Open Services Gateway initiative - OSGi

Student : Dan Birsan : grupa 7982

Tehnologia OSGi - exemple şi aplicaţii
Lucrare de dizertaţie - Master APS (Algoritmi şi Produse Software)

Profesor îndrumător: Scheiber Ernest

18/03/2009

Abstract

Pentru parcurgerea acestui material sunt necesare cunoştinţe prealabile din disciplinele de programare: Programarea în limbajul Java.

Contents

Capitolul 1

Introducere

1.1 Despre Alianţa OSGi

Alianţa OSGi este un consorţiu internaţional de inovatori in domeniul tehnologiei software. Acest grup promovează un proces matur si fiabil care asigură interoperabilitatea aplicaţiilor şi servicilor. Interoperabilitatea se realizeaza pe baza unei platfome de integrare a componentelor. Platforma de servicii OSGi este distribuită în multe aplicaţii şi servicii ale companiilor de top cât şi pe diverse pieţe din industria informatică. Astfel piaţa telefoniei mobile cea a automatizărilor sau piaţa telecomunicaţiilor sunt câteva dintre industriile care folosesc această tehnologie.

Alianţa furnizează specificaţii, implementări de referinţă, teste şi verificări pentru a ajuta la o integrare uniformă a tehnologiei în oricare industrie. Companiile membre colaborează în cadrul unui mediu echitabil şi transparent beneficiind de expertiza altor utilizatori şi de forumuri de discuţii.

Alianţa OSGi este o corporaţie non profit înfiinţată în Martie 1999.

1.2 Architectura 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 ascunda implementarea faţă de alte componente în timp ce permit comunicarea prin servicii, acestea din urmă fiind obiecte ce sunt oferite componentelor în mod special. Acest model surprinzător de simplu are efecte cu bătaie

îndelungată pentru aproape oricare aspect al procesului de dezvoltare soft.

Cu toate că asfel de componente sunt disponibile de mai multă vreme pe piaţa soft, până în prezent ele nu au confirmat aşteptărilor, eşuând în a fi adoptate pe scară largă. OSGi este prima tehnologie care în fapt a reuşit ca şi componentă de sistem ce rezolvă multe probleme reale în dezvoltarea soft. Cei care au adoptat tehnologia OSGi beneficiază de o complexitate redusă în aproape toate aspectele de dezvoltare. Codul este mai uşor de scris şi de testat, reutilizarea sa este crescută, construirea de sisteme devine semnificativ mai simplă, implementarea este mai usor de gestionat, defectele de programare sunt depistate mai devreme în ciclul de dezvoltare, iar rularea programelor conferă o transparenţă totală a ceea ce se execută. Însă cea mai importantă mărturie o reprezintă larga adopţie a tehnologiei şi implemntarea acesteia în aplicaţii precum Eclipse şi Spring.

Tehnologia OSGi a fost creată pentru a oferi un mediu colaborativ de dezvoltare soft. Dezvoltatorii nu au fost preocupaţi de posibilitatea rulării a multiple aplicaţii într-o singură maşină virtuală. Serverele de aplicaţii sunt capabile astăzi de aşa ceva cu toate că ele nici nu existau pe vremea când a fost început acest proiect în 1998. Problema pe care OSGi a încercat să o rezolve a fost mai dificilă. Sa dorit ca o aplicaţie să se formeze prin unirea mai multor componente reutilizabile care nu au cunostinţe prealabile unele faţa de celelalte. Chiar mai mult, sa dorit ca aplicaţiile să se nasca din asamblarea dinamică a unui set de componente. De exemplu, aveţi acasă un **server capabil să gestioneze lumini şi dispozitive**. O componentă v-ar permite să comutati întrerupătorul de curent prin intermediul unei pagini web. O altă componentă v-ar permite controlul dispozitivelor prin intermediul mesajelor text de pe mobil. Scopul a fost acela de a permite adăugarea unor noi funcţiuni fără ca programatorii să aibă cunostinţă unul de ce lucrează celălalt, implementând aceste componente independent.

1.3 Stratificarea

OSGi are un model stratificat care este ilustrat în figura următoare.

Lista următoarea conţine o scurtă definire a acestor termeni:

- Bundles - Pachete = sunt acele componente OSGi dezvoltate de programatori.

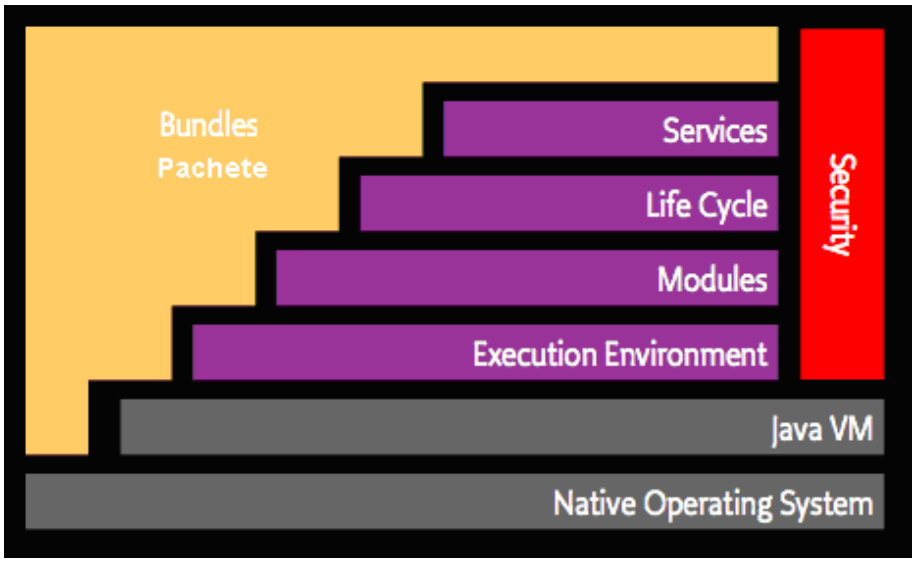

Figure 1.1: Stratificarea OSGi

- Services - Stratul de servicii care conecteaza pachetele într-un mod dinamic oferind un model "publică-găseşte-leagă" folosit pentru obiectele de tip Java obişnuite (POJO).
- Life-Cycle - Este interfaţa pentru instalarea, pornirea, oprirea, actualizarea, şi dezinstalarea de pachete = bundles.
- Modules - Stratul care defineşte cum un pachet - bundle poate importa şi exporta cod.
- Security - Stratul care se ocupa de aspectele de securitate
- Execution Environment - Defineşte ce metode şi clase sunt disponibile pe o anumită platformă.

Aceste concepte sunt explicate mai pe larg în sectiunile următoare.

1.4 Modules

Conceptul fundamental care permite funcţionarea unui astfel de sistem este modularitatea. Modularitatea, explicată la modul simplist, se referă la a presupune şi a-ţi asuma cât mai puţine lucruri. Modularitatea se referă la

păstrarea lucrurilor la nivel local fără a le pune la comun. Este greu să greşeşti atunci când te referi la nişte lucuri pe care nu le cunoşti şi asupra cărora nu poţi face presupuneri. De aceea modularitatea este baza, nucleul specificaţiilor OSGi find integrată în conceptul de pachet - bundle. În termenii folosiţi în Java, un bundle este un simplu fişier JAR obişnuit (POJO). În orice caz, dacă în varianta "Java standard" într-un JAR totul este complet vizibil tuturor celoralalte JAR-uri, OSGi ascunde totul dintr-un JAR mai puţin atunci când opţiunea de accesare este menţionată explict. Un pachet - bundle care doreşte să folosească un alt JAR trebuie să importe explicit părţile de care are nevoie. Implicit sau automat nu există nici un fel de moştenire de la un pachet la altul.

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

1.5 Servicii

Motivul pentru care avem nevoie de modelul de servicii este datorat faptului că Java ne arată cât este de dificil să scrii un model colaborativ doar prin accesarea în comun a claselor. Soluţia standard în Java este folosirea fabricilor **"factories"** care folosesc încărcarea dinamica a claselor. De exemplu, dacă vrei un **DocumentBuilderFactory** atunci invoci metoda statică de fabricare: **DocumentBuilderFactory.newInstance()**. În spatele acelei faţade, metoda newInstance încearcă fiecare truc pentru încărcarea unei clase şi pentru a crea o instanţă a unei subclase a clasei **DocumentBuilderFactory**. Încercarea de a influenţa ce implementare să fie folosită, nu este o sarcină uşoară. De asemenea acesta este un model pasiv. Codul implementat nu poate să facă totul de unul singu pentru a-şi face cunoscută disponibilitatea şi nici utilizatorul nu poate afişa o listă a soluţilor din care să o aleaga pe cea optimă. De asemenea modelul clasic nu este dinamic. Din momentul în care o implementare înaintează o instanţă, acel obiect nu se mai poate retrage. Cel mai rău însă este faptul că mecanismul factory este o convenţie utilizată în sute de locuri în maşina virtuală VM unde fiecare factory are propria interfaţă API şi propriul mecanism de configurare. Nu există o vedere centralizată a implementărilor la care codul dumneavoastră este legat.

Soluţia la toate aceste neajunsuri este serviciul OSGi registry. Un pachet poate creea un obiect şi să-l înregistreze cu ajutorul serviciului OSGi registry

sub una sau mai multe interfeţe. Alte pachete pot apela registry ca să listeze toate obiectele care sunt înregistrate sub o anume interfaţă sau clasă. De exemplu, un pachet furnizează o implementare a DocumentBuilder-ului. Când acesta este lansat crează o instanţă a clasei **DocumentBuilderFactory-Impl** şi o înregistrează cu registry sub clasa **DocumentBuilderFactory**. Un pachet care are nevoie de **DocumentBuilderFactory** poate apela registry pentru a cere o listă cu toate serviciile disponibile care extind clasa **DocumentBuildFactory**. Chiar mai mult, un pachet poate aştepta să apară un anumit serviciu, după care să primească un răspuns.

Un pachet poate înregistra un serviciu, poate primi un serviciu şi poate urmări dacă un serviciu apare sau dispare. Oricâte pachete pot înregistra acelaşi tip de serviciu, şi oricâte pachete pot primi acelaşi serviciu. Aceasta este ilustrata în figura următoare.

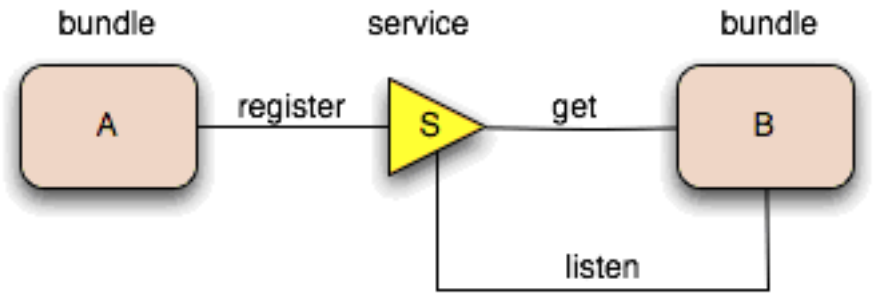

Figure 1.2: Bundle - Service

Ce se întampla cand pachete multiple înregistreaza obiecte sub aceeaşi interţată sau clasă? Cum pot fi ele diferenţiate? Raspunsul este: prin proprietăţile acestora. Un filtru de selecţie este disponibil pentru a selecta numai serviciile asupra c˘arora eşti interesat. Proprietăţiile pot fi utilizate pentru a găsi serviciul adecvat sau pot juca alte roluri la nivelul aplicaţiei.

Serviciile sunt dinamice. Aceasta înseamnă că un pachet poate sa-şi retragă serviciile din registry în timp ce alte pachete continuă să folosească acest serviciu. Pachetele folosind un astfel de serviciu trebuie să se asigure că nu mai folosesc serviciul şi că au fost eliminate toate referinţele către acesta. Stim că acest lucru implică o complexitate ridicată dar este dovedit faptul că, clase precum Service Tracker şi cadre de lucru precum iPOJO, Spring şi Declarative Services, pot uşura implementarea în timp ce avantajele sunt considerabile. Dinamica serviciilor a fost adaugată astfel încat să putem instala şi dezinstala pachete din mers, în timp ce alte pachete se pot adapta. Aceasta înseamnă că un pachet ar putea funcţiona în continuare

chiar dacă de exemplu serviciul http cade.

În orice caz am descoperit în timp că lumea reală este dinamică şi multe probleme sunt mult mai uşor de modelat cu servicii dinamice decât cu factories statice. De exemplu, un serviciu al unui dispozitiv poate fi un dispozitiv din reţeaua locală. Dacă dispozitivul este înlăturat atunci serviciul care îl reprezintă este scos din registry. În acest fel disponibilitatea unui serviciu modelează disponibilitatea unei entităţi din lumea reală. Aceasta functionează foarte bine în modelul OSGi distribuit unde un serviciu poate fi retras dacă conexiunea la distanţa cu o maşină dispare. De asemenea se releva faptul că această dinamică rezolvă problema iniţializării. Aplicaţiile OSGi nu necesită un pachet specific pentru iniţializare.

Efectul pe care la avut serviciul de înregistrare a fost acela că multe interfeţe API specializate au putut fi modelalte folosind acest serviciu. Nu numai că aceasta a simplificat per ansamblu aplicaţia, dar a şi permis standardizarea uneltelor folosite la depanarea şi investigarea sistemelor pentru a vedea cum sunt acestea conectate.

Cu toate că serviciul de înregistrare acceptă orice obiect ca serviciu, cel mai bun mod de a obţine reutilizarea este înregistrarea acestor obiecte sub interfeţe standard pentru decuplarea implementatorului de codul clientului. Acesta este motivul pentru care Alianta OSGi a publicat un compendium cu specificaţii. Toate aceste servicii standardizate sunt descrise cu lux de amănunte.

1.6 Instalarea în mediul de lucru

Pachetele sunt amplasate într-un cadru de lucru OSGi, numit mediu de rulare în timp real a pachetelor (bundle runtime environment). Acesta nu este un mediu precum cel al aplicaţiilor server Java. Acesta este un mediu colaborativ. Pachetele rulează în aceeaşi maşina virtuala VM şi pot în fapt să interschimbe codul. Cadrul de lucru foloseşte importurile şi exporturile definite explicit pentru a conecta pachetele astfel încat acestea să nu se îngrijească de încarcarea claselor. O alta diferentă faţă de serverul de aplicaţii este faptul că serverul de aplicaţii este standardizat. O simpla interfaţa API permite pachetelor să instaleze, pornească, oprească şi să actualizeze alte pachete, precum şi enumerarea pachetelor şi folosirea lor de către un serviciu. Interfata API este folosită de multi agenti de managament pentru a controla cadrul de lucru OSGi. Agenţii de management sunt diversi precum

"Knopflerfish desktop" sau "IBM Tivoli management server".

1.7 Implementări

Procesul de specificare al OSGi necesită o referinţă de implementare pentru fiecare specificaţie. Cu toate acestea, de la primele specificaţii au existat întodeauna companii comerciale care au implementat specificaţiile alături de implementări Open Source. Actualmente, există patru implementari Open Source ale cadrului de lucru şi prea multe implementări ale serviciilor OSGi, pentru a le mai putea număra. Industria software Open Source a descoperit tehnologia OSGi şi din ce în ce mai multe proiecte livrează soluţii la pachet cu această tehnologie.

1.8 Concluzii

Specificaţiile OSGi oferă un model matur şi atotcuprinzător cu o interfaţă API pe cât de eficientă pe atât de mică. Trecerea unor sisteme monolitice sau imature la modelul OSGi furnizează adesea mari îmbunătăţiri în întregul proces de dezvoltare a softului.

