şi concluziile cuprinse în acest proiect de diplomă constituie efortul meu propriu, mai puţin acele elemente ce nu îmi aparţin, pe care le indic şi recunosc ca atare.

Declar de asemenea că, după ştiinţa mea, lucrarea în această formă este originală şi nu a mai fost niciodată prezentată sau depusă în alte locuri sau alte instituţii decât cele indicate în mod expres de mine.

Data: 25 iunie 2010

Autor: **Oxana Maria HOTEA**
Număr matricol: 21010308
Semnătura:_____

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DOMENIUL CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI
SPECIALIZAREA CALCULATOARE

SINTEZA

proiectului de diplomă cu titlul:

TEHNOLOGIA OPEN SERVICES GATEWAY INITIATIVE FOLOSITA IN DEZVOLTAREA UNUI SISTEM SMART HOME

Autor: **Oxana Maria HOTEA**
Coordonator: **Şef lucrări ing. Cosmina IVAN**

1. **Cerinţele temei:** Proiectul îşi propune dezvoltarea unui sistem *smart home* bazat pe aplicaţia open source HomeRun, aplicaţie care permite gestionarea interacţiunii cu mediul fizic înconjurător. HomeRun este proiectat ca un sistem modular ceea ce permite adăugarea cu uşurinţă a unor funcţionalităţi noi.

2. **Soluţii alese:** Pentru a funcţiona, majoritatea modulelor au nevoie de dispozitive electronice care nu sunt la îndemâna oricărui utilizator. Din această cauză am ales dezvoltarea unor module care să nu necesite echipamente tehnice suplimentare, doar o conexiune la internet, şi care îşi dovedesc utilitatea în diverse situaţii.

3. **Rezultate obţinute:** Pe parte practică am reuşit să extind funcţionalităţile aplicaţiei HomeRun prin adăugarea unor module noi.

4. **Testări şi verificări:** Pentru partea de testare a fost instalată aplicaţia server pe un calculator, iar aplicaţia client pe mai multe calculatoare pentru a verifica funcţionarea într-o manieră concurentă a acestora.

5. **Contribuţii personale:** Dezvoltarea modulelor „SMS” şi „News” în cadrul aplicaţiei HomeRun, precum şi modificări aduse modulului existent „Weather”.

6. **Surse de documentare:** Principalele surse sunt:
Alianţa OSGi, http://www.osgi.org/
N. Bartlett, „OSGi in practice”, Londra, 2009
HomeRun, http://homerun.sourceforge.net/

Data: 25 iunie 2010 Semnătura autorului_____

Semnătura coordonatorului_____

Cuprins

.

Lista de figuri

Capitolul 1 INTRODUCERE

1.1 Motivaţie

Ideea de la care am pornit în vederea realizării lucrării de licenţă consta în dezvoltarea unei aplicaţii care să folosească tehnologia OSGi. Conceptele de bază referitoare la această tehnologie au fost studiate şi în cadrul cursului de Calcul paralel şi distribuit, prezentând interes pentru a fi aprofundate.

Mulţi dintre noi folosesc tehnologia OSGi fără a fi conştienţi de acest lucru. Beneficiile folosirii acestei tehnologii le-am observat pentru început lucrând în mediul de dezvoltare Eclipse. Abia mai târziu am aflat că plug-in-urile pe care le foloseşte Eclipse IDE urmează modelul bundle-urilor OSGi.

Am căutat dupa aceea alte domenii şi aplicaţii în care tehnologia OSGi a reuşit să vină cu soluţii noi şi eficiente. Tehnologia a fost adoptată de o gamă largă de pieţe: de la domeniul afacerilor, al telefoniei mobile sau chiar domeniul auto, până la cel al automatizării locuinţelor sau domeniul sanitar.

Oprindu-mă la domeniul automatizării locuinţelor am constatat că multe firme de renume au ales tehnologia OSGi pentru dezvoltarea produselor. Printre acestea se numără Bosh şi Siemens cu sistemul BSH sau Philips cu produsul Philips iPronto. Poate singurul dezavantaj pe care l-am întâlnit la aceste produse a fost preţul relativ ridicat, un sistem iPronto putând depăşi 2000$.

Din dorinţa dezvoltării unei aplicaţii de automatizare a locuinţelor, sau cel puţin a unei aplicaţii care să vină în ajutorul oamenilor, am pornit în căutarea unei alternative la produsele costisitoare amintite mai sus. Astfel am găsit aplicaţia open source HomeRun, de altfel distribuită gratuit, pe care am hotărât să o dezvolt. Aplicaţia este împărţită pe mai multe module, fiecare cu o funcţionalitate bine definită. Folosirea câtorva module presupune existenţa unor dispozitive din categoriile Insteon şi X10, cum ar fi senzori de mişcare, de lumină etc. In consecinţă am optat pentru varianta dezvoltării unor module care să nu presupună echipamente specializate, având la dispoziţie doar accesul la internet.

Există nenumărate avantaje care derivă din utilizarea unui sistem open source. Printre acestea câteva ar fi următoarele: independenţa faţă de furnizor, stabilitatea (când codul sursă este public, el este testat de o multitudine de utilizatori, problemele sunt detectate rapid şi rezolvate la fel de repede, fără a fi ascunse în produs), informaţiile nu sunt ascunse în software (toate informaţiile sunt disponibile, oricine poate să testeze şi să utilizeze toate funcţionalităţile) şi, de asemenea, nu există costuri de licenţiere pentru aplicaţie, ceea ce înseamnă un cost de achiziţie şi întreţinere mult mai redus faţă de sistemele comerciale.

1.2 Context

Arhitectura unei aplicaţii convenţionale pentru locuinţe inteligente (smart home) este, de obicei, concentrată în jurul unui server şi acest lucru cauzează multe probleme. Dispozitivele mobile şi serviciile dinamice creează un mediu într-o continuă şi dinamică schimbare care poate genera interacţiuni dificil de realizat. Pe lângă acest lucru, furnizarea de servicii eficiente şi corespunzătoare este întotdeauna o problemă importantă pentru locuinţele inteligente. Pentru a rezolva problemele cauzate de o arhitectură tradiţională, pentru a face faţă mediului dinamic şi pentru a furniza serviciile corespunzătoare, s-a dezvoltat o arhitectură orientată pe servicii bazată pe tehnologia OSGi. Mecanismul orientat pe servicii e folosit în interacţiunea dintre componentele sistemului.

O aplicaţie smart home tradiţională este de obicei bazată pe o arhitectură centralizată, conform figurii 1.1. Dispozitivele (aparatele) din locuinţă sunt conectate prin intermediul unei reţele interne (Home Network) şi controlate de către Home Gateway, care este platforma

de furnizare a serviciilor pentru locatari. In cazul în care apar probleme la nivel de Gateway, întreg sistemul este compromis [14].

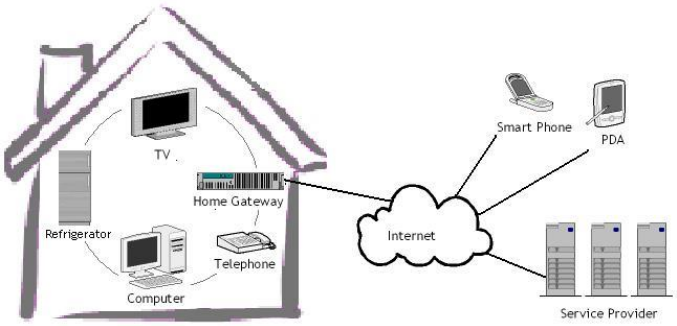

Figura 1.1 Arhitectura tradiţională a unei aplicaţii smart home

Pentru a dezvolta o aplicaţie smart home, este important să se respecte standardele deschise. Altfel, vor apărea proprietăţi ale sistemului ce vor cauza incompatibilitate. In acest sens, OSGi pare a fi o bună alegere, dând posibilitatea dezvoltatorilor de a adăuga pe parcurs noi componente, de a le actualiza sau şterge fără a fi nevoie să se modifice întreaga structură a aplicaţiei.

1.3 Temă

HomeRun este o aplicaţie software care ne permite să gestionăm interacţiunea cu mediul fizic înconjurător. Această interacţiune poate lua mai multe forme precum automatizarea şi orchestrarea dispozitivelor electronice sau monitorizarea şi comunicarea schimbărilor de mediu, aplicaţia putând să devină „centrul de comandă“ al oricărei locuinţe.

HomeRun este destinat tuturor celor interesaţi să exploreze domeniul automatizării proceselor. Fără îndoială că performanţele unui sistem automatizat rezultă din dotarea cu echipamente electronice de automatizare a acestuia, însă partea de comandă şi control nu este deloc de neglijat.

Pe lângă funcţionalităţile deja existente cum ar fi: monitorizarea senzorilor de lumină, de căldură, controlarea dispozitivelor (controlul luminii) etc. se pot adăuga alte operaţiuni ca de exemplu: trimiterea automata de email-uri prietenilor la zile de naştere sau doar trimiterea unor mesaje care să îţi reamintească lucrurile importante, trimiterea ştirilor actualizate din domeniile care prezintă interes etc. HomeRun are o arhitectură modulară, ceea ce permite instalarea doar a acelor pachete având funcţionalităţile dorite de către utilizator.

Aplicaţia este gândită într-o manieră client – server, ceea ce în termeni non-tehnici înseamnă doar că este compusă din două părţi independente dar care cooperează. Partea server trebuie să ruleze pe un singur calculator care să fie în permanenţă pornit şi conectat la internet. Aplicaţia client, pe de altă parte, poate să ruleze pe oricâte calculatoare în acelaşi timp (inclusiv pe cel pe care s-a instalat serverul) şi acestea nu trebuie să fie în permanenţă pornite, ci doar atunci când este nevoie de ele.

1.4 Obiective

Primul obiectiv este acela de a furniza o scurtă introducere în domeniul arhitecturii OSGi. Fără cunoştinţele privind acest topic ar fi imposibil de înţeles şi atins următoarele obiective. Al doilea obiectiv este mai important şi constă în înţelegerea conceptelor referitoare la arhitectura OSGi (înţelegerea noţiunilor de bundle, serviciu, cadru de lucru etc.). Cel de al treilea şi cel mai important obiectiv îl constituie aplicaţia HomeRun în sine.

Aplicaţia HomeRun este proiectată ca un sistem modular şi fiecare modul are propria funcţionalitate. Majoritatea modulelor sunt specializate pe parte hardware, pentru

funcţionarea lor fiind nevoie de echipamente de automatizare aparţinând tehnologiilor Insteon şi X10. Astfel de echipamente sunt senzorii de mişcare, dispozitivele pentru controlul luminii sau a căldurii etc.

Fiind un proiect open source, funcţionalităţile sale pot fi extinse prin adăugarea de noi module specializate pe diverse operaţii. Obiectivul este de a extinde funcţionalităţilor existente prin adăugarea unor module specializate pe parte software care să nu necesite existenţa echipamentelor mai sus menţionate, acestea nefiind la îndemâna tuturor potenţialilor utilizatori. Acest obiectiv conduce spre un altul: acela de a realiza o documentaţie de calitate a aplicaţia spre a servi viitorilor dezvoltatori sau altor persoane interesate.

Ultimul obiectiv este mai puţin legat de partea teoretică şi practică a lucrării, întrucât este unul personal. Obiectivul constă în acumularea de noi cunoştinţe, abordarea şi aprofundarea unei noi tehnologii cum este OSGi. Este în acelaşi timp obiectivul de a învăţa să îmi planific eficient şi efectiv lucrul, de a lucra independent cu minimum de supervizare şi de a dobândi abilităţi de evaluare critică.

1.5 Structura lucrării

Această lucrare include şase secţiuni majore: fundamentare teoretică, concepte generale ale aplicaţiei HomeRun, arhitectura şi proiectarea aplicaţiei, utilizarea sistemului, testarea şi evaluarea performanţelor sistemului, concluzii şi dezvoltări ulterioare.

In capitolul 2 este făcută o scurtă incursiune în aspectele tehnologiei OSGi. Se prezintă succint arhitectura bazată pe bundle-uri şi servicii, principalele cadre de lucru OSGi, avantajele tehnologiei şi câteva alternative la tehnologia OSGi. In final se prezintă câteva informaţii cu privire la unealta open source de dezvoltare a software-ului, Apache Ant.

Capitolul 3 prezintă conceptele generale ale aplicaţiei HomeRun. Sunt descrise resursele aplicaţiei şi modul în care se interacţionează cu acestea.

Capitolul 4, „Arhitectura şi proiectarea aplicaţiei”, prezintă arhitectura aplicaţiei gândită şi construită într-o manieră client - server, câteva cazuri de utilizare şi detalii de implementare, precum şi o scurtă descriere a pachetelor cu funcţionalităţile corespunzătoare.

In capitolul 5 este prezentat un studiu de caz cu privire la instalarea serverului si a aplicaţiei client, configurarea HomeRun şi un exemplu de utilizare a modulului Twitter SMS.

In capitolul 6 se testează şi se evaluează performanţele aplicaţiei HomeRun, iar în final, în ultimul capitol, sunt relatate cele mai importante concluzii şi câteva idei cu privire la dezvoltările ulterioare.

Lucrarea conţine în încheiere o scurtă listă de acronime folosite în cadrul redactării, iar în anexă sunt detaliate câteva clase ale aplicaţiei HomeRun.

1.6 Metodologie

Abordarea acestei lucrări a fost planificată în felul următor: pentru început cercetare în domeniul tehnologiei OSGi şi al aplicaţiilor care folosesc această tehnologie, urmând dezvoltarea unei astfel de aplicaţii. Etapa de implementare a aplicaţiei trebuie să coincidă cu începerea scrierii documentaţiei (atât partea de analiză şi proiectare a aplicaţiei, cât şi fundamentele teoretice cu privire la domeniul studiat). Se va finaliza cu scrierea ideilor introductive şi a concluziilor. Obiectivele au fost îndeplinite, dar procesul de elaborare nu a urmat cu desăvârşire aceşti paşi.

Conform planificării, la început a fost multă muncă de cercetare, majoritatea fiind în domeniul tehnologiei OSGi. Informaţiile cu privire la acest subiect au fost găsite în majoritate pe internet. Fiind o tehnologie relativ nou apărută, nu există multe cărţi care să o abordeze. Cunoştinţele mele despre acest subiect erau foarte limitate, de aceea etapa de cercetare a

necesitat o perioadă îndelungată de timp în care am citit despre fundamentele tehnologiei şi avantajele pe care aceasta le oferă. In acestă perioadă am înţeles foarte clar faptul pentru care tehnologia OSGi a avut un asemenea impact asupra programării software. Partea de cercetare a inclus de asemenea cadrele de lucru OSGi. Existând multe cadre de acest gen, m-am oprit doar la cele de referinţă. La început mi-a fost greu să fac trecerea de la modularitatea Java bazată pe JAR-uri, la modularitatea OSGi care are la bază bundle-urile, conceptele fiind puţin diferite. Pentru a înţelege modul în care bundle-urile acţionează şi interacţionează între ele a trebuit să citesc câteva tutoriale orientate pe diferitele cadre de lucru OSGi.

Următoarea etapă a constituit-o alegerea unei aplicaţii care să folosească tehnologia OSGi. Deşi tehnologia este relativ nouă, domeniile în care a fost integrată sunt destul de numeroase. M-am oprit asupra domeniului automatizării locuinţelor unde am găsit aplicaţia open source HomeRun pe care am hotărât să o dezvolt.

La început, partea de implementare a prezentat câteva probleme întrucât, deşi citisem noţiunile teoretice, partea practică s-a dovedit a nu fi atât de uşoară. Problemele au apărut la partea de comunicare între bundle-uri, însă după o altă etapă de cercetare, am reuşit să soluţionez şi aceste probleme.

Metodologia pe care am abordat-o în realizarea proiectului se numeşte SCRUM. SCRUM este o metodologie de dezvoltare Agile care se bazează pe o abordare iterativă, incrementală. Intâi a fost întâlnirea cu coordonatorul lucrării, doamna profesoară Cosmina Ivan, întâlnire în care s-a planificat o temă generală a lucrării. Al doilea pas l-a constituit planificarea sarcinilor şi estimarea viitoarelor întâlniri. Perioada dintre două întâlniri, care erau la distanţă de una-două săptămâni, reprezenta un ciclu de lucru, fiecare finalizându-se cu nişte rezultate mai mult sau mai puţin promiţătoare. La fiecare întâlnire rezultatele erau analizate şi dezbătute urmând stabilirea sarcinilor următoare, până la definitivarea aplicaţiei şi îndeplinirea obiectivelor.

Capitolul 2 FUNDAMENTARE TEORETICA

2.1 Alianţa OSGi şi standardele impuse

OSGi – „Open Services Gateway initiative" – este un standard definit de către o alianţă de aproximativ patruzeci de companii. Specificaţiile sunt disponibile gratuit, fiind destul de cuprinzătoare şi uşor de înţeles astfel încât să se poată realiza cu uşurinţă o implementare doar pe baza documentelor de referinţă [1].

Alianţa OSGi este un consorţiu internaţional de inovatori în domeniul tehnologiei software. Acestă organizaţie non-profit a fost fondată de IBM, Oracle şi Sun Microsystems în luna martie a anului 1999 cu scopul de a promova o platformă Java care să asigure crearea de aplicaţii complexe bazate pe conceptul de modularitate şi care să fie în acelaşi timp uşor de dezvoltat şi întreţinut. Specificaţiile permit aplicaţiilor să ascundă implementarea de alte componente în timp ce comunică cu acestea prin servicii. Se urmăreşte în continuare o îmbunatăţire a actualului sistem de module reprezentat prin clasicele jar-uri.

Rolul alianţei OSGi este de a defini specificaţii pentru noile versiuni ale platformei şi de a certifica implementările versiunilor curente. Munca tehnică este realizată de câteva grupuri de experţi (Expert Groups – Egs) repartizaţi pe mai multe domenii: Core Platform EG, Mobile EG, Vehicle EG, Enterprise EG.

2.2 Arhitectura OSGi

Tehnologia OSGi este definită ca fiind un set de specificaţii ce definesc un sistem de componente Java dinamice. Aceste specificaţii compun un model de dezvoltare în care aplicaţiile sunt compuse dinamic din mai multe componente diferite şi reutilizabile. Specificaţiile OSGi permit componentelor sa-şi ascundă implementarea faţă de alte componente în timp ce permit comunicarea prin servicii, acestea din urmă fiind obiecte ce sunt oferite componentelor în mod special [13].

Deşi astfel de componente sunt disponibile pe piaţa de software de mai multă vreme, până în prezent ele nu au fost adoptate pe scară largă. OSGi este prima tehnologie care a reuşit, ca şi sistem bazat pe componente, să rezolve multe probleme reale în dezvoltarea de soft-uri. Cei care au adoptat tehnologia OSGi au observat reducerea semnificativă a complexităţii în aproape toate aspectele de dezvoltare. Codul este mai uşor de scris şi de testat, posibilitatea de reutilizare a componentelor este crescută, dezvoltarea sistemelor se simplifică semnificativ, erorile sunt depistate cu uşurinţă, iar rularea programelor conferă o transparenţă ridicată a ceea ce se execută. Dar poate unul dintre cele mai importante lucruri e că OSGi este folosit în aplicaţii ca Eclipse şi Spring.

2.2.1 Stratificare

Arhitectura OSGi are un model stratificat, ilustrat în figura 2.1.

- Bundles – Pachete; sunt componente OSGi realizate de dezvoltatori
- Services – Serviciile conectează modulele într-un mod dinamic oferind un model "publish-find-bind" pentru obiectele Java
- Life-Cycle – API-ul pentru a instala, porni, opri, actualiza şi dezinstala module
- Modules – Defineşte cum un pachet - bundle poate importa şi exporta cod
- Security – Defineşte aspectele legate de securitate
- Execution Environment – Defineşte ce metode şi clase sunt disponibile pe o anumită platformă.

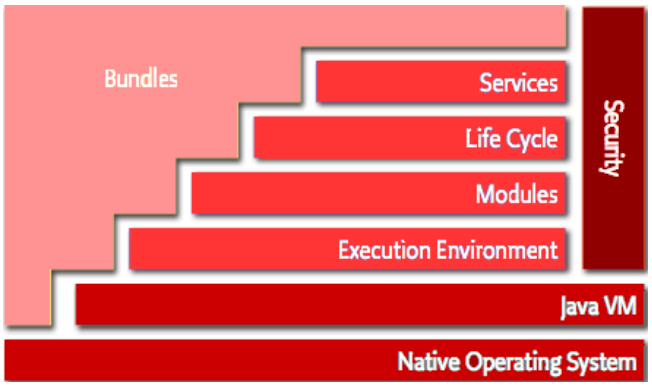

Figura 2.1 Arhitectura OSGi

2.2.2 Module dinamice

Conceptul fundamental care stă la baza arhitecturii OSGi este modularitatea. Modularitatea, explicată la modul simplist, se referă la păstrarea lucrurilor la nivel local, fără a le pune la comun. În Java, un modul poate fi văzut ca un fişier JAR obişnuit (POJO). Deşi în Java tot ce este într-un JAR este complet vizibil tuturor celorlalte JAR-uri, OSGi ascunde datele dintr-un JAR, mai puţin atunci când opţiunea de accesare este menţionată explicit. Un modul care doreşte să folosească un alt JAR trebuie să importe explicit părţile care îl interesează. In mod implicit nu există nici un fel de legătura între module.

OSGi nu este doar un sistem de module Java. Este un sistem de module Java dinamice. In OSGi modulele pot fi instalate, actualizate si dezinstalate fără a modifica întreaga aplicaţie. Acest lucru face posibilă actualizarea, de exemplu, a unui singur modul din cadrul unei aplicaţii fără a afecta funcţionalitatea altor părţi. De asemenea, aplicaţiile desktop pot descărca noi versiuni fără a fi necesară repornirea aplicatiei, astfel că utilizatorul nu va fi intrerupt [12].

Cu toate că ascunderea codului şi accesarea lui în comun menţionată explicit aduc multe beneficii (de exemplu permit folosirea mai multor versiuni ale aceleiaşi librării ca să fie folosite de o singură maşină virtuală), accesarea la comun a codului a fost introdusă pentru a suporta modelul de servicii OSGi. Modelul de servicii se referă la module (bundle-uri) care colaborează.

Modularitatea este unul dintre principalele avantaje ale tehnologiei OSGi, însă nu ar fi un motiv suficient pentru a alege această tehnologie întrucât există şi alte posibilităţi de a realiza modularizarea. In plus faţă de modularitate, tehnologia OSGi defineşte un software complet de gestiune a componentelor, cu alte cuvinte un ciclu de gestionare a "vieţii" modulelor (life-cycle). Se oferă o soluţie pentru a gestiona stările unui bundle pe durata vieţii acestuia. Ciclul de viaţă al unui modul OSGi este compus din mai multe stări, după cum se poate observa în figura 2.2.

Managementul ciclului de viaţă este destul de complex întrucât există mai multe stări şi chiar mai multe tranziţii, care se desfăşoară înainte şi înapoi. In practică nu e chiar atât de greu de realizat deoarece nu toate tranziţiile trebuie să fie executate manual, unele fiind automatizate. Ciclul de viaţă începe cu instalarea bundle-ului care schimbă starea lui în "installed". Din această stare există două posibilităţi: una este de a porni bundle-ul, cealaltă opţiune este de a-l dezinstala din nou. Tranzacţiile din starea "installed" în starea "active" se execută automat de către OSGi. După ce bundle-ul este pornit el poate fi oprit, ceea ce îi schimbă starea în "resolved". Din această stare, bundle-ul poate fi dezinstalat sau repornit [3]. Acest ciclu descris este doar un exemplu. Există mai multe alternative într-un ciclu de viaţă.

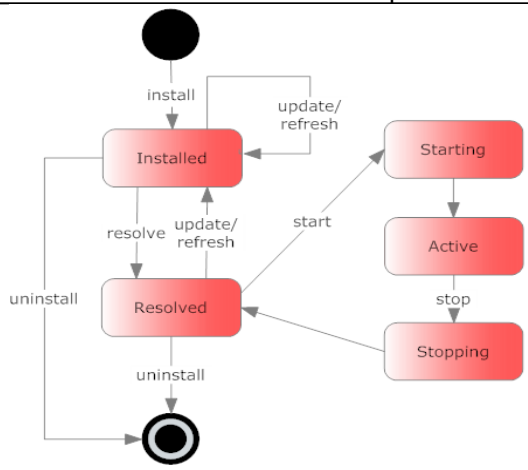

Figura 2.2 Ciclul de viaţă al unui bundle

2.2.3 Servicii

Motivul pentru care este nevoie de modelul de servicii e datorat faptului că Java ne arată cât este de dificilă scrierea unui model colaborativ doar prin accesarea în comun a claselor. Soluţia standard adoptată în Java este utilizarea fabricilor "Factories" care folosesc încărcarea dimamică a claselor. Incercarea de a infuenţa ce implementare să fie folosită, nu este o sarcină uşoară. Mecanismul Factory este o convenţie utilizată în sute de locuri în maşina virtual unde fiecare factory are propria interfaţă API şi propriul mecanism de configurare. Nu există o vedere centralizată a implementărilor.

Soluţia la aceste neajunsuri este serviciul OSGi registry. Un modul poate crea un obiect şi să-l înregistreze cu ajutorul serviciului OSGi registry sub una sau mai multe interfeţe. Alte module pot apela registry pentru a lista toate obiectele care sunt înregistrate sub o anumită interfaţă sau clasă. Chiar mai mult, un modul poate aştepta să apară un anumit serviciu, după care să primească un răspuns [13].

Un modul poate înregistra un serviciu, poate primi un serviciu şi poate urmări dacă un serviciu apare sau dispare. Mai multe module pot înregistra acelaşi tip de serviciu şi oricâte module pot primi acelaşi serviciu. Acest lucru este ilustrat în figura 2.3.

Serviciile sunt dinamice. Aceasta înseamnă că un modul poate să-şi retragă serviciile din registry în timp ce alte module continuă să folosească acest serviciu. Modulele care folosesesc un astfel de serviciu trebuie apoi să se asigure că nu mai folosesc serviciul si ca au fost eliminate toate referinţele către acesta.

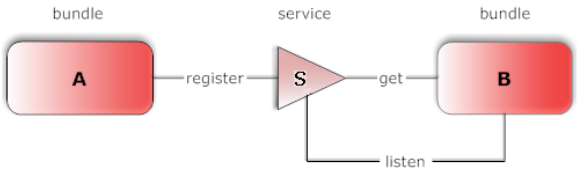

Figura 2.3 Bundle – Serviciu

După cum este ilustrat în figura 2.4, serviciul OSGi registry permite o formă de arhitectură orientată pe servicii (SOA). Cu toate acestea, spre deosebire de multe interpretări ale arhitecturilor orientate pe servicii, care se bazează pe servicii web pentru partea de comunicare, serviciile OSGi sunt publicate şi consumate în aceeaşi maşină virtuală Java, adică nu sunt distribuite. Astfel, tehnologia OSGi este uneori descrisă ca fiind „SOA într-o maşină virtuală Java" („SOA is a JVM").

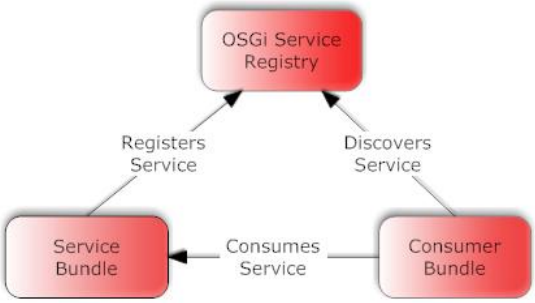

Figura 2.4 SOA într-o maşină virtuală Java

Punând la dispoziţie o arhitectură orientată pe servicii într-o maşină virtuală Java, OSGi permite modulelor să publice servicii şi să depindă de servicii publicate de alte module. Serviciile sunt cunoscute de către interfeţele publicate ale acestora, nu de către implementarea în sine. Acest lucru înseamnă că nivelul de cuplare între modulele care publică servicii şi cele care le consumă este redus.

2.2.4 Securitate

Nu toate componentele OSGi sunt create egale, ceea ce pune imediat problema securităţii. Componentele trebuie să fie protejate unele de altele. Securitatea începe cu o autentificare adecvată. Autentificarea bundle-urilor poate fi bazată pe semnarea unor certificate digitale sau pe localizarea bundle-urilor. In funcţie de tipul de autentificare, bundle-ul primeşte un set de autorizaţii numite permisiuni. De exemplu, o astfel de permisiune este aceea de a exporta şi importa pachete Java.

Securitatea OSGi se bazează pe securitatea Java, dar furnizează şi câteva extensii. După cum am arătat anterior, arhitectura OSGi are un model stratificat. Unul din aceste straturi este cel de securitate. Nivelul de securitate rulează în paralel cu nivelele de funcţionalitate cum sunt nivelul de servicii sau nivelul de gestionare a ciclului de viaţă.

Modelul de securitate OSGi este extrem de bine conturat. De asemenea este opţional, iar implementările care nu necesită un grad ridicat de siguranţă pot ignora acest model. Bundle-urile pot conţine propriile lor permisiuni care se intersectează cu permisiunile pe care le primesc pe baza identităţii lor. Acest model la prima vedere paradoxal, permite fiecărui utilizator controlul asupra securităţii domeniului, fără a compromite securitatea per ansamblu.

OSGi specific patru tipuri de permisiuni speciale: AdminPermission, BundlePermission, PackagePermission şi ServicePermission. Prima permisiune oferă posibilitatea bundle-urilor de a executa funcţii administrative privilegiate sau de a obţine informaţii "sensibile" despre alte bundle-uri. O acţiune de acest tip ar fi *execute* care permite oprirea şi pornirea altor bundle-uri, acţiune care în mod normal îi revine administratorului. Comanda *context* permite vizualizarea fişierului BundleContext ce conţine informaţii cu privire la locaţia unui bundle şi permisiunile asignate lui. Folosirea iresponsabilă a acestor informaţii poate compromite funcţionarea bundle-ului.

Si celelalte permisiuni sunt aproape la fel de importante ca şi AdminPermission. PackagePermission, de exemplu, are doar două acţiuni: *export* şi *import*. Aceste permisiuni sunt importante când un bundle vrea să expună un serviciu pentru a putea fi folosit de către alt bundle. Pentru a face acest lucru are nevoie de PackagePermission cu acţiunea *export*.

BundlePermission este asemănător cu PackagePermission, deşi acţiunile sunt diferite. BundlePermission permite acţiuni cum ar fi *require* (care este similară cu acţiunea *import* şi permite unui bundle să aibă acces la pachetul unui alt bundle) sau *provide* (similară cu *export*). In general se preferă PackagePermission deoarece are o abordare mai simplă, dar BundlePermission oferă nişte funcţionalităţi speciale, printre care aceea de a expune doar un fragment din serviciul unui bundle.

A patra permisiune este ServicePermission care permite realizarea unor acţiuni de înregistrare a unui bundle. Această permisiune nu este necesară pentru proiectanţii unei aplicaţii, numai în cazul în care se doreşte implementarea unui întreg cadru de lucru OSGi.

Inainte de a da permisiuni unui bundle, acesta trebuie să îndeplinească o condiţie. Condiţiile specificate de OSGi sunt BundleLocationCondition care verifică originea bundle-ului şi BundleSignerCondition care analizează semnătura bundle-ului pentru a o valida.

După cum îi spune şi numele, BundleLocationCondition verifică dacă bundle-ul provine dintr-o locaţie specificată. Aceasta înseamnă că bundle-urile care provin din locaţii nesigure pot fi excluse. Specificaţiile pentru determinarea locaţiei pot conţine caractere de genul "*" (toate fişierele dintr-un director) sau "-" (toate fişierele dintr-un director şi toate subdirectoarele lui).

Cu ajutorul condiţiei BundleSignerCondition se poate verifca dacă bundle-ul are semnătura specificată. Este o condiţie inflexibilă, întrucât semnătura unui bundle nu poate fi schimbată, cel puţin până când acesta nu este actualizat sau dezinstalat şi instalat din nou. Dar după fiecare instalare permisiunile sunt redistribuite. Si această condiţie poate folosi caracterele "*" şi "-" menţionate mai sus. In timpul validării, textul condiţiei este comparat cu semnătura bundle-ului. Dacă cele două se potrivesc, condiţia este îndeplinită.

Pe lângă cele două tipuri de condiţii, programatorul poate crea condiţii personalizate. Pentru a crea o astfel de condiţie, o clasă trebuie să conţină funcţia *getCondition* cu parametrii *Bundle* şi *ConditionInfo*. Tipul returnat va fi o implementare a interfeţei *Condition*. Următorul fragment de cod defineşte implementarea condiţiei BundleLocationCondition:

```
public static Condition getCondition(final Bundle bundle,ConditionInfo info) {
if (location.matches())
{return Condition.TRUE;}
 else {return Condition.FALSE;}}
```

Dacă instanţierea eşuează, este creat obiectul predefinit Condition.FALSE şi în fişierul de log-uri se înregistrează o eroare.

2.3 Cadre de lucru

Cadrul de lucru este mediul în care aplicaţiile pot rula, beneficiind de toate avantajele oferite de tehnologia OSGi. In prezent exista mai multe implementări OSGi de referinţă. Câteva dintre acestea sunt:

- Equinox
- Knopflerfish
- Apache Felix
- Concierge
- Spring Dynamic Modules
- Newton
- Oscar

2.3.1 Equinox

Equinox este în prezent cel mai răspândit cadru de lucru OSGi datorită utilizării sale în cadrul aplicaţiei Eclipse, începând cu versiunea 3.0. Poate fi regăsit de asemenea în cadrul multor aplicaţii şi produse cum ar fi IBM WebSphere Application Server şi Lotus Notes [5].

Equinox este practic un sistem plug-in care permite dezvoltatorilor să implementeze aplicaţii ca o mulţime de bundle-uri care colaborează prin intermediul serviciilor.

La instalarea aplicaţiei Eclipse, mediul OSGi Equinox este şi el instalat automat. Lansarea acestui mediu se face prin comanda Run -> Open Run Dialog... de unde se alege opţiunea OSGi Framework. După lansarea comenzii, în consola din Eclipse va apărea prompt-ul: osgi>. Equinox poate rula şi independent de mediul Eclipse.

Câteva din comenzile folosite în cadrul de lucru Equinox sunt:

- **ss** – afişează lista modulelor OSGi instalate
- **exit** – părăseşte şi închide cadrul de lucru
- **install file:modulOSGi.jar** – instalează modulul OSGi; la instalare i se asignează automat un număr natural ca identificator – id
- **start id** – lansează modulul OSGi cu identificatorul id
- **stop id** – opreşte modulul OSGi cu identificatorul id
- **uninstall id** – dezinstalează modulul OSGi cu identificatorul id

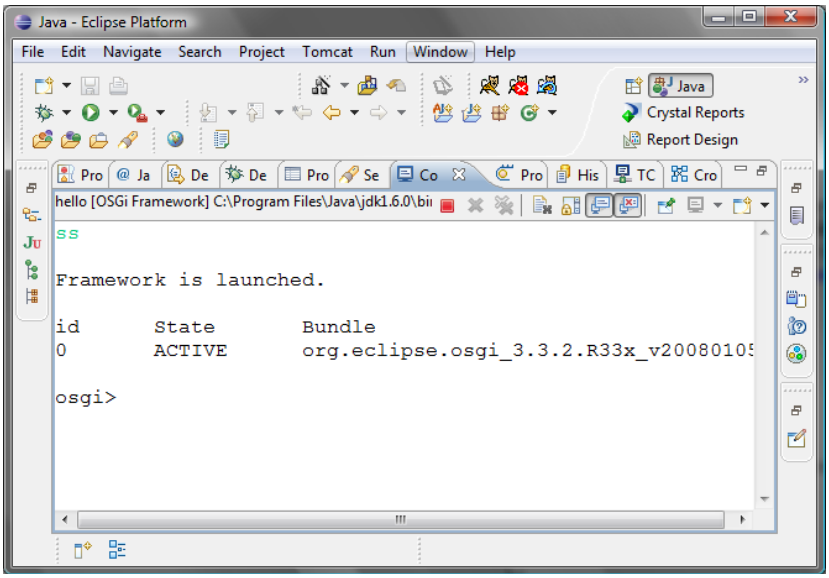

Figura 2.5 Cadrul de lucru Equinox în consola Eclipse

2.3.2 Knopflerfish

Knopflerfish este un cadru de lucru dezvoltat de compania Makewave AB din Suedia. Makewave pune la dispoziţie o variantă comercială a acestui cadru de lucru, Knopflerfish Pro. Acest cadru de lucru permite mai multor aplicaţii să ruleze într-o singură maşină virtuală concomitent şi sigur. Aplicaţiile pot împărţi aceleaşi resurse, funcţionalităţi şi thread-uri [10].

Knopflerfish poate fi instalat în Eclipse ca şi plug-in sau ca şi aplicaţie de sine stătătoare. Aplicaţia se caracterizează printr-o interfaţă grafică ce ne permite pornirea şi oprirea bundle-urilor, vizualizarea mesajelor de eroare cât şi alte operaţii similare. Listarea pachetelor instalate se poate face şi din consolă ca şi în cazul cadrului de lucru precedent. Knopflerfish dispune de o consolă pentru comenzile date în mod text.

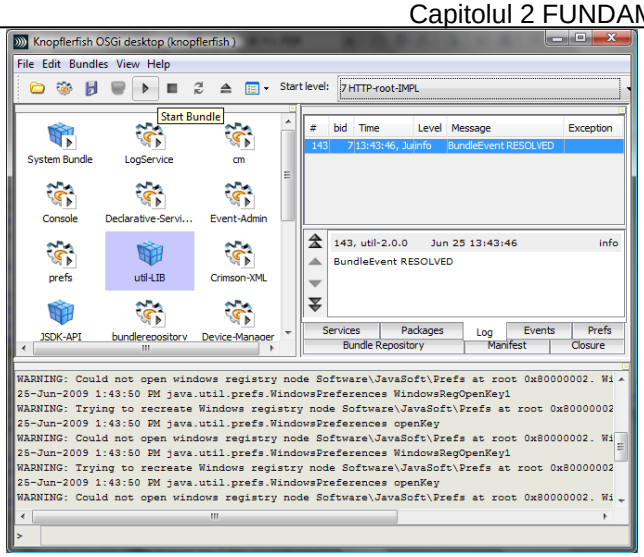

Figura 2.6 Aplicaţia desktop Knopflerfish

2.3.3 Apache Felix

Apache Felix este o implementare open source a platformei OSGi Release 4. Specificaţiile OSGi vizate iniţial încorporau principii de modularitate, orientarea pe componente şi/sau pe servicii. Aspectele principiilor menţionate mai sus au fost combinate pentru a defini un cadru dinamic potrivit pentru lucrul la distanţă [6].

Proiectul Apache Felix este organizat în subproiecte, fiecare având ca ţintă o specificaţie OSGi, de exemplu: jPOJO – componentă orientată pe servicii, HTTP Service – implementare a specificaţiei OSGi HTTP Service, Log – implementare bazată pe memorie a serviciului OSGi Log etc.

Câteva din comenzile folosite în cadrul de lucru Apache Felix sunt:

ps – afişează lista modulelor OSGi instalate
refresh – actualizează modulele
start id – lansează modulul OSGi cu identificatorul id
stop id – opreşte modulul OSGi cu identificatorul id
uninstall id – dezinstalează modulul OSGi cu identificatorul id

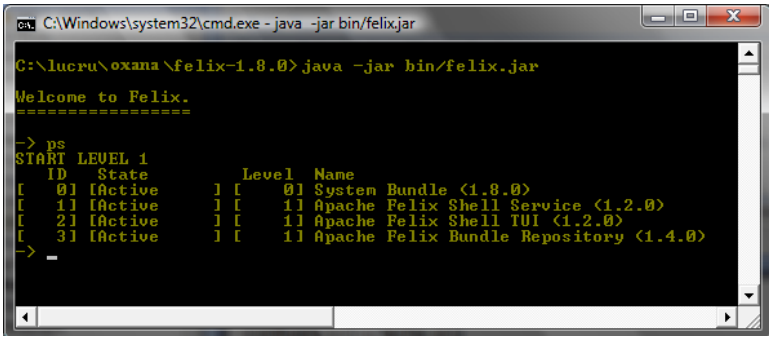

Figura 2.7 Consola Felix OSGi

2.3.4 Comparaţie între cadrele de lucru

Cadrele de lucru enumerate anterior nu sunt direct comparabile. De exemplu, Sping Dynamic Modules este construit pe mediul OSGi Equinox, iar Newton poate folosi Equinox, Felix sau Knopflerfish (care sunt cele 3 cadre de baza). Oscar este cel mai nou aparut şi mai puţin dezvoltat cadru de lucru, iar Concierge poate implementa doar versiunea 3 a platformei

OSGi (versiunea curentă fiind 4.2) [4]. Cadrul de lucru Apache Felix se confruntă cu câteva probleme la nivel de servicii, în timp ce Equinox şi Knopflerfish par a fi cele mai complete din punct de vedere al implementării platformei OSGi 4.2.

In ceea ce priveşte mediul de dezvoltare a cadrelor de lucru şi uneltele pe care le pune la dispoziţie, care sunt cruciale din punct de vedere al vitezei de dezvoltare, Eclipse oferă cea mai bună integrare. Pentru cei care vor să evite folosirea mediului de dezvoltare Eclipse, se recomandă alegerea cadrului de lucru independent Apache Felix. Dezavantajul în ceea ce priveşte cadrul Felix este acela că nu suportă fragmentarea (posibilitatea de a adăuga cod într-un bundle care rulează).

Argumentele în favoarea folosirii cadrului de lucru Equinox susţin faptul că ar fi o bună implementare în ceea ce priveşte platforma OSGi 4.2 şi posibilitatea unei configurări de nivel înalt. In comparaţie cu Knopflerfish prezintă şi unele dezavantaje cum ar fi o slabă documentare, implementarea nu este pur OSGi, Eclipse aducându-şi contribuţia în multe cazuri. Knopflerfish beneficiază şi de o bună interfaţă de gestionare a bundle-urilor.

In figura 2.8 se prezintă o comparaţie a cadrelor de lucru Equinox, Knopflerfish şi Apache Felix din punct de vedere al specificaţiilor şi serviciilor adoptate la aderarea la versiunea 4 a platformei OSGi.

R4 Core	Framework Spec							Core Services				
Container	Version	Framework Spec	OSGi Certified	Security Layer	Module Layer	Lifecycle Layer	Service Layer	Package Admin	Conditional Permission	Permission Admin	Start Level Service	URL Handler
Eclipse Equinox	3.2	R4	Yes	OK	OK	OK	OK	OK	OK	OK	OK	OK
knopflerfish	2.0	R4	No	OK	OK	OK	OK	OK	OK	OK	OK	OK
Apache Felix	0.9	R4	No	Partial	OK-	OK-	OK-	OK-	-	-	OK	OK

Figura 2.8 Comparaţie între cadrele de lucru OSGi de bază

2.3.5 Container OSGi vs. Container Web

Comportamentul unui container (cadru de lucru) OSGi este similar cu cel al unui container Web, cu toate că în primul caz componentele pot fi pornite, oprite şi actualizate într-un mod dinamic. Container-ul OSGi poate, de asemenea, să ruleze orice fel de aplicaţie Java şi permite o comunicare directă cu aceasta [13].

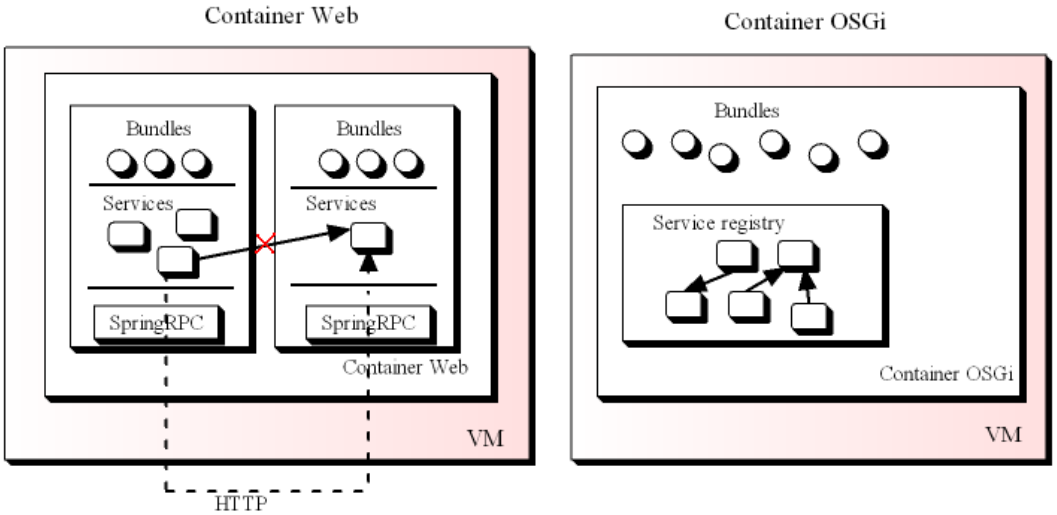

Figura 2.9 Container Web vs. Container OSGi

Container-ul Web (ex:Sun Java System Web Server, Tomcat etc.):

- Nu poate partaja JAR-urile
- Nu poate comunica direct

Container-ul OSGi:
- Partajează JAR-urile
- Partajează serviciile
- Inlocuire dinamică a bundle-urilor

2.4 Avantaje ale tehnologiei OSGi

Tehnologia OSGi prezintă un set de specificaţii care definesc şi comunică modularitatea unei aplicaţii Java într-un mod mult mai dinamic. In mod tradiţional o aplicaţie Java este modularizată ca şi pachet JAR. Lucrul cu fişiere JAR prezintă însă anumite limitări:

- Nu există un cadru de lucru pentru actualizarea pachetelor JAR în mod dinamic în timpul rulării atunci când are loc o modificare de cod.
- Pachetele JAR nu pot fi etichetate cu o anumită versiune şi nu se poate urmări o istorie a pachetelor nou create sau modificate.
- Pachetele JAR sunt rezolvate prin intermediul unei variabile de mediu class path care nu oferă un cadru robust pentru gestionarea dependinţelor dintre pachete.

Pentru a rezolva problemele menţionate, se foloseşte tehnologia OSGi deoarece aceasta redefineşte sistemul modular introdus de Java. Un sistem OSGi prezintă următoarele avantaje faţă de sistemul tradiţional al modulelor JAR:

- OSGi furnizează un mediu robust integrat în care modulele pot fi publicate ca servicii şi exportate pentru a fi folosite de către alte module.
- OSGi furnizează un mecanism de etichetare a versiunii fiecărui modul pentru fiecare nouă implementare.
- Cu OSGi se pot actualiza în mod dinamic pachetele în timpul rulării de câte ori are loc o schimbare de cod.

Alte beneficii pe care le ofera tehnologia OSGi sunt:

Complexitate redusă - a dezvolta folosind tehnologia OSGi înseamnă a dezvolta componentele OSGi. Componentele reprezintă de fapt module. Acestea îşi ascund structura internă şi comunică prin intermediul unor servicii bine definite. Ascunderea acestor aspecte oferă mai multă libertate de a schimba anumite lucruri mai târziu. Acest lucru nu numai că reduce numărul de bug-uri, de asemenea face ca modulele să fie mai uşor de dezvoltat deoarece fiecare funcţionalitate este implementată de către un modul prin intermediul unei interfeţe bine definite.

Reutilizare – componentele OSGi pot fi utilizate şi în alte aplicaţii. Un număr tot mai mare de proiecte open source furnizează JAR-urile deja pregătite pentru OSGi.

Transparenţă – bundle-urile si serviciile sunt componentele de bază ale unei aplicaţii dezvoltate cu tehnologia OSGi. Este permis accesul la structura interna a bundle-urilor. Modul de comunicare cu alte bundle-uri este de asemenea transparent.

Versionare – are o mare importanţă în cadrul tehnologiei OSGi. Modulele şi librăriile sunt într-o continuă schimbare ori de câte ori apar modificări în rândul cerinţelor sau sunt adăugate noi proprietăţi. De aceea este necesară cunoaşterea versiunii modulului care se doreşte a fi folosită.

Actualizare dimanică – modelul componentei OSGi este un model dinamic. Modulele pot fi instalate, pornite, oprite, actualizate si dezinstalate fără a afecta întregul sistem.

Simplitate – interfaţa OSGi este una foarte simplă. Există un singur pachet şi mai puţin de 30 de clase. API-ul este suficient pentru a scrie module, a le instala, porni, opri, actualiza sau a le dezinstala şi include toţi listener-ii şi clasele pentru securitate.

Dimensiune redusă - ultima versiune a framework-ului OSGI poate fi implementată într-un JAR de aproximativ 300 KB, aproape nesemnificativ ca mărime având în vedere funcţionalităţile de care poate beneficia o aplicaţie care foloseşte OSGI. Totodata OSGI rulează pe o gama largă de dispozitive: de la cele mai mici, până la mainframe-uri.

Rapiditate - una dintre responsabilităţile de bază ale framework-ului OSGI este încărcarea claselor din resurse (bundle-uri). In Java, JAR-urile sunt complet vizibile şi plasate într-o listă liniară. A căuta o clasă înseamnă a căuta prin această listă. OSGI preîncarcă bundle-urile şi ştie pentru fiecare bundle exact ce resursă furnizează clasa respectivă. Această „lipsă" de a mai căuta este foarte importantă la startup, reprezentând un factor de viteză mai mare.

Optimizare - este un lucru util în software şi tehnologia OSGi are multe mecanisme care permit realizarea unor lucruri doar atunci când sunt cu adevărat necesare. De exemplu, unele resurse pot fi încărcate mai devreme, dar pot fi de asemenea configurate să pornească numai atunci când o altă resursă are nevoie de ele. Serviciile pot fi înregistrate, dar sunt create numai atunci când este nevoie de ele. Specificaţiile au fost optimizate de mai multe ori pentru a permite astfel de comportamente care pot salva costuri enorme la runtime.

Securitate - Java are la bază un foarte puternic model de securitate , dar practic este dificil de configurat. In general, OSGi oferă probabil unul dintre cele mai sigure medii de aplicaţii.

2.5 Alternative la tehnologia OSGi

La baza acesteia, tehnologia OSGi este foarte simplă şi, ca şi pentru toate ideile bune şi simple, oamenii au inventat diverse alternative. Nici una din ele nu a atins însă nivelul de maturitate şi răspândire de care se bucură în prezent OSGi.

2.5.1 Maven şi Ivy

Maven şi Ivy sunt ambele instrumente populare care au anumite caracteristici ale unui sistem de module, dar sunt mai degrabă tool-uri folosite la build-time decât cadre de lucru la runtime. Astfel, ele nu concurează direct cu OSGi. De fapt, sunt complementare şi mulţi dezvoltatori folosesc Maven şi Ivy pentru a construi sisteme care au la bază tehnologia OSGi.

Maven este un instrument de sine stătător, în timp ce Ivy este o componentă ce poate fi integrată într-o construcţie de tip Ant. Ambele instrumente încearcă să facă JAR-urile mai uşor de gestionat prin adăugarea de caracteristici de modularitate, cum ar fi versionarea JAR-urilor folosind metadate în fişiere XML. Din păcate, formatul metadatelor nu este compatibil cu formatul utilizat de OSGi dar se fac eforturi în acest sens, pentru o mai bună integrare a instrumentului Maven în tehnologia OSGi.

2.5.2 Sistemul de plug-in-uri Eclipse

Cum s-a mai precizat anterior în cadrul acestei lucrări, Eclipse IDE este bazat pe o implementare OSGi. Acest lucru nu se întâmpla însă şi înainte de versiunea 3.0 când Eclipse folosea propriul său sistem personalizat de module.

In terminologia Eclipse, un modul este un „plug-in". De fapt, dezvoltatorii Eclipse folosesc adesea termenul „plug-in" ca şi o alternativă la noţiunea de „bundle" din OSGi. In

vechiul sistem Eclipse, un plug-in era un director care conţinea la nivel superior un fişier numit plugin.xml. Conţinutul acestui fişier consta în metadate, în mare parte asemănătoare cu metadatele din fişierul manifest care apare în OSGi.

Cea mai mare deficienţă a sistemului de plug-in-uri Eclipse era incapacitatea de a instala, actualiza şi dezinstala în mod dinamic plug-in-urile: de câte ori se schimba structura unui plug-in, era nevoie de o schimbare generală. In anul 2004 dezvoltatorii Eclipse au început să lucreze la proiectul intitulat Equinox care îşi propunea elaborarea unui sistem dinamic de plug-in-uri. Atunci a fost ales şi adaptat sistemul de module OSGi, Equinox devenind imaginea acestuia.

2.5.3 JSR 277

Java Specification Request (JSR) numărul 277 este numit şi Java Module System. Se aşteaptă ca acesta să încerce să rezolve o mulţime similară de probleme ca şi OSGi. Cu toate acestea, JSR 277 le abordează într-un mod diferit.

Punctul cel mai important de reţinut este acela că, în acest moment, JSR 277 are o specificaţie incompletă, fără a fi pus încă în aplicare. Este programat să fie inclus în Java 7 dar nu se ştie încă dacă acest lucru se va întâmpla şi când.

Ca şi vechiul sistem de plug-in-uri, Eclipse, JSR 277 nu suportă dependenţe la nivel de pachet. De asemenea, nu este dinamic, fiind nevoie să se repornească sistemul la fiecare instalare, actualizare sau dezinstalare a unui modul.

JSR 277 vine câţiva ani mai târziu faţă de OSGi, prezintă unele caracteristici de proiectare discutabile şi este în mod substanţial mai puţin ambiţios, în ciuda posibilităţii de a schimba principiile de bază ale tehnologiei Java.

2.6 Apache Ant

Apache Ant este o unealtă open-source de dezvoltare a software-ului, scrisă în întregime în Java. Este o unealtă similară utilitarului **make** utilizat la dezvoltarea aplicaţiilor scrise în limbajul C [2].