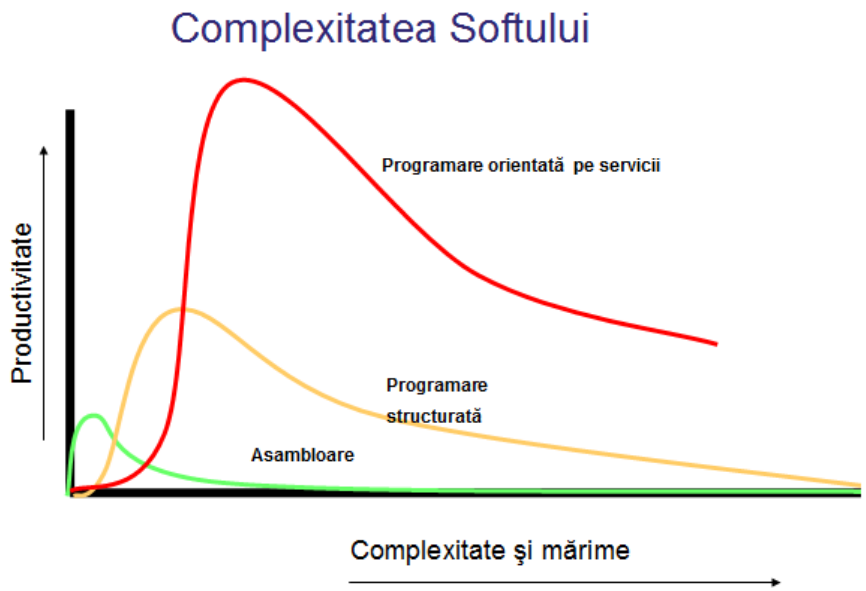

Figure 1.3: Grafic comparativ

Din unele studii statistice rezultă faptul că productivitatea în programare este rezultatul unei combinaţii de factori precum complexitatea, mărimea şi tehnica de programare folosită. Astfel nu este de mirare ca odată cu creşterea complexităţii şi mărimii softului, scade şi productivitatea. În graficul anterior se poate observa cum programarea orientată pe servicii oferă productivitate crescută faţă de tehnicile de programare tradiţionale (mai vechi).

Acest lucru se datorează în cea mai mare parte faptului că OSGi se integrează perfect cu sistemul de operare, maşina virtuală JAVA, clasele de librării ale sistemului şi alte aplicaţii si librării preexistente.

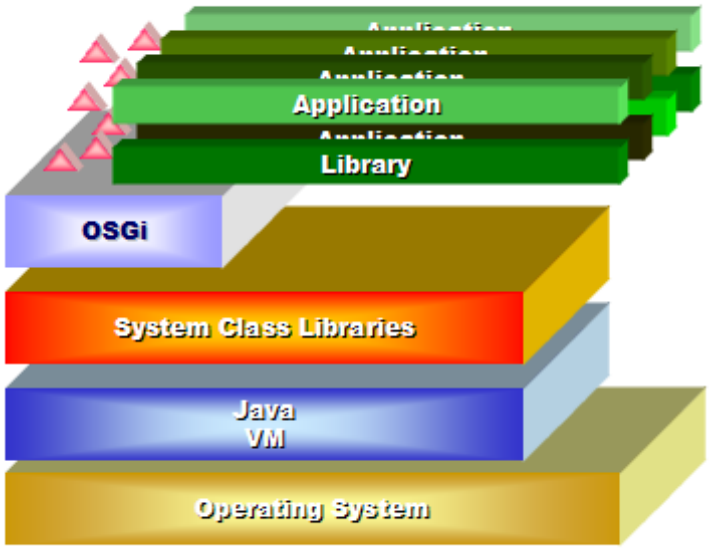

Figure 1.4: Layere constructie aplicatii

Capitolul 2

Cadre de lucru

Când spunem cadru de lucru ne gândim la un mediu în care aplicaţiile pot rula, beneficiind de toate avantajele oferite de tehnologia OSGi care este 100% compatibilă cu specificaţiile <OSGi R4 Service Platform>. În prezent există mai multe implementări OSGi.

Aşadar există la ora actuală mai multe implementări OSGi de referinţă. Dintre acestea vom prezenta trei care sunt şi cele mai răspândite şi cunoscute:

- Equinox
- Knopflerfish
- Felix

2.1 Equinox

La instalarea programului 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 apare promptul: osgi>

Instalare: Pentru a rula independent de mediul Eclipse, dintr-o distribuţie Eclipse, se copiază în directorul propriu fişierul org.eclipse.osgi *.jar, schimbând numele în equinox.jar. Alternativ, fişierul menţionat mai sus se poate descărca de la adresa web următoare:

http://download.eclipse.org/eclipse/equinox .

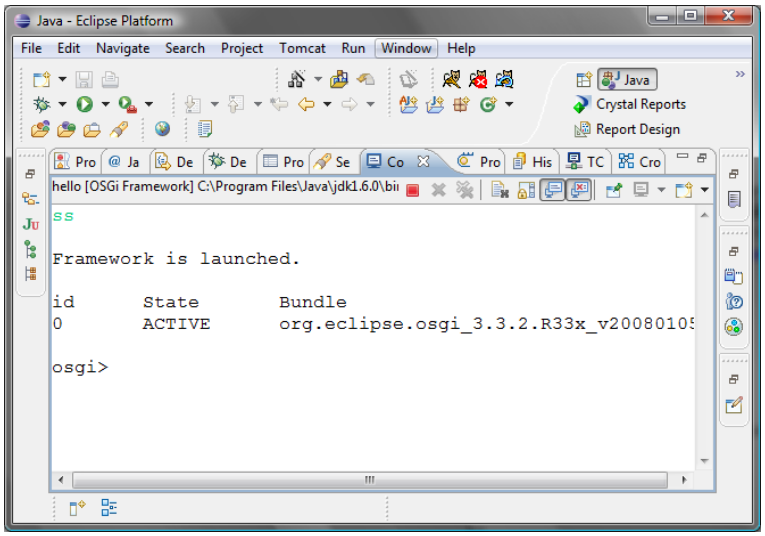

Figure 2.1: Equinox Eclipse

Utilizare: Cadrul de lucru se lansează prin comanda:

java -jar equinox.jar -console

apărând prompt-ul osgi>. Cadrul de lucru foloseşte subcatalogul configuration al catalogului de lucru.

În continuare se folosesc comenzile OSGi:

2.2 Knopflerfish

Kitul de instalare al pachetului Knopflerfish se găseşte la următoarea adresă web:

http://www.knopflerfish.org/download.html.

După copierea fişierului jar de instalare pe local acesta se instalează şi se lansează în execuţie cu următoarea comandă:

java -jar framework.jar

În urma acestei comenzi la prima pornire se instalează şi se pornesc toate

Comanda	Funcţionalitatea
ss	*short status* - Asează lista modulelor OSGi instalate.
exit	Părăseşte şi închide cadrul de lucru.
install file:modulOSGi.jar	Instalează modulul OSGi. La instalare unui modul i se atribuie în vederea identificării un număr natural - id.
start id	Lansează modulul OSGi id.
stop id	Opreşte modulul OSGi id.
uninstall id	Dezinstalează modulul OSGi id.

Table 2.1: Comenzi Equinix Consolă

pachetele programului. Vezi figura: Start Knopflerfish

Mediul de lucru Knopflerfish se caracterizează printr-o interfaţă grafică ce ne permite pornirea sau oprirea pachetelor, vizualizarea mesajelor de eroare cât şi alte operaţiuni similare. Aşa cum am văzut în exemplul precedent listarea pachetelor instalate se făcea şi din consola. Acest lucru este posibil şi în Knopflerfish care dispune de o consola pentru comenzile date in mod text. Vezi figura: Desktop Knopflerfish:

Exemplu de implementare al unei aplicaţii: Vezi aplicaţia HomeRun.

2.3 Felix

Instalarea se face prin simpla dezarhivare a pachetului Felix într-un director la alegere. Pachetul se poate găsi pe internet la următoarea adresă:

http://felix.apache.org/site/downloads.cgi.

După dezarhivare se porneşte cadrul de lucru Felix din directorul rădăcină cu comanda:

java -jar bin/felix.jar

Comenzile OSGi sunt diferite faţă de cele utilizate de Equinox dar asemănă-

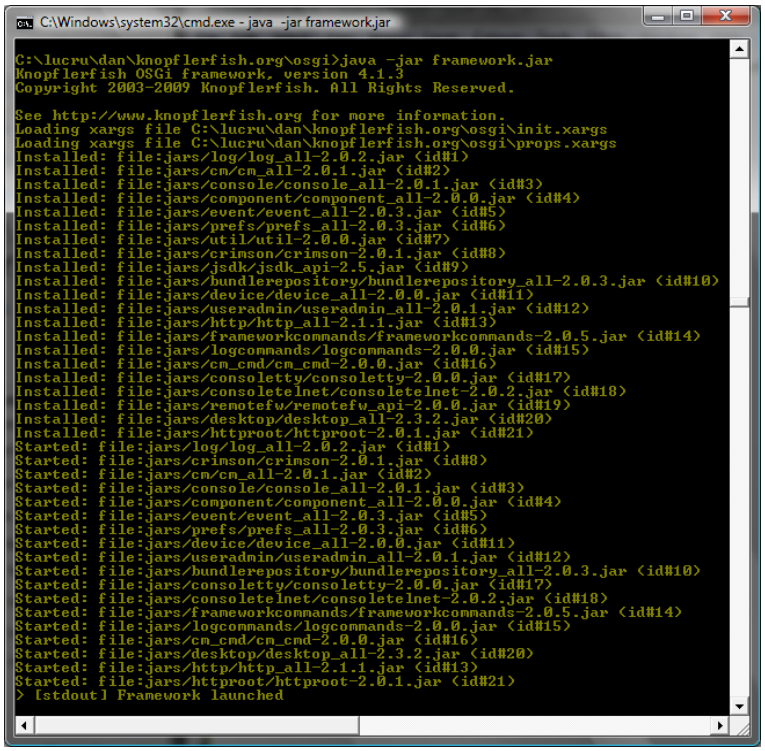

Figure 2.2: Start Knopflerfish

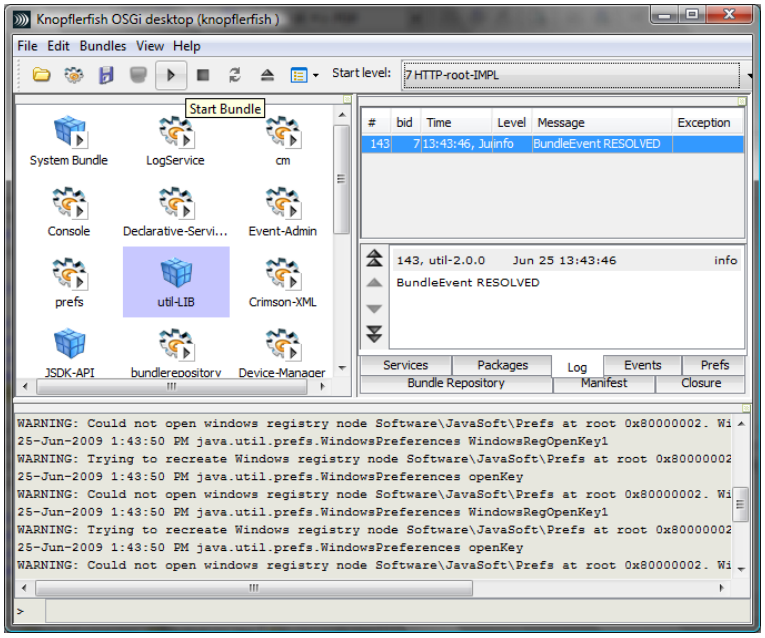

Figure 2.3: Knopflerfish Desktop

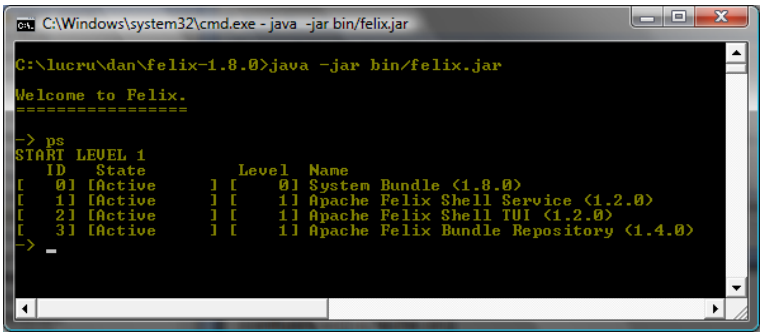

Figure 2.4: Consola Felix OSGi

Comanda	Funcţionalitatea
ps	*process status* - Afişează lista modulelor OSGi instalate.
refresh	Actualizează pachetele.
start	Lansează modulul OSGi id.
stop	Opreşte modulul OSGi id.
uninstall	dezinstalează un pachet.
requires	Lista pachetelor care oferă servicii.
requirers	Lista pachetelor care cer servicii.
help	Afişează comenzile ce pot fi folosite
shutdown	Părăseşte şi închide cadrul de lucru.

Table 2.2: Comenzi Felix Consolă

toare parţial cu cele folosite de Knopflerfish.

Capitolul 3

Apache Felix şi Spring DM OSGi

3.1 Avantaje ale OSGi.

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

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

Pentru a rezolva problemele menţionate mai sus, puteţi folosi cadrul de lucru OSGi deoarece acesta 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 pachetele pot fi publicate ca servicii şi exportate pentru a fi folosite de către alte pachete.
- OSGi furnizează un mecanism de etichetare al versiunii fiecărui pachet pentru fiecare nouă implementare.
- Cu OSGi putem actualiza dinamic pachetele în timpul rulării de câte ori are loc o schimbare de cod.

În următoarele exemple vom parcurge şi exemplifica folosirea pachetului Felix ca şi container OSGi.

Partea 1 demonstrează dezvoltarea de componente folosind OSGi API.

Partea 2 va exemplifica folosirea tehnologiei Spring într-un container OSGi şi înlăturarea dependinţelor OSGi API din clasele componentelor. Astfel aceste componente vor deveni obiecte POJO. Avantajul oferit de acest mediu este că ne putem concentra doar pe scrierea de "business metods" în clasa componentelor noastre lasând gestionarea ciclului de viaţă al componentelor în grija fişierelor de configurare Spring.

3.2 Felix şi OSGi - Partea întâi

3.2.1 Introducere

În acest articol din partea întâi vom dezvolta o aplicaţie cu componente pentru partea de client şi server. După aceasta vom împacheta aceste componente ca şi pachete-bundle OSGi. Clientul va invoca componenta serviciu pentru a procesa un ordin. Componenta serviciu conţine o metodă care procesează ordinul şi tipăreşte numărul de ordine - order ID. După citirea acestui articol veţi putea aplica conceptele şi facilitaţile produsului Apache Felix pentru a construi şi împacheta clasele componentelor Java ca pachete OSGi.
Aplicaţia Order este una pur demonstrativă. Ea constă în simpla tipărire al unui id = ORD123.

3.2.2 Cerinţe ale sistemului

Pentru a rula exemplele din acest articol, verificaţi programele instalate pe calculator şi setarile lor:

- Java 5 sau mai nou
- Ant build tool
- Distribuţia Apache Felix binary 1.0.4

În continuare setaţi următoarele variabile de mediu:

- JAVA_HOME (pentru Java)

- ANT_HOME (pentru Ant)

În continuare adăugaţi următoarele variabile de mediu PATH:

- JAVA_HOME \ bin
- ANT_HOME \ bin

3.2.3 Aplicaţia Order

Să vedem cum putem creea un pachet pentru o aplicatie de comenzi folosind un cadru OSGi Felix. Aplicaţia are doua componente: OrderClient.java (pe partea de client) şi OrderService.java (pe partea de server). Componenta client este împachetată ca şi client.jar iar componenta server ca order.jar. Să inspectăm mai întâi clasa OrderClient.

Aşa cum vedem in Listing 1., OrderClient invocă metoda processOrder prin intermediul componentei OrderService pentru a tipări orderID atunci când pachetul client.jar este lansat. Clasa implementează interfaţa BundleActivator care prezintă două metode complementare start() şi stop(). Containerul Felix invocă aceste două metode implementate atunci când pachetul client este pornit sau oprit.

Să aruncăm o privire mai în detaliu la metoda start(). În metoda start() mai întâi primim o referinţă către clasa ServiceTracker. Putem să ne gândim la aceasta ca la o clasă de tip factory. După aceasta primim referinţa serviciului de la această clasă factory prin apelarea cu propria-i metodă getService. ServiceTracker este construit cu următorii parametrii: contextul pachetului şi numele clasei OrderService.

Fişierul order.jar folosit pe partea de server conţine două componente: Interfaţa OrderService şi clasa OrderServiceImpl. Această interfaţă implementează de asemenea o interfaţă BundleActivator care este invocată de către containerul Felix pentru a porni şi opri pachetul order.jar. Metoda start() înregistrează componenta OrderService în regiştrii pentru a fi folosită de pachetul client.

```
public class OrderClient implements BundleActivator{

private ServiceTracker orderTracker;
private OrderService orderService;

public void setService(OrderService orderService)
this.orderService = orderService;
}
public void removeService()
this.orderService = null;
}
public void start(BundleContext context) throws Exception
torderTracker = new ServiceTracker(context, OrderService.class.getName(), null);
torderTracker.open();
tOrderService order = (OrderService) orderTracker.getService();

if (order == null)
System.out.println("Order service not available");
} else {
order.processOrder();
}
}
public void stop(BundleContext context) {
System.out.println("Bundle stopped");
orderTracker.close();
}
}
```

Table 3.1: Listing 1. Componenta client OrderClient

```
public class OrderServiceImpl implements OrderService, BundleActivator {

private ServiceRegistration registration;

public void start(BundleContext context) {
registration = context.registerService(OrderService.class.getName(), this, null);
System.out.println("Order Service registered");
}

public void stop(BundleContext context) {
System.out.println("Order Service stopped");
}

public void processOrder() {
System.out.println("Order id: ORD123") ;
}
}
```

Table 3.2: Listing 2. Implementarea OrderService

3.2.4 Comunicarea prin intermediul fişierelor Manifest

Întrebarea care rămîne este, cum ştie pachetul client despre pachetele de servicii înregistrate? Această comunicare este gestionată prin intermediul fişierelor Manifest. În fişierele Manifest, furnizăm referinţe despre pachetele care pot fi importate şi exportate. Pachetul client Manifest în mod normal importă pachetele de servicii disponibile în timp ce pachetul service Manifest exportă propriul package. De notat faptul că atunci când pachetul A importă un package al pachetului B, şi pachetul B trebuie să exporte propriul package. Comunicarea va eşua fără o definire corectă a importurilor şi exporturilor.

În aplicaţia noastră pachetul client.jar în fişierul său Manifest are definită partea; Import-Package: org.osgi.framework, org.osgi.util.tracker, order. Acest lucru înseamnă efectiv că pachetul client importă package-urile descrise în continuare: core OSGi packages şi OrderService package. În mod similar pachetul order.jar are definit in fişierul Manifest aferent partea de export prin linia următoare: Export-Package: order. Acest lucru înseamnă că pachetul exportă package-urile proprii pentru a fi utilizate de client. OSGi va da un mesaj de eroare dacă importurile şi exporturile nu sunt definite explicit.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Order Service Client
Bundle-SymbolicName: orderclient
Bundle-Version: 1.0.0
Bundle-Activator: order.client.OrderClient
Import-Package: org.osgi.framework, org.osgi.util.tracker, order
```

Table 3.3: Listing 3. Fişierul Manifest Client

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Order Service
Bundle-SymbolicName: orderservice
Bundle-Version: 1.0.0
Export-Package: order
Bundle-Activator: order.impl.OrderServiceImpl
Import-Package: org.osgi.framework
```

Table 3.4: Listing 4. Fişierul Manifest Service

Fisierul Manifest conţine şi alte informaţii precum numele pachetului clasei de activare. Clasa de activare este responsabilă pentru invocarea metodelor start() şi stop() în pachete. În acest caz, clasa activator pentru pachetul client.jar este OrderClient şi OrderService este pentru pachetul order.jar.

3.2.5 Instalarea

Pentru început creaţi structura de directori afişată în figura următoare:

Plasaţi componentele programului descrise anterior în această structură. Codul Java se copiază în directorii aferenţi atât pentru client cât şi pentru service.

Ambele fişiere Manifest client şi service sunt copiate în directorul META-INF.

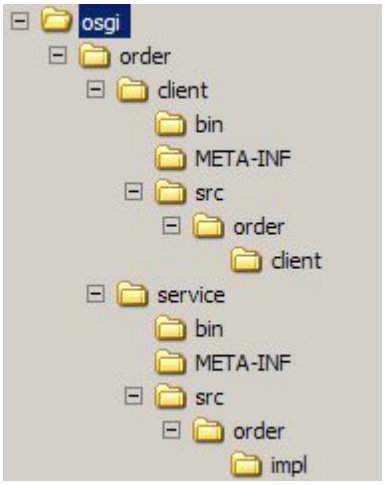

Figure 3.1: Structura directorilor

Următorul pas este să creăm aceste pachete. Pentru aceasta folosim containerul Felix OSGi care va genera pachetele client şi service urmând aceşti paşi:

- Rulăm build.xml folosind ANT în directorul service.
- Rulăm build.xml folosind ANT în directorul client.

După execuţia acestor fişiere de construcţie sunt create pachetele client.jar şi order.jar în directorii client/bin şi service/bin.

Rulăm startfelix.bat in consola command prompt Microsoft® Windows® pentru a lansa în execuţie Felix.

De fiecare dată când pornim Felix în modul run time suntem apelaţi de către program pentru a furniza un nume de profil. Acest nume este folosit ca un nume de proiect. Putem furniza orice nume pentru profilul proiectului. La fiecare pornire ulterioară a sistemului, dacă furnizăm acelaş nume de proiect, setările anterioare sunt păstrate şi Felix încarcă toate pachetele asociate numelui proiectului. Dacă furnizăm un nume nou de proiect, atunci va trebuii să instalăm în mod explicit pachetele aferente proiectului nostru:

Pachetele se instalează folosind următoarele comenzi în consola Felix:

- install file:service/bin/order.jar
- install file: client/bin/client.jar
- start service_bundle_id
- start client_bundle_id

Pentru a indica ID-ul pachetului putem folosi comanda ps. Întâi trebuie să pornim pachetul serviciu şi pe urmă pachetul client. După lansarea pachetelor vor fi afişate următoarele: Vezi figura următoare:

```
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (1.0.4)
[   1] [Active     ] [    1] Apache Felix Shell Service (1.0.1)
[   2] [Active     ] [    1] Apache Felix Shell TUI (1.0.1)
[   3] [Active     ] [    1] Apache Felix Bundle Repository (1.0.3)
-> install file:service/bin/order.jar
Bundle ID: 4
-> install file:client/bin/client.jar
Bundle ID: 5
-> start 4
DEBUG: WIRE: 4.0 -> org.osgi.framework -> 0
Order Service registered
-> start 5
DEBUG: WIRE: 5.0 -> order -> 4.0
DEBUG: WIRE: 5.0 -> org.osgi.util.tracker -> 0
DEBUG: WIRE: 5.0 -> org.osgi.framework -> 0
Order id: ORD123
->
```

Figure 3.2: Felix Dos output

Putem de asemenea să actualizăm pachetele prin comanda update pachet id dată în consola programului Felix. Pachetul este actualizat din mers în timpul rulării.

3.2.6 Concluzii

Aceasta primă parte descrie sumar capacităţiile şi conceptul cadrului de lucru OSGi şi demonstrează cum putem să folosim acest cadru pentru a crea pachete JAR dinamice. Am învăţat despre construirea şi împachetarea componentelor în pachete OSGi, am învăţat despre rularea în mediul Felix şi am aruncat o privire asupra fişierelor pachet Manifest care acţionează ca o interfaţă de comunicare între pachete.

OSGi oferă o nouă perspectivă la modul în care pachetele JAR sunt construite şi gestionate. În partea a doua vom vedea cum cadrul de lucru Spring va prelua responsabilitatea asupra managementului pachetelor OSGi într-un mediu OSGi.

3.3 OSGi şi Spring - Partea a doua

3.3.1 Introducere

În acest articol vom revedea aplicaţia de comenzi prezentată în prima parte. Aplicaţia va folosi acum Spring DM pentru a construii şi împacheta pachetele-bundle. Aplicaţia client va invoca componenta service pentru a procesa comanda şi componenta server va tipării numărul de ordine order ID. După parcurgerea acestor paşi vom înţelege conceptul Spring DM şi modalitatea de folosire al acestuia cu containerul OSGi bazat pe Felix.

3.3.2 Despre Spring DM

Spring DM este prescurtarea de la "Spring Dynamic Modules". Această tehnologie face posibilă integrarea uşoară a aplicaţiilor dezvoltate cu tehnologiile Spring şi OSGi. O aplicaţie Spring scrisă cu ajutorul acestor două tehnologii oferă următoarele avantaje:

- O separare mai bună a modulelor
- Posibilitatea de a adăuga, şterge sau actualiza module într-un sistem care rulează în timp real.
- Facilitatea de a instala şi rula mai multe versiuni de module simultan astfel încât să nu se creeze conflicte între versiuni iar legăturile să se realizeze automat.

- Un model de servicii dinamice.

Spring DM are următoarele cerinţe software.

- Java 1.4 (sau mai nou)
- Platforma OSGi R4 (sau mai nouă). Spring DM este testata zilnic cu Eclipse Equinox 3.2.x, Knopflerfish 2.2.x şi Apache Felix 1.x
- Spring Framework 2.5.6 (sau mai nou)

Aici putem face download la această tehnologie:

http://www.springsource.org/osgi

Spring DM include JAR-uri sau pachete care ne vor ajuta să instalăm aplicaţiile Spring în mediu de rulare OSGi. Aplicaţiile Spring DM pot face uz de serviciile oferite de mediul de lucru OSGi. Aceste tipuri de aplicaţii furnizează o mai mare uşurinţă şi comoditate la construcţia aplicaţiilor bazate pe OSGi. Spring DM oferă următoarele beneficii în mediul OSGi:

- Modularizează aplicaţia în pachete dinamice şi furnizează servicii în timp real acestor pachete.
- Furnizează capacităţi de etichetare a versiunilor aplicabile modulelor.
- Modulele sunt pachete definite ca servicii care pot fi descoperite şi consumate dinamic
- Modulele pot fi instalate, actualizate şi dezinstalate dinamic
- Profită de cadrul de lucru Spring în configurarea aplicaţiei ca module OSGi
- Furnizează separarea părţilor de "business logic" şi "configurare" astfel facând setarea şi instalarea uşoară şi comodă.

3.3.3 Cerintele sistemului

Pentru a rula exemplele din acest articol, verificaţi programele instalate pe calculator şi setările lor:

- Java 5 sau mai nou
- Ant build tool

- Distribuţia Apache Felix binary 1.0.4
- Pachetele Spring DM

După ce pachetele menţionate anterior au fost instalate, setaţi variabilele de mediu: (de exemplu <set ANT_HOME=C:\apache-ant-1.7.0>).

- JAVA_HOME (for Java)
- ANT_HOME (for Ant)

Va fi nevoie de următoarele cărări definite ca variabile de mediu PATH:

- JAVA_HOME\bin
- ANT_HOME\bin

3.3.4 Refacerea aplicaţiei Order

Vom revizita aplicaţia order dezvoltată în prima parte. Aşa cum puteţi observa clasele sunt strâns legate de cadrul de lucru OSGi. Acum vom elimina această legătură strânsă şi vom face ca acestea să fie simple clase POJO cu ajutorul cadrului "Spring DM".

Să privim la noul fişier OrderClient.java, de mai jos.

Dependinţa de cadrul de lucru OSGi este complet eliminată. Clasa este o simplă clasă POJO cu metoda start() care procesează comanda. În cazul acesta nu este folosită clasa ServiceTracker .

Similar, OrderServiceImpl va fi transformat şi el într-un simplu POJO cu metoda processOrder(). Nu există nici o asociere cu componente nucleului OSGi. De asemenea nici clientul şi nici clasele service nu implentează BundleActivator. Ciclul de viaţă al pachetelor este gestionat aici de Spring DM.

Aşadar, cum interacţionează aceste simple POJO cu componentele OSGi? Secretul aici îl reprezintă fişierele de configurare Spring. Aici este definită partea legată de OSGi.

Înainte de a defini fişierele XML trebuiesc făcute câteva schimbări la fişierele manifest. Manifestul va conţine includeri ale importului de packageuri ce conţin fişiere Spring DM. Acest lucru va asigura preluarea managementului pachetelor OSGi de către Spring. Codul de mai jos ilustrează fişierul manifest al clientului şi al service-ului.

Acum putem crea două fişiere XML unul pentru service şi unul pentru componenta client. Pentru service, fisierul XML orderservice.xml va defini

```
package order.client;
import order.OrderService;

public class OrderClient {

private OrderService orderService;

public void setOrderService(OrderService orderService) {
this.orderService = orderService;
}

public void removeService() {
this.orderService = null;
}

public void start() {
orderService.processOrder();
}

public void stop() {
System.out.println("Bundle stopped");
}

}
```

Table 3.5: Listing 1. Componenta client OrderClient

```
package order.impl;

import order.OrderService;

public class OrderServiceImpl implements OrderService {

public void start() {
System.out.println("Order Service registered");
}

public void stop() {
System.out.println("Order Service stopped");
}

public void processOrder() {
System.out.println("Order id: ORD123") ;
}
}
```

Table 3.6: Listing 2. Componenta service OrderService

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Order Service
Bundle-SymbolicName: orderservice
Bundle-Version: 1.0.0
Import-Package: org.springframework.beans.factory.xml,
org.springframework.aop, org.springframework.aop.framework,
org.aopalliance.aop, org.xml.sax, org.osgi.framework,
org.springframework.osgi.service.importer.support,
org.springframework.beans.propertyeditors,
org.springframework.osgi.service.exporter.support,
org.springframework.osgi.service.exporter
Export-Package: order
```

Table 3.7: Listing 3. Service Manifest

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Order Service Client
Bundle-SymbolicName: orderclient
Bundle-Version: 1.0.0
Import-Package: org.springframework.beans.factory.xml,
org.springframework.aop, org.springframework.aop.framework,
org.aopalliance.aop, org.xml.sax, org.osgi.framework,
org.springframework.osgi.service.importer.support,
org.springframework.beans.propertyeditors,
org.springframework.osgi.service.importer,
org.springframework.osgi.service.exporter.support,
order
```

Table 3.8: Listing 4. Fişierul Client Manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean name="orderService" class="order.impl.OrderServiceImpl"/>

</beans>
```

Table 3.9: Listing 5. Fişierul Service XML - orderservice.xml

serviciul bean de implementare a order-ului iar fişierul XML orderservice-osgi.xml va defini interfaţa order service ce urmează a fi implementată. În mod similar pentru fişierele XML client vom defini order client bean şi referinţa către componentele order service. Fişierele service şi client se găsesc în directorul corespunzător META-INF/spring. Codul de mai jos evidenţiază conţinutul fişierelor XML.

Putem de asemenea să comasăm intr-un singur fişier conţinutul definiţiei bean-ului şi cel al OSGi-ului. Nu este nevoie să construim pentru aceasta două fişiere separate. În continuare vom crea două fişiere separate pentru a ilustra mai uşor definirea interfeţei şi configurarea. Gestionarea şi menţinerea acestor fişiere este mai uşoară în felul acesta. Frumuseţea unei configurări cu

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean name="orderService" class="order.impl.OrderServiceImpl"/>

</beans>
```

Table 3.10: Listing 6. Fişierul Service OSGi XML - orderservice.xml

Spring este că ne va permite să testăm serviciul order în afara containerului OSGi.

Pentru a fi capabil sa facem acest lucru va trebuii să instalăm câteva componente Spring DM în Felix.

După cum se poate vedea din cele demonstrate mai sus, fişierul de configurare Felix va avea pachetul Spring DM definit. Putem să descărcăm pachetul Spring DM şi să-l punem în orice director. Odată copiate fişierele jar vom face setările în fişierele de configurare Felix specificând locul acestor fişiere jar care vor fi instalate la pornire.

Acum putem trece la următoarea etapa şi să instalăm pachetele client şi service în mediul de rulare Felix. Odată instalate aceste pachete vom putea să le pornim şi să le oprim pentru a vedea rezultatele. Vom observa că spre deosebire de exemplul precedent când Felix gestiona ciclul de viaţă al pachetelor de data aceasta Sprind DM face acest lucru.

3.3.5 Concluzii

Puterea cadrului de lucru Spring a făcut ca dezvoltarea aplicaţilor OSGi să se simplifice şi să devină mai eficientă. OSGi a revoluţionat felul în care aplicaţiile Java sunt împachetate. OSGi în asociere cu Spring furnizează o fundaţie solida pentru dezvoltarea la scară largă a aplicaţiilor de tip enterprise.

```
felix.auto.start.1= \
file:/felix-1.0.4/bundle/org.apache.felix.shell-1.0.1.jar \
file:/felix-1.0.4/bundle/org.apache.felix.shell.tui-1.0.1.jar \
file:/felix-1.0.4/bundle/org.apache.felix.bundlerepository-1.0.3.jar \
file:/osgi_spring/order/springlib/aopalliance.osgi-1.0-SNAPSHOT.jar \
file:/osgi_spring/order/springlib/jcl104-over-slf4j-1.4.3.jar \
file:/osgi_spring/order/springlib/log4j.osgi-1.2.15-SNAPSHOT.jar \
file:/osgi_spring/order/springlib/org.apache.felix.main-1.0.1.jar \
file:/osgi_spring/order/springlib/slf4j-api-1.4.3.jar \
file:/osgi_spring/order/springlib/slf4j-log4j12-1.4.3.jar \
file:/osgi_spring/order/springlib/spring-aop-2.5.1.jar \
file:/osgi_spring/order/springlib/spring-beans-2.5.1.jar \
file:/osgi_spring/order/springlib/spring-context-2.5.1.jar \
file:/osgi_spring/order/springlib/spring-core-2.5.1.jar \
file:/osgi_spring/order/springlib/spring-osgi-core-1.0.2.jar \
file:/osgi_spring/order/springlib/spring-osgi-extender-1.0.2.jar \
file:/osgi_spring/order/springlib/spring-osgi-io-1.0.2.jar
```

Table 3.11: Listing 7. Partea de configurare Felix

Capitolul 4

HomeRun

4.1 Introducere

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.