Spre deosebire de make care învaţă să execute noi acţiuni prin intermediul comenzilor shell-ului (command.com, bash), Ant se poate extinde scriind noi clase Java. Fişierele de configurare se bazează pe XML. Fiecare operaţie este executată de un obiect ce implementează o interfaţă Task specifică.

Pentru a putea executa comenzi shell, independent de sistemul de operare instalat pe calculatorul unde rulează programul, Ant pune la dispoziţie operaţia <exec>.

Distribuţia Ant este utilizată cu success pe mai multe platforme, de la Linux, Solaris, Windows până la Novell Netware6 sau MacOS X. Versiunea compilată conţine ultima versiune a parserului XML Apache Xerces2.

Versiunea curentă necesită prezenţa unei variante de JDK (>1.1) instalată pe calculatorul gazdă. Unele operaţii nu funcţionează decât cu Java 1.2 sau o versiune ulterioară a acesteia.

Pentru a folosi Ant în scopul compilării automatizate a distribuţiei unui produs este nevoie de un fişier de configurare build.xml, fişier ce trebuie să respecte în primul rând sintaxa limbajului XML şi apoi să respecte modul de formare pentru a putea fi înţeles de către Ant.

2.6.1 Sintaxa unui fişier build

Fişierele Ant de build conţin un proiect şi cel puţin un target (default). Target-urile conţin elemente de tip task. Fiecare element de tip task dintr-un fişier de build poate avea un

atribut *id* prin intemediul căruia poate fi referit ulterior. Valoarea acestui identificator trebuie să fie unică.

Proiecte

Sintaxa:

```
<project name="MyProject" default="dist" basedir=".">
  <description>
     Scurtă descriere a proiectului
  </description>
</project>
```

Un proiect are următoarele trei atribute:

- name – numele proiectului (nu este obligatoriu)
- default – numele target-ului implicit utilizat atunci când la rulare nu e furnizat nici un target
- basedir – directorul de bază relativ la care sunt calculate toate căile. Acest atribut poate fi suprascris setând proprietatea "basedir" mai înainte. Dacă proprietatea a fost setată înainte, acest atribut poata lipsi. Dacă nici atributul nici proprietatea nu au fost setate se va considera directorul părinte al fişierului.

Opţional se poate furniza o descriere a proiectului în interiorul tag-ului <description>.

Fiecare proiect defineşte unul sau mai multe target-uri. Un target este format dintr-o mulţime de task-uri care pot fi executate. La rularea programului Ant putem selecta target-ul/target-urile pe care dorim să le executăm. Dacă nu este specificat nici un target, atunci se utilizează target-ul implicit al proiectului.

Target-uri

Sintaxa:

```
<target name="targetName" depends="lista target-uri" if="nume propritate 1"
unless="nume propritate 2" description="Descriere target"/>
```

Un target are următoarele atribute:

- name – numele target-ului (este obligatoriu)
- depends – lista de target-uri de care depinde target-ul curent, separate prin virgulă (nu este obligatoriu)
- if – numele proprietăţii care trebuie să fie setată pentru ca target-ul să se execute (nu este obligatoriu)
- unless – numele proprietăţii care nu trebuie să fie setată pentru ca target-ul să se execute (nu este obligatoriu)
- description – o scurtă descriere a activităţii realizate de target (nu este obligatoriu)

Un target poate să depindă de alte target-uri. De exemplu, putem avea un target pentru realizarea compilării şi unul pentru crearea distribuţiei proiectului. Insă nu putem realize distribuţia proiectului dacă nu a fost compilat în prealabil. Astfel, spunem că target-ul de distribuţie depinde de target-ul de compilare. Ant poate să rezolve astfel de dependenţe. Pentru a specifica că un target depinde de unul sau mai multe target-uri utilizăm atributul *depends* pentru a arăta ordinea în care trebuie să se execute target-urile.

Task-uri

Un task este un fragment de cod (o acţiune) care poate fi executat. Un task poate avea mai multe atribute (sau arguments). Valoarea unui atribut poate face referire la o proprietate definită în prealabil.

Task-urile au următoarea stuctură:

```
<name attribute1="value1" attribute2="value2" ... />
```

unde *name* este numele task-ului, *attributeN* este numele atributului N, iar *valueN* este valoarea acestui atribut.

Proprietăţi

Intr-un proiect se pot defini o mulţime de proprietăţi (variabile). Acestea se pot defini fie în interiorul fişierului de build utilizând tag-ul *property*, fie în afara Ant-ului. Sintaxa definirii unei variabile este următoarea:

<property name="numePropietate" value="valoare" />
<property name="numePropietate" location="cale" />

O proprietate are un nume şi o valoare. Numele este case-sensitive. Valorile acestor propietăţi se pot utiliza ca atribute în cadrul task-urilor. Referirea valorii unei variabile se fac printr-o construcţie de forma ${numeVariabilă}.

In afara variabilelor definite de către utilizator, există o serie de variabile predefinite, majoritatea dintre ele având nume identice cu cele utilizate în System.getProperties().
Exemplu:

- basedir - calea absolută către directorul rădăcină al proiectului
- ant.file - calea absolută a fişierului ‘build.xml’
- ant.version - versiunea programului Ant
- ant.project.name - numele proiectului aflat în desfăşurare
- ant.java.version - versiunea maşinii virtuale Java – JVM - ce a fost detectată.

Capitolul 3 CONCEPTE GENERALE ALE APLICATIEI HOMERUN

3.1 Obiecte şi componente

Aproape toate operaţiile din HomeRun constă în interacţiunea a ceea ce se cheamă obiecte, de aceea înţelegerea conceptului de „obiect" şi a modului în care obiectele funcţionează este esenţială. Unele pachete furnizează obiecte gata create pentru a fi utilizate, dar în general acestea trebuie create de către utilizator.

Un obiect este doar o resursă căreia i s-a dat un nume. Fiecare obiect este unic însă toate sunt instanţe ale unei singure clase, cunoscută sub numele de tipul obiectului. Tipul obiectului este important atunci când se creează un nou obiect care este adăugat sistemului. Aplicaţia dispune de o fereastră numită „Object Editor" care afişează toate obiectele cunoscute de sistem. Tot în această fereastră este posibilă crearea de noi obiecte în funcţie de tipul dorit.

Fiecărui obiect îi corespund o scurtă descriere şi câteva proprietăţi care sunt moştenite de toate obiectele dintr-un anumit tip.

Tipurile sunt importante pentru crearea obiectelor, însă în majoritatea situaţiilor prezintă interes categoria in care se află obiectul. Categoria este modalitatea prin care se grupează toate obiectele în interfaţa programului.

Tipurile sunt şi ele grupate în domenii. In HomeRun, „Domains" este cel mai înalt nivel de grupare a tipurilor, categoriilor şi obiectelor şi ajută la o mai bună organizare a sistemului. Toate obiectele sunt relative la domeniile în care se încadrează.

Obiectele pot avea trei tipuri de componente, după cum urmează:

Proprietăţi

Acestea sunt în principal descrieri ale atributelor care pot fi generice (true of the type) sau specifice (true of the individual). Unele proprietăţi sunt obligatorii, altele opţionale. Unele au valori implicite (de exemplu pictogramele utilizate pentru reprezentarea obiectelor primesc valori implicite de la tipul părintelui).

Proprietăţile sunt folosite pentru atribute ale obiectelor care nu se schimbă, adică sunt statice. Pentru atributele care pot să îşi schimbe valoarea, HomeRun foloseşte o altă componentă, numită *model,* care stochează valorile acestor atribute.

Controller-e

Interesul major într-o clasă largă de obiecte (precum este cea a luminilor) este acela de a le controla. O instalaţie de iluminat, spre exemplu, are un întrerupător folosit pentru aprindere/stingere. Fiecare din aceste operaţiuni se numeşte *punct* astfel că întrerupătorul are două puncte de control. In funcţie de tipul lui, un obiect poate avea zero, unul sau mai multe controller-e. Homerun foloseşte o reprezentare şi o terminologie standard pentru controller-e, după cum urmează: un controller este o mulţime de operaţii interconectate pe care le putem efectua asupra unui obiect.

Emiţători

Spre deosebire de obiectele cărora li se atribuie controller-e, alte obiecte se comportă ca „observatori". Un exemplu de acest tip ar putea fi detectorul de fum din locuinţă. Deşi aceste obiecte nu au controller-e, au în schimb proprietatea de a observa şi raporta anumite situaţii. HomeRun numeşte aceste activităţi produse independent evenimente, iar facilitatea de a produce un set de evenimente, emiţător. Spre deosebire de controller-e, emiţătorii nu pot fi opţionali: dacă aparţin unui obiect, se vor adăuga automat la crearea acestuia.

3.2 Acţiuni şi legături

HomeRun foloseşte acţiuni pentru a transforma modul de control într-un proces automatizat. Astfel obiectele pot fi manipulate fără a necesita prezenţa unui individ. HomeRun furnizează o interfaţă cu utilizatorul intuitivă pentru lucrul cu acţiuni, numită „Action Builder“. O acţiune în forma ei cea mai simplă constă într-o listă de sarcini (minim una) asociate unui obiect. Acţiunea se execută în momentul în care condiţia specificată este îndeplinită.

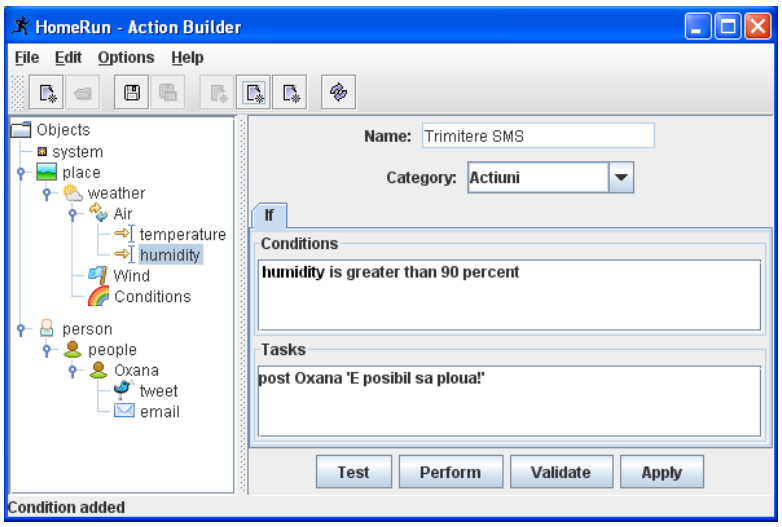

Figura 3.1 Fereastra Action Builder

Adăugarea sarcinilor este numai o chestiune de navigare în explorer panel-ul din stânga obiectului de interes, după care se face click pe modelul de control care se doreşte a fi adăugat. Odată ce acţiunile sunt definite, ele pot fi invocate manual în Control Panel, dar de obicei sunt invocate automat în diferite moduri. Aceste moduri sunt cunoscute ca „trigger-e“ în sensul că apariţia lor declanşează acţiunea. In HomeRun se preferă termenul de „legături“, acest lucru însemnând că acţiunea e legată de o circumstanţă a cărei apariţii o declanşează. Desigur, aceeaşi acţiune poate fi legată de diferite şi multiple circumstanţe.

Unul dintre tipurile comune de legături este acela dintre o acţiune şi o dată/oră specificată, care nu se repetă (vreau ca acţiunea să aibă loc în data de 28 iunie, ora 17:30). Acestea sunt cunoscute ca şi legături de calendar, de vreme ce ar fi scrise firesc în calendarul personal. HomeRun furnizează un program simplu pentru crearea legăturilor de calendar (din Consolă: Automate -> Calendar).Se face click pe dată pentru a afişa orice legături curente, apoi se apasă butonul ’New Entry’ pentru a adăuga o nouă legătură (legăturile sunt salvate automat pe măsura ce sunt adăugate). Se pot adăuga legături pe viitor şi acţiunea e executată doar o dată. Oricum, introducerea rămâne vizibilă după ce acţiunea a fost executată, aşa încât se pot revedea introducerile precedente.

Un alt tip de legătură foarte des întâlnit este acela dintre o acţiune şi o anumită unitate de timp, dar care se repetă la un interval standard. Cel mai des utilizat interval de acest gen este săptămâna, fapt pentru care HomeRun furnizează o unealtă pentru a defini acţiuni repetabile săptămânal, care se numesc legături de program (din Consola: Automate -> Schedule). Spre deosebire de Calendar, se pot crea numeroase programe diferite corespunzătoare aceluiaşi interval de timp. De aceea se asignează un nume fiecărui program. Aceasta este o caracteristică foarte puternică, de vreme ce se poate schiţa un program pretabil fiecărei ocazii (de exemplu un program de vacanţă) care trebuie doar activat, mai degrabă decât reprogramarea întregului sistem de câte ori apar schimbări ale rutinei. Un singur program este activ la un moment dat.

Un ultim tip de legătură este legătura eveniment. Aceasta determină execuţia unei acţiuni, fără a depinde de vreun moment specific. Trebuie doar să aibă loc evenimentul emiţător. De vreme ce nu se ştie când sau dacă un anumit eveniment se va întâmpla, legăturile eveniment au o natură condiţională, comparativ cu legăturile de calendar şi program care se execută garantat la momentul stabilit. Spre deosebire însă de legăturile de program, pot să existe mai multe planuri active simultan. HomeRun pune la dispoziţie un planificator (din Consolă: Automate -> Plan) pentru crearea legăturilor de acest tip.

3.3 Modele şi scene

Pentru atributele care pot să îşi schimbe valoarea, HomeRun foloseşte o componentă numită *model*, care stochează valorile acestor atribute. Obiectele pot avea orice număr de modele asociate lor. Majoritatea software-urilor de automatizare tratează atributele variabile într-un mod destul de limitat, de exemplu, ca având un dispozitiv „status” cu valori fixe (on şi off). Problema care apare este că aceste software-uri nu reuşesc să surprindă mai multe modalităţi complexe pentru a reprezenta starea internă, precum şi faptul că de multe ori trebuie urmărite mai multe variabile, nu doar un singur „status”. Acest deficit se încearcă a fi rezolvat prin adăugarea de suport pentru limbaje de scripting, dar acest lucru ridică în mod semnificativ nivelul de complexitate şi calificare cerut pentru utilizarea sistemului.

HomeRun vine cu o alternativă simplă, flexibilă şi foarte puternică în privinţa modelelor. Se pot defini propriile modele şi asocia cât mai multe dintre ele cu un obiect. Modelele pot fi vizualizate prin intermediul interfeţei cu utilizatorul sau pot fi utilizate pentru a influenţa comportamentul acţiunilor.

Scene

Scenele sunt construcţii in HomeRun create pentru vizualizarea modelelor (din Consolă: View -> Scene). Se poate selecta orice număr de modele care vor apărea într-un cadru cu o structură standard. Câteva elemente de control din fereastra Scene Viewer: săgeata circulară care înseamnă „refresh” permite actualizarea scenei cu cele mai recente valori ale modelului. Scenele reprezintă o evaluare a modelului la un moment dat, dar evaluarea se poate actualiza cu „refresh”. Butonul „capture” permite înregistrarea valorilor unei scene pentru o revedere ulterioară. De asemeni, există posibilitatea de a defini o acţiune care să înregistreze automat valorile scenei. Acţiunea se adaugă la un program.

Condiţii de acţiune

Alt mod în care modelele pot fi folosite este de a adăuga logică condiţională acţiunilor întreprinse. Astfel, putem face ca performanţa acţiunii să depindă de „status-ul” modelelor, indiferent de momentul în care acţiunea este efectuată. In secţiunea „Conditions” a ferestrei Action Builder se poate adăuga orice număr de teste asupra unui model pentru a controla modul în care se realizează acţiunea.

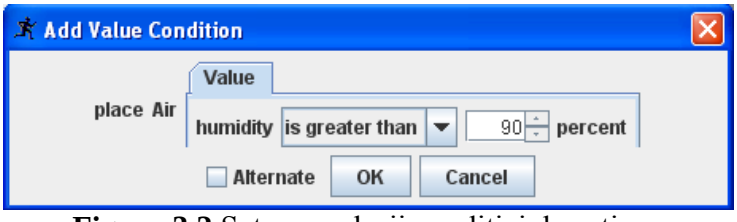

Figura 3.2 Setarea valorii condiţiei de acţiune

Tipuri

HomeRun simplifică lucrul cu modele prin stabilirea familiilor sau tipurilor de modele şi sprijină multe operaţii de build-in în funcţie de tip.

Modelul *număr* este poate cel mai simplu de înţeles, reprezentând un simplu număr, fără nici o unitate de măsură. Modelul este folosit în cazul în care se doreşte captarea unui singur număr întreg. O altă caracteristică la îndemână a unui model număr este un mijloc de asociere a valorilor de referinţă cu care valoarea modelului poate fi comparată. Acestea sunt denumite limite în HomeRun şi fiecare model poate avea orice număr de limite care au fiecare o valoare, un nume şi o pictogramă. Existenţa limitelor face să fie mai uşoară şi intuitivă adăugarea condiţiilor de acţiune (de exemplu, dacă umiditatea depăşeste 90%, anunţă-l pe John prin SMS).

Spre deosebire de cele numerice, modelele *valoare* au o unitate de măsură – precum grade Celsius – şi de obicei reprezintă cantităţi fizice măsurate, mai degrabă decât valori calculate sau atribuite, specifice modelelor de tip număr. Unităţile sunt descrise de tipuri valoare care includ nume, simboluri, pictograme etc., astfel încât vizualizarea acestor modele să poată furniza succint informaţii despre unitate. Pentru că adesea există imprecizie sau fluctuaţie în măsurătorile fizice care stau la baza modelelor valoare, acestea nu utilizează limite.

Modelele de *stare* reprezintă situaţii care constă într-una dintr-un număr fix de stări, care sunt descrise prin nume (şi pictogramă, opţional). Cele mai simple modele de stare sunt cele numite variabile booleene (adică stări true/false). De exemplu, un model de stare ar putea reprezenta locaţia unui individ: este acasă sau nu. Un model de stare poate reprezenta acest fapt prin două stări: acasă(„home“) şi plecat(„away“) şi mai departe furnizează nume pentru tranziţiile dintr-o stare în alta: „sosire“ este trecerea de la „away“ la „home“, „plecare“ este trecerea inversă. Modelele de stare pot avea desigur mai mult de două stări, dar majoritatea problemelor se pot rezolva cu modele de stare binare.

Când există o listă de una sau mai multe valori, se consideră un model *mulţime* (set). De reţinut este faptul că modelele de stare pot fi într-o mulţime fixă de stări. Mulţimile pot conţine de la zero la un număr nelimitat de valori. Fiecare valoare este reprezentată de un nume şi singura restricţie într-o mulţime este aceea că acelaşi nume nu poate apărea de mai multe ori.

Modelele *calendar* sunt folosite pentru a reprezenta timpul exprimat în forma următoare: zi din săptămână: Luni – Duminică, zi a lunii: 1 – 31, luna anului: Ianuarie – Decembrie.

Modelele de *timp*, de asemenea, exprimă o perioadă de timp dar în format ceas şi nu calendar. Aceste modele pot fi utilizate pentru a reprezenta intervale de timp.

Modele de informare

Modelul de informare ne ajută să înţelegem cum sunt actualizate şi întreţinute valorile modelelor. HomeRun utilizează termenul de „informare“ a unui model şi are un sistem prin care unui model i se atribuie unul sau mai mulţi „informatori“.

Informatorul intern – un obiect poate avea un model care este strâns legat de identitatea lui: de exemplu un obiect de tip „Soare“ cu modelele proprii „răsărit“ şi „apus“. Aceste valori pot fi calculate dacă se cunosc latitudinea şi longitudinea deci nu trebuie să existe alte informaţii externe pentru ca modelul valoare să fie determinat. Informaţiile externe nu includ datele de configurare iniţiale care trebuie introduce, cum ar fi setarea coordonatelor.

Informator controlat – la crearea unui obiect se poate specifica faptul că individul doreşte să actualizeze el însuşi starea modelului.

Informator proiectat – este cel mai comun caz şi permite asocierea unei valori modelului în contextul unei acţiuni în Action Builder.

Informator „wired“ – se creează un circuit între control şi model astfel încât să rezulte o legătură între componentele unui obiect.

3.4 Roluri

HomeRun foloseşte un set restrâns de roluri şi limitează accesul la anumite funcţionalităţi pe baza acestora. La crearea unui utilizator se asignează unul sau mai multe roluri.

Admin – poate să administreze sistemul;

Editor – poate să utilizeze aplicaţiile de editare pentru a crea noi obiecte şi componente;

User – poate utiliza panoul de control (Control Panel) sau poate lucra cu mesaje;

Other – rol folosit de către utilizatorii anonimi (care nu sunt logaţi).

Rolurile nu se autoinclud, în sensul că un administrator nu este automat şi editor etc. Fiecare rol trebuie alocat în parte fiecărui utilizator. Administrarea rolurilor se face in secţiunea Admin -> User -> Manage din Consolă.

Capitolul 4 ARHITECTURA SI PROIECTAREA APLICATIEI

4.1 Arhitectura aplicaţiei

4.1.1 Arhitectura de ansamblu a aplicaţiei

Aplicaţia HomeRun este construită şi gândită într-o manieră client – server. Cu alte cuvinte, este compusă din două părţi distincte care interacţionează. Pe lângă arhitectura client – server, aplicaţia este gândită modular. Aproape toate componentele sale sunt cuprinse în pachete independente şi sunt adăugate la server după instalare, în procesul de configurare a aplicaţiei.

Postul *client* se caracterizează prin faptul că:

- prezintă o interfaţă utilizator care este de obicei grafică (GUI);
- "formulează" cereri sau comenzi pe care le "înaintează" serverului;
- transmite comenzile respective serverului prin intermediul unei tehnologii de comunicaţie. Comenzile de control şi gestiune pot fi transmise şi de pe telefonul mobil, pentru anumite module ale aplicaţiei HomeRun;
- analizează datele din rezultatele comenzilor primite de la server.

Serverul, pe de altă parte, are următoarele caracteristici:

- furnizează un serviciu clientului;
- răspunde la comenzile clientului;
- ascunde detaliile sistemului client/server, făcând transparent dialogul dintre aceştia.

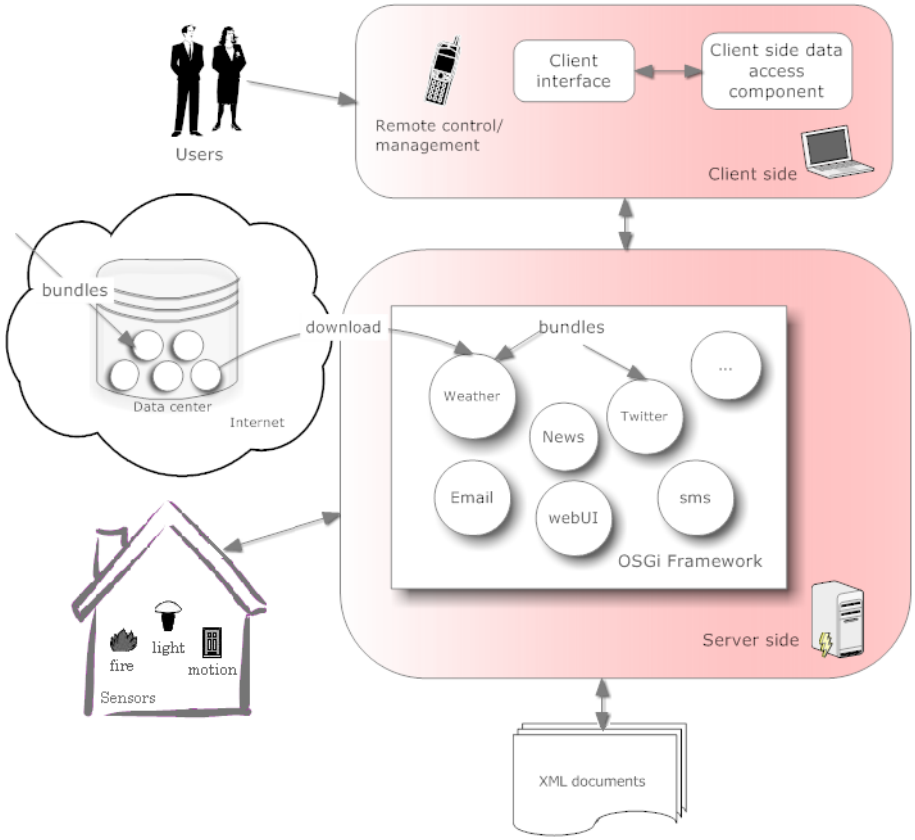

Figura 4.1 Arhitectura generală a aplicaţiei HomeRun

Comunicarea între client şi server se realizează printr-un mecanism pus la dispoziţie de către Java RMI (Remote Method Invocation) prin care se transmit informaţii în ambele direcţii. Astfel de aplicaţie este uneori referită şi ca „aplicaţie cu obiecte distribuite”.

Aplicaţia trebuie să execute următoarele sarcini:

- Localizarea obiectelor remote – există diverse mecanisme pentru a obţine referinţe la obiectele remote. Aplicaţia HomeRun îşi înregistrează obiectele remote într-un RMI Registry, facilitate oferită de RMI.
- Comunicarea cu obiectele remote – detaliile comunicării între obiecte remote sunt manipulate de RMI. Din punctul de vedere al programatorului, comunicarea remote este similară cu invocarea metodelor Java obişnuite.
- Incărcarea definiţiilor pentru clasele obiectelor care sunt transferate – deoarece RMI permite ca obiectele să fie trecute dintr-o parte în alta, furnizează mecanisme pentru încărcarea definiţiei clasei unui obiect şi pentru transmiterea datelor obiectului.

Conform figurii 4.2, aplicaţia HomeRun foloseşte RMI registry pentru a obţine o referinţă la obiectul remote. Serverul apelează RMI registry pentru a asocia un nume cu obiectul remote. Clientul caută obiectul remote după numele său în registry-ul serverului după care invocă o metodă asupra lui. In figură este ilustrat de asemenea faptul că sistemul RMI foloseşte un server web existent pentru a încărca definiţiile claselor, de la client la server şi invers când este nevoie.

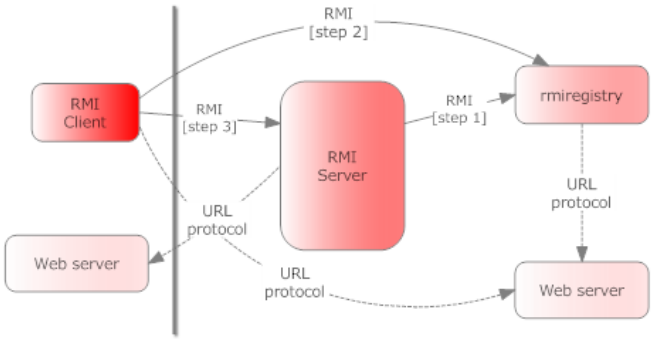

Figura 4.2 Comunicarea RMI

Una dintre calităţile importante ale RMI este posibilitatea de a descărca definiţia clasei unui obiect în cazul în care clasa nu este definită în maşina virtuală a receiver-ului. Toate tipurile şi comportamentul unui obiect, care până acum erau disponibile doar într-o singură maşină virtuală Java, pot fi transmise către o altă maşină virtuală, posibil remote. RMI transferă obiectele în funcţie de clasa lor reală, astfel încât comportamentul obiectelor nu se schimbă când ele sunt trimise către o altă maşină virtuală Java. Această calitate a RMI permite introducerea unor tipuri şi comportamente noi într-o maşină virtuală remote, prin urmare extinderea dinamică a comportamentului unei aplicaţii.