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.

Acest software este gratuit şi distribuit sub termenii descrişi de licenţa Apache 2. Există posibilitatea ca servicii care se integrează cu această platformă să fie distribuite contra unui cost de anumite companii.

Acest software poate fi descărcat de la următoarea adresă web:

http://homerun.sourceforge.net/

HomeRun este proiectat ca un sistem modular şi fiecare modul are propria sa documentaţie pentru instalare, configurare şi rulare. Acest ghid are menirea să ne arate modul de utilizare şi operare al acestui soft.

4.2 Cerinţe

Pentru a rula HomeRun avem nevoie de nişte cerinţe minimale. Este posibil însa ca anumite componente să necesite o atenţie specială. Verificaţi cu atenţie cerinţele sistemului dumneavoastră înainte de instalarea unor echipamente speciale. Platforma a fost gândită ca să ruleze întro retea LAN sau WAN conectată prin intermediul unui modem sau al unui router. Oricare computer care rulează HomeRun client sau server va avea instalată JAVA versiunea 5 sau mai nouă. Java este suportată pe o gamă largă de sisteme de operare precum Windows, Apple sau Linux.

4.2.1 Hardware

Partea de server este extrem de modestă în ceea ce priveşte cerinţele hard. Cam orice computer fabricat în ultimii cinci ani poate să ruleze acest soft având suficientă memorie, putere şi spaţiu pe disc. În orice caz performaţa sistemului este dată de performaţa tuturor pachetelor instalate şi rulate simultan.

Pentru comanda echipamentelor electronice este nevoie de un port serial la care acestea să se conecteze. Este posibil ca şi portul USB să fie capabil sa emuleze interfaţa serială. O placă de retea este de asemenea necesară pentru conectarea la mediul distribuit în care rulează aplicaţia. Pentru dispozitivele de automatizare, precum componentele X10, este posibil să avem de asemenea cerinţe speciale.

4.2.2 Software

HomeRun nu necesită nici un alt soft in afară de Java. El 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.

4.3 Instalare

4.3.1 Considerente generale

HomeRun este construit şi gândit întro manieră client - server. Cu alte cuvinte el este compus din două părţi distincte care interacţionează. Instalarea programului se va realiza atât pentru partea de client cât şi pentru cea de

server. Pe lângă arhitectura client - server, HomeRun este gândit modular. Aproape toate componentele sale sunt cuprinse în pachete independente şi sunt adăugate la server după instalare. Aşadar putem spune ca în realitate sunt trei faze ale instalării. Instalarea serverului, clientului şi pe urmă configurarea aplicaţiei.

4.3.2 Alegerea unui server

Serverul trebuie să ruleze pe un singur computer în reţeaua locală şi să fie conectat la reţea permanent. Este de dorit pentru o accesare la distanţă ca server sa fie conectat şi la internet. Un alt considerent la alegerea serverului este faptul dacă maşina suportă echipamentul fizic de comandă. In general este vorba de un port serial pe care unele calculatoare precum laptopurile recente sau MacIntosh-urile nu le 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 sa nu fie unul cu consum mare de energie. HomeRun nu necesită servere puternice iar calculatoarele economice se pretează foarte bine pentru această sarcină.

4.3.3 Instalarea server-ului

După ce aţi ales serverul pe care se va face instalarea vizitaţi pagina web: http://homerun.sourceforge.net/. Urmaţi linkul de download şi pe pagina respectivă veţi găsi un tabel cu două rânduri intitulate Server şi Client. Făceţi clic pentru a descărca cele două arhive zip pe calculatorul dumneavoastră. Alegeţi un director în care să făceţi dezarhivarea. Aplicaţiile client şi server vor fi instalate în două directoare diferite. Vezi în imaginea de mai jos.

Navigaţi prin structura de directori şi fişiere şi alegeţi directorul **fwork** din aplicaţia de server. Întro fereastră DOS tastaţi comanda hrserver.bat care va lansa aplicaţia server. Dacă rulaţi un sistem de operare non-DOS folosiţi comanda hrserver.sh. Dupaă lansare veţi observa în fereastra de consolă următoarea linie:
HomeRun boot strap on port: 8070

Dacă aeastă linie este afişată înseamnă că aplicaţia server a fost lansată cu succes.
În unele cazuri după dezarhivare este necesară resetarea permisiunilor fişierelor de execuţie. Aceasta se face utilizând comanda: $chmod u +x hrserver.sh.

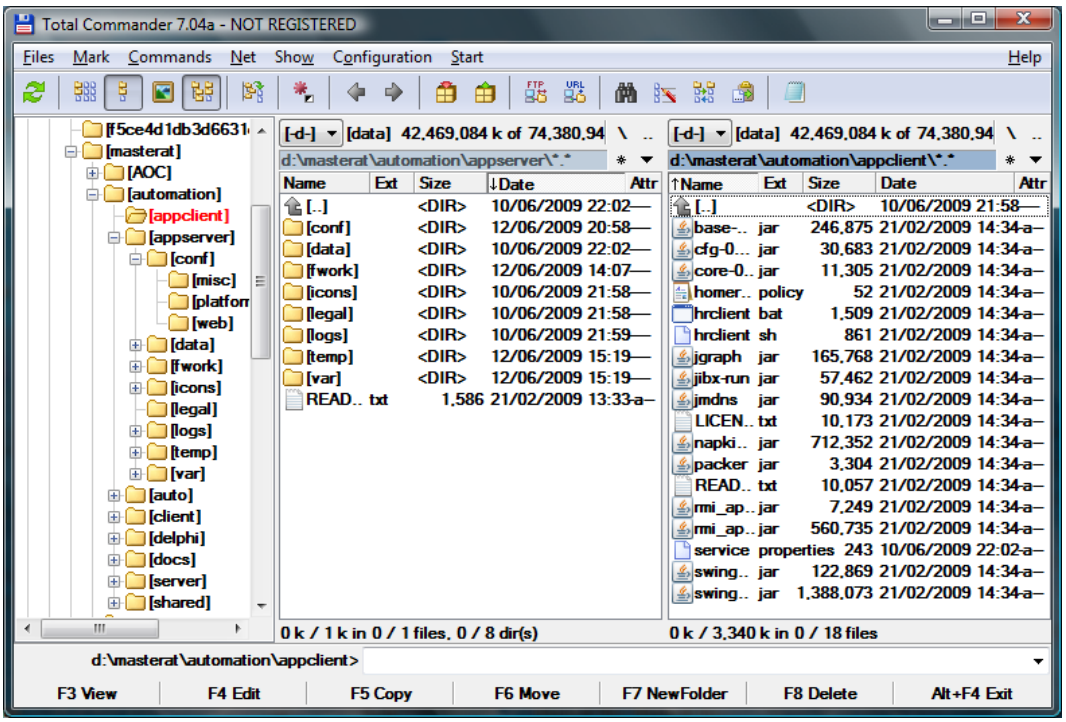

Figure 4.1: Instalarea pe directori

4.3.4 Instalarea clientului

Spre deosebire de aplicaţia server, clientul poate să ruleze pe oricare calculator din reţea. Acesta nu necesită un calculator dedicat care să fie pornit tot timpul. Aplicaţia client poate să ruleze doar atât cât este nevoie de ea pentru a configura sistemul. Se pot instala mai multe aplicaţii client pe mai multe calculatoare insă numai unul este necesar pentru configurarea sistemului. Instalarea se face prin dezarhivarea fişierului client.zip într-un director separat de cel al aplicaţiei server. La fel ca şi la aplicaţia server pentru a da drepturi de execuţie fişierului executabil se rulează comanda: $chmod u +x hrclient.sh. Pentru sistemele Windows lansarea aplicaţiei client se realizează cu ajutorul comenzii hrclient.bat. Dacă programul este lansat cu succes o fereastră asemănătoare celei de mai jos va fi afişată.

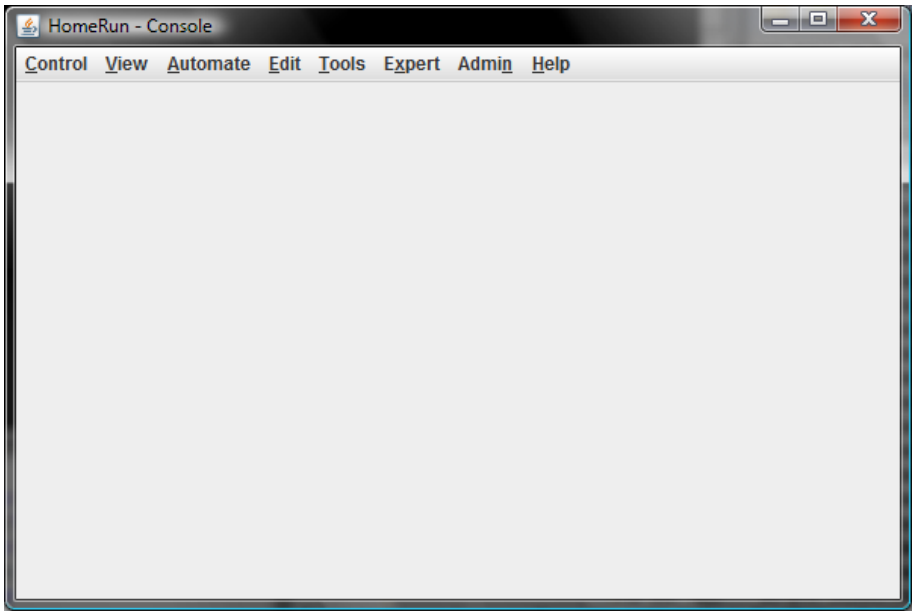

Figure 4.2: Aplicaţia client

După lansare aplicaţia client va arăta în mare la fel pe toate sistemele de operare. Acest lucru se datorează proprietăţilor mediului de programare şi interpretare Java. In Windows puteţi folosi comanda: <hrclient.bat native> pentru a lansa programul cu un "look and feel" Windows.

4.3.5 Configurarea sistemului HomeRun

După instalarea programului server şi client urmează configurarea acestuia. Acest lucru se realizează prin instalarea unor pachete independente care ne permit specificarea anumitor lucuri în acest sistem. Pentru aceasta puteţi vizita website-ul HomeRun unde vă puteţi informa asupra pachetelor disponibile. După ce reţinem numele unui pachet care ne interesează putem să-l lansăm din aplicaţia client. Comanda <Admin -> Setup> va deschide o fereastră asemănătoare cu cea pe care o ilustram în continuare. În Setup, vom alege din meniu comanda: <Package -> List>

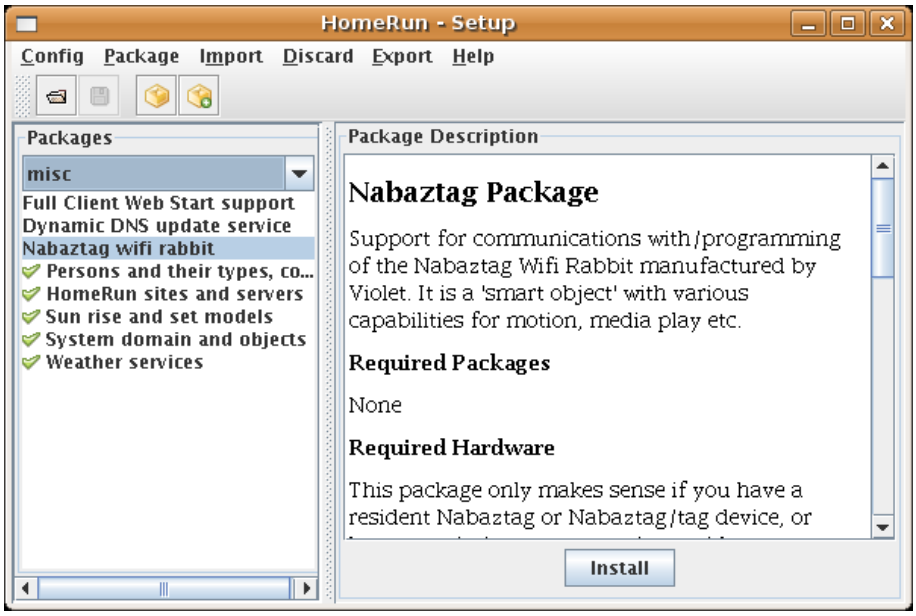

Figure 4.3: Fereastra setup

Pentru a instala un pachet selectaţi pachetul dorit şi apăsaţi butonul <Intsall> aflat în josul ferestrei care descrie pachetul selectat. În unele cazuri se cere configurarea la instalare a pachetului respectiv. În acest caz completaţi câmpurile text şi apăsaţi butonul <Done>. Pentru a accepta valorile implicite apăsaţi butonul <Skip>. Aceste ferestre de dialog ne permit localizarea unor câmpuri text ce for fi afişate în interfaţa UI.

4.4 Obiecte şi componente

4.4.1 Obiecte, Tipuri, Categorii şi Domenii

Aproape toate operaţiile în HomeRun au legătură cu interacţiunea dintre obiecte. Pentru a inţelege cum funcţionează programul trebuie sa înţelegem obiectele.

Un obiect este pur şi simplu o resursă cu un nume în universul HomeRun. Fiecare obiect este unic însă toate sunt instanţe ale unei singure clase cunoscută sub numele de tipul obiectului. Ca şi analogie putem spune că toţi indivizii sunt unici dar sunt cu toţii membrii ai speciei umane. Tipul obiectului este important atunci când dorim sa creăm noi obiecte (indivizi sau instanţe) pe care să le adăugăm sistemului nostru. Aplicaţia dispune de o fereastră intitulată 'Object Editor' care afişează toate obiectele cunoscute de sistem. Tot în această fereastră ni se dă posibilitatea să creăm noi obiecte de un anumit tip. Un exemplu puteţi vedea în fereastra care urmează.

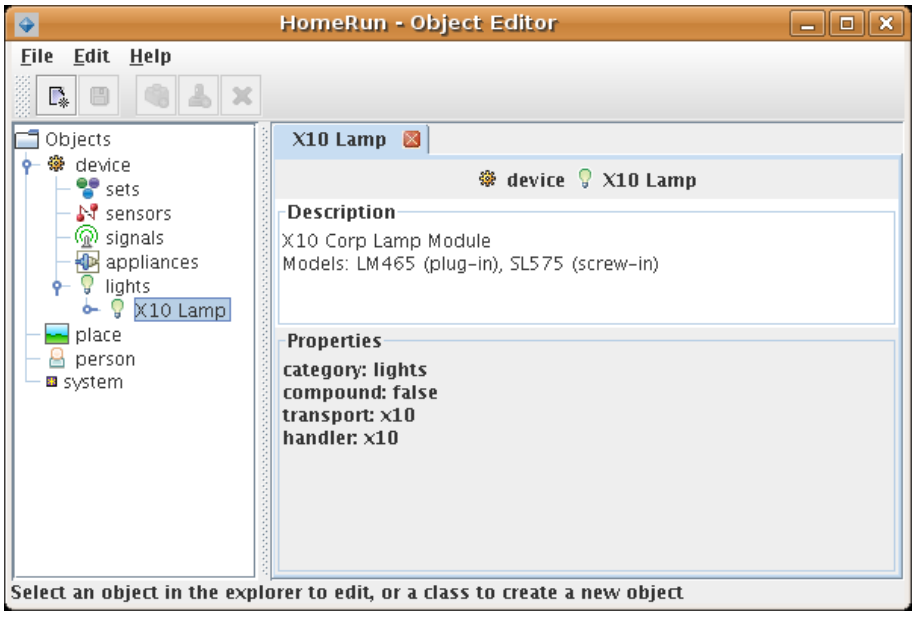

Figure 4.4: Proprietati X10 Lamp

Tabul etichetat 'X10 Lamp' afişează informaţii despre tipul obiectului adaugând o descriere şi informaţii despre diverse proprietăţi. Aceste proprietăţi sunt moştenite de către toate obiectele create.

Tipurile sunt importante pentru crearea obiectelor însă în majoritatea situaţiilor ne interesează categoria în care se află un obiect. Observaţi că tipul obiectului 'X10 Lamp' aparţine categoriei 'lights'.
Aşadar 'Categoria' este modalitatea prin care se grupează toate obiectele în interfaţa UI a programului. Acest lucru ne permite de exemplu să tratăm toate luminile ca şi un grup logic indiferent că discutăm de lumini X10, Insteon sau Z-Wave.

Observaţi că tipurile sunt şi ele grupate în domenii. In HomeRun 'Domains' este cel mai înalt nivel de grupare a tipurilor, categoriilor şi obiectelor. De fapt toate obiectele sunt relative la domeniile în care se încadrează. Sunt doar câteva domenii, însă ele ne ajută să organizăm toate obiectele din sistemul nostru.

4.4.2 Anatomia unui obiect

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

a) Proprietăţi - Acestea sunt în principal descrieri ale atributelor care pot fi generice (true of the type) sau specifice (true of the individual). A fi biped e o proprietate umană generică, dar culoarea părului e o proprietate specifică. Am vazut în imaginea precedentă proprietăţile generice ale X10 Lamp, iar aici e una din cele specifice:
De vreme ce dispozitivele electronice X10 sunt accesate de către codul de casă şi codul dispozitivului, acestea sunt proprietăţi specifice. Unele proprietăţi sunt obligatorii (marcate mai sus cu asterisc), altele sunt opţionale (de exemplu, Lamp Wattage). Unele au valori implicite, spre ex. pictograma utilizată pt. a reprezenta lumina primeşte valoarea implicită de la tipul părintelui.

Proprietăţile sunt folosite pentru atribute ale lucrurilor care nu se schimbă, cum sunt cele statice. Pentru atribute care se schimbă, de ex. dacă lumina este activată sau dezactivată, HomeRun foloseşte o altă componentă, numită model, pentru a capta valoarea acestor atribute variabile care vor fi discutate în propriul capitol.
b) Controller-e
Interesul nostru major într-o clasă largă de obiecte precum este cel al luminilor este acela de a le controla. Instalaţia de iluminare, spre exemplu, are un întrerupător pe care îl aprindem/stingem. Funcţie de tipul lui, un obiect poate avea zero, unul sau mai multe controller-e.

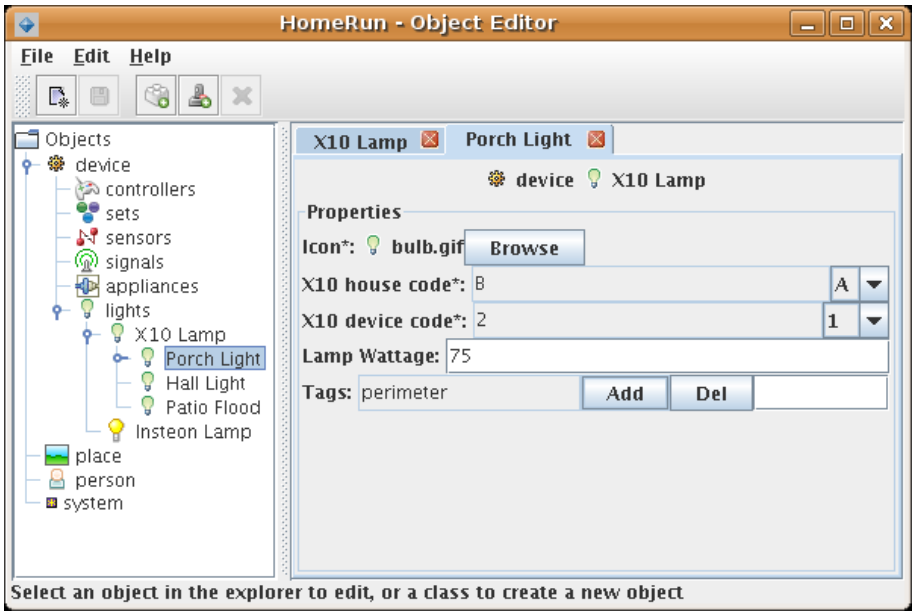

Figure 4.5: Propritati specifice ale X10 Lamp

HomeRun foloseşte o reprezentare şi terminologie standard pentru controllere; aceasta fiind un set conectat de operaţiuni care se pot efectua asupra unui obiect.

Conectat înseamnă a fi cu adevărat logic legate, la modul că fiecare operaţiune se referă la acelaşi lucru. Un corp de iluminat poate avea avea două stări (aprins/stins) dar amândouă afecteaza puterea de a lumina. Fiecare din aceste operaţiuni se numeşte punct astfel că întrerupătorul are două puncte de control. Tipurile de puncte de control mai complexe pot avea valori adiţionale asociate (de exemplu, citirea de la tastatură), iar acestea sunt menţionate ca intrări.

Pentru numeroase tipuri de obiecte, şi ne gândim din nou la corpul de iluminat - este definitoriu ca acesta să aibă un anumit controller. Când acest lucru este adevărat, controller-ul va fi asociat automat cu obiectul când îl creăm. Pentru altele, controllere-le sunt opţionale, în sensul că fie nu toate obiectele de acel tip le au (de exemplu, reglajul de intensitate al luminii), fie nu ne interesează să folosim aplicaţia HomeRun relativ la ele. Object Editor poate fi utilizat pentru a adăuga un controler unui obiect în aceste cazuri.

c) Emiţători

Numeroase obiecte se comportă ca 'observatori' sau 'raportori'. Un exemplu ar putea fi detectorul de fum din locuinţă. Deşi aceste obiecte nu au

cu adevarat controller-e (poate un buton de testare a bateriei), au în schimb proprietatea de a suna alarma în anumite condiţii. HomeRun numeşte aceste activităţi produse independent evenimente şi numeşte facilitatea de a produce un set legat de evenimente emiţător. Din nou, depinzând de tipul obiectului, poate deţine zero, unul sau mai mulţi emiţători. Spre deosebire de controller-e, emiţătorii nu pot fi opţionali: dacă aparţin unui obiect, vor fi adăugaţi lui când creaţi unul.

Aici este o imagine a unui obiect cu controller-e şi emiţători:

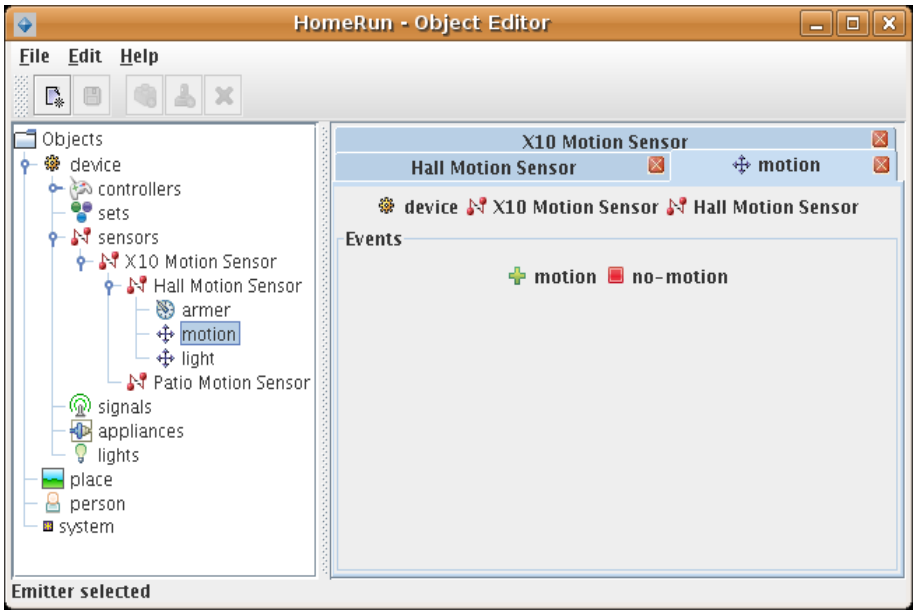

Figure 4.6: Editorul de obiecte

Senzorul de miscare X10 are 2 emiţători, unul pentru evenimente mişcare sau ne-mişcare (figură), şi unul pentru evenimentele lumină sau întuneric.

d) Utilizarea Controller-elor şi a Emiţătorilor

Până în prezent am discutat doar ce sunt aceste componente şi cum le gestionăm în editoare. Desigur, întrebarea mai importantă este cum le utilizăm în software. Amânăm discuţia despre evenimente pentru capitolul următor, dar pentru controller-e aceasta este un UI (User Interface) pe care îl putem folosi pentru a manipula direct controller-ele obiectelor.

Se numeşte Control Panel (Control - >Panou de la Consolă) şi arată astfel: Vezi figura.

Obiectele sunt grupate pe categorii şi puteţi apăsa butoanele reprezentând punctele de control.

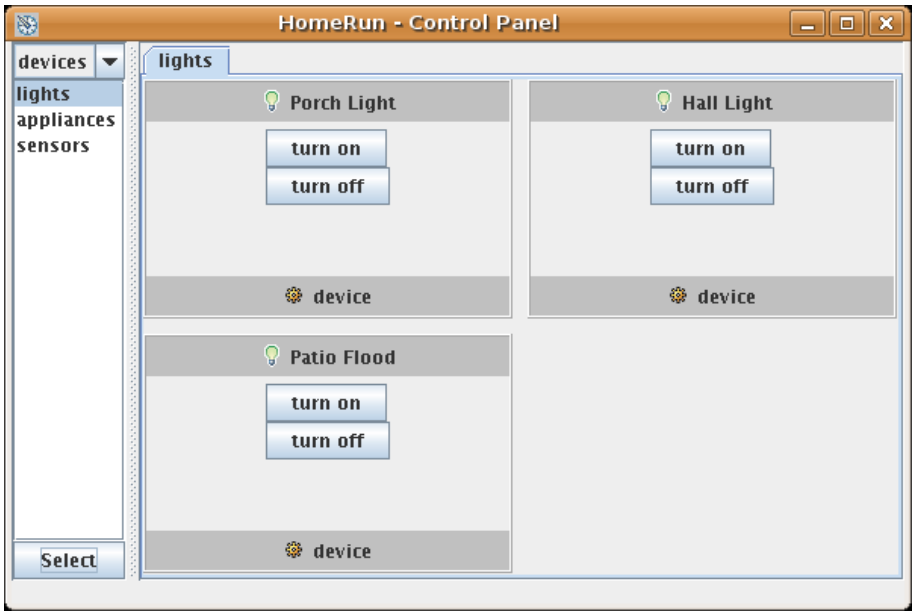

Figure 4.7: Control Panel

4.5 Acţiuni şi Legături

În ultimul capitol, am explorat lumea obiectelor din HomeRun şi părţile care le alcătuiesc. Am făcut şi o scurtă trecere în revistă a modului în care obiectele sunt expuse în UI, astfel încât modul lor de control să poată fi manipulat. Dar cum procedăm dacă dorim ca modul de control să opereze când nu suntem prezenţi, ca parte a unui proces automatizat?

Răspunsul în HomeRun este să foloseşti 'acţiuni', care sunt numite colecţii ale "object control invocations" şi mai mult de atât puteţi defini acţiuni potrivite oricărui scop, oar acţiunile pot fi incluse în multe alte construcţii. Să începem un tur al acţiunilor cu una foarte simpla.

4.5.1 Acţiuni Simple

Avem un obiect 'Lumina din Verandă' şi ştim că acesta are un mod de control 'întrerupător'. Acest mod de control are două 'puncte' - închis şi deschis. Cea mai 'atomică' operaţiune apoi, poate fi exprimată prin specificaţia (Porch Light, switch, on). HomeRun numeşte aceste specificaţii 'tasks' (sarcini) şi ele constituie cel mai mic element de lucru pe care sistemul îl poate executa.

O acţiune în forma ei cea mai simplă este o listă numită "de sarcini"

(minim una).
Haide să numim (Porch Light, switch, on) cu numele imaginar de "Porch Light On". Acum putem folosi această stenografie oriunde avem de selectat un obiect, mod de control şi punct. Spre exemplu, putem pune acţiuni în Control Panel în locul de reprezentare al obiectelor. Un singur task nu realizează prea multe dar dacă concatenăm mai multe la un loc, acţiunile devin mai puternice.

HomeRun furnizează un UI intuitiv pentru lucrul cu acţiuni, numit 'Action Builder' (din Consola: Edit -> Action). Aici este o imagine a acţiunii noastre:

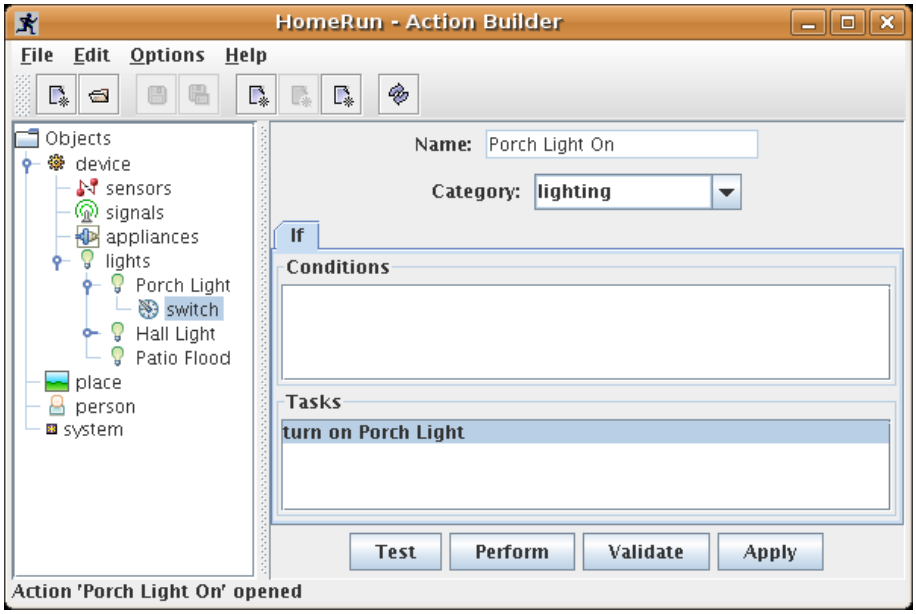

Figure 4.8: Action builder

Adăugarea sarcinilor este numai o chestiune de navigare în explorer panelul obiectului în stanga obiectului de interes, şi apoi facem click pe modul de control pe care dorim să-l adaugăm. Odată ce acţiunile sunt definite, ele pot fi invocate manual în Control Panel, dar tipic sunt invocate automat în diferite moduri. La aceste moduri ne referim adesea ca la 'triggers' (trăgaci) în sensul că apariţia lor dă foc acţiunii. HomeRun preferă termenul de 'legături', însemnând că acţiunea e legată de o circumstanţă a cărei apariţii o porneşte. Desigur, aceeaşi acţiune poate fi legată de diferite şi multiple circumstanţe. Să examinăm tipurile primare de legături în HomeRun, şi cum să le conducem.

Pentru început, putem lega o acţiune de o dată şi oră specifice, non-repetabile: Vreau ca acţiunea să aibă loc în data de 25 Iunie, ora 19:30. Acestea sunt cunoscute ca şi legături de calendar, de vreme ce ar fi scrise firesc în calendarul tău. HomeRun furnizează un program simplu pentru crearea legăturilor de calendar (din Consola: Automate - >Calendar):

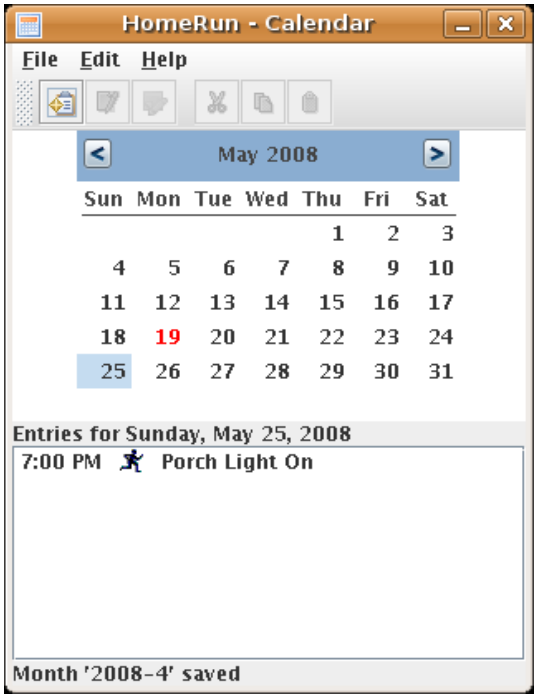

Figure 4.9: Calendar

Click pe dată pentru a afişa orice legături curente, apoi apăsăm butonul 'New Entry' pentru a adăuga una (ele sunt salvate automat pe masură ce le adaugăm). Poţi adăuga entries pe viitor, şi acţiunea e executată doar odată. Oricum, introducerea rămâne vizibilă după ce acţiunea a fost executată, aşa încât poţi revedea introducerile precedente.