A folosi XML (*Extensible Markup Language*) este asemănător cu a alege SQL pentru baza de date. Si în acest caz trebuie construită baza de date şi procedurile care permit folosirea ei. XPath (XML Path Language) [11] este un limbaj de expresii utilizat pentru a selecta porţiuni dintr-un document XML, sau pentru a calcula valori pe baza conţinutului unui document XML. Câteva avantaje faţă de SQL ar fi acelea că XML funcţionează pe orice platformă, nu are nevoie de licenţă şi permite accesul la date aflate la distanţă (remote data). XML permite lucrul cu mai multe date provenind de la mai multe surse şi obţinerea mai multor rezultate pe baza acestor date. Documentele XML au un caracter semi-structurat al datelor (structura internă nu este atât de regulată). Mai trebuie să se asigure stocarea eficientă a datelor, securitatea şi integritatea acestora. XML este tehnologia preferată pentru multe din aplicaţiile de transfer de informaţii deoarece are abilitatea de a coda informaţia într-un format

care este uşor de citit, procesat şi generat. Java suportă XML. Java şi XML au caracteristici comune: ambele limbaje au o evoluţie similară, au aceleaşi scopuri (simplicitate, portabilitate şi flexibilitate) şi continuă să fie dezvoltate în toate mediile de activitate. Din aceste motive se poate spune că Java este mediul cel mai preferat pentru dezvoltarea de aplicaţii pe partea de server şi partea de client bazate pe XML.

4.1.2 Arhitectura modulului SMS

Twitter este un nou tip de tehnologie care permite utilizatorilor să comunice între ei. Foloseşte mai multe metode de comunicare dar cea mai eficientă este trimiterea de tweet-uri (notificări) prin SMS. Cu tehnologia SMS se pot trimite actualizările către telefonul mobil. Pentru a folosi această metodă excelentă de comunicare utilizatorul nu mai are neapărată nevoie de un calculator personal conectat la internet, ci doar de un telefon mobil. In rest, toată treaba este făcută de Twitter.

Aplicaţia HomeRun foloseşte serviciul social de mesagerie Twitter pentru a trimite utilizatorilor notificări direct pe telefonul mobil. Aceasta este funcţionalitatea modulul „Twitter". Fluxul evenimentelor care se desfăşoară este prezentat în figura 4.3, sensul fiind de la dreapta la stânga (Computer -> GSM phone).

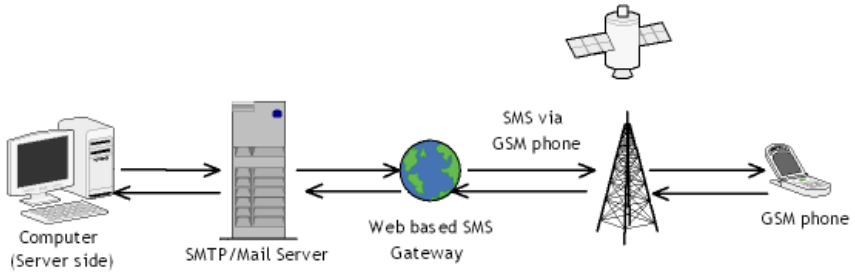

Figura 4.3 Comunicare între calculator (aplicaţie desktop) şi telefonul mobil

Modulul pune la dispoziţia utilizatorului două controale: *tweet* şi *direct*. Controlul *tweet* permite actualizarea statusului pentru contul configurat în pachet. Controlul *direct* foloseşte facilitatea „direct messaging" pusă la dispoziţie de Twitter care permite trimiterea de mesaje private către alt utilizator. Primul control permite trimiterea de notificări SMS către mai mulţi utilizatori în acelaşi timp, întrucât toţi cei care urmăresc contul căruia i se trimite notificarea de către aplicaţia HomeRun vor primi mesaj pe telefonul mobil. Controlul *direct* direcţionează mesajul text spre un singur utilizator.

Serviciile de email-uri de pe internet vin sub formă de „gateway service" caz în care mesajele nu sunt stocate, sau „mailbox" în care mesajele sunt stocate. In cazul serviciului gateway, platforma wireless de email transferă pur şi simplu mesajul de la SMTP, protocolul de internet pentru email, sub formă de SMS către centrul de SMS-uri. In cazul serviciului mailbox, email-urile sunt de fapt stocate şi utilizatorul are acces la ele numai conectându-se la internet (eventual de pe telefonul mobil).

Modulele SMS şi Twitter folosesc serviciul gateway de mesagerie care este pus la dispoziţie de aplicaţia Twitter şi a cărei arhitectură este prezentată în figura 4.4. Serviciul accesează periodic contul utilizatorilor folosind metoda RSS şi verifică dacă au apărut mesaje noi, caz în care preia aceste mesaje şi le transferă către un centru de SMS-uri de unde vor fi redirecţionate către destinatari.

Figura 4.4 Arhitectura sistemului Twitter de trimitere a SMS-urilor

SMS gateway este un serviciu care oferă transferul mesajelor între două surse care nu folosesc acelaşi protocol de comunicare. Un caz obişnuit de utilizare ar fi simpla trimitere a unui email către un telefon mobil. Operatorii de reţele wireless folosesc SMS gateways pentru a conecta centrele de SMS (SMSCs). Un centru SMS este o porţiune a reţelei wireless care gestionează operaţiunile asupra SMS-urilor, cum ar fi rutarea, transmiterea şi stocarea mesajelor text aflate în curs de trimitere spre destinaţia finală.

Dacă modulul Twitter al aplicaţiei HomeRun descrie fluxul evenimentelor din figura 4.3 doar într-un singur sens, şi anume direcţia comunicării fiind dinspre calculator către telefonul mobil, modulul SMS completează fluxul evenimentelor cu o comunicare în sens invers (GSM phone -> Computer). Acest modul oferă posibilitatea utilizatorilor de a comunica cu aplicaţia HomeRun prin intermediul telefonului mobil. Este nevoie ca utilizatorul să trimită un mesaj text pentru a cere anumite informaţii pe care le primeşte în scurt timp pe telefonul mobil.

Figura 4.5 ilustrează procedurile de comunicare între componentele sistemului. Există două canale de comunicare, fiecare fiind corespunzător unui sens al fluxului de evenimente. Când un mesaj ajunge la gateway se trimite automat o cerere HTTP către serverul de e-mail pentru a actualiza statusul utilizatorului Twitter sau pentru a trimite un mesaj direct altui utilizator. Această cerere poate să returneze un răspuns sub forma unui mesaj SMS anunţând utilizatorul că cererea a fost procesată sau un mesaj care conţine informaţiile solicitate, puse la dispoziţie de aplicaţia desktop de pe calculatorul personal.

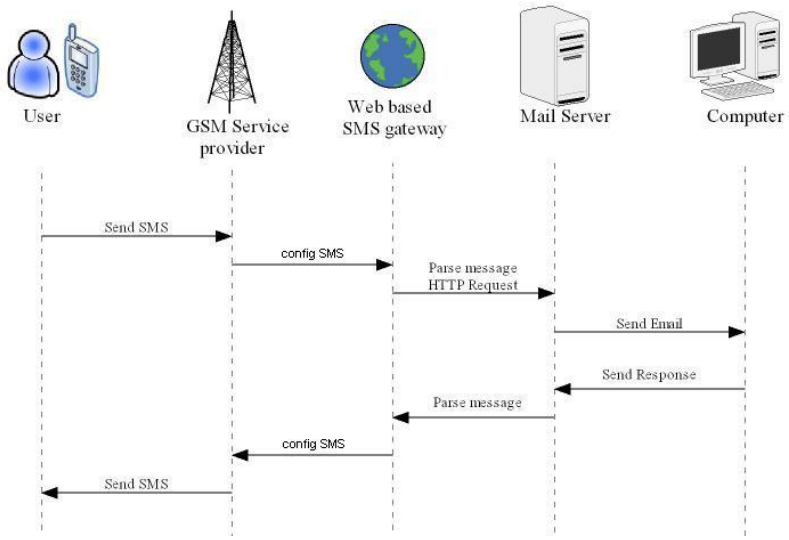

Figura 4.5 Procedură de comunicare între utilizator şi calculatorul personal

4.1.3 Arhitectura modulului News

RSS este un protocol deschis de publicare a informaţiilor pe Internet. Un RSS – Rich Site Summary feed – este un fişier XML care descrie conţinutul unui site, fiind actualizat odată cu acesta. RSS este considerat şi prescurtarea de la Really Simple Syndication datorită uşurinţei cu care utilizatorii pot accesa informaţia actualizată de pe internet. Un fişier RSS include titlul, link-ul, descrierea site-ului şi itemii de noutate. Fiecare item constă din URL-ul articolului respectiv, titlul articolului şi o scurtă descriere.

Structura unei reţele RSS constă din trei componente principale: un număr mare de furnizori de conţinut, descris prin fişierul RSS propriu, un număr relativ mic de agregatoare, care citesc şi indexează fişierele RSS şi un număr mare de aplicaţii de citire. Un utilizator al unei astfel de aplicaţii poate vizualiza descrierile şi vizita fiecare articol.

Sisteme numite agregatoare – sisteme online sau RSS Reader-e instalate local, citesc RSS-urile şi informează abonaţii asupra modificărilor de pe site, informaţia fiind astfel transmisă unei largi audienţe – syndicated. Informaţia din fişierele RSS este stocată în baze

de date, care pot fi accesate cu multiple criterii de căutare. Prin abonarea la RSS-uri, utilizatorul este la curent cu noutăţile despre site-urile care oferă acest mecanism, fără a mai fi necesar să le viziteze periodic pentru a vedea ultimele noutăţi. Agregatoarele fac parte dintr-o categorie mai largă de aplicaţii, numite *harvesters*. Acestea preiau metadate aflate pe diferite servere şi le furnizează într-o formă specifică diferitelor aplicaţii.

Motoarele de căutare specifice RSS-urilor fac vizibilă informaţia din acestea la câteva secunde/minute de la publicarea ei, spre deosebire de motoarele clasice de căutare, care indexează informaţia la câteva zile de la apariţia ei. Din această cauză, RSS-urile constituie World Live Web.

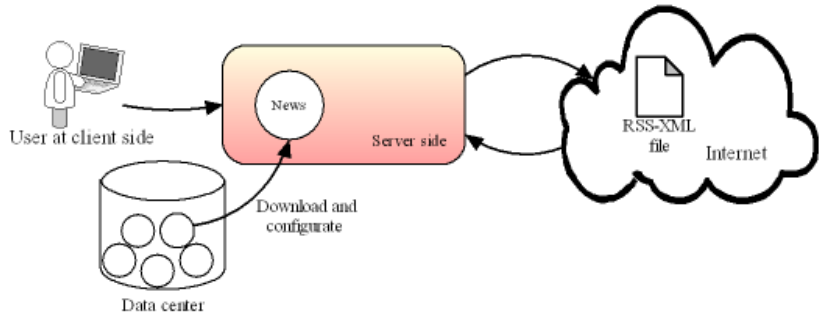

Figura 4.6 Arhitectura modulului News

Conform figurii 4.7, în aplicaţia HomeRun, rolul agregatorului este jucat de scene, construcţiile pentru vizualizarea modelelor. Acesta este un *agregator personal*, întrucât este o aplicaţie care ruleaza local. Poate accesa un agregator centralizat sau direct un canal RSS. In cazul de faţă accesează direct canalul introdus în etapa de configurare a modulului News, http://newsrss.bbc.co.uk/rss/newsonline_world_edition/technology/rss.xml. Prin alegerea acestui canal RSS, utilizatorul se abonează (Subscribe) pentru a primi ştiri din domeniul respective la intervale de timp stabilite, de asemenea, în etapa de configurare.

Pentru a selecta şi extrage informaţii din fişierul XML se utilizează limbajul de expresii XPath care foloseşte o sintaxă de adresare bazată pe o cale prin structura logică a documentului sau sintaxă bazată pe ierarhie. Acest lucru face scrierea expresiilor de programare mai uşoară decât în cazul în care fiecare expresie ar trebui să cunoască şi să înţeleagă sintaxa XML şi ordinea informaţiilor într-un document. Xpath permite de asemenea programatorului să lucreze cu un document într-o formă mai înaltă de abstractizare. Câteva concept folosite de limbajul XPath sunt următoarele: nod concept (este punctul din care începe adresa căii), arbore logic (care este parte constitutivă a unui document XML) şi concept reprezentând relaţii logice între noduri, cum ar fi: strămoş, atribut, copil, părinte. XPath include şi un set redus de expresii pentru specificarea funcţiilor matematice, putând fi adăugate şi alte funcţii.

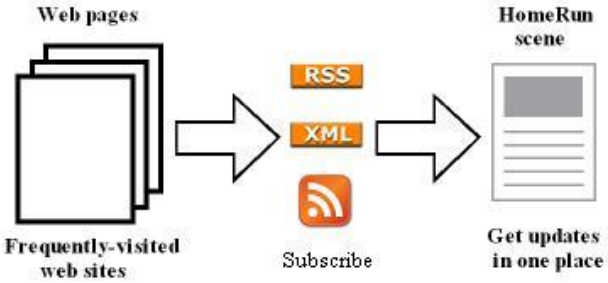

Figura 4.7 Preluarea şi centralizarea automată a informaţiilor din canalele RSS

RSS este un instrument important pentru sindicarea informaţiilor. Lipsa câtorva facilităţi face însă să nu fie o soluţie robustă pentru localizarea şi organizarea informaţiei: lipsa informaţiei de copyright atât în articolele originale, cât şi în RSS feeds (RSS presupune

că articolele şi metadatele sunt publicate pe pagini gratuite, deci doar resursele gratuite pot fi accesate prin RSS) şi inabilitatea agregatoarelor RSS de a analiza date conforme diferitelor standarde.

4.2 Cazuri de utilizare

4.2.1 Instalarea unui modul HomeRun

In continuare se vor identifica paşii pe care un utilizator trebuie să îi urmeze pentru a se putea bucura de funcţionalităţile pe care aplicaţia i le pune la dispoziţie. Utilizatorii care prezintă interes privind acest caz de utilizare sunt utilizatorii cu drepturi de administrator, în cazul în care sistemul se află în stare închisă (se cere autentificarea utilizatorilor). In caz contrar, dacă sistemul se află în stare deschisă şi utilizatorii nu necesită logare (crearea şi configurarea contului sunt opţionale), aceştia se pot bucura de toate funcţionalităţile aplicaţiei.

Inainte de instalarea modulelor, trebuie să fie îndeplinite câteva precondiţii: aplicaţia server să fie instalată pe un calculator care să îndeplinească cerinţele hardware şi software specificate în capitolul 5, aplicaţia client să fie instalată pe unul sau mai multe calculatoare care, de asemenea, să îndeplinească cerinţele software.

Se aplică şi câteva postcondiţiile care trebuie să fie îndeplinite la încheierea fluxului de evenimente: sistemul trebuie să rămână neschimbat în cazul în care aplicaţia eşuează şi de asemenea, sistemul rămâne într-o stare consistentă dacă operaţiile se încheie cu succes.

Fluxul de evenimente de bază (Basic Flow)

Acest caz de utilizare începe când un utilizator doreşte să instaleze un modul (bundle) al aplicaţiei HomeRun căruia să îi testeze funcţionalitatea.

1. Utilizatorul lansează în execuţie aplicaţia server pe calculatorul-server.
2. Utilizatorul lansează în execuţie aplicaţia client pe calculatorul-client.
3. Pe ecran este afişată consola aplicaţiei client.
4. In consola client utilizatorul introduce IP-ul calculatorului-server pentru a se conecta la aplicaţia server.

4.1 Dacă nu a fost găsit IP-ul introdus sau serverul nu a fost instalat pe calculatorul cu IP-ul introdus, se afişează un mesaj de eroare şi se revine la pasul 4.

4.2 Dacă IP-ul introdus e corect şi s-a localizat serverul, se trece la pasul următor.

5. Se afişează o pictogramă în colţul din stânga al consolei care indică faptul că serverul a fost localizat şi pornit, aplicaţiile putând de acum interacţiona.
6. Utilizatorul alege opţiunea de creare a unui cont nou (opţional).
7. Utilizatorul îşi configurează contul creat (opţional).
8. Utilizatorul alege din lista de opţiuni modulul cu funcţionalitatea dorită.
9. In consolă apare descrierea detaliată a funcţionalităţii modulului selectat.
10. Utilizatorul alege opţiunea de instalare a modulului.
11. Utilizatorul configurează setările modulului (de exemplu, în cazul modulului „Weather”, se va introduce un link către site-ul de meteorologie de unde se preiau informaţiile privind starea vremii şi intervalul de timp la care vor fi reînnoite informaţiile).
12. Pe ecran este afişată din nou lista de module.

12.1 Dacă utilizatorul doreşte să instaleze şi alte module se reia pasul 8.

12.2 Altfel se trece la pasul 13.

13. Utilizatorul revine în consolă.

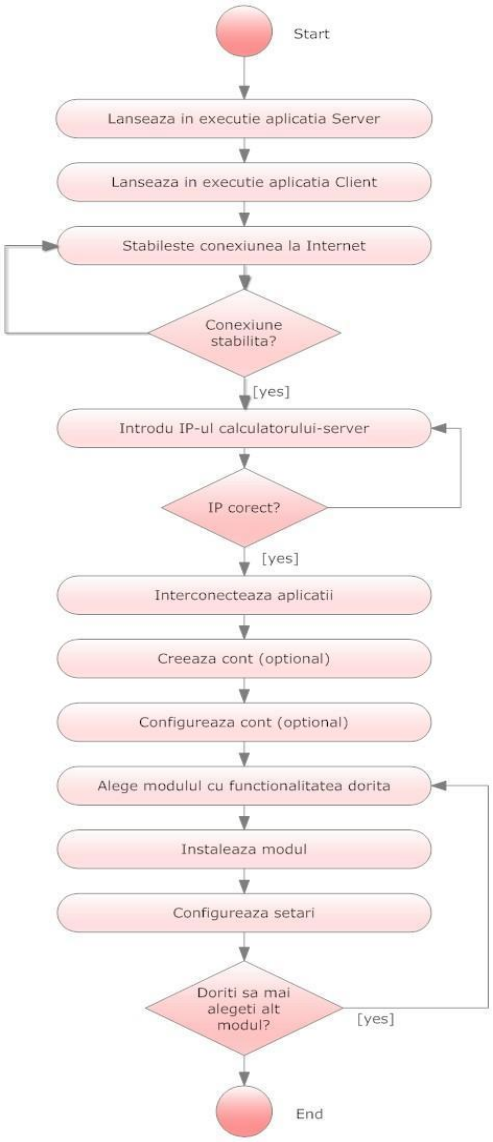

Figura 4.8 Fluxul de evenimente pentru instalarea unui modul HomeRun

Fluxul de evenimente alternativ (Alternative Flow)

1’. Aplicaţia eşuează

Acest lucru poate să se întâmple în unul din următorii paşi: 4, 6 sau 8.

1’.1. Nu se găseşte serverul, în consecinţă nu se poate face interconectarea dintre aplicaţiile client şi server.

1’.2. Utilizatorul selectează comanda Cancel şi părăseşte pagina de configurare a contului.

1’.3. Utilizatorul selectează comanda Cancel şi părăseşte pagina de configurare a modulului.

Sistemul rămâne neschimbat şi într-o stare consistentă.

4.2.2 Utilizarea modulului SMS

Inainte de instalarea modulului SMS trebuie să fie îndeplinite câteva precondiţii: instalarea şi configurarea în prealabil a pachetelor Twitter, News şi Weather, presupunând că aplicaţiile server şi client au fost deja instalate şi configurate.

Postcondiţiile sunt aceleaşi ca pentru orice alt caz de utilizare din cadrul aplicaţiei HomeRun: sistemul trebuie să rămână neschimbat în cazul în care aplicaţia eşuează şi de asemenea, sistemul rămâne într-o stare consistentă dacă operaţiile se încheie cu succes.

Cazul de utilizarea al modulului SMS este iniţiat de către un utilizator obişnuit, în cazul în care sistemul este deschis (nu se solicită identificarea şi autentificarea utilizatorilor). In caz contrar, actorii implicaţi sunt administratorul, editorul sau utilizatorii autentificaţi. Administratorul este cel care instalează şi configurează modulul, după care ceilalţi utilizatori cu drepturi se pot bucura de funcţionalităţile modulului. După cum se poate observa în figura 4.9, rolurile nu se autoinclud. Adică administratorul nu este atomat şi utilizator. La crearea unui nou cont HomeRun se asignează de către administrator unul sau mai multe roluri: admin, editor, user sau other.

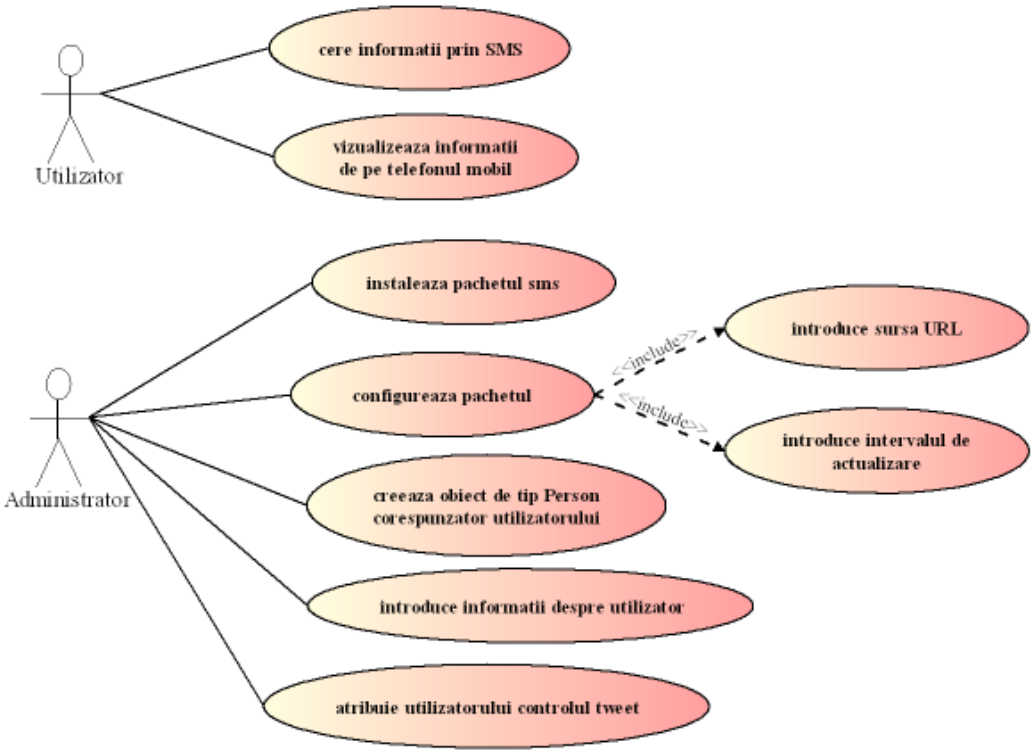

Figura 4.9 Diagrama cazului de utilizare pentru modulul SMS

Scenariul asociat cazului de utilizare ilustrat în figura 4.9:

1. Administratorul alege opţiunea „Twitter SMS” din lista de pachete şi apasă butonul Install.
2. Pentru configurarea pachetului, administratorul alege din fereastra Setup opţiunea Config -> Open.. şi introduce proprietăţile specifice modulului SMS.

2.1 Se introduce sursa URL de unde vor fi extrase informaţiile cu privire la textul SMS-ului trimis de către utilizator aplicaţiei HomeRun.

2.2 Se introduce o valoare întreagă care reprezintă frecvenţa cu care se verifică apariţia unui nou mesaj pe Twitter.

3. Din fereastra Object Editor, administratorul creează un obiect de tip Person corespunzător utilizatorului.
4. Se introduc informaţii cu privire la acest utilizator: contul Twitter asociat.
5. Obiectului creat i se atribuie controlul „tweet”.
6. Utilizatorul poate de acum, în orice moment, să trimită mesaje text de pe telefonul mobil pentru a cere informaţii aplicaţiei HomeRun.

6.1 Dacă textul mesajului este „weather”, utilizatorul cere informaţii în legătură cu starea vremii la domiciliu.

6.2 Dacă textul mesajului este „news”, utilizatorul cere informaţii în legătură cu ultima ştire apărută pe site-ul introduc la configurarea modulului News.

7. Utilizatorul vizualizează informaţiile primite de la aplicaţia HomeRun direct de pe telefonul mobil.

Există posibilitatea ca software-ul asociat modulului să eşueze. Este cazul în care, de exemplu, sursa URL nu a fost introdusă corect sau nu este recunoscută. De asemenea, textul mesajelor trimise de către utilizator trebuie să respecte cuvintele cheie „weather” şi „news” deoarece numai acestea sunt recunoscute în cadrul aplicaţiei. In caz contrar, nu se va realiza nici o acţiune.

Oricare ar fi cauza eşecului, sistemul trebuie să rămână într-o stare consistentă.

4.2.3 Utilizarea modulului News

Pentru funcţionarea modulului News nu sunt necesare alte componente. Acest pachet foloseşte o metodă alternativă de accesare a informaţiei disponibile pe internet, RSS sau „Really Simple Syndication (RSS 2.0)“. Astfel, în loc să se acceseze o sursă de informaţii de fiecare dată când e nevoie, prin intermediul unei subscrieri informaţia este trimisă către utilizatorul aplicaţiei chiar în momentul publicării ei.

Precondiţiile care trebuie îndeplinite înainte de utilizarea modulului sunt: aplicaţia server să fie instalată pe un calculator care să îndeplinească cerinţele hardware şi software specificate în capitolul 5, aplicaţia client să fie instalată pe unul sau mai multe calculatoare care, de asemenea, să îndeplinească cerinţele software.

Postcondiţiile sunt aceleaşi ca pentru orice alt caz de utilizare din cadrul aplicaţiei HomeRun: sistemul trebuie să rămână neschimbat în cazul în care aplicaţia eşuează şi de asemenea, sistemul rămâne într-o stare consistentă dacă operaţiile se încheie cu succes.