Un alt tip de legătură foarte des intâlnit se referă la o anume unitate de timp, dar repetitivă la un interval standard. Pentru cei mai mulţi dintre noi, intervalul important în viaţa noastră este săptămâna, asa că HomeRun furnizează o unealtă pentru a conduce acţiuni repetate săptămânal, care se numesc legături de orar. Aici este un screenshot al programului care conduce aceste legături, numit Planificare (Scheduler) (din Consola: Automate -> Orar):

Aici am creat un orar numit Săptămâna Normală de Lucru şi am adăugat legăturilor zilei de luni acţiunea Porch Light. Desigur, poţi avea atâtea

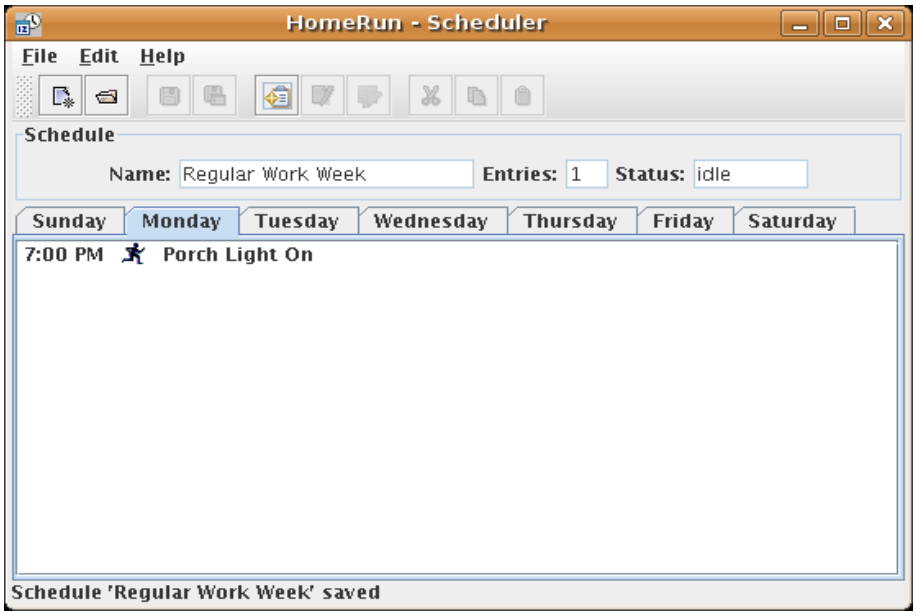

Figure 4.10: Orar - scheduler

legături câte doreşti (chiar în acelaşi timp). Spre deosebire de Calendar, putem crea numeroase orare diferite (de aceea asignăm un nume fiecăruia). Aceasta este o caracteristică foarte puternică, de vreme ce putem schiţa un orar pretabil fiecărei ocazii (de exemplu unul pentru vacanţă etc). , pe care doar trebuie să-l activăm cand e necesar, mai degrabă decât să reprogramăm întreg sistemul de câte ori rutina ni se schimbă . Un singur orar este activ la un moment dat şi observăm că Săptămâna Normală de Lucru are acum status-ul de inactiv.

Legătura finală pe care vrem s-o analizăm este legătura eveniment. Legătura eveniment determină execuţia unei acţiuni, nedepinzând de vreun moment specific, ci doar când se întamplă evenimentul emiţător. De vreme ce nu ştim când sau dacă un anume eveniment se va întampla, legăturile eveniment au o natură condiţională comparată cu legăturile calendar şi orar, care se întamplă garantat la momentul fixat. După cum banuiţi, HomeRun are un UI care să vă ajute să conduceţi aceste legături, care cheamă aplicaţia Planner, de vreme ce grupuri de evenimente legături relaţionate sunt cunoscute ca planuri (în acelaşi fel în care grupuri de legături orar sunt cunoscute ca orare).

Planner (din Consola Automate -> Plan) arată astfel; Vezi figura.

Puteţi observa similaritatea cu Planificare UI, care este intenţionată.

Oricum în acest caz, tab-ul este etichetat cu numele şi icon-ul evenimen-

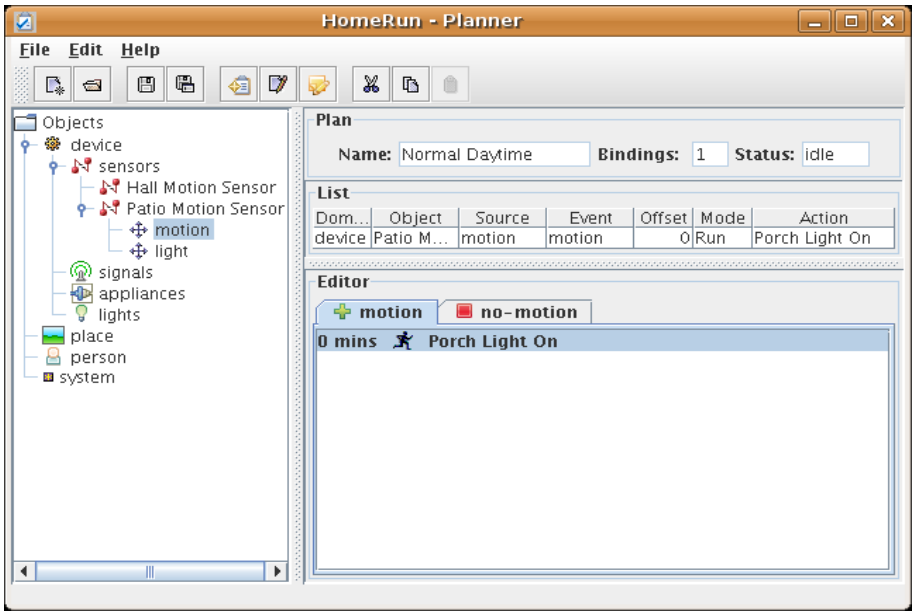

Figure 4.11: Planificator

tului emiţător care a fost selectat în obiect explorer-ul din panelul stânga. Notaţia "0 mins" înainte de numele acţiunii se referă la capacitatea de a întârzia execuţia acţiunii pentru un timp configurabil. Observaţi deasemeni că asemenea orarelor, planurile trebuie să fie activate înainte ca legăturile să funcţioneze: dar spre deosebire de orare, puteţi avea mai multe planuri active simultan. Puteţi chiar inlănţui cele două în moduri utile. Spre exemplu, puteţi avea un plan (=set de acţiuni) bazat pe detectarea mişcării, valid atunci când dormiţi şi altul atunci când sunteţi treji. Puteţi crea intrări (entries) în orarul vostru pentru a porni planul noaptea, apoi il opriţi şi îl porniţi pe celălalt dimineaţa.

4.6 Modele şi Scene

Am învăţat că obiectele cu atribute variabile au folosit componente HomeRun numite modele. Obiectele pot avea orice număr de modele asociate lor, şi revenim acum cu detalii asupra modelelor şi domeniilor de utilizare. Reamintiti-vă că modelul reprezintă orice atribut variabil (mutabil) al unui obiect. Majoritatea software-ului de automatizare tratează această clasă de date într-un mod destul de limitat, de exemplu, ca având un dispozitiv "status", cu unele valori fixe (închis sau deschis). Problema este faptul că nu

reuşeşte să surprindă mai multe modalităţi complexe pentru a reprezenta starea internă, precum şi faptul că de multe ori am vrea să urmărim mai multe variabile, nu doar un singur "status". Acest deficit este de câteva ori abordat prin adăugarea de suport pentru limbaje de scripting, dar aceasta ridică în mod semnificativ nivelul de calificare şi complexitate cerut pentru a utiliza sistemul.

HomeRun vă oferă o alternativă simplă, flexibilă şi foarte puternică în privinţa modelelor. Puteţi defini propriile modele şi asocia cât mai multe dintre ele cu un obiect, după cum doriţi. Să luăm un exemplu simplu, Lumina din Verandă. Ne-am aştepta la un model pentru cazul în care lumina este aprinsă sau stinsă. Dar poate de asemenea, vrem să ştim când a fost ultima oară când a fost pornit, sau chiar folosit. Poate dorim doar să calculăm de câte ori întrerupătorul a fost operat, sau cine a fost ultima persoană care l-a aprins. Există cinci modele simple pe care le-am putea dori, iar noi le putem crea pe oricare sau toate dintre ele şi să le ataşăm oricărui obiect dorim, toate acestea fără folosirea unui limbaj scripting.

4.6.1 Utilizare

Înainte de a studia mai amănunţit gama de modele, trebuie să adresăm prima întrebare: cum le utilizaţi în HomeRun? Există două moduri: puteţi să le inspectaţi într-un UI, sau puteţi utiliza valorile lor pentru a afecta comportamentul acţiunilor.

4.6.2 Scene

Scenele sunt UI construcţii în HomeRun care vizualizează modele. Puteţi selecta orice număr de modele, şi puneţi-le pe o "pânză" cu câteva machete standard. Apoi, folosind un program separat, vă selectaţi scena şi se afişează vizualizarea formatată a fiecărui model utilizând valoarea actuală. Iată un screenshot de aplicaţie Scene Viewer (View-> Scene de la Consolă):

Această scenă foarte simplă numită "Unde este toată lumea" e alcătuită din şapte modele distincte: pe rândul de jos e un model pentru ziua săptămânii, data şi ora curentă. Pe rândul de sus, există un model "locaţie" pentru fiecare din cei patru rezidenţi imaginari. Vizualizarea aici este o pictogramă reprezentând locaţia lor actuală, care, în toate cazurile este "acasă". În engleza simplă, toţi Jetsons sunt acasă.

Alte câteva UI de control pentru a observa în Scene Viewer: săgeata circulară, care înseamnă "refresh" vă permite să actualizaţi scena cu cele mai

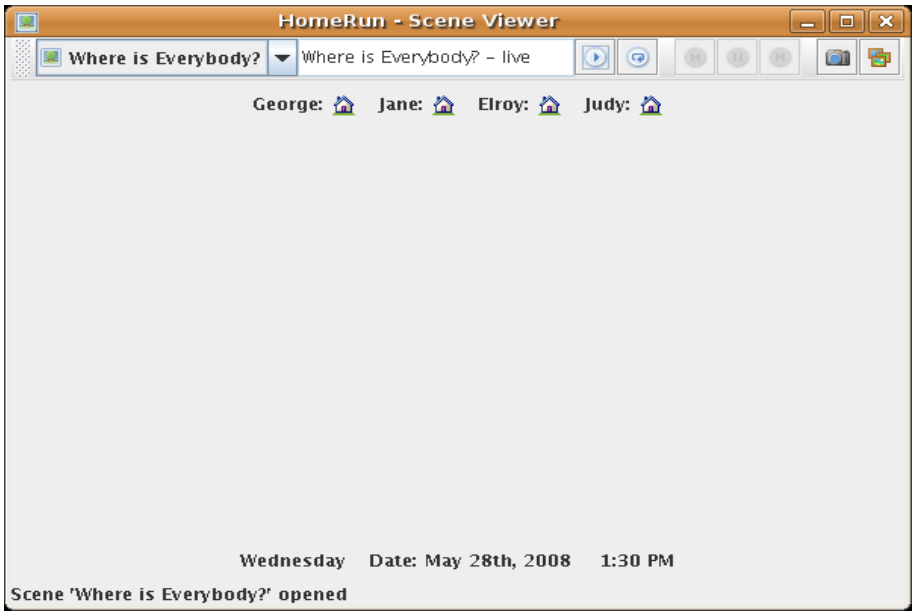

Figure 4.12: Fereastra Scene Viewer

recente valori ale modelului. Scenele reprezintă o evaluare a modelului la un moment dat (când atingeţi butonul 'play'), dar puteţi să le actualizaţi cu refresh. Butonul cu pictograma 'aparat de fotografiat' (captare) vă permite să înregistraţi valorile unei scene pentru o revedere ulterioară - gândiţi-vă ca la o "data instantanee" a scenei. Aceste "snapshots" pot fi vizualizate folosind butonul cu "mai multe imagini snapshots"' - care va deschide un dialog pentru a alege între capturile disponibilele ale scenei selectate. Reţineţi, de asemenea, că aveţi posibilitatea de a defini o acţiune pentru a lua un instantaneu, aşa că aţi putea adăuga aceasta la un orar (de exemplu, cine este acasă la ora 17:00 în fiecare zi?).

Scene Designer are aspectul familiar pentru editoarele HomeRun, cu un obiect Explorer în panoul din stânga. Aici suntem pe cale de a adăuga locaţia lui Astro la scenă.

Să aruncăm o privire la UI-ul pentru crearea de scene (Edit -> Scene de la Consolă):

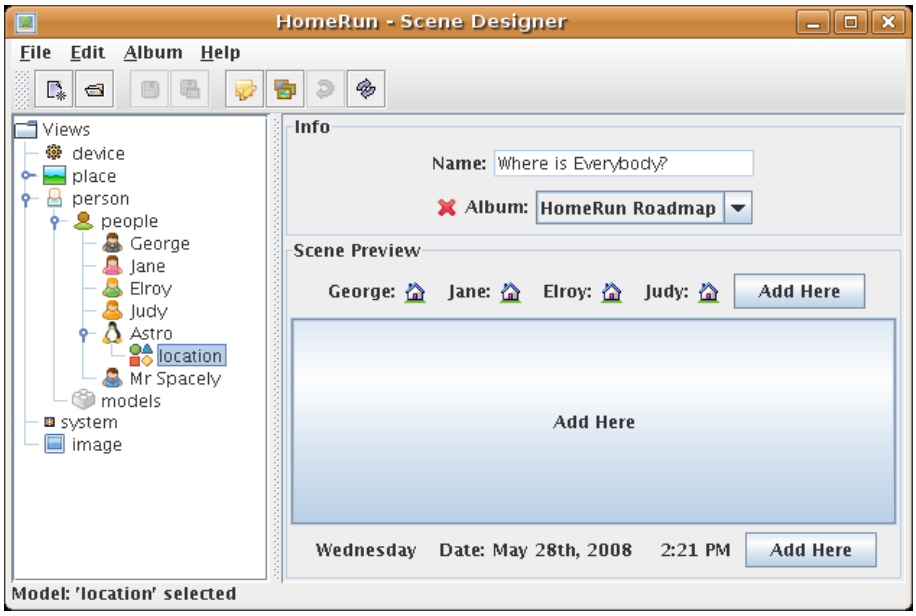

Figure 4.13: Fereastra Scene Designer

4.6.3 Condiţii de acţiune

Alt mod în care folosim modele este de a adăuga logică condiţională acţiunilor noastre. Putem face ca performanţa acţiunii să depindă de 'status-ul' modelelor, indiferent de momentul în care acţiunea este efectuată. S-ar putea să fi observat o secţiune în generatorul de acţiuni etichetată "Conditions". Aici putem adăuga orice număr de teste ale modelului pentru a controla modul de lucru al acţiunii. Iată un exemplu în UI:

Noi tocmai am adăugat o condiţie pentru acţiunea noastră, în aşa fel încât Lumina din Verandă să fie în funcţiune numai dacă George nu-i acasă. Acţiunea Builder permite o logică foarte sofisticată, unde putem avea mai multe condiţii, condiţii care alternează (această condiţie sau acea condiţie), precum şi alte clauze de ansamblu (ALTFEL DACĂ conditii, activităţi). Dacă apăsaţi pe butonul 'Test', UI va afişa dacă condiţiile sunt adevărate.

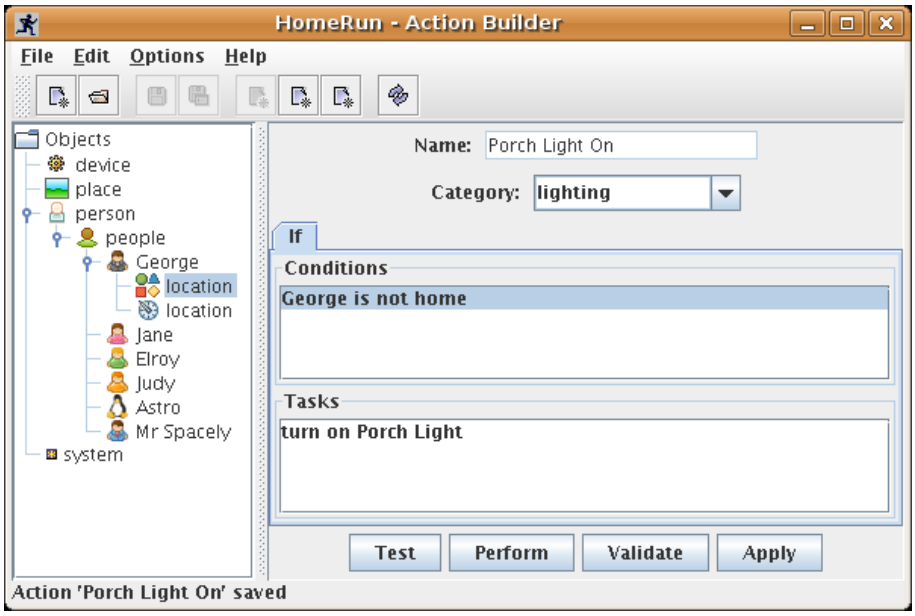

Figure 4.14: Fereastra Action Builder

4.6.4 Tipuri

HomeRun face să fie simplu de lucrat cu modele, prin stabilirea familiilor sau tipurilor de modele şi suportă multe built-in operaţiuni în funcţie de tip. Să vedem cum funcţionează în practică. Există 6 tipuri de model în HomeRun, pe care le vom trece rapid în revistă.

Modele Număr

Poate că cel mai simplu model de înţeles este modelul număr, care reprezintă un simplu număr fără nici o unitate. Ne-ar folosi acest model când dorim să captăm un singur număr întreg. Există un mic set de tipuri de număr, cum ar fi "contor", sau "variabilă" asociate cu fiecare model de număr, care determină modul în care modelul poate fi informat. De exemplu, un "contor" poate fi informat doar folosind 3 verbe: "creşte", "descreşte" şi "resetează" ('increment', 'decrement' şi 'reset').

O altă caracteristică la îndemână a unui model de 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. Având limite face să fie foarte uşor şi intuitiv să adaugi condiţii de acţiune de genul: Să presupunem că avem un obiect cu un 'duty - cycle' de 100, ceea

ce înseamnă că trebuie să reparăm sau să verificăm un subansamblu după 100 utilizări. Am putea crea o limită numită "control", cu o valoare de 100, şi apoi într-o acţiune, am putea cu uşurinţă exprima restricţia ca:

"în cazul în care utilizarea numără este sub limita de control, continuă cu sarcinile, altfel notifică-l pe Bob prin e-mail".

Modele Valoare

În contrast cu cele numerice, modelele valoare au o unitate - precum grade Fahrenheit - şi de obicei reprezintă cantităţi fizice măsurate, mai degrabă decât valorile calculate sau atribuite tipice de modele de numar. Unităţile sunt descrise de tipurile de valoare, care includ nume, simboluri, pictograme etc. astfel incât vizualizări ale acestor modele pot expune unitatea de informaţii succint.

Pentru că adesea există imprecizie sau fluctuaţie în măsurătorile fizice care stau la baza modelelor de valoare, acestea nu utilizează limite. Imaginaţivă, de exemplu, în cazul în care aţi folosit o condiţie de acţiune că "dacă temperatura > 68", şi citirile v-au venit la 67, 69, 68 pe eşantioane succesive. O modalitate mai naturală de a manipula acest caz, este de a utiliza intervale de valori, care sunt ceea ce foloseşte HomeRun cu modelele de valoare. Astfel s-ar putea defini un şir ca "zona de confort" 67 - 72 de grade. Apoi condiţia de acţiune ar putea citi: "Dacă temperatura rămâne în zona de confort", etc

Modele de Stare

Modelele de stare reprezintă situaţii care consistă într-una dintr-un număr fix de stări, care sunt descrise de nume şi pictograme, dacă doriţi. Cele mai simple modele de stare sunt cele care se numesc variabile boolene (adică stări TRUE/FALSE). De exemplu, un model de stare ar putea reprezenta locaţia ta: esti acasă sau nu. Un model de stare ar exprima acest lucru ca două stări: acasă şi plecat, ar furniza mai multe nume pentru tranziţia de la o stare la alta: "sosire" este trecerea de la distanţă la domiciliu, "de plecare" de acasă la distanţă. Modelele de stare pot avea desigur mai mult de două stări, dar poţi să rezolvi destul de multe cu modelele de stare binare.

Modele Set

Când aveţi o listă de una sau mai multe valori, consideraţi un model set. Ţineţi minte că modelele de stare pot fi într-un set fix de stări; seturile pot conţine de la zero la un număr nelimitat de lucruri. Fiecare lucru este

reprezentat de un nume, şi singura restricţie într-un set este că acelaşi nume nu poate apărea de două ori.

Modele Calendar

Aceste modele sunt folosite pentru a reprezenta timpul exprimat într-un calendar. Modelele Calendar au câteva sub - tipuri, după cum urmează. "zi de săptămână": Luni - Duminică "zi a lunii": 1-a - 31-a. "luna anului": Ianuarie - Decembrie.

Modele de Timp

Modelele de timp, de asemenea, exprimă o perioadă de timp, dar după cum sunt văzute de un ceas, mai degrabă decât un calendar. Aceste modele pot fi, de asemenea, utilizate pentru a reprezenta intervale de timp, la fel ca şi "count down timers".

4.6.5 Modele de Informare

Un subiect foarte important peste care am trecut în acest tur al modelelor este întrebarea simplă: cum sunt actualizate şi întreţinute valorile modelului?
Cum ştie modelul locaţie al lui George când vine acasă?

HomeRun utilizează termenul de "informează" un model, şi are un sistem prin care unui model i se dau unul sau mai mulţi "informatori". Iată care sunt mecanismele de bază la lucru:

Informatorul Intern

Un obiect poate avea un model care este strâns legat de identitatea lui: un bun exemplu este obiectul "Soare" cu modelele "rasarit" şi "apus" pentru răsăritul şi apusul soarelui. Aceste valori pot fi calculate (dacă longitudinea şi latitudinea sunt cunoscute), deci nu trebuie să fie nici un fel de informaţii externe pentru ca modelul valoare să fie determinat. Acestea sunt modele de informare interne şi nu trebuie să faceţi nimic pentru a asigura valoarea lor (cu excepţia de configurare iniţială incluzând setarea coordonatelor).

Control Informator

Când adăugaţi un model unui obiect în Object Editor, există 3 proprietăţi pe care le-aţi putea atribui, şi este o etichetă "Expune Control". Dacă bifaţi

această casetă, HomeRun creează un "control virtual" al cărui scop este pur şi simplu de a actualiza modelul. I se da acelaşi nume ca şi modelului. Acest lucru vă permite în Control Panel sau într-o acţiune să updataţi un model dvs. inşivă.

Informator Proiectat

Probabil, cel mai comun caz, din nou set cu o proprietate numită "Proiectul Informator" verificată în Object Editor, acest lucru vă va permite să asociaţi o valoare unui model în contextul unei acţiuni în Action Builder.

Wired Informator

O caracteristică avansată a lui HomeRun este abilitatea de a "wire up" componentele intr-un obiect. Un bun exemplu ar fi power status model pt. Lumina din Verandă. Are sens ca ori de câte ori comutatorul este activat, modelul ar trebui să fie actualizat pt. a fi "on". Putem legifera aceasta prin setarea unui "circuit" între control şi model. UI pentru acest lucru este reglementat în capitolul Advanced Topics.

4.7 Administrare

HomeRun tinde să fie foarte usor de rulat şi gestionat şi furnizează mai multe instrumente şi UIs pentru a vă ajuta să vă administraţi sistemul. Înainte de a face un tur, iată o scurtă prezentare a arhitecturii HomeRun ce se referă la administrare.

4.7.1 Server Life -Cycle

După cum am menţionat anterior, software-ul HomeRun este divizat într-un software server şi client care comunică cu acesta. Serverul are un ciclu de viaţă intern, în sensul că poate fi în una din cele doua stari pe care le puteti controla. Când porniţi serverul cu fişierul batch hrserver.bat, nu rulează complet, ci numai într-o stare cunoscută sub numele de bootstrap state. Când, sunt în această stare, cele mai multe aplicaţii client nu pot funcţiona. În fapt, în cazul în care Consola se lansează prima dată, veţi observa că aproape toate opţiunile din meniu sunt dezactivate, cu excepţia celor din grupul Admin.

Din fericire, una dintre opţiunile activate din Admin e Server -> Start care aduce serverul la o stare pe deplin operaţională. De cele mai multe ori, acesta va rămâne în această stare, şi voi în mod normal, nu va trebui

sa-l opriţi. Există, cu toate acestea câteva cazuri în care, după instalarea de pachete noi, veţi fi instruiţi pentru a reporni serverul (de exemplu, stop şi porneşte), ca şi în cazul altor instalări de software.

4.7.2 Utilizatorii şi Autentificarea

Un alt concept important de înţeles este modelul de utilizatori "totul - sau - nimic" adoptat de către HomeRun. Când este instalat prima data, HomeRun este considerat a fi în stare deschisă, ceea ce înseamna, pur şi simplu: nu are noţiunea de utilizatori distincţi de sistem, asa că nu va cere să vă identificaţi sau să vă autentificaţi. În consecinţă, orice persoană care utilizează software-ul poate face orice. De fapt, s-ar putea să vrei să laşi sistemul deschis pentru o oarecare perioadă de timp, mai ales în timp ce inveţi: nu va trebui să te deranjezi să te loghezi pentru a utiliza sistemul. În cazul în care toţi utilizatorii sunt cunoscuţi şi de încredere, e mai simplu să rămână deschis întotdeauna pentru simplitate. Cu toate acestea, veţi vrea să rezervaţi anumite drepturi anumitor persoane, şi HomeRun suportă un rol - modelul bazat pe autoritate. În momentul în care defineşti un singur utilizator (de la Consola de administrare -> Utilizator -> Administrare), atunci sistemul este considerat a fi în stare închisă şi utilizatorii trebuie să se identifice şi să se autentifice ei înşişi ca să funcţioneze sistemul. Aceasta, de asemenea, creează un potenţial risc; ce se intamplă în cazul în care primului utilizator creat îi lipseşte autoritatea administrativă? Atunci, cum mai creăm administratorul? Pentru a evita acest lucru, HomeRun insistă asupra faptului că primul utilizator definit să fie un administrator, pentru ca el sau ea să poate defini apoi utilizatorii de mai mică putere. Puteţi reda întregul proces şi în sens invers: Asiguraţi-vă că dacă ştergeţi toţi utilizatorii pentru a reveni la starea deschisă, ultimul utilizator rămas este un administrator.

4.7.3 Roluri

HomeRun înţelege un set foarte simplu de roluri, şi limitează accesul la anumite programe şi interfeţe UI bazate pe aceste roluri. La crearea unui utilizator, trebuie doar să atribui unul dintre roluri pentru el/ea. Rolurile sunt:

- admin poate administra sistemul
- editor poate utiliza aplicaţiile editor pentru a crea noi obiecte şi componente

- user poate rula panoul de control sau mesaje
- other, de asemenea utilizat pentru utilizatorii anonimi (= nu este conectat)

Puteţi nota faptul că rolurile nu se autoinclud, în sensul că un administrator nu este automat şi un editor, etc. Trebuie să alocaţi fiecare rol pentru fiecare utilizator. Utilizatorii sunt gestionaţi într-o aplicaţie UI numită Manager (Admin -> Utilizator -> Gestionare de la Consolă) unde tot ce ai de făcut este să aloci numele utilizatorului şi să stabileşti rolurile de mai sus. Noilor utilizatori le sunt date parola implicită "homerun", dar ar trebui să-şi selecteze propria parolă într-un dialog (Admin -> User -> Schimbare parolă).

4.7.4 Îndatoriri Administrative

Am discutat deja una dintre principalele sarcini, care este selecţia şi instalarea pachetelor HomeRun, care determină funcţionalitatea sistemului. Cele mai multe pachete necesită configurare mai amanunţită, care este accesată prin intermediul programului de instalare. Puteţi, de asemenea, importa ambele imagini (JPEG, GIFs) pentru utilizarea în scene, precum şi pictograme pentru a fi utilizate în diferite locuri, de exemplu obiecte, modele de stare, modele etc. Setup are, de asemenea, opţiunile pentru a şterge vechi capturi = snapshots.

4.7.5 Conectarea

HomeRun are un sistem flexibil de logare şi oferă un instrument, jurnalul Viewer, jurnal de gestionare a datelor. În mod normal, sunt păstrate trei jurnale:
- Jurnalul de sistem păstrează date despre funcţionarea internă a serverului.
- Jurnalul de activitate are un registru al utilizarii sistemului HomeRun, atunci când sunt acţiuni sau apar evenimente etc.
- În cele din urmă, un jurnal de utilizator este menţinut în scopul exclusiv de a înregistra date pe care le furnizaţi în acţiuni.

Jurnalul de vizualizare are trei file, unul pentru listarea fişierelor jurnal (unul pe zi pe tip), jurnalul de înregistrări (demonstrat), precum şi o listă a jurnalelelor arhivate. Instrumentele includ crearea de arhive (care comprimă jurnalele şi economisesc spaţiu pe disc) şi căutarea datelor de jurnal pentru cuvinte cheie. Jos sunt filtre, incluzând severitatea de nivel "log entry".

Iată o imagine a Log Viewer, (Admin -> Rapoarte de la consolă):

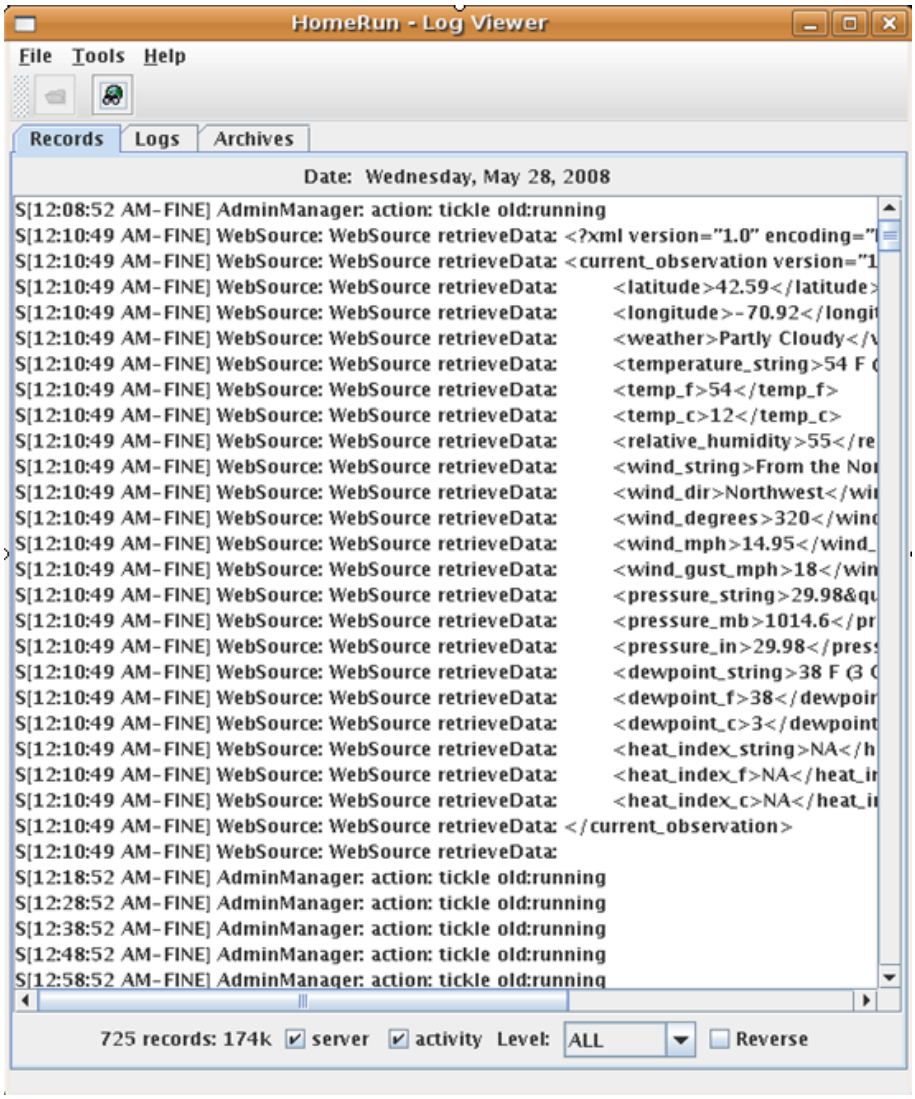

Figure 4.15: Fereastra Log Viewer

4.8 Exemplu - Modulul RSS

Pe lângă posibilităţiile de automatizare a controlului dispozitivelor electronice, HomeRun mai oferă şi posibilitatea personalizării unor informaţii şi distribuirea acestora pe pagina principală a serverului web.