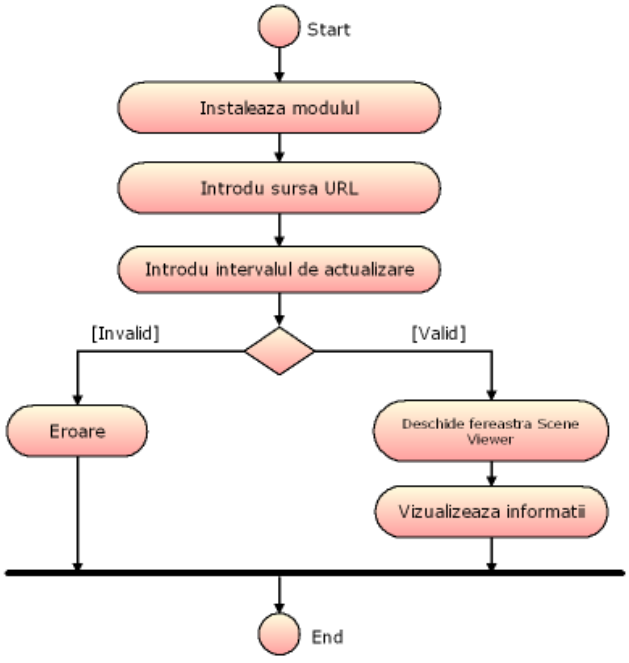

Figura 4.10 Fluxul de evenimente pentru utilizarea modulului News

Fluxul de evenimente:

1. Administratorul alege opţiunea „BBC news” din lista de pachete şi apasă butonul Install.
2. Pentru configurarea pachetului, administratorul alege din fereastra Setup opţiunea Config -> Open.. şi introduce proprietăţile specifice modulului News.

2.1 Se introduce sursa URL reprezentând canalul RSS de unde vor fi extrase ştirile.

2.2 Se introduce o valoare întreagă care reprezintă frecvenţa cu care se verifică apariţia unei noi ştiri.
3. Dacă informaţiile introduse sunt greşite se afişează un mesaj de eroare şi aplicaţia eşuează.
4. Dacă informaţiile sunt corecte se trece la pasul 5.
5. Utilizatorul deschide fereastra Scene Viewer.
6. Utilizatorul vizualizează informaţiile extrase din canalul RSS.

Dacă aplicaţia eşuează, sistemul revine la starea iniţială şi rămâne într-o stare neschimbată şi consistentă.

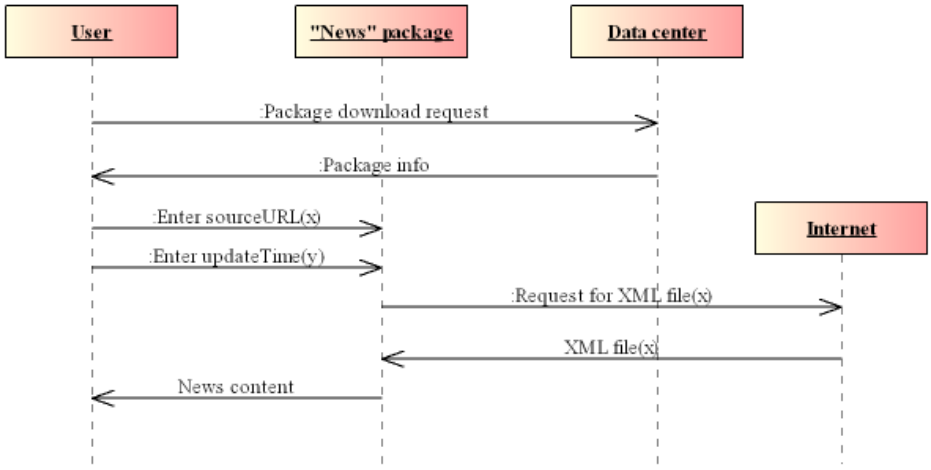

Figura 4.11 Diagrama de secvenţă pentru utilizarea modulului News

In figura 4.11 e prezentată succesiunea detaliată a paşilor care se execută în vederea obţinerii funcţionalităţii modulului News, conform fluxului de evenimente.

Modulul News poate fi folosit împreună cu alte module, ca de exemplu SMS sau Email. Astfel, utilizatorul poate să primească notificări cu privire la ultimele ştiri apărute prin intermediul unui mesaj text direct pe telefonul mobil sau sub forma unui email pe internet.

4.3 Proiectarea aplicaţiei

4.3.1 Aplicaţia client

In figura 4.12 e prezentată diagrama de pachete a aplicaţiei client cu câteva clase reprezentative care implementează interfaţa cu utilizatorul şi componentele de comunicare cu aplicaţia server.

Pachetul bootapp

BootTalker – ajută la efectuarea comunicării de nivel scăzut cu serverul aflat în stare bootstrap (serverul este încărcat dar încă nu rulează) folosind protocolul socket.

Console – este o aplicaţie grafică foarte simplă de unde se lansează toate modulele HomeRun şi se controlează serverul. De aici se poate porni şi opri serverul, se poate lansa în execuţie orice aplicaţie remote. De asemenea în consolă se afişează informaţii cu privire la starea serverului.

Setup – este o aplicaţie grafică folosită pentru a edita valorile ce constituie parametrii de configurare ai serverului. Gestionează instalarea componentelor serverului şi alte operaţii de mentenanţă a sistemului. Comunică prin serverul în stare bootstrap, deci poate să ruleze chiar dacă serverul nu este activ.

LogViewer – este o aplicaţie pentru vizualizarea şi gestionarea log-urilor. Utilizatorul poate selecta log-urile după dată şi tip, poate să le vizualizeze sau să le caute, să le arhiveze,

să le şteargă sau să le transfere de pe server. Comunică prin serverul în stare bootstrap, deci poate să ruleze chiar dacă serverul nu este activ.

Pachetul netutl

LocationListener – este o interfaţă folosită de componentele care au nevoie de locaţia unui serviciu care ar putea deveni disponibilă.

ServiceLocator – are rolul de a descoperi servicii pentru aplicaţiile client. Serviciile de bază sunt cele proprii ale serverului HomeRun şi anume „bootstrap admin service", „RMI registry" şi „web service".

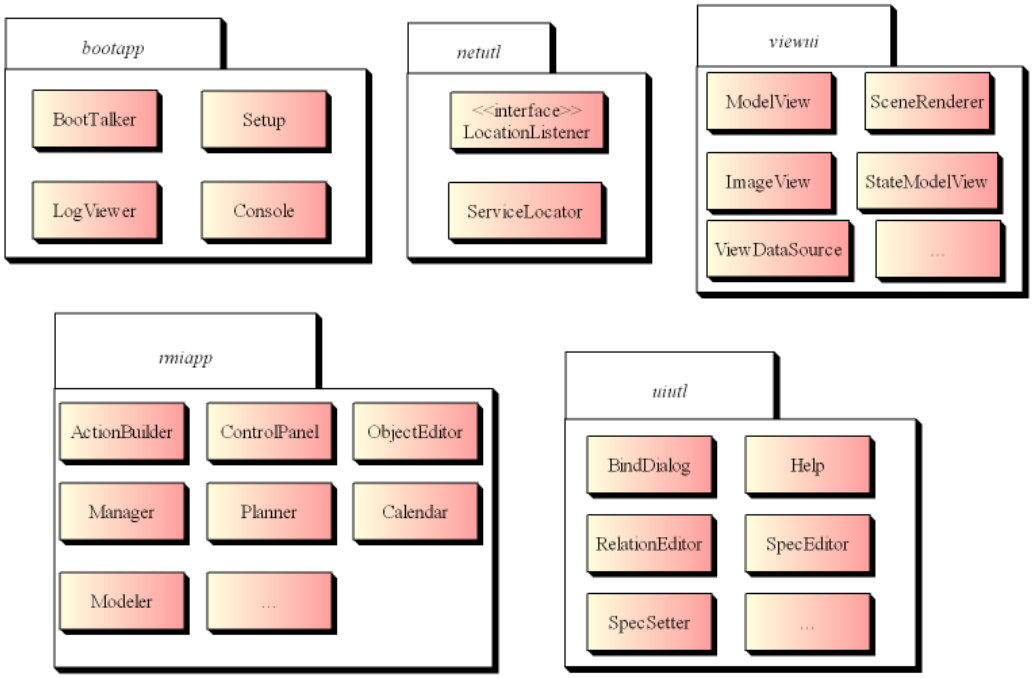

Figura 4.12 Diagrama de pachete a aplicaţiei client

Pachetul viewui

DateModelView – afişează un model de tip dată într-un panel pe baza formatului stabilit pentru afişarea elementelor. Modelul e constituit dintr-un singur „obiect" de tip dată.

ImageView – afişează o imagine într-un panel.

ModelView – interfaţă implementată de StateModelView, TimeModelView, NumericModelView, ScalarModelView etc.

ViewDataSource – oferă metode pentru obţinerea datelor necesare pentru a reda vizualizări (Views).

SceneRenderer – afişează scenele HomeRun care sunt ansambluri de date. Scenele nu se actualizează automat la schimbarea modelului, dar se pot actualiza prin invocarea metodei „refresh" la apăsarea butonului corespunzător.

Pachetul rmiapp

ActionBuilder – este o aplicaţie care gestionează regulile de acţiune. Utilizatorul poate crea noi reguli, poate modifica regulile deja existente, le poate testa, aplica, şterge etc.

ControlPanel – afişează panel-uri, care sunt mulţimi de obiecte cărora li s-au asociat controale şi/sau reguli de acţiune.Aceste panel-uri sunt folosite pentru a controla manual dispozitive şi pentru a executa reguli de acţiune.

ObjectEditor – este o aplicaţie grafică folosită pentru gestionarea obiectelor HomeRun. Utilizatorul selectează un obiect în Object Explorer pe care îl poate edita (poate să îi seteze anumite valori, să îi adauge sau să îi şteargă controale şi modele etc.)

Manager – este o aplicaţie grafică folosită pentru funcţii administrative cum ar fi gestionarea utilizatorilor.

Planner – este o aplicaţie care permite asignarea de acţiuni unor evenimente.

Calendar – este o aplicaţie care permite utilizatorului să asigneze o acţiune unei anumite date într-un calendar sau să modifice acţiunile care au fost programate în viitor.

Modeler – este o aplicaţie care permite crearea, editarea şi gestionarea modelelor. Utilizatorul selectează un domeniu şi o clasă, după care introduce numele stărilor, valorile etc. Modelele pot fi adăugate, modificate şi salvate sau şterse din „depozitul" de modele.

Pachetul uiutl

Help – conţine informaţii care vin în ajutorul utilizatorului.
RelationEditor – permite editarea relaţiilor de tip model.
SpecEditor – permite editarea de specificaţii.
SpecSetter – este o interfaţă grafică pentru asignarea şi editarea de specificaţii.

4.3.2 Aplicaţia server (cadrul de lucru OSGi)

Odată cu instalarea unui pachet, acesta este stocat în partea de server a aplicaţiei HomeRun. Acest lucru prezintă un mare avantaj în ceea ce priveşte gestionarea memoriei, deoarece memoria sistemului va fi ocupată doar de pachetele de care este nevoie, fără ca utilizatorul să fie nevoit să copieze şi să instaleze toate pachetele existente. In figura 4.13 este prezentată o diagramă a câtorva pachete care pun la dispoziţia utilizatorilor diverse funcţionalităţi.

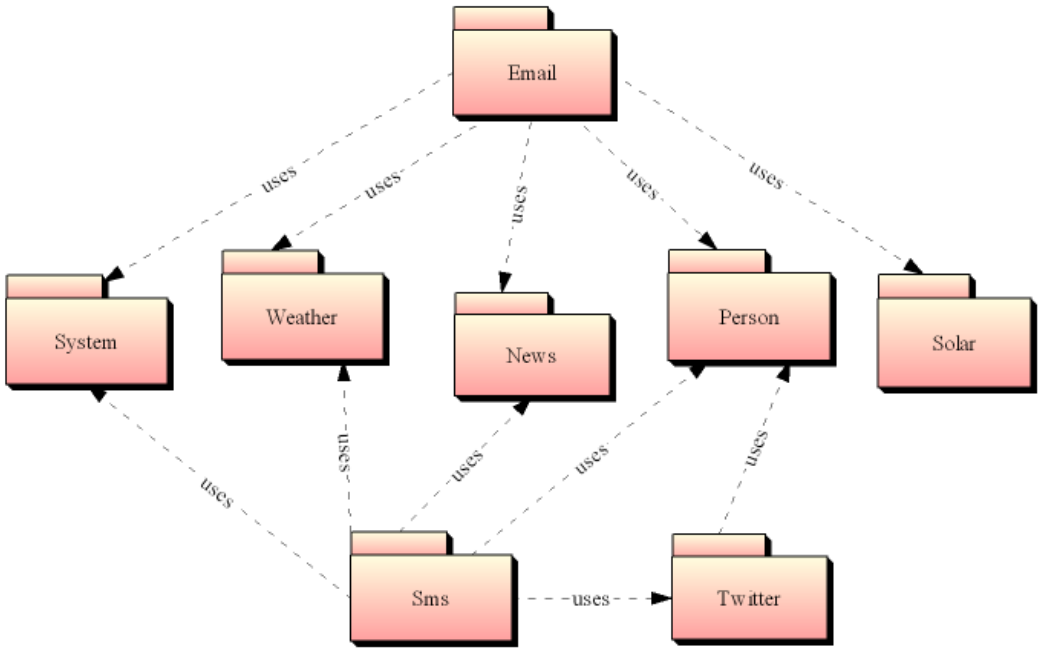

Figura 4.13 Diagrama de pachete din cadrul de lucru OSGi (Server side)

Pentru ca un pachet să folosească funcţionalităţile altuia nu este suficient să îl importe, cum se întâmplă în cazul aplicaţiilor Java obişnuite. Comunicarea se realizează prin intermediul serviciilor. OSGi permite modulelor să publice servicii şi să depindă de servicii publicate de alte module. Serviciile sunt cunoscute de către interfeţele publicate ale acestora, nu de către implementarea în sine. Acest lucru înseamnă că nivelul de cuplare între modulele

care publică servicii şi cele care le consumă este redus. In figura 4.14 este prezentată modalitatea prin care un modul (Service Provider) publică un serviciu care v-a fi înregistrat în Service Registry, după care orice alt modul (Service Consumer) are acces la el.

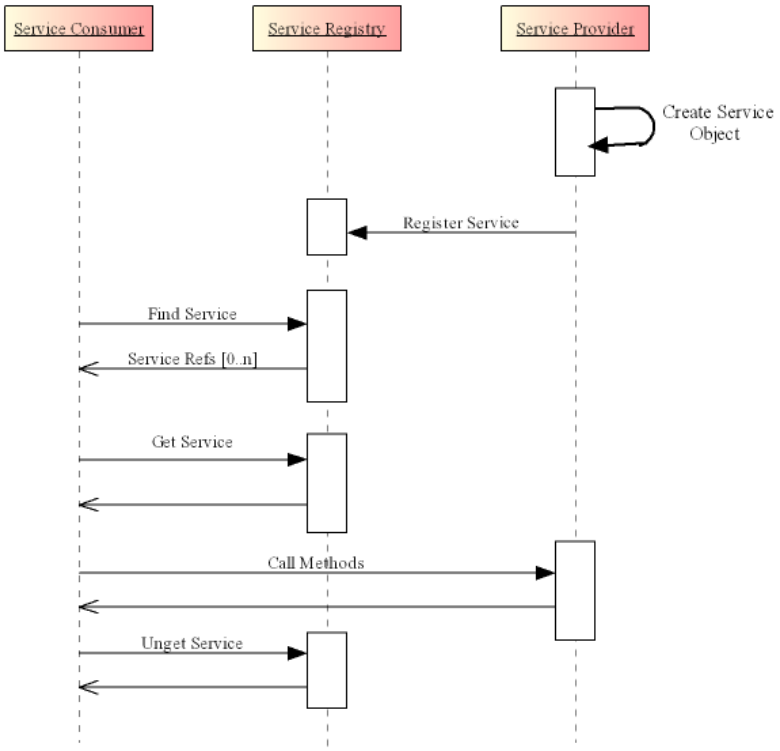

Figura 4.14 Diagrama de secvenţă pentru crearea şi consumarea unui serviciu

4.3.3 Specificaţiile pachetelor

- **Pachetul Twitter**

Pachetul conţine mijloacele de a folosi serviciul social de mesagerie Twitter pentru notificări HomeRun. Având în vedere că Twitter permite trimiterea de mesaje text (SMS), acest pachet poate funcţiona şi ca un serviciu de notificare prin SMS către telefonul mobil.

Utilizarea acestui pachet nu necesită neapărat instalarea altor pachete suplimentare, dar în general se foloseşte împreună cu pachetul Person.

Conţinutul pachetului

In plus faţă de cod şi o interfaţare configurabilă cu Twitter, sunt prevăzute două controale, fiecare conceput pentru a fi utilizat cu obiecte de tip Person. Controlul *tweet* permite actualizarea status-ului pentru contul configurat în pachet. In funcţie de setările contului Twitter, această actualizare va putea fi văzută de către toată lumea sau numai de către cei care urmăresc acest cont. Controlul *direct* foloseşte facilitatea Twitter "direct messaging" (mesaje directe) pentru a trimite un mesaj privat altui utilizator Twitter.

Configurare

După instalare, modulul Twitter trebuie configurat. In acest scop se alege din meniu opţiunea Setup -> Config -> Open, după care se selectează din listă "messaging". In continuare se selectează "twitter", apoi providers -> twitter din arborele care apare. Este obligatorie completarea ambelor câmpuri: *User Name* se referă la numele de utilizator asociat contului Twitter şi *Password* este parola corespunzătoare contului care va fi utilizat. Este

indicată crearea unui nou cont Twitter care să reprezinte aplicaţia HomeRun instalată pe calculatorul personal.

Mod de folosire

Pentru controlul *tweet* se selectează mai întâi persoana corespunzătoare contului care a fost introdus în partea de configurare. Dacă o astfel de persoană nu există, se creează în Object Editor. In continuare se adaugă controlul *tweet* persoanei, după care se poate actualiza status-ul contului, fie din Control Panel sau din Action Builder. Pentru trimiterea de mesaje directe, trebuie să se cunoască username-ul persoanei căreia i se trimit mesajele, iar mesajele pot fi trimise doar cu aprobarea acelei persoane. Se adaugă controlul *direct* persoanei dorite şi se introduce username-ul acesteia în câmpul *properties*. După aceea controlul poate fi folosit în acţiuni după modelul prezentat în capitolul 3. Pentru a primi notificări prin SMS trebuie configurat contul personal de Twitter [8].

- **Pachetul Email**

Pachetul conţine mijloace pentru a trimite notificări din partea aplicaţiei HomeRun sub formă de email-uri. Funcţionarea acestui pachet nu necesită instalarea altor pachete suplimentare, dar poate fi utilizat împreună cu pachetele Weather, News, Person, System şi Solar.

Conţinutul pachetului

In plus faţă de cod şi o interfaţare configurabilă a serverelor de email, este furnizat un control *email* care poate fi atribuit, spre exemplu, obiectelor de tip Person.

Configurare

După instalare, pentru a configure modulul se alege din meniu opţiunea Setup -> Config -> Open, apoi se selectează din listă “messaging”. In continuare se selectează “email”, apoi servers -> default din arborele care apare. Trebuie introduse toate valorile care se solicit, în special adresa serverului de mail. Se poate personaliza subiectul tuturor email-urilor trimise şi adresa expeditorului. Se setează de asemenea tipul de autentificare şi dacă se alege opţiunea “password” trebuie introduse username-ul şi parola contului de pe care se trimit notificări, pentru a putea accesa serverul de email-uri.

Mod de folosire

Obiectului asociat fiecărei persoane care trebuie să primească notificări de la aplicaţia HomeRun i se atribuie controlul *email* în Object Editor. Obiectului trebuie să i se asigneze două proprietăţi: un calificativ (qualifier) şi adresa de email. Adresa trebuie să fie în format standard (de exemplu, oxana.hotea@gmail.com). De vreme ce o persoană poate avea mai multe adrese de email, controlul *email* se poate atribui unui obiect de mai multe ori, de aceea fiind nevoie de asignarea unor “calificative”. Fiecare calificativ este un nume pentru a distinge legăturile între persoană şi controale. De exemplu unei persoane i se pot atribui calificativele *work* şi *home*. Acestea nu influenţează trimiterea email-urilor. Ele pur şi simplu apar în arborele de explorare astfel încât să se poată alege cu uşurinţă email-ul corect atunci când e nevoie [8].

- **Pachetul Person**

Pachetul conţine informaţii despre obiectele de tip person, tipuri, controluri comune şi modele care pot fi asociate cu aceste obiecte. Funcţionarea acestui pachet nu necesită instalarea altor pachete suplimentare.

Conţinutul pachetului

In afara definiţiei domeniului *person*, pachetul conţine un tip pentru persoane individuale şi un tip pentru grupuri de persoane. In cele din urmă, sunt furnizate două modele care sunt de folos în lucrul cu obiectele de tip persoană: *location* este un model de stare simplu care permite localizarea persoanei (dacă aceasta se află acasă sau nu) şi modelul de stare *waking* determină dacă persoana este trează sau doarme. Utilizatorul pachetului trebuie să stabilească modul în care sunt informate aceste modele.

Configurare şi mod de folosire

Pachetul nu necesită nici un fel de configurare. In ceea ce priveşte modul de folosire, se creează doar obiecte de tip persoană sau de tip grup de persoane pentru a reprezenta locatarii unei clădiri [8].

- **Pachetul Weather**

Pachetul adună şi modelează date meteo preluate de pe internet. Funcţionarea acestui pachet nu necesită instalarea altor pachete suplimentare.

Conţinutul pachetului

Pachetul conţine un software care interfaţează diferite surse ce furnizează informaţii despre vreme. Sursa *web* obţine informaţii meteo de la serviciul de meteorologie internaţional www.weather.com.

Datele obţinute sunt introduse în modelele obiectelor pe care pachetul le furnizează, toate în categoria *weather* a domeniului *place*. Obiectele şi modelele sunt următoarele:

- Air (subcategoria *air*) reprezintă atmosfera de afară. Are ca modele temperatura şi umiditatea relativă a aerului.
- Wind (subcategoria *wind*) reprezintă mişcarea masei de aer şi are ca modele viteza şi direcţia vântului.
- Conditions (subcategoria *conditions*) reprezintă condiţiile atmosferice. Are un model pentru presiunea barometrică şi altul pentru gradul de acoperire al norilor.

Configurare şi mod de folosire

Pentru a obţine starea locală a vremii folosind o sursă web va trebui mai întâi setată proprietatea de configurare “sourceURL”. Prima parte a URL-ului (până la ultimul “/”) reprezintă site-ul serviciului de meteorologie şi nu tebuie modificată. Ultima parte, constituită dintr-un cod, este însă specifică, în cazul de faţă, oraşului Cluj-Napoca. Această parte poate fi înlocuită cu oricare alta corespunzătoare unui alt oraş din România sau din afară.

Cel mai simplu mod de utilizare este acela de a adăuga condiţii unor acţiuni bazate pe starea vremii. De exemplu, dacă procentul de umiditate a aerului depăşeşte 90%, ceea ce înseamnă că sunt şanse mari de precipitaţii, se trimite un mesaj către proprietarul locuinţei care nu se află acasă pentru a-l înştiinţa despre eventualele probleme care pot să apară. Alte acţiuni mai complexe pot avea loc dacă se folosesc şi dispozitive de control precum senzorii. In acest caz, dacă temperatura înregistrată este sub un anumit prag, se porneşte automat dispozitivul de încălzire din locuinţă [8].

- **Pachetul News**

Acest pachet foloseşte o metodă alternativă de accesare a informaţiei disponibile pe internet, RSS sau „Really Simple Syndication (RSS 2.0)“. Astfel, în loc să se acceseze o sursă de informaţii de fiecare dată când e nevoie, prin intermediul unei subscrieri informaţia

este trimisă către utilizatorul aplicaţiei chiar în momentul publicării ei. Pentru funcţionarea acestui modul nu sunt necesare şi alte componente.

Configurare şi mod de folosire

In fereastra de configurare a pachetului trebuie completat în primul rând câmpul „sourceURL“ pentru a stabili sursa de unde sunt extrase informaţiile la un anumit interval de timp, specificat de asemenea în fereastra de configurare.

Informaţiile extrase de pe internet pot fi vizualizate în cadrul unei scene (din Consolă: View -> Scenes -> current rss), direct din consola aplicaţiei client. Există posibilitatea de a vizualiza informaţiile direct de pe telefonul mobil. In acest caz trebuie instalate şi configurare mai întâi pachetele Twitter şi Twitter SMS. La trimiterea unui mesaj de pe telefonul mobil cu textul „news“, utilizatorul va primi o notificare prin SMS cu ultimele noutăţi apărute pe site-ul introdus în etapa de configurare a pachetului.

- **Pachetul SMS**

Pachetul conţine un software care permite trimiterea de mesaje text (SMS) direct de pe telefonul mobil către aplicaţia HomeRun. Funcţionarea lui necesită instalarea şi configurarea în prealabil a pachetului Twitter.

Configurare şi mod de folosire

In fereastra de configurare a pachetului trebuie completat câmpul „sourceURL“ care reprezintă sursa de pe internet unde sunt publicate mesajele primite de pe telefonul mobil. Intr-un alt câmp trebuie completată frecvenţa cu care se verifică apariţia unui nou mesaj pe Twitter.

Pentru testarea funcţionalităţii modulului se instalează şi configurează pachetele News şi Weather. De asemenea, utilizatorul trebuie să aibă un cont Twitter. Utilizatorul trimite un mesaj cu textul „weather” către aplicaţia Twitter. Software-ul pachetului SMS verifică la anumite intervale de timp apariţia de mesaje noi. La primirea acestui mesaj va face legătura cu modulul Weather de unde preia datele meteo în starea actuală, după care face legătura cu modulul Twitter căruia îi transmite aceste date. Informaţiile vor fi publicate în contul Twitter al utilizatorului şi, de asemenea, prin intermediul serviciului de mesagerie pus la dispoziţie de Twitter, vor fi trimise sub formă de mesaj text pe telefonul utilizatorului.

- **Pachetul Solar**

Pachetul conţine mijloace folosite pentru a determina momentele la care răsare şi apune soarele. Funcţionarea acestui pachet nu necesită instalarea altor pachete suplimentare.

Conţinutul pachetului

Pachetul conţine metode prin care sunt create obiecte *Sun* din categoria *nature*, domeniu *place*. Sun are trei modele de tip „time“: *rise*, *set* şi *next rise* care conţin informaţii referitoare la timpul asociat apusului şi răsăritului pentru ziua curentă şi ziua următoare.

Configurare şi mod de folosire

Pentru ca să se poată determina cu exactitate momentul la care soarele răsare sau apune, HomeRun trebuie să cunoască poziţionarea exactă a locuinţei pentru care se foloseşte aplicaţia. In acest scop trebuie introduse latitudinea şi longitudinea corespunzătoare în fişierul de configurare al pachetului Solar. După instalare, se accesează fişierul de configurare (Setup -> Config -> Open). Se alege opţiunea „misc“ şi mai departe „solar“. Din arborele de explorare se alege settings -> location. In câmpurile text corespunzătoare se introduc

latitudinea şi longitudinea. Pentru valori negative se foloseşte semnul „-“. Aceste coordonate se pot obţine cu uşurinţă de pe internet pe baza unui cod poştal. Dacă informaţiile au fost introduse după pornirea serverului, acesta trebuie repornit pentru a lua în considerare noile coordonate.

Modelele *sunrise* şi *sunset* pot fi folosite în Action Builder ca şi condiţii în realizarea unor acţiuni, de exemplu când este atins momentul „sunset“, se aprind luminile pe verandă.

- **Pachetul System**

Pachetul este constituit dintr-un număr de tipuri şi obiecte pre-construite cu controale şi modele care sunt de folos şi care expun serviciile sistemului într-un domeniu sistem dedicat. Instalarea acestui pachet este recomandată. Funcţionarea lui nu necesită instalarea altor pachete suplimentare.

Conţinutul pachetului

In afara definiţiei domeniului, mai există şase categorii de obiecte de bază care se introduc: *activators, agents, writers, points, timers* şi *factories*.

Activators – unelte care conferă acces la serviciile sistemului. Dintre acestea câteva sunt Plan, Schedule şi Calendar. Plan are un control *run* care porneşte şi opreşte planurile de acţiune şi un model *plans* pentru a reprezenta toate planificările în desfăşurare. Schedule are un control *activate* care porneşte şi opreşte programările şi un model *schedules* pentru a reprezenta toate programările active la momentul respectiv. Calendar are un control *enable* care activează şi dezactivează evenimentele de tip calendaristic şi un model *caledar* pentru a reprezenta stările active. Folosirea acestor obiecte este singurul mod de a face planificări şi programări. Pentru aceasta trebuie instalat pachetul System.

Writers – unelte care permit înregistrarea activităţilor HomeRun. Următoarele obiecte sunt de acest tip: User Log, Action Tracer şi SceneCam. User Log are un control *log* care scrie mesaje în log-ul *user*. Action Tracer are un control *trace* care înregistrează datele detaliate de execuţie ale acţiunilor specificate de utilizator. Datele sunt scrise într-un fişier temporar care poate fi vizualizat cu ajutorul aplicaţiei Action Builder. Acest obiect este util pentru a urmări acţiunile ce s-au desfăşurat în lipsa utilizatorului. SceneCam are un control *capture* care păstrează valorile modelelor dintr-o scenă într-un fişier temporar de unde vor putea fi vizualizate mai târziu folosind aplicaţia Scene Viewer.

Points – diverse obiecte ale căror modele lucrează cu puncte de timp şi date calendaristice. Sunt două astfel de tipuri de obiecte: *clock* şi *calendar*. Obiectul Clock are un singur model, *timeOfDay,* pentru a reprezenta momentul curent (timp nu dată). Obiectele de tip Calendar sunt *day* (cu modelele *dayOfWeek* şi *dayOfMonth*), *month* (cu modelul *monthOfYear*) şi *date* (cu modelul *date*).

Timers – unelte pentru măsurarea şi gestionarea timpului.

Factories – această categorie conţine tipuri de obiecte pentru crearea de *factories,* care sunt obiecte capabile să creeze alte obiecte. Ele pot reduce semnificativ efortul care ar fi necesar altfel (în Object Editor) pentru a crea obiecte cu valoare temporară.

Agents – sunt obiecte care au rolul de a automatiza anumite sarcini repetitive care se execută în HomeRun.

Configurare şi mod de folosire

Pachetul System nu necesită nici un fel de configurare. Folosirea pachetului constă doar în utilizarea de controale şi modele asociate obiectelor, în cadrul definirii acţiunilor din Action Builder [8].

4.3.4 Proiectarea modulului SMS

Diagrama de clase pentru fişierul sursă SMS este reprezentată într-o formă extinsă la secţiunea Anexe, figura 1.

Pachetul sms

Report – este un container pentru datele colectate din sursa twitter introdusă ca sourceURL în etapa de configurare a pachetului.

SMSService – descrie serviciile oferite de sistemul twitter.

Observation – descrie un set de valori extrase din fişierul twitter.

Source – este interfaţa implementată de furnizorii datelor extrase.

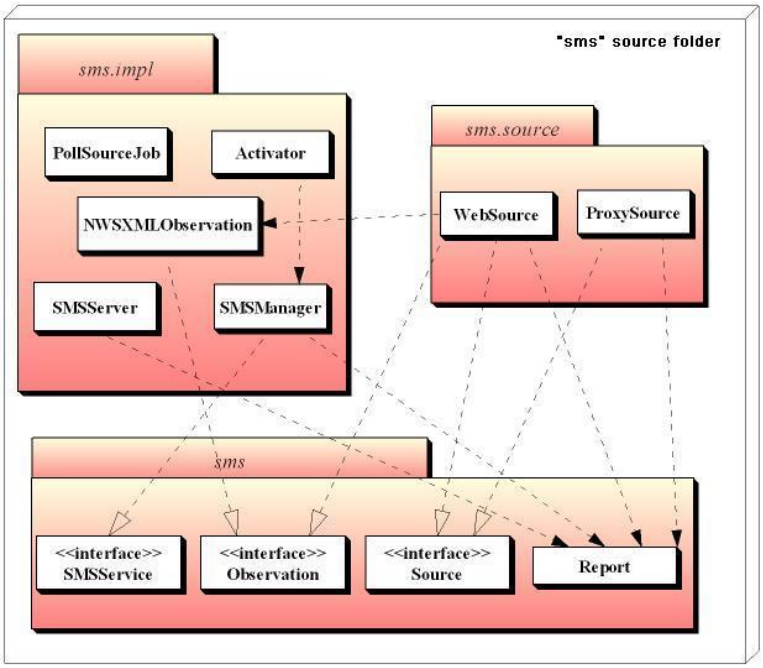

Figura 4.15 Diagrama de clase pentru fişierul sursă SMS

Pachetul sms.source

WebSource – obţine un raport cu mesajele primite extrase din sursa introdusă în etapa de configurare.

ProxySource – este implementarea unei surse care rulează local şi preia datele de la un SMSServer remote. La un anumit interval de timp deschide un socket către serverul remote, preia datele şi le stochează în SMSManager.

Pachetul sms.impl

PollSourceJob, Activator

NWSXMLObservation – încapsulează codificarea specifică a unei înregistrări folosită de către SMSService.

SMSServer – container pentru SMSSource care ii permite să opereze remote faţă de server şi să comunice cu serverul prin SourceProxy.

SMSManager – adună informaţii din sursa introdusă în etapa de configurare.

Pentru ca pachetul SMS să poată funcţiona, este nevoie să se instaleze în prealabil pachetul Twitter a cărui diagramă de clase este prezentată în figura 4.16.

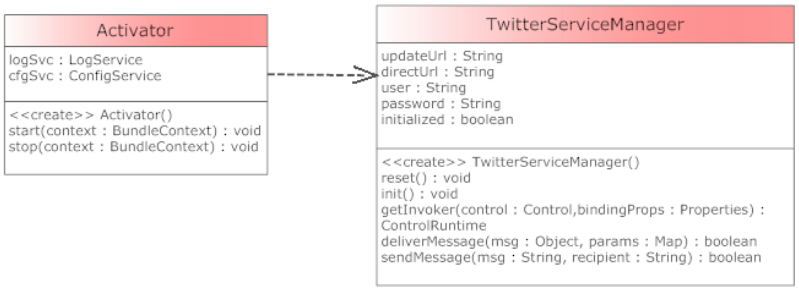

Figura 4.16 Diagrama de clase pentru pachetul Twitter

Clasa TwitterServiceManager asigură accesul la sistemul de mesagerie Twitter şi permite trimiterea de mesaje directe către utilizatorii Twitter sau actualizarea statusului.

4.4 Implementarea aplicaţiei

4.4.1 Aplicaţia client

Comanda de instalare a pachetelor este dată din interfaţa client care va iniţializa cu metoda addActionListener modulul selectat. Comanda se găseşte în fişierul setup.java din pachetul com.monad.homerun.bootapp:

instButton.addActionListener(this);

La instalarea fiecărui pachet se verifică dacă există pachetele de care depinde execuţia pachetului selectat şi care nu sunt iniţializate. In cazul în care sunt depistate astfel de pachete neiniţializate, este afişat un mesaj de eroare.

msgLabel.setText(“Installation failed: “+ status)

unde status reprezintă cauza pentru care instalarea a eşuat. Spre exemplu, dacă un pachet este necesar a fi preinstalat, va fi generat mesajul: ‘Installation failed: missing requirement’ com.monad.homerun.pkg.<nume pachet>. Acest lucru se petrece în fişierul PackageInstaller.java din pachetul com.monad.homerun.admin.impl.

ab.error = “missing requirement’” + req + “’”;

Pachetele de care depinde execuţia altuia sunt amintite în fişierul build.xml corespunzător pachetului respectiv:

<attribute name = "Homerun-Require"
value = "${base.pkg}.pkg.twitter;version="${homerun.version}"" />

4.4.2 Aplicaţia server

Odată lansat în execuţie modulul Twitter SMS, acesta se va conecta la sursa twitter specificată în fişierul de configurare sms.xml.

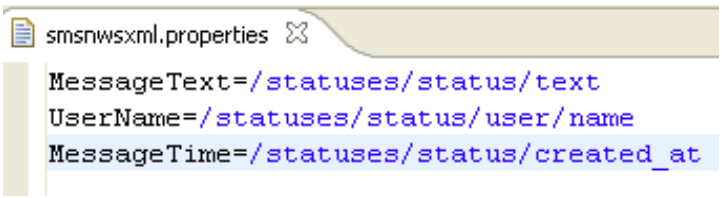

Figura 4.17 Fişierul smsnwsxml.properties

Fişierul http://twitter.com/statuses/user_timeline/... .xml va fi parcurs şi din el vor fi extrase tag-urile marcate în fişierul smsnwsxml.properties care specifică structura fişierului XML ce va fi interogat. Pentru ca aplicaţia să funcţioneze, trebuie să existe o legătură internet activă.

sms.xml

```
<?xml version="1.0"?>
<smsConf>
  <channels>
    <channel name="primary">
      <source type="pick" limiters="/sources" desc="SMS source">web</source>
    </channel>
    <channel name="alternate">
      <source type="pick" limiters="/sources" desc="SMS source">web</source>
    </channel>
  </channels>
  <sources>
    <source name="web" enabled="true">
      <class type="string" level="2" desc="Source class">com.monad.homerun.pkg.sms.source.WebSource</class>
      <tag type="string" desc="Display Tag">NWS via Web</tag>
      <!-- URL for sms: needs to be configured for each installation -->
      <sourceURL type="string" desc="URL for sms">http://twitter.com/statuses/user_timeline/124827426.xml</sourceURL>
      <dataFormat type="string" desc="Data Format">SMS NWS XML</dataFormat>
      <updateMins type="number" desc="Update frequency (mins)">1</updateMins>
    </source>
  </sources>
</smsConf>
```

Figura 4.18 Fişierul de configurare sms.xml

In terminologia XML, fişierul smsnwsxml.properties conţine definiţia XPath a tag-urilor ce urmează a fi interogate.

Toate aceste fişiere de configurare plus fişierele ce compun aplicaţia în sine vor fi împachetate în fişierul sms-0.4.1.jar după rularea Ant a fişierului build.xml din directorul <app-root>/server/packages/sms.

4.4.3 Mecanismul OSGi

Lansarea pachetului Twitter SMS este realizată din fişierul Activator.java care conţine metodele start şi stop [7]. Activator.java are rolul de a iniţializa câteva servicii, după cum urmează:

```
logSvc = (LogService)bc.getService(ref);
cfgSvc = (ConfigService)bc.getService(ref);
timingSvc = (TimingService)bc.getService(ref);
modelSvc = (ModelService)bc.getService(ref);
```

Metoda getService este de tip ServiceReference şi face parte din pachetul OSGi org.osgi.framework.BundleContext. Referinţa ref pe care metoda o primeşte ca parametru este obţinută de la serviciile iniţializate prin <NumeServiciu>.class.getName().

Aceste servicii vor fi folosite de aplicaţia Twitter SMS pentru a executa diverse acţiuni precum înregistrarea unor activităţi (logging), configurarea unor setări de sistem, temporizarea şi sincronizarea cu alte servicii şi interacţiunea cu alte obiecte din sistem.

Pachetele OSGi folosite de Activator.java sunt declarate la început după cum urmează:

```
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
```

De altfel, clasa Activator este singura clasă din pachetul SMS care consumă şi este direct dependent de serviciile OSGi. Toate celelalte clase sau interfeţe precum: SMSManager, SMSService, SMSServer fac apel indirect la cadrul de lucru OSGi, acesta având doar rolul de a gestiona serviciile LogService, ConfigService, TimingService şi ModelService iniţializate.

Lansarea în execuţie a aplicaţiei Twitter SMS se face prin apelarea constructorului ce iniţializează clasa SMSManager.

```
//register our service
SMSManager mgr = new SMSManager();
//start him up
mgr.init(true);
smsSvc = mgr;
```

După lansarea clasei SMSManager este apelată funcţia de construcţie a raportului ce urmează a fi afişat. Acest lucru se realizează prin comanda : curReport = source.reportSMS(); din fişierul SMSManager.java. reportSMS este o metodă a clasei WebSource care apelează la rândul ei clasa SMSXmlObservation ce extrage fizic informaţiile necesare din sursa web menţionată la configurare. SMSXmlObservation foloseşte la rândul său nişte pachete speciale pentru interpretarea fişierelor XML. Acestea sunt importate la început:

```
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
```

Capitolul 5 UTILIZAREA APLICATIEI

5.1 Cerinţe hardware şi software

HomeRun, ca şi platformă, are cerinţe minimale de operare dar pachetele individuale au şi ele cerinţe specifice care trebuie avute în vedere şi îndeplinite înainte de instalarea lor. Platforma presupune totuşi un mediu ideal care să permită folosirea aplicaţiei HomeRun la capacite maximă. Acest mediu constă într-un calculator pe care să ruleze software-ul dedicat server-ului, care să fie conectat la o reţea (reţea locală sau internet, prin intermediul unui modem sau router). Internetul dial-up nu este acceptat. Orice calculator care rulează aplicaţia HomeRun (client sau server) trebuie să aibă instalat Java Runtime Environment (JRE), cel puţin versiunea 5.0. Java este acceptat pe sistemele de operare moderne, de la Windows (Windows 98 ediţia a 2-a, până la Vista), Mac OS (distribuit de Apple), Linux şi până la Sun Solaris 10.

Hardware

Partea de server este foarte modestă în ceea ce priveşte cerinţele hardware: aproape orice calculator fabricat în ultimii cinci ani poate să ruleze această aplicaţie, având o unitate centrală de procesare care satisface nevoile, memorie suficientă etc. Memoria ocupată de aplicaţia server depinde de numărul de pachete instalate. In ceea ce priveşte interfaţa hardware a calculatorului, este nevoie doar de o conexiune IP la reţea. Unele dispozitive de automatizare, precum componentele Insteon şi X10, pot impune cerinţe adiţionale, cea mai comună fiind existenţa unui port serial. Dispozitivele USB sunt capabile să emuleze interfeţe seriale pentru majoritatea sistemelor de operare.

Software

HomeRun nu necesită nici un alt soft în afară de Java. Aplicaţia conţine în cadrul pachetelor sale toate componentele necesare pentru a rula, incluzând şi o distribuţie Knopflerfish OSGi framework, versiunea 4.0.0. Acest lucru este valabil pentru toate sistemele de operare. O excepţie face portul serial care are drivere specifice sistemelor de operare Windows, Mac şi Linux.

5.2 Utilizare

5.2.1 Alegerea şi instalarea unui server

Serverul trebuie să ruleze pe un singur calculator în reţeaua locală şi să fie conectat la reţea în permanenţă. Pentru o accesare la distanţă este de dorit ca serverul să fie conectat şi la internet. Un alt considerent în alegerea calculatorului îl constituie prezenţa echipamentelor fizice necesare pentru funcţionarea serverului. In general este vorba de un port serial pe care unele calculatoare precum laptop-urile recente sau macintosh-urile nu îl au. Un ultim considerent este consumul de energie. Având în vedere că serverul va rula 24 de ore din 24, este de dorit ca acesta să nu fie unul cu consum mare de energie. HomeRun nu necesită servere puternice, iar calculatoarele economice se pretează foarte bine pentru această sarcină.

După ce s-a ales calculatorul pe care va rula serverul, se copiază şi se dezarhivează aplicaţia Server. Din aplicaţie se alege directorul *fwork*, după care se face dublu click pe hrserver.bat. Va apărea o fereastră terminal cu următorea informaţie: HomeRun bootstrap on port: 8070.

5.2.2 Instalarea clientului

Spre deosebire de aplicaţia server, clientul poate să ruleze pe orice calculator din reţea. Acesta nu necesită un calculator dedicat care să fie pornit tot timpul. Aplicaţia client poate să ruleze doar atât timp cât este nevoie pentru a configura sistemul. Se pot instala mai multe aplicaţii client pe mai multe calculatoare, însă numai unul este necesar pentru configurarea sistemului. Instalarea se face prin dezarhivarea fişierului client.zip într-un director separat faţă de cel al aplicaţiei server. Pentru sistemele Windows lansarea în execuţie aplicaţiei client se realizează cu ajutorul comenzii hrclient.bat. Dacă programul este lansat cu succes va aparea o fereastră asemănătoarele celei din figura 5.1.

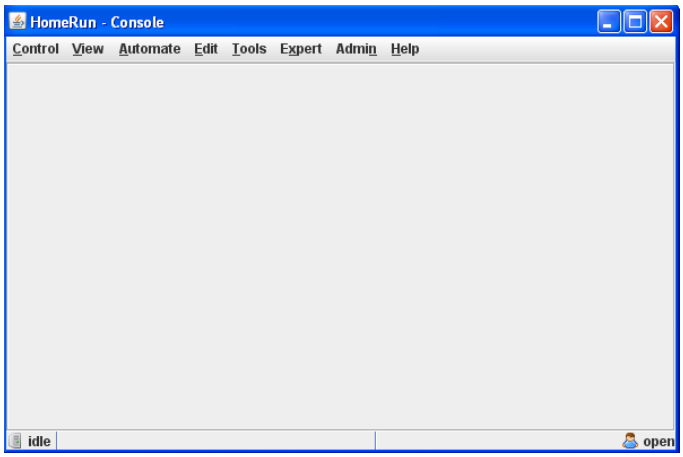

Figura 5.1 Consola aplicaţiei client

Pasul următor constă în crearea unei legături între aplicaţiile client şi server. Pentru aceasta, din consolă se selectează opţiunea Admin -> Server -> Find. In fereastra care apare trebuie introdus IP-ul calculatorului pe care este instalată aplicaţia server. Dacă IP-ul este corect şi a fost găsit, serverul este pornit şi se poate trece în continuare la configurarea sistemului HomeRun.

5.2.3 Configurarea sistemului HomeRun

După instalarea aplicaţiilor server şi client urmează configurarea acestora. Acest lucru se realizează prin instalarea unor pachete independente, fiecare cu o funcţionalitate distinctă. Pachetele împreună cu o scurtă descriere a lor pot fi vizualizate din consolă (Admin -> Setup). In Setup se alege din meniu comanda Package ->List.

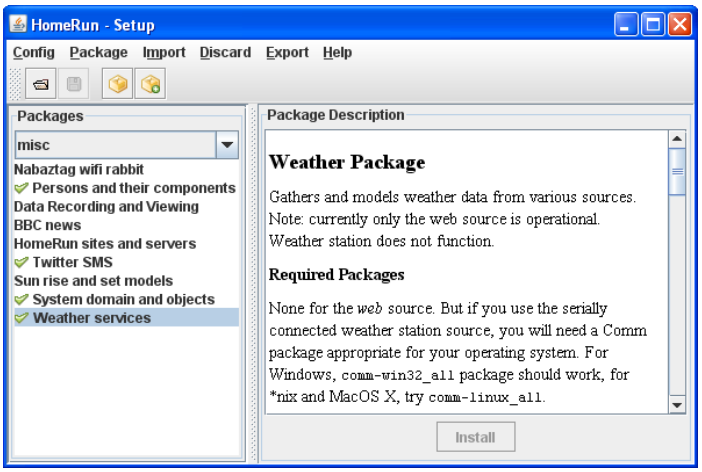

Figura 5.2 Fereastra setup

Pentru instalare se selectează pachetul dorit şi se apasă butonul *Install* aflat în partea de jos a ferestrei, sub specificaţia pachetului selectat. In unele cazuri se cere configurarea la instalare a pachetului respectiv. In acest caz se completează câmpurile text şi se apasă butonul *Done*. După cum se observă în figura 5.2, toate pachetele care au fost deja instalate sunt marcate cu o bifă de culoare verde.

Când este instalată prima data, aplicaţia HomeRun este considerată a fi în stare deschisă, ceea ce înseamnă că nu are noţiunea de utilizatori distincţi de sistem, aşa că nu va cere date de identificare sau de autentificare. In consecinţă, orice persoană care utilizează software-ul poate face orice. In cazul în care toţi utilizatorii sunt cunoscuţi şi de încredere, e mai simplu să rămână deschis întotdeauna pentru simplitate. Cu toate acestea,dacă se doreşte atribuirea de drepturi anumitor persoane, HomeRun suportă un model bazat pe autoritate. In momentul în care se defineşte un singur utilizator (din Consola: Admin -> User -> Manage), atunci sistemul este considerat a fi în stare închisă si utilizatorii trebuie să se identifice şi să se autentifice ei înşişi ca să aibă acces la sistem. Acest lucru, de asemenea, creeaza un potenţial risc; ce se întâmplă în cazul în care primului utilizator creat îi lipseşte autoritatea administrativă? Atunci cum se mai creează administratorul? Pentru a evita acest lucru, HomeRun insistă asupra faptului ca primul utilizator definit să fie un administrator, pentru ca el să poată defini apoi utilizatorii cu drepturi mai puţine. Procesul poate fi redat şi în sens invers: trebuie să existe siguranţa că dacă toţi utilizatorii sunt şterşi pentru a reveni la starea deschisă, ultimul utilizator rămas trebuie să fie un administrator.

5.2.4 Exemplu de utilizare a modulului SMS

Funcţionalitatea modulului „Twitter SMS” (sau SMS) constă în posibilitatea de a controla prin intermediul telefonului mobil alte câteva module, cum ar fi „BBC News”, „Twitter” sau „Weather”. Pentru ca utilizatorul să se bucure de funcţionalitatea acestui modul, trebuie mai întâi să îndeplinească anumite cerinţe. Printre acestea se numără existenţa unui cont Twitter şi a unei cartele Vodafone. Aplicaţia HomeRun beneficiază de asemenea de un cont Twitter.

Twitter este un site web care permite utilizatorilor scrierea şi transmiterea de mesaje de maxim 140 de caractere. Este uneori descris ca fiind „SMS-ul Internetului”. Twitter dispune şi de un serviciu de mesagerie pe care clienţii îl pot folosi dacă deţin o cartelă Vodafone. Astfel, clienţii pot să posteze mesajele lor pe Twitter prin intermediul SMS-urilor. De asemenea ei pot opta să primească notificări pe telefon în legătură cu mesajele primite, tot prin intermediul SMS-urilor.

Modulul „Twitter SMS” urmăreşte să facă posibilă comunicarea dintre aplicaţia HomeRun şi utilizator prin intermediul telefonului mobil. De exemplu, utilizatorul trimite un mesaj cu textul „weather” către Twitter. Mesajul va fi transmis şi către contul aplicaţiei HomeRun care urmăreşte contul clientului. Modulul „Twitter SMS” are sarcina de a verifica la anumite intervale de timp modificările care apar în contul Twitter al aplicaţiei. La sesizarea mesajului „weather” se face legătura cu modulul „Weather”, prin intermediul unui serviciu, pentru a prelua de aici datele meteo (temperatură, umiditate, viteza şi direcţia în care bate vântul) asociate locaţiei stabilite în configuraţia aplicaţiei HomeRun (de obicei este vorba de locaţia domiciliului utilizatorului). Un alt serviciu va fi folosit pentru transmiterea datelor meteo spre modulul „Twitter” care le va trimite mai apoi, sub forma unui mesaj, către contul Twitter al aplicaţiei. Conform configurărilor din Twitter, utilizatorul urmăreşte mesajele pe care le postează aplicaţia HomeRun şi la fiecare mesaj nou postat de aceasta primeşte notificare printr-un SMS pe telefonul mobil. Astfel, după trimiterea de către utilizator a unui mesaj cu textul „weather”, el va primi înapoi prin SMS o notificare privind starea vremii la locaţia domiciliului său. De multe ori acest lucru este util când utilizatorul este plecat pentru mai multă vreme şi doreşte să monitorizeze condiţiile meteo.

Un exemplu asemănător este cel în care utilizatorul trimite către Twitter un mesaj cu textul „news” şi primeşte înapoi o notificare prin SMS cu ultima ştire apărută pe un anumit site pe care l-a precizat în prealabil la configurarea modulului „BBC News”.

Pentru a putea instala modulul „Twitter SMS”, trebuie instalat mai întâi modulul „Twitter”, după care, pentru a ne putea bucura de funcţionalităţile lui, se instalează şi modulele „BBC News” şi „Weather”.

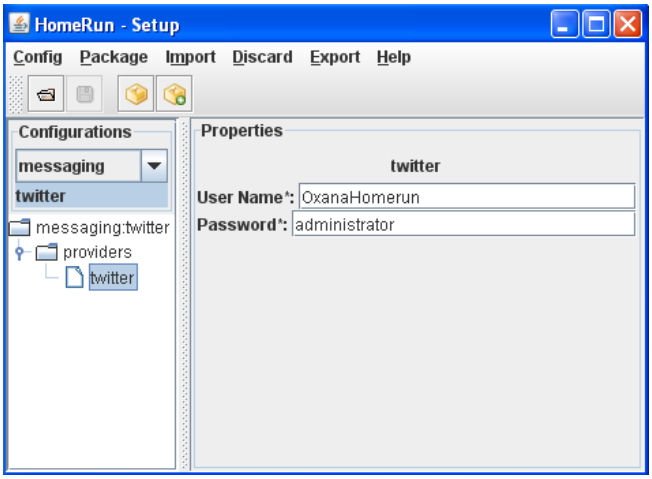

Figura 5.3 Configurarea modulului „Twitter”

Modulele „Twitter” şi „SMS” se configurează conform figurilor 5.3 şi 5.4. Din fereastra Setup se alege opţiunea Config -> Open.. şi se introduc proprietăţile specifice fiecărui modul. O proprietate a modulului „Twitter SMS” este frecvenţa cu care se verifică apariţia unui nou mesaj pe Twitter şi a fost setată la 1 minut.

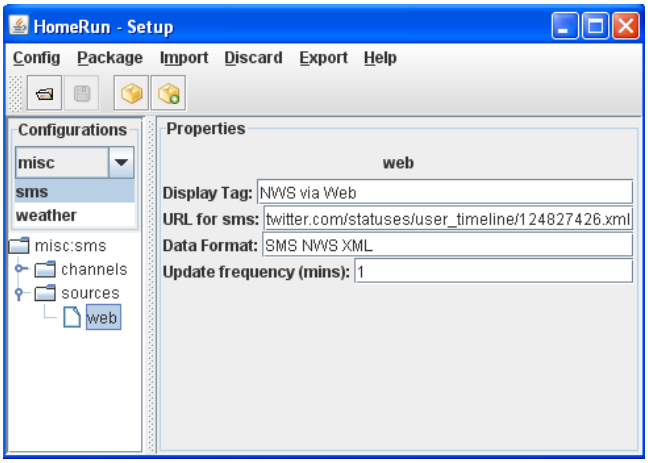

Figura 5.4 Configurarea modulului „SMS”

Trimiterea unui SMS constituie o acţiune şi după cum aminteam într-unul din capitolele anterioare, o acţiune în forma ei cea mai simplă constă într-o listă de sarcini (minim una) asociate unui obiect. Acţiunea se execută în momentul în care o anumită condiţia specificată este îndeplinită. In cazul de faţă obiectul asupra căruia se aplică acţiunea este un obiect de tip Person, iar condiţia este primirea unui mesaj cu textul „weather“ de la persoana respectivă.

Mai întâi trebuie creat un obiect de tip Person. Se deschide fereastra Object Editor (din Consolă: Edit -> Object). In partea stângă a ferestrei există o listă cu tipurile de obiecte care pot fi create. De această dată voi crea un obiect de tip Person (person -> people -> single). Se alege opţiunea File -> New... şi se creează un obiect numit „Oxana“. După ce obiectul a fost salvat (File -> Save) i se atribuie un control (Edit -> Add Control..). Din lista

de controale se selectează „tweet“. După realizarea acestor paşi, fereastra Object Editor trebuie să arate conform figurii 5.5.

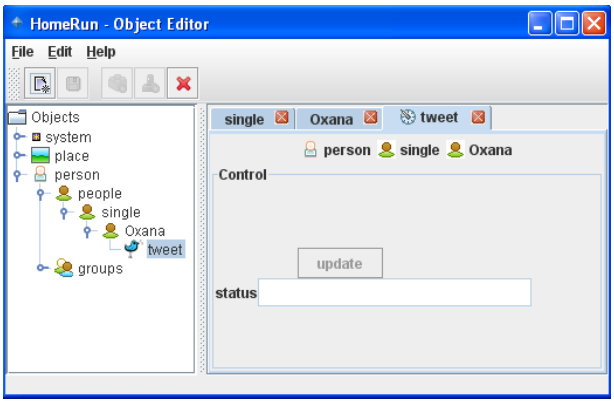

Figura 5.5 Fereastra Object Editor

Acum nu ne rămâne decât să trimitem un SMS cu textul „weather“ către Twitter. In figura 5.6 este ilustrat acest lucru. Pe contul de Twitter al utilizatorului OxanaHotea a fost trimis un mesaj text (SMS) cu textul „weather“. Mesajul a fost transmis mai departe către contul aplicaţiei Homerun, OxanaHomerun. După câteva minute apare şi răspunsul din partea aplicaţiei. Este vorba de starea vremii la momentul respectiv în Cluj-Napoca [4]. In momentul primirii răspunsului, Twitter trimite o notificare prin SMS utilizatorului OxanaHotea cu textul mesajului primit de la OxanaHomerun.

Figura 5.6 Mesaje primite pe Twitter

Capitolul 6 TESTAREA SI EVALUAREA PERFORMANTELOR APLICATIEI

6.1 Testarea aplicaţiei

HomeRun intenţionează să fie uşor de utilizat şi gestionat, de aceea se furnizează câteva unelte pentru a ajuta utilizatorii în administrarea sistemului.

Ciclul de viaţă al serverului

După cum a fost menţionat şi în capitolele precedente, aplicaţia HomeRun este împărţită în componentele client şi server care comunică între ele. Serverul are un ciclu de viaţă intern, în sensul că poate trece prin mai multe stări pe care utilizatorul le poate controla. La pornirea serverului prin rularea fişierul batch hrserver.bat (fişier care conţine o serie de comenzi ce se execută de către interpretorul de comenzi), acesta nu rulează ci se află într-o stare „ready" cunoscută şi sub numele de „bootstrap state" (serverul este încărcat dar încă nu rulează). Cât timp serverul se află în această stare, aproape toate opţiunile din consola client sunt inactive, excepţie făcând opţiunile din grupul Admin. Una din aceste opţiuni, Server -> Start oferă posibilitatea de a aduce serverul la starea lui pe deplin operaţională. In majoritatea timpului va rămâne în această stare şi, în mod normal, utilizatorul nu va fi nevoit să îl oprească. Există totuşi câteva cazuri, de exemplu după instalarea unui nou pachet, când utilizatorului i se cere să repornească serverul (adică să îl oprească şi să îl pornească din nou) pentru a fi încărcate noile configurări.

Una dintre obligaţiile administrative majore ale utilizatorului este selectarea şi instalarea pachetelor HomeRun care determină funcţionarea sistemului. Majoritatea pachetelor necesită mai departe configurare, care se face accesând fereastra de Setup.

Sistemul de logging

Java Logging API facilitează aplicaţiile bazate pe servicii şi întreţinere pe partea de utilizator prin generarea de rapoarte pentru analiză destinate utilizatorilor finali, administratorilor de sistem şi pentru echipele de dezvoltare. Logging API captează informaţiile, cum ar fi căderi ale securităţii, erori de configurare, congestii şi/sau erori în aplicaţie sau platformă şi le expune utilizatorului în diverse forme [9].

Aplicaţia HomeRun include suport pentru distribuţia de rapoarte sub formă de text simplu sau format XML în memorie şi consolă pentru o interfaţă directă cu utilizatorul. API-ul Java Logging este capabil de interacţiune cu serviciile de efectuare a rapoartelor existente în aplicaţia gazdă.

Intrucât sistemul a fost implementat pentru a primi cereri de la mai multe aplicaţii client în mod concurent, a fost creat şi un sistem de logging pentru a facilita „diagnosticarea" rapidă a problemelor care apar la runtime. Acest sistem foloseşte pachetul Java.util.logging care furnizează un mecanism sigur bazat pe thread-uri pentru a păstra şi menţine înregistrările. Aceste înregistrări detaliază stările componentelor din arhitectura aplicaţiei HomeRun şi interacţiunile dintre ele. Rolul sistemului de logging este şi acela de a determina cu uşurinţă sursa erorilor, mai ales pentru o aplicaţie cu un grad ridicat de dezvoltare cum este HomeRun.

Pachetul Java.util.logging foloseşte handler-e pentru a comunica cu mediul extern. Diferitele tipuri de handler-e permit transmiterea log-urilor sub diverse forme: un handler de tip fişier crează un fişier căruia îi atribuie log-uri, un handler de tip socket trimite log-urile printr-o conexiune remote folosind un socket TCP. Ambele tipuri de handler-e au fost folosite în aplicaţia HomeRun pentru a permite utilizatorilor să vizualizeze log-urile din propria lor sesiune de evaluare.

Sistemul de logare este utilizat ca mecanism de raportare a excepţiilor aruncate. Cum aplicaţiile client rulează remote, un mecanism de raportare a excepţiilor de tip consolă ar fi insuficient, mai ales că mai mulţi utilizatori folosesc în mod concurent sistemul la un moment dat. Din acest motiv, sesiunile individuale sunt separate şi log-urile sunt introduse într-un sistem de fişiere, fiind indexate în funcţie de dată şi oră, separat pentru fiecare sesiune particulară, pentru a putea fi găsite mai uşor.

O altă proprietate importantă este aceea că log-urile sunt împărţite în funcţie de nivelul de severitate a informaţiei pe care o descriu (warning-uri, informaţii, erori etc.). Aceasta înseamnă că mesajele care nu prezintă prea mare interes pot fi cu uşurinţă filtrate pentru a vedea, de exemplu, doar problemele grave care apar.

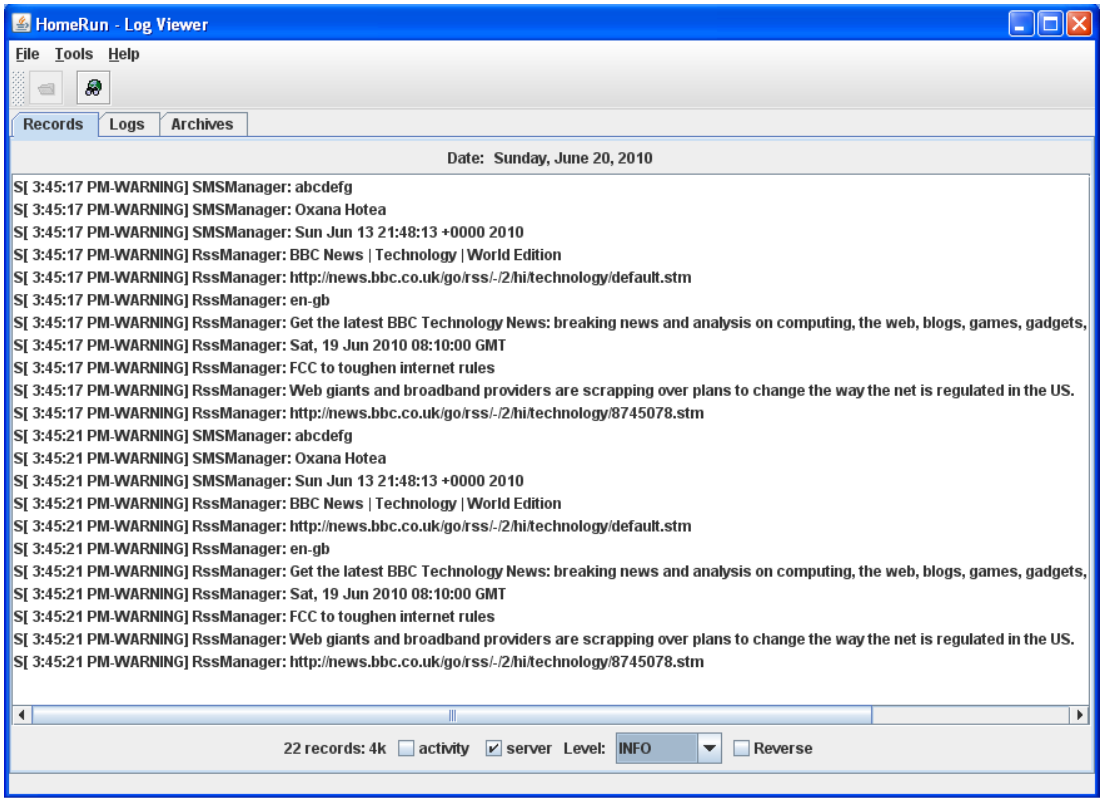

Figura 6.1 Fereastra Log Viewer a aplicaţiei HomeRun

HomeRun are un sistem flexibil de logging şi furnizează o unealtă, Log Viewer, pentru gestionarea log-urilor. In mod normal se păstrează trei tipuri de log-uri: log-uri de sistem care conţin date cu privire la operaţiile interne ale serverului, log-uri de activitate care păstrează înregistrări cu privire la acţiunile care se desfăşoară, evenimentele care apar etc. Ultimul tip îl constituie log-urile de tip user care înregistrează datele pe care utilizatorul le foloseşte în acţiunile sale.

Fereastra Log Viewer are trei tab-uri: unul pentru afişarea fişierelor log (unul din fiecare tip pentru fiecare zi), unul pentru înregistrările log şi unul pentru afişarea log-urilor arhivate. In fereastră se găseşte şi opţiunea de creare a arhivelor (care comprimă log-urile şi le salvează într-un spaţiu pe disc) şi o altă opţiune de căutare a log-urilor după anumite cuvinte cheie.

6.2 Evaluarea performanţelor aplicaţiei

Evaluarea aplicaţiei se poate face din mai multe perspective: fie o evaluare în comparaţie cu alte aplicaţii din aceeaşi clasă care folosesc aceeaşi tehnologie sau în comparaţie cu aplicaţii din aceeaşi clasă (în cazul de faţă, aplicaţii smart home), dar care folosesc altă tehnologie de dezvoltare.

Am amintit în partea de introducere două aplicaţii smart home care folosesc tehnologia OSGi. Este vorba de sistemul BSH al celor de la Bosh şi Siemens sau produsul Philips iPronto de la Philips. Din păcate, există puţine avantaje pe care HomeRun le are în faţa acestor aplicaţii. Unul din ele este lipsa unei licenţe de folosire care, în cazul celor două aplicaţii, poate depăşi chiar şi 2000$. Un alt avantaj de care se bucură aplicaţia HomeRun este stabilitatea (când codul sursă este public, el este testat de o multitudine de utilizatori, problemele sunt detectate rapid şi rezolvate la fel de repede, fără a fi ascunse în produs).

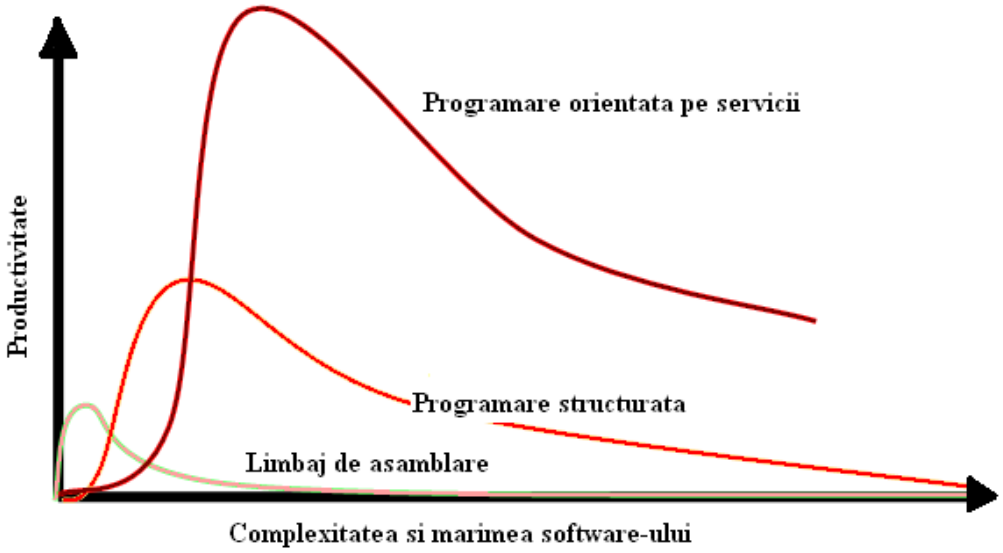

Figura 6.2 Nivelul productivităţii în funcţie de tehnica de programare folosită

Trecerea la modelul OSGi aduce adesea mari îmbunătăţiri în procesul de dezvoltare a soft-ului. Din studiile statistice rezultă faptul că productivitatea în programare este rezultatul unei combinaţii de factori precum complexitatea, mărimea aplicaţiei şi tehnica de programare folosită. Astfel, nu este de mirare că odată cu creşterea complexităţii şi mărimii soft-ului, scade şi productivitatea. In figura 6.2 se poate observa cum programarea orientată pe servicii, care se află la baza tehnologiei OSGi, oferă productivitate crescută faţă de alte tehnici mai vechi de programare cum sunt programarea structurată sau programarea în limbaj de asamblare. Productivitatea ridicată a aplicaţiilor realizate cu tehnologia OSGi se datorează şi faptului că OSGi se integrează perfect cu sistemul de operare, maşina virtuală Java, librăriile aplicaţiei şi alte librării preexistente.

O altă perspectivă de evaluare ar fi să se facă o comparaţie între o aplicaţie tradiţională smart home şi una realizată cu tehnologia OSGi. O arhitectură de acest tip a fost prezentată în primul capitol al lucrării. O aplicaţie smart home tradiţională este de obicei bazată pe o arhitectură centralizată. Dispozitivele din locuinţă sunt conectate prin intermediul unei reţele interne (Home Network) şi controlate de către Home Gateway, care este platforma de furnizare a serviciilor pentru locatari. In cazul în care apar probleme la nivel de Gateway, întreg sistemul este compromis.

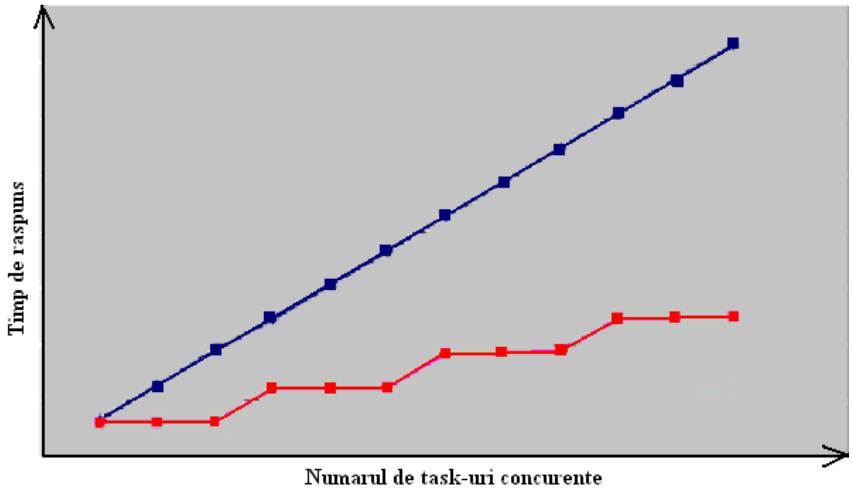

Figura 6.3 Evaluarea comparativă a performanţei între o aplicaţie smart home tradiţională şi una dezvoltată cu tehnologia OSGi

In figura 6.3 se face comparaţie între timpul de răspuns în funcţie de numărul de sarcini (task-uri) executate concurent în cazul unei aplicaţii tradiţionale cu arhitectura centralizată în jurul gateway-ului (culoarea albastră) şi o aplicaţie dezvoltată cu tehnologia OSGi, având o arhitectură modulară (culoarea roşie). Se poate observa că în cazul arhitecturii modulare, timpul de răspuns este considerabil mai scazut odată cu creşterea numărului de task-uri concurente.

Intr-o oarecare măsură, performanţa aplicaţiei HomeRun este datorată şi tehnologiei client – server pe care o foloseşte. Beneficiile pe care această tehnologie le aduce sunt următoarele: asigură suportul pentru noi componente, permite suportul noilor tehnologii, mai ales cele orientate pe obiecte.

Capitolul 7 CONCLUZII SI DEZVOLTARI ULTERIOARE

7.1 Concluzii

In urma revizuirii literaturii de specialitate, fie că a fost vorba de cărţi, lucrări de cercetare sau diverse pagini web scrise de către studenţii care abordau aceeaşi tehnologie, au început să prindă contur ideile despre ceea ce reprezintă tehnologia OSGi, care este structura ei, cum funcţionează şi, cel mai important lucru, care sunt avantajele pe care le oferă în procesul de dezvoltare software.

In Java, soluţia clasică pentru modularitate (plug-in ability) este OSGi care reprezintă standardul incontestabil pentru acest domeniu. OSGi permite crearea de bundle-uri (care sunt de fapt jar-uri cu un manifest bine definit) care reprezintă plugin-urile ce se pot instala în mod dinamic. Tot în mod dinamic în runtime se rezolvă dependenţele unui bundle de alte bundle-uri.

Desigur că au existat, ca pentru orice tehnologie, argumente pro şi contra în vederea utilizării ei. De exemplu, punctele forme sunt modularitatea, simplitatea, reutilizabilitatea etc. Pe de altă parte, dacă un modul nu este declarat explicit în fişierul său manifest ca fi exportat, nu poate fi văzut şi folosit de către alte module.

După studiul domeniului, următorul pas l-a constituit alegerea unei aplicaţii care să se poată bucura de avantajele oferite de tehnologia OSGi: modularitate, reutilizabilitate, actualizare dinamică etc. Domeniul de aplicabilitate al tehnologiei este foarte vast dar m-am oprit la aplicaţiile „smart home”, din dorinţa dezvoltării unui sistem de automatizare a locuinţelor, sau cel puţin a unui sistem care să vină în ajutorul oamenilor. HomeRun este un astfel de sistem, distribuit gratuit. Fiind open source, el a putut fi dezvoltat prin adăugarea de module cu noi funcţionalităţi, de sine stătătoare sau care folosesc avantajele altor module. Este vorba aici de modulele SMS şi News care au fost prezentate în cadrul lucrării.

Aplicaţia HomeRun prezintă, de asemenea, avantaje şi dezavantaje. Avantajul major îl constituie faptul că, fiind o aplicaţie open source, nu există costuri de licenţiere şi, de asemenea. este o aplicaţie stabilă. Când codul sursă este public, el este testat de o multitudine de utilizatori, iar eventualele probleme apărute sunt repede soluţionate. Pentru ca utilizatorul să se bucure de toate funcţionalităţile aplicaţiei HomeRun, este nevoie de câteva echipamente tehnice, care nu sunt la îndemâna tuturor. Pentru a compensa aceste lipsuri, am dezvoltat nişte module care să nu necesite existenţa unor echipamente suplimentare, ci doar o conexiune la internet.

Modulul SMS aduce un mare avantaj aplicaţiei, întrucât permite controlul la distanţă al câtorva module, ceea ce nu s-a mai realizat până în prezent. Pentru comunicarea dintre telefonul mobil şi calculator am folosit serviciul de mesagerie Twitter deoarece este printre puţinele servicii disponibile în România şi cu un cost relativ redus. Au apărut în schimb probleme uneori la trimiterea mesajelor, serviciul fiind foarte încărcat.

Pe tot parcursul dezvoltării aplicaţiei şi lucrării de prezentare am încercat să urmăresc obiectivele pe care de la bun început le-am conturat. Am reuşit astfel să acumulez noi cunoştinţe în domeniul dezvoltării de software-uri, să abordez şi să aprofundez noi tehnologii cum este OSGi. In ceea ce priveşte obiectivele practice, am reuşit să extind funcţionalităţile aplicaţiei HomeRun prin adăugarea unor module noi.

7.2 Dezvoltări ulterioare

Acest subcapitol prezintă câteva idei în vederea îmbunătăţirii sau extinderii funcţionalităţilor aplicaţiei HomeRun. Multe dintre ele sunt inspirate din nevoile de zi cu zi ale fiecăruia şi sunt gândite pentru a uşura munca utilizatorilor şi pentru a realiza o comunicare cât mai uşoară între acesta şi mediul fizic înconjurător.

HomeRun este proiectat ca un sistem modular ceea ce permite adăugarea cu uşurinţă a unor funcţionalităţi noi, fără a fi nevoie să se modifice structura sau conţinutul modulelor deja existente.

Ca şi dezvoltări ulterioare se poate lua în considerare fie dezvoltarea şi îmbunătăţirea modulelor deja existente sau adăugarea altor module cu funcţionalităţi noi. Câteva exemple ar fi următoarele: îmbunătăţirea modulului News prin posibilitatea de a urmări concomitent mai multe canale RSS (în prezent este posibilă urmărirea unui singur canal), extinderea funcţionalităţii modulului SMS (prin adăugarea unor senzori de mişcare în interiorul locuinţei, proprietarul poate să primească mesaj pe telefonul mobil daca senzorii sesizează mişcare). Tot de pe telefonul mobil, prin intermediul unui mesaj text, utilizatorul să poată controla alte dispozitive din locuinţă (de lumină, căldură) când nu se află acasă.

Un alt exemplu ar fi dezvoltarea unui modul care să permită aplicaţiei HomeRun să primească comenzi vocale din partea utilizatorului. S-ar putea aduce modificări şi în cadrul interfeţei grafice cu utilizatorul, făcând-o mai „prietenoasă”.

In ceea ce priveşte trimiterea de notificări, s-ar putea adăuga şi alte servicii personale de mesagerie cum ar fi Yahoo Messenger (utilizatorul să poată trimite mesaje către aplicaţia HomeRun direct din fereastra de Messenger şi de asemenea să poată primi notificări).

Pe parte de echipamente tehnice, o altă îmbunătăţire ar consta în adăugarea de echipamente precum camere web sau dispozitive care să permită înregistrarea audio pentru a putea controla în permanenţă şi de la distanţă activităţile care se desfăşoară în cadrul locuinţei.

Dezvoltări ulterioare s-ar putea face şi pe parte de securizare a bundle-urilor pentru a proteja componentele unele de altele şi pentru a evita folosirea iresponsabilă a acestora. In acest caz, folosirea lor ar fi permisă numai după îndeplinirea anumitor condiţii stricte.

O aplicaţie având toate aceste funcţionalităţi s-ar ridica la nivelul celor realizate de Siemens sau Bosh, dar cu un cost mult mai scăzut ce nu include costul de licenţiere şi cu o posibilitate de dezvoltare continuă.

7.3 Reflecţii personale

Prin realizarea acestui proiect am acumulat multe cunoştinţe. Pe lângă cunoştinţele teoretice şi practice acumulate, am învăţat cum să documentez un proiect prin înţelegerea fundamentelor teoretice care au stat la bază. Am înţeles şi pus în practică planul de cercetare bazat pe studiul cărţilor de referinţă, a jurnalelor şi articolelor şi pe compararea argumentelor aduse de autorii acestora.

In al doilea rând am învăţat câte ceva despre gestionarea timpului, şi anume cum să aloc şi să utilizez timpul pentru îndeplinirea sarcinilor. Acest lucru este foarte important pentru buna desfăşurare a unui proiect.

Am învăţat să îmi organizez eficient şi efectiv lucrul prin abordarea unei metodologii de dezvoltare Agile, SCRUM, metodologie folosită de proiectanţii din domeniul IT.

Nu în cele din urmă, mi-am îmbunătăţit abilităţile de programare în Java. Am învăţat câteva noţiuni avansate de programare consultând cărţi şi căutând pe internet, am învăţat cum să testez şi să evaluez un sistem, toate acestea sperând să mă ajute într-o carieră viitoare.

Bibliografie

[1] Alianţa OSGi, http://www.osgi.org/

[2] Apache Ant, http://ant.apache.org/

[3] **N. Bartlett**, „OSGi in practice”, Londra, 2009

[4] Concierge framework, http://concierge.sourceforge.net/

[5] Equinox framework, http://www.eclipse.org/equinox/

[6] Felix framework, http://felix.apache.org/

[7] **S. Haiges**, „OSGi Tutorial – A step by step introduction to OSGi programming based on the open source Knopflerfish OSGi framework”, 2004

[8] HomeRun, http://homerun.sourceforge.net/

[9] Java Logging API, http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/

[10] Knopflerfish framework, http://www.knopflerfish.org/

[11] Limbajul XPath, http://ro.wikipedia.org/wiki/XPath

[12] **C. Walls**, „Modular Java – Creating flexible applications with OSGi and Spring”, The Pragmatic Bookshelf, SUA, 2009

[13] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/OSGi

[14] **C.L. Wu, C.F. Liao, L.C. Fu**, „Service-Oriented Smart Home Architecture based on OSGi and Mobile Agent Technology”, http://www.try.idv.tw/try/files/smcc06.pdf

Acronime

API - Application Programming Interface
EG – Expert Group
GSM – Global System for Mobile communications
GUI – Graphical User Interface
HTTP – Hypertext Transfer Protocol
IDE – Integrated Development Environment
IP – Internet Protocol
JAR – Java Archive
JDK – Java Development Kit
JRE – Java Runtime Environment
JSR – Java Specification Request
JVM – Java Virtual Machine
OS – Operating System
OSGi – Open Service Gateway initiative
POJO – Plain Old Java Object
RMI – Remote Method Invocation
RPC – Remote Procedure Call
RSS – Really Simple Syndication / Rich Site Summary
SMS – Short Message Service
SMSC – Short Message Service Center
SMTP – Simple Mail Transfer Protocol
SOA – Service Oriented Architecture
SQL – Structured Query Language
URL – Uniform Resource Locator
XML – Extensible Markup Language
XPath – XML Path Language

Anexe

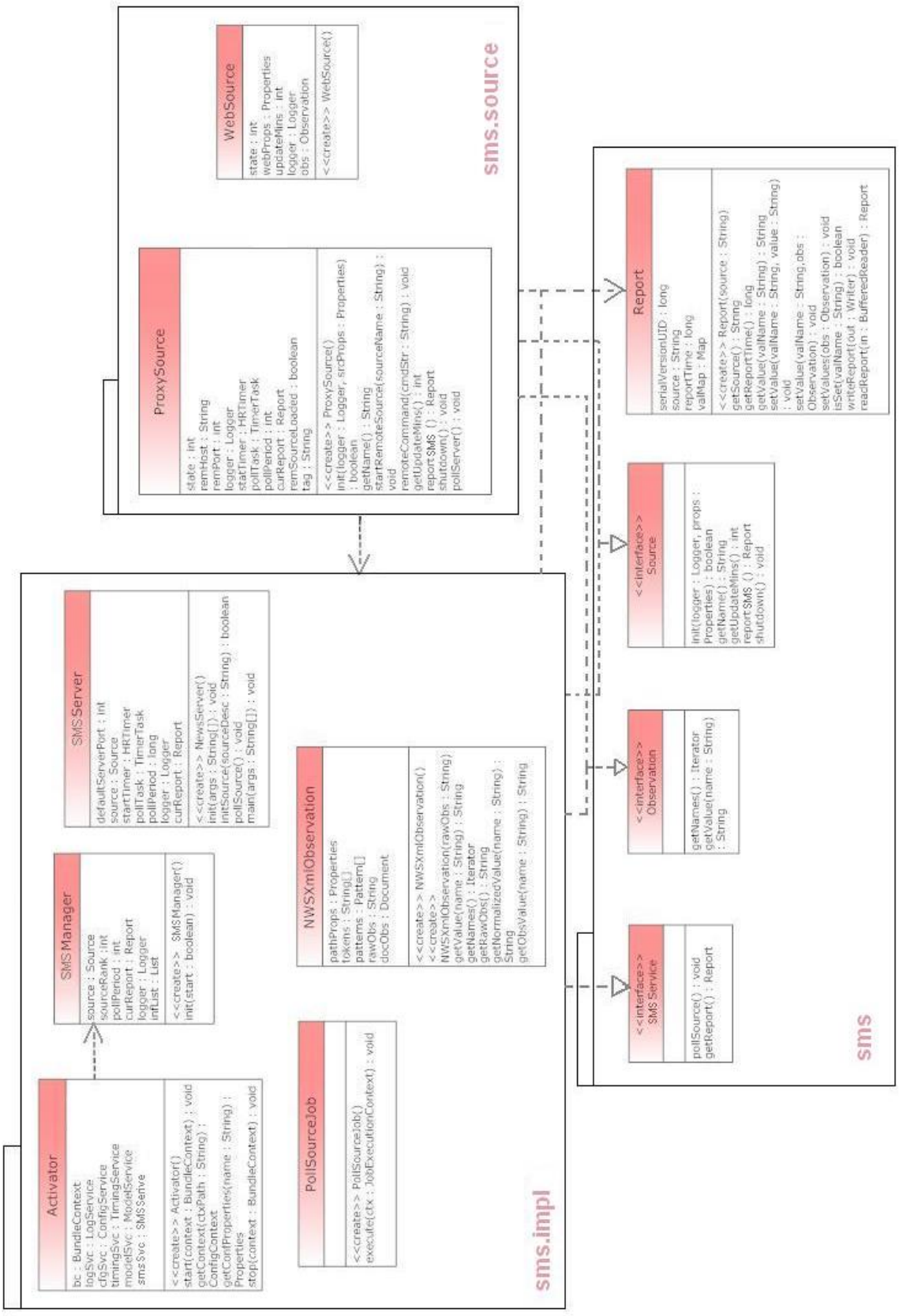

Figura 1 Diagrama de clase pentru fişierul sursă SMS

Câteva din clasele modulului SMS:

Activator.java

```
package com.monad.homerun.pkg.sms.impl;

import java.io.IOException;
import java.net.URL;
import java.util.Dictionary;
import java.util.Hashtable;
import java.util.Properties;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;

import com.monad.homerun.core.GlobalProps;
import com.monad.homerun.config.ConfigContext;
import com.monad.homerun.config.ConfigService;
import com.monad.homerun.config.Installer;
import com.monad.homerun.log.LogService;
import com.monad.homerun.modelmgt.ModelService;
import com.monad.homerun.objmgt.ObjectService;
import com.monad.homerun.pkg.sms.SMSService;
import com.monad.homerun.timing.TimingService;
/**
* OSGi Activator class
*/
public class Activator implements BundleActivator
{
  // bundle context
  private static BundleContext bc = null;
  // logging service
  static LogService logSvc;
  // config service
  static ConfigService cfgSvc;
  // timing service
  static TimingService timingSvc;
  // model services
  static ModelService modelSvc;
  // my own service
  static SMSService smsSvc;
  static ObjectService objSvc;

  // no-arg constructor
  public Activator()
    {}

  // activate the bundle
  public void start( BundleContext context ) throws Exception
    {
      bc = context;
     // get services we need
     ServiceReference ref = bc.getServiceReference(
LogService.class.getName());
     logSvc = (LogService)bc.getService( ref );
     ref = bc.getServiceReference( ConfigService.class.getName() );
     cfgSvc = (ConfigService)bc.getService( ref );
     ref = bc.getServiceReference( TimingService.class.getName() );
     timingSvc = (TimingService)bc.getService( ref );
     ref = bc.getServiceReference( ModelService.class.getName() );
     modelSvc = (ModelService)bc.getService( ref );
```

```
     ref = context.getServiceReference(ObjectService.class.getName());
     objSvc = (ObjectService)context.getService(ref);

     if ( GlobalProps.DEBUG )
      {
       // DEbug
      System.out.println( "SMS activator start: " + bc );
      }
     // if I am not yet installed, 'pre-install' config file,
     // since config parameters needed in SMSManager initialization
     // RLR TODO review this logic
     Installer.installConf( context.getBundle() );
     try
      {
       // register our service
       SMSManager mgr = new SMSManager();
       // start him up
       mgr.init( true );
       smsSvc = mgr;
       Dictionary props = new Hashtable();
       bc.registerService( SMSService.class.getName(),smsSvc, props );
      }
     catch ( Throwable t )
      {
       if ( GlobalProps.DEBUG )
      {
        t.printStackTrace();
        System.out.println( "Caught Exception: " + t.toString() );
      }}
    if ( GlobalProps.DEBUG )
     {
       System.out.println( "SMS activator started" );
     }}

  // get a config context
  public static ConfigContext getContext( String ctxPath )
   {
     // determine conf directory
     Dictionary dct = bc.getBundle().getHeaders();
     String confPath = dct.get( "Bundle-Category" ) + "/" + dct.get(
"Bundle-Name" );
     try
      {
       return cfgSvc.getContext( confPath, ctxPath );
      }
     catch ( IOException ioE )
      {
       if ( GlobalProps.DEBUG )
      {
        System.out.println( "SMS Activator init error can't read: " +
confPath );
        ioE.printStackTrace();
      }}
     return null;
    }

 public static Properties getConfProperties( String name )
  {
    String propName = "conf/" + name + ".properties";
    if ( bc.getBundle().getResource( propName ) != null )
     {
       URL propUrl = bc.getBundle().getResource( propName );
       Properties props = new Properties();
```

```
      try
       {
         props.load( propUrl.openStream() );
       }
      catch ( IOException ioE )
       {
         if ( GlobalProps.DEBUG )
         {
          System.out.println( "Error reading props: " + name );
         }}
     return props;
     }
    return null;
   }

  //deactivate the bundle
  public void stop( BundleContext context ) throws Exception
   {
     bc = null;
   }
}
```

SMSManager.java

```
package com.monad.homerun.pkg.sms.impl;

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.ArrayList;
import java.util.Properties;
import java.util.logging.Logger;
import java.util.logging.Level;
import com.monad.homerun.base.Value;
import com.monad.homerun.config.ConfigContext;
import com.monad.homerun.core.GlobalProps;
import com.monad.homerun.event.Event;
import com.monad.homerun.event.Emitter;
import com.monad.homerun.event.EventDesc;
import com.monad.homerun.model.Model;
import com.monad.homerun.model.scalar.ScalarModel;
import com.monad.homerun.model.set.SetModelStatus;
import com.monad.homerun.modelmgt.ModelInformer;
import com.monad.homerun.msgmgt.DeliveryService;
import com.monad.homerun.pkg.sms.SMSService;
import com.monad.homerun.pkg.sms.Source;
import com.monad.homerun.pkg.sms.Report;

/**
* RssManager gathers information by polling the
* configured 'sms source', which may be an internet sms service.
*/

 public class SMSManager implements SMSService, ModelInformer
  {
   // the source for sms data
   private Source source = null;
   // current source rank (no source=0, primary=1, alternate=2, etc)
   private int sourceRank = 0;
   // polling frequency in minutes
   private int pollPeriod = 0;
   // current sms report from source
   private Report curReport = null;
```

```
  // system logger
  private Logger logger = null;
  // list of models to inform
  private List<InformModel> infList = null;

  // no-arg constructor
  public SMSManager()
   {
    logger = Activator.logSvc.getLogger();
   }

  // IStatMonitor methods
  public void init( boolean start )
   {
    if ( GlobalProps.DEBUG )
     {
      logger.log( Level.FINE, "initializing" );
     }
    infList = new ArrayList<InformModel>();
   // initialize members
   source = null;
   pollPeriod = 0;
   curReport = null;
   // first try to instantiate the primary source
   if ( ! initSource( "primary" ) )
   {
    logger.log( Level.WARNING, "Can't start primary source" );
    // try the alternate source
    if ( ! initSource( "alternate" ) )
     {
      logger.log( Level.SEVERE, "Can't start any source" );
     }
    }
   // let ModelService know I am an Informer
   Activator.modelSvc.registerInformer( this );
  }

 private boolean initSource( String rank )
  {
   // if another source active, shut him down
   if ( source != null )
    {
     source.shutdown();
     // kill any pending polling job
     Activator.timingSvc.killJob( "SMSPoll" );
    }
   // get config info
   ConfigContext chanCtx = Activator.getContext( "channels/@" + rank );
   // instantiate the class configured to be the sms data source
   String sourceName = chanCtx.getProperty( "source" );
   ConfigContext srcCtx = Activator.getContext( "sources/@" +sourceName );
   Properties sourceProps = srcCtx.getProperties();
   String sourceClass = sourceProps.getProperty( "class" );
   try
    {
      source = (Source)Class.forName( sourceClass ).newInstance();
    }
   catch (Exception e )
    {
     logger.log( Level.SEVERE, "caught exception loading class '" +
     sourceClass + "': " + e.getMessage() );
     //e.printStackTrace();
     source = null;
```

```
    }
   if ( source == null )
    {
     sourceRank = 0;
     return false;
    }
   // initialize source
   if ( ! source.init( logger, sourceProps ) )
    {
     return false;
    }
   // get the updateTime to set up the poll of the source
   // get the polling period & start a poll timer
   pollPeriod = source.getUpdateMins();
   Activator.timingSvc.scheduleJob( "SMSPoll", PollSourceJob.class,
   "delay", pollPeriod );
   // add a one-off delayed poll to let the source initialize & gather
   data
   long firstPoll = System.currentTimeMillis() + 1000 * 5;
   Activator.timingSvc.scheduleJob( "FirstSMSPoll", PollSourceJob.class,
   "date", new Date( firstPoll ) );
   // adjust rank
   sourceRank = rank.equals( "primary" ) ? 1 : 2;
   return true;
}

public void pollSource()
 {
  curReport = source.reportSMS();
  if ( GlobalProps.DEBUG && curReport != null )
   {
    logger.log( Level.FINE,"SMS: "+curReport.getValue("MessageText") );
   }
  checkSanity();
  // always notify models, even if the status state
  // has not changed. This is to prevent the problem that
  // they weren't observing when this monitor started,
  // and therefore never got the first value.
  // if any models, inform them
  for ( InformModel infModel : infList )
  {
   informModel( infModel, curReport );
  }}

private void informModel( InformModel infModel, Report report )
 {
  // inform them based on their type
  Object event = null;
  String type = infModel.model.getModelType();
  // if there is a value to report
  if ( report != null )
   {
    String valStr = report.getValue( infModel.type );
    if ( valStr != null )
     {
      if ( GlobalProps.DEBUG )
       {
        System.out.println( "WethM informModel: " + infModel.type );
       }
     if ( "value".equals( type ) )
      {
       event = new Value( convertValue( valStr, infModel.type ) );
      }
```

```
     else if ( "state".equals( type ) )
      {
       event = new Value( valStr, 0L );
      }
     // add more cases here - when not integers
     if ( event != null )
     {
      Activator.modelSvc.informModel( infModel.domain, infModel.object,
      infModel.model.getModelName(),
      new Event( event, getInformerName() ) );
     }}
     else
     {
      if ( GlobalProps.DEBUG )
      {
       System.out.println( "SMS informModel - report lacks data for: " +
       infModel.type );
      }}
     }
   else
   {
    if ( GlobalProps.DEBUG )
    {
     System.out.println( "SMS informModel - lacks report for: " +
     infModel.type );
    }}}

  private String convertValue( String valStr, String type )
   {
    logger.log(Level.WARNING, valStr);
    if ( "MessageText".equals( type ))
     {
      if (valStr.equals("wth"))
      {
        String msg="",msg1, msg2;
        msg1 = (Activator.objSvc.getModelStatus("place",
"Air","temperature").getModelValue()).getStringValue();
        msg2=(Activator.objSvc.getModelStatus("place",
"Air","humidity").getModelValue()).getStringValue();
     msg="Cluj weather: temperature "+msg1+" degrees, humidity "+msg2+" %";

        HashMap params = new HashMap();
        params.put("status", msg);
        Activator.objSvc.controlObject("person", "Oxana", "tweet",
"update", params, new HashMap());

        return msg;
   }
    else
        return valStr+"no";
   }
  else return valStr;
 }

  public void reset()
  {
   // cancel polling job
   Activator.timingSvc.killJob( "SMSPoll" );
   // unregister with ModelManager
   Activator.modelSvc.unregisterInformer( this );
  // now redo init
  init( true );
  logger.log( Level.INFO, "SMSManager reset" );
```

```
  }

  // save any unsaved state data and release all resources
  public void shutdown()
   {
    // forward command to sms source
    if ( source != null )
     {
      source.shutdown();
     }
    // cancel polling job
    Activator.timingSvc.killJob( "SMSPoll" );
    // unregister with ModelManager
    Activator.modelSvc.unregisterInformer( this );
    logger.log( Level.INFO, "SMSManager shutdown" );
  }

  // End IStatMonitor methods
  // ModelInformer methods
  public boolean addModel( String type, String domain,
  String objectName, Model model )
   {
    if ( GlobalProps.DEBUG )
     {
      System.out.println( "SMSMth: addmdel: " + model.getModelName() );
     }
    InformModel infModel = new InformModel(type,domain,objectName,model );
    // inform model now
    informModel( infModel, source.reportSMS() );
    infList.add( infModel );
    return true;
   }

  public void removeModel( String domain, String objectName, String
  modelName )
   {
    for ( int i = 0; i < infList.size(); i++ )
     {
      InformModel infModel = infList.get( i );
      if ( infModel.domain.equals( domain ) &&
          infModel.object.equals( objectName ) &&
          infModel.model.getModelName().equals( modelName ) )
       {
        infList.remove( i );
        break;
       }}
   }

  public Emitter canInform( Model model )
   {
    Emitter emitter = null;
    // is this a model we can inform?
    if ( "value".equals( model.getModelType() ) )
     {
      if ( GlobalProps.DEBUG )
       {
         System.out.println( "canInform" );
       }
      ScalarModel smd = (ScalarModel)model;
      String vtName = smd.getValueType();
  return emitter;
}
```

```
  public String getInformerName()
   {
    return "SMSManager";
   }

  // Monitor may also be queried for the last retrieved sms data
  public Report getReport()
   {
    // if we don't have a report from the source yet, send a stub
    if ( curReport == null )
    {
     return ( new Report( "Manager" ) );
    }
    return curReport;
  }

  // determine whether source is operating correctly
  private void checkSanity()
   {
    boolean sane = true;
    // if we haven't yet received any report, it is an error
    if ( curReport == null )
     {
      logger.log( Level.WARNING, "Check failed - no report" );
      sane = false;
     }
    else
     {
      // see how recent the last report is
      long now = System.currentTimeMillis();
      long reportTime = curReport.getReportTime();
      // allow for some clock skew - require that last report is within
      // twice the polling period
      long ppMSecs = pollPeriod * 60 * 1000;
      if ( ( now - reportTime ) > ( ppMSecs * 2 ) )
       {
        logger.log( Level.WARNING, "Check failed - late report" );
        sane = false;
       }
      else if ( ! curReport.isSet( "MessageText" ) )
      {
       logger.log( Level.WARNING, "Check failed - bad outdoor temp" +
       " : " + curReport );
       sane = false;
      }}
      // see if there's anything we can do
      if ( ! sane && sourceRank == 1 )
       {
        logger.log( Level.INFO, "Starting alternate source" );
        initSource( "alternate" );
      }}

  private class InformModel
   {
      public String type = null;
      public String domain = null;
      public String object = null;
      public Model model = null;
      public InformModel( String type, String domain, String object, Model
model )
      {
       this.type = type;
       this.domain = domain;
```

```
        this.object = object;
        this.model = model;
      }
  }}
```

Fişierul build.xml pentru pachetului SMS

```
<?xml version="1.0"?>
<!-- ======================================== -->
<!-- build file for homerun sms package     -->
<!-- ======================================== -->

<!DOCTYPE project [
  <!ENTITY common SYSTEM "../../common.xml">
]>

<project name="hrp-sms" default="bundle" basedir=".">

  <!-- include common stuff -->
  &common;

  <!-- local properties -->
  <property name="bname" value="sms" />
  <property name="pname" value="pkg.sms" />
  <property name="cname" value="misc" />

  <!-- external properties files -->
  <property file="${3up}homerun.properties" />

  <path id="lclass.path">
    <!-- any server-specific generated jars -->
      <fileset dir="../../${build.dir}/jars">
         <include name="**/*.jar"/>
      </fileset>
      <!-- any server-specific 3rd party jars -->
      <fileset dir="../../jars">
        <include name="**/*.jar"/>
      </fileset>
      <!-- any shared generated jars -->
      <fileset dir="${3up}shared/${build.dir}/jars">
        <include name="**/*.jar"/>
      </fileset>
      <!-- quartz jar -->
      <fileset dir="../../bundles/timing/src/main/resources">
        <include name="**/*.jar"/>
      </fileset>
</path>
<!-- scompile target - compiles the java sources with a more extensive
class path -->
   <target name="scompile" depends="init">
    <javac srcdir="src" destdir="${build.dir}/classes">
      <classpath refid="lclass.path" />
    </javac>
  </target>

  <!-- bundle target - creates an OSGi bundle, which is simply
       a jar with specific manifest properties -->
  <target name="bundle" depends="scompile" >
      <jar jarfile="${build.dir}/${bname}-${homerun.version}.jar">
           <fileset dir="${build.dir}/classes" />
         <fileset dir="${rsc.path}"/>
           <manifest>
             <attribute name = "Bundle-Name"
```

```
                              value = "${bname}"/>
            <attribute name = "Bundle-SymbolicName"
                              value = "${base.pkg}.${pname}"/>
            <attribute name = "Bundle-Version"
                              value = "${homerun.version}"/>
            <attribute name = "Bundle-Description"
                              value = "Twitter SMS"/>
            <attribute name = "Bundle-Vendor"
                              value = "homerun"/>
            <attribute name = "Bundle-Activator"
                              value =
"${base.pkg}.${pname}.impl.Activator"/>
            <attribute name = "Bundle-Category"
                              value = "${cname}"/>
            <attribute name = "Import-Package"
                              value =
"${fw.pkg},${quartz},${base.pkg}.action,${base.pkg}.base,${base.pkg}.config
,${base.pkg}.control,${base.pkg}.core,${base.pkg}.event,${base.pkg}.log,${b
ase.pkg}.model,${base.pkg}.object,${base.pkg}.timing,${base.pkg}.util,${bas
e.pkg}.app,${base.pkg}.objmgt,${base.pkg}.objmgt.impl,${base.pkg}.modelmgt,
${mdl-impls.pkgs},${base.pkg}.wiring,${base.pkg}.pkg.twitter"/>
            <attribute name = "Export-Package"
                              value = "${base.pkg}.${pname}"/>
            <attribute name = "Homerun-Type"
                              value = "regular"/>
            <attribute name = "Homerun-Conf"
                              value = "sms.xml,smsnwsxml.properties" />
            <attribute name = "Homerun-Require"
                               value =
"${base.pkg}.pkg.twitter;version=&quot;${homerun.version}&quot;" />
            <attribute name = "Homerun-Icons"
                         value = "domain/picture.png,category/sms.png />
            <attribute name = "Homerun-Load"
                               value = "place-dom,value-types-twitter,
twitteritem,scenes" />
          </manifest>
      </jar>
      <!-- copy up to common staging area -->
      <copy file="${build.dir}/${bname}-${homerun.version}.jar"
            todir="../../${build.dir}/pkgs/local" />
  </target>
</project>
```

Fişierul de configurare sms.xml

```
<?xml version="1.0"?>
<smsConf>
  <channels>
    <channel name="primary">
      <source type="pick" limiters="/sources" desc="SMS
source">web</source>
    </channel>
    <channel name="alternate">
      <source type="pick" limiters="/sources" desc="SMS
source">web</source>
    </channel>
  </channels>
  <sources>
    <source name="web" enabled="true">
<class type="string" level="2" desc="Source class">
com.monad.homerun.pkg.sms.source.WebSource </class>
      <tag type="string" desc="Display Tag">NWS via Web</tag>
      <!-- URL for sms: needs to be configured for each installation -->
```

```
      <sourceURL type="string" desc="URL for sms">
http://twitter.com/statuses/user_timeline/124827426.xml</sourceURL>
      <dataFormat type="string" desc="Data Format">SMS NWS XML</dataFormat>
      <updateMins type="number" desc="Update frequency
(mins)">1</updateMins>
    </source>
  </sources>
</smsConf>
```

Fişierul de configurare smsnwsxml.properties

```
MessageText=/statuses/status/text
UserName=/statuses/status/user/name
MessageTime=/statuses/status/created_at
```

Fişierul de configurare twitteritem.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<loadSet>
  <description>Type, object, and models representing twitteritem
conditions</description>
  <loadCmd action="add" type="com.monad.homerun.object.Type"/>
  <objType name="twitteritem" enabled="true">
    <domain>place</domain>
    <description>Objects of this type represent the Twitter
Items</description>
    <create>single</create>
    <factory>platform</factory>
    <class>com.monad.homerun.objmgt.impl.SimpleObject</class>
    <icon>rss_item.png</icon>
    <typeProps>
      <typeProp name="category">
        <value>sms</value>
        <type>string</type>
        <indexed>true</indexed>
      </typeProp>
      <typeProp name="compound">
        <value>false</value>
        <type>boolean</type>
        <indexed>false</indexed>
      </typeProp>
    </typeProps>
    <specifiers>
      <spec name="icon">
        <type>icon</type>
        <limiters>object</limiters>
        <description>Icon</description>
        <required>true</required>
      </spec>
    </specifiers>
    <controls/>
    <emitters/>
    <models>
      <constraint directive="1">
        <group>value</group>
        <component>*</component>
        <properties size="1">
          <entry key="implClass">
com.monad.homerun.modelmgt.impl.ValueModel</entry>
        </properties>
      </constraint>
    </models>
  </objType>
```

```
  <loadCmd action="add" type="com.monad.homerun.model.scalar.ScalarModel"/>
  <scalarModel name="messagetext">
    <category>place</category>
    <modified>1086550158899</modified>
    <note>Provided by homerun in 'sms' package</note>
    <icon>value.gif</icon>
    <valueType>messagetext</valueType>
  </scalarModel>
  <loadCmd action="add" type="com.monad.homerun.model.scalar.ScalarModel"/>
  <scalarModel name="username">
    <category>place</category>
    <modified>1086550158899</modified>
    <note>Provided by homerun in 'sms' package</note>
    <icon>value.gif</icon>
    <valueType>username</valueType>
  </scalarModel>
    <loadCmd action="add"
type="com.monad.homerun.model.scalar.ScalarModel"/>
  <scalarModel name="messagetime">
    <category>place</category>
    <modified>1086550158899</modified>
    <note>Provided by homerun in 'sms' package</note>
    <icon>value.gif</icon>
    <valueType>messagetime</valueType>
  </scalarModel>
  <loadCmd action="add" type="com.monad.homerun.object.Instance"/>
  <instance name="Twitteritem">
    <category>sms</category>
    <modified>1086550158899</modified>
    <note>Provided by homerun in 'sms' package</note>
    <domain>place</domain>
    <type>twitteritem</type>
    <status>internal</status>
    <controls></controls>
    <emitters></emitters>
    <models>
      <binding name="messagetext">
        <compType>value</compType>
        <compName>messagetext</compName>
        <properties size="7">
          <entry key="emit">false</entry>
          <entry key="infType">MessageText</entry>
          <entry key="informer">SMSManager</entry>
          <entry key="expose">false</entry>
          <entry key="project">false</entry>
          <entry key="noCore">true</entry>
          <entry key="implClass">
com.monad.homerun.modelmgt.impl.ValueModel</entry>
        </properties>
      </binding>
      <binding name="username">
        <compType>value</compType>
        <compName>username</compName>
        <properties size="7">
          <entry key="emit">false</entry>
          <entry key="infType">UserName</entry>
          <entry key="informer">SMSManager</entry>
          <entry key="expose">false</entry>
          <entry key="project">false</entry>
          <entry key="noCore">true</entry>
          <entry
key="implClass">com.monad.homerun.modelmgt.impl.ValueModel</entry>
        </properties>
```

```
      </binding>
      <binding name="messagetime">
        <compType>value</compType>
        <compName>messagetime</compName>
        <properties size="7">
          <entry key="emit">false</entry>
          <entry key="infType">MessageTime</entry>
          <entry key="informer">SMSManager</entry>
          <entry key="expose">false</entry>
          <entry key="project">false</entry>
          <entry key="noCore">true</entry>
          <entry
key="implClass">com.monad.homerun.modelmgt.impl.ValueModel</entry>
        </properties>
      </binding>
    </models>
    <properties size="1">
      <entry key="icon">rss_item.png</entry>
    </properties>
  </instance>
</loadSet>
```