În exemplul dezvoltat vom arăta cum HomeRun prin intermediul unui modul implementat cu ajutorul tehnologii OSGi va extrage informaţii de pe un site de ştiri şi le va afişa pe pagina web.

4.8.1 Instalarea pachetului RSS

Pentru instalare, din fereastra 'HomeRun Console' se selectează optiunea Admin -> Setup. Vezi figura următoare:

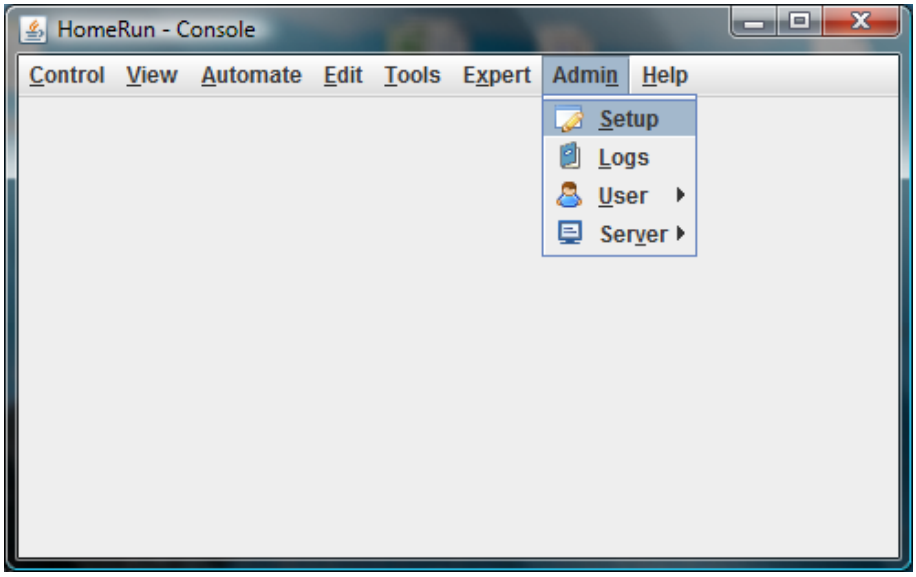

Figure 4.16: Fereastra Admin Setup

Instalarea modulului se va face prin selectarea grupului de pachete "misc" şi selecţia pachetului BBC News. "misc" provinde de la "miscellaneous". După selecţie se face clic pe butonul 'Install' care va instala pachetul selectat. Acesta este modul de instalare a tuturor pachetelor suplimentare oferite de HomeRun. În continuare vom descrie mecanismul care stă la baza in-

stalării acestui pachet.

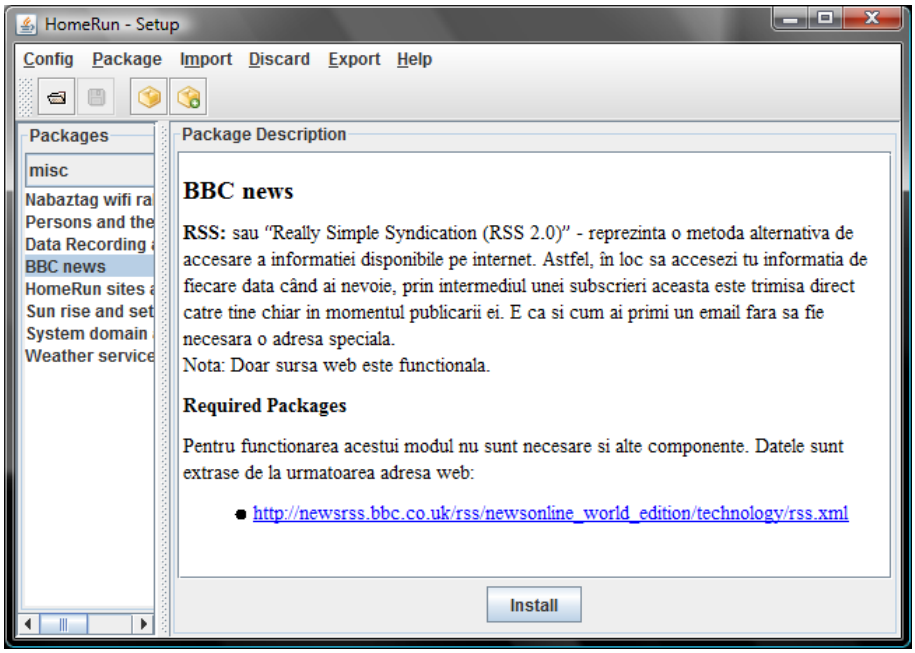

Figure 4.17: Instalarea modulului RSS

4.8.2 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 package-ul : com.monad.homerun.bootapp; la linia 401.

```
instButton.addActionListener( this );
```

La instalare se verifică dacă există pachete de care depinde rularea pachetului selectat şi care nu sunt iniţializate. În cazul în care sunt depistate astfel de pachete neiniţializate un mesaj de eroare este afişat - **Setup.java** linia 941.

```
msgLabel.setText( "Installation failed: " + status );
```

status reprezintă cauza pentru care instalarea a eşuat. Spre exemplu dacă un anumit 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** la linia 253 din package-ul: com.monad.homerun.admin.impl.

```
ab.error = "missing requirement '" + req + "'";
```

După instalare, butonul de 'Install' din fereastra va fi înlocuit cu unul gri pentru a arăta faptul că pachetul a fost instalat. De asemenea o bifă verde va apare în dreptul pachetului respectiv.

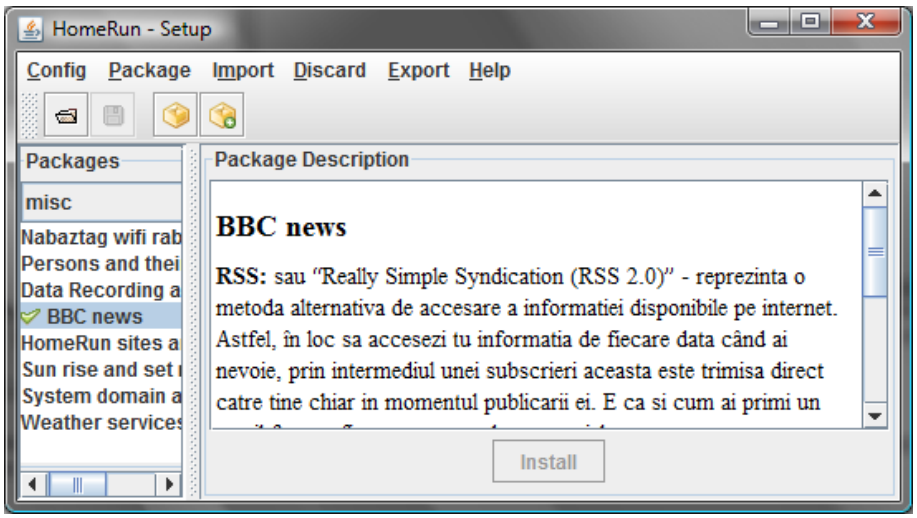

Figure 4.18: Pachet instalat

4.8.3 Aplicaţia server

Odată lansată în execuţie aplicaţia RSS, ea se va conecta la canalul de ştiri setat în fişierul de configurare rss.xml

Acesta va specifica locaţia de la care se vor extrage ştirile. În cazul exemplului nostru ştirile vor fi extrase de pe site-ul BBC: http://newsrss.bbc.co.uk/rss/newsonline_world_edition/technology/rss.xml secţiunea "Technology".

PackageInstaller.java *rss.xml

```
<?xml version="1.0"?>
<rssConf>
  <channels>
    <channel name="primary">
      <source type="pick" limiters="/sources" desc="Rss source">web</source>
    </channel>
    <channel name="alternate">
      <source type="pick" limiters="/sources" desc="Rss source">web</source>
    </channel>
  </channels>
  <sources>
    <source name="web" enabled="true">
      <class type="string" level="2" desc="Source class">
      com.monad.homerun.pkg.rss.source.WebSource</class>
      <tag type="string" desc="Display Tag">NWS via Web</tag>
      <!-- URL for rss: needs to be configured for each installation -->
      <sourceURL type="string" desc="URL for rss">
      http://newsrss.bbc.co.uk/rss/newsonline_world_edition/technology/rss.xml</sourceURL>
      <dataFormat type="string" desc="Data Format">BBC NWS XML</dataFormat>
      <updateMins type="number" desc="Update frequency (mins)">4</updateMins>
    </source>
  </sources>
</rssConf>

```

Figure 4.19: Fisier configurare rss.xml

Versiunea=/rss/channel/ttl
Media=/rss/channel/ttl
Limba=/rss/channel/language
DescriereCanal=/rss/channel/description
Titlu=/rss/channel/title
DataUltimeiCompilari=/rss/channel/lastBuildDate
Linc=/rss/channel/link
TitluArticol=/rss/channel/item/title
DescriereArticol=/rss/channel/item/description
LincArticol=/rss/channel/item/link

Table 4.1: Fişierul rssnwsxml.properties

Fişierul http://newsrss.bbc.co.uk/.../rss.xml va fi parcurs şi din el vor fi extrase tag-urile marcate în fişierul rssnwsxml.properties care specifică structura fişierului xml ce va fi interogat. Pentru ca aplicaţia să funcţioneze trebuie să existe o legătură internet activă.

În terminolgia XML fişierul rssnwsxml.properties conţine definiţia XPath a tagurilor ce urmează a fi interogate.

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

În următoarele fişiere se găseşte codul sursă al aplicaţiei.

```
----rss
    -   Observation.java
    -   Report.java
    -   RssService.java
    -   Source.java
    -   TempUtil.java
    ----impl
    -       Activator.java
    -       NWSXmlObservation.java
    -       PollSourceJob.java
    -       RssManager.java
    -       RssServer.java
    ----source
            ProxySource.java
            WebSource.java
```

Aceste fişiere pot fi regăsite în Anexa A. sau pe CD-ul care însoţeşte această lucrare.

4.8.4 Mecanismul OSGi

Lansarea pachetului - bundle RSS este realizată din fişierul Activator.java care conţine metodele start şi stop.

Activator.java are rolul de a iniţializa mai multe 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". Aceasta se iniţializează după modelul <ServiceReference org.osgi.framework.BundleContext. getServiceReference (String arg0)> unde **arg0** este obţinut de la serviciile iniţializate prin <NumeServiciu>.class.getName().

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

Apelarea la installer se face după atribuirea tuturor valorilor elementului context de tip BundleContext. Vezi linia 88 din fisierul Activator.java:
Installer.installConf(context.getBundle());

Installerul foloseşte şi el capacitatea cadrului de lucru OSGi pentru a instala şi configura pachetul RSS. Funcţia **getBundle** este o metodă aparţinând pachetului - Bundle org.osgi.framework.BundleContext.getBundle().

Lansarea în execuţie a programului RSS se face prin apelarea constructorului ce iniţializează clasa RssManager.

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

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;

Dealtfel clasa Activator este singura clasă din pachetul RSS care consumă şi este direct dependentă de serviciile OSGi. Toate celelalte clase sau interfeţe precum RssManager, RssService, RssServer fac apel indirect la cadrul de lucru OSGi, acesta având doar rolul de a gestiona serviciile LogService, ConfigService, TimingService şi ModelService iniţializate.

După lansarea clasei RssManager este apelată funcţia de construcţie a raportului ce urmeaza a fi afişat. Aceasta se realizează prin coamanda : curReport = source.reportRss(); linia 158 din fişierul RssManager.java.

reportRss este o metodă a clasei WebSource care apelează la rândul ei clasa NWSXmlObservation ce extrage fizic informaţiile necesare din sursa web menţionată la configurare. NWSXmlObservation foloseşte la rândul său nişte pachete speciale pentru interpretarea fişierelor XML. Acestea sunt importate la început prin declararea lor ca importuri.

- 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;

4.8.5 Pachete pentru afişarea web

HomeRun este distribuit cu propriul server web care foloseşte tehnologia velocity pentru compilarea paginilor de prezentare. Componenta velocity-dep-1.5.jar se regăseşte în directorul <root-app>\server\packages\webui\src\main\resources\. Pentru afişarea pe web a ştirilor este necesară instalarea următorului modul:

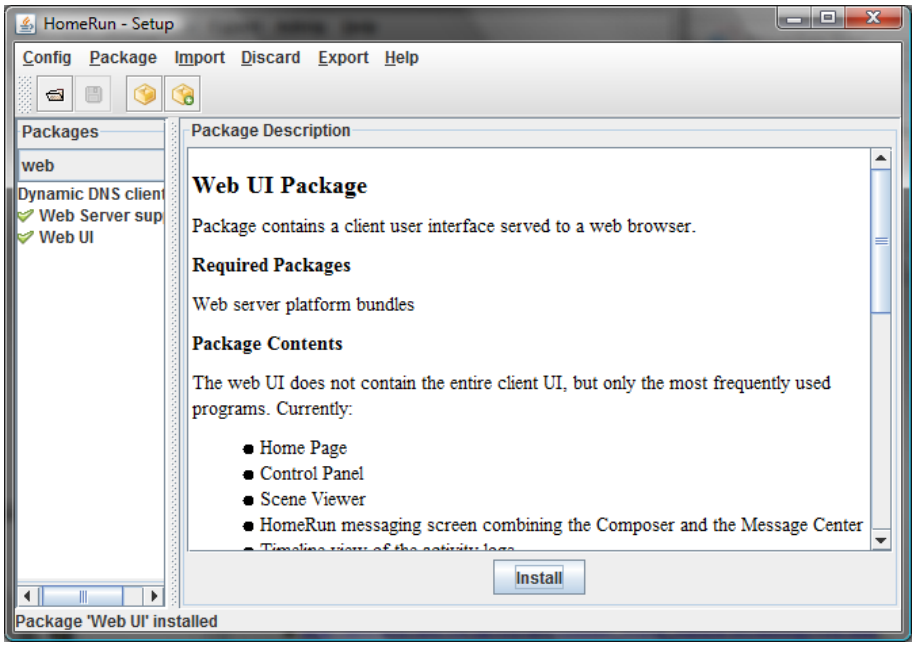

Figure 4.20: Instalarea modulului web al aplicaţiei HomeRun

După deployment fişierele velocity şi cele de stil sunt instalate la adresa următoare:

```
<root-app>fwork\parts\bs\17\jar0\WEB-INF\
+---styles
|       default.css
|
+---templates
        control.vm
        control.vt
        denied.vt
        footer.vt
        header.vt
```

```
home.vt
login.vt
message.vt
scene.vm
timeline.vt
topnav.vt
view.vt
```

După inițializarea serviciului web şi pornirea serviciului RSS se poate accesa pagina web pe portul 8080 care este setat implicit. Vezi imaginea următoare:

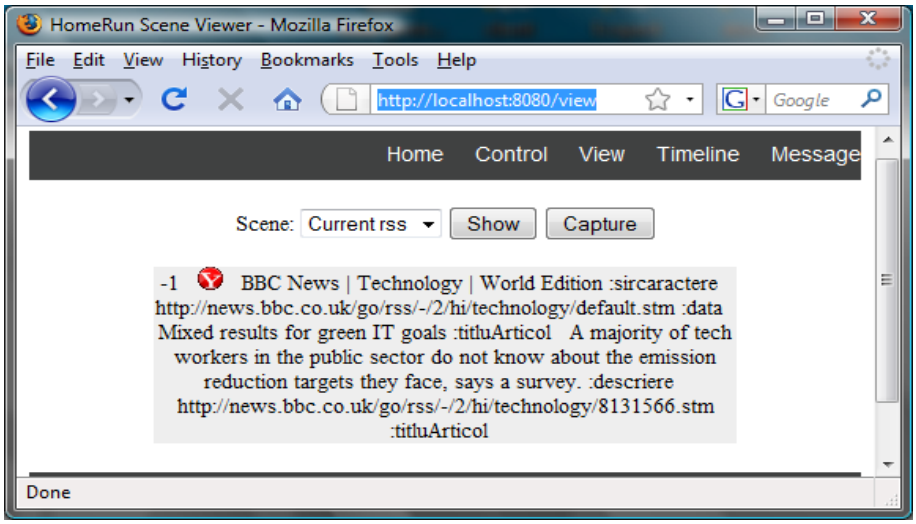

Figure 4.21: Interfaţa web

Appendix A

Apendix

În continuare vom afişa un listing al unor fişiere ce compun pachetul-bundle RSS.

A.1 rss

A.1.1 Fişierul Observation.java

```
package com.monad.homerun.pkg.rss;

import java.util.Iterator;

/**
 * Observation descrie un set de valori rss.
 */

public interface Observation
{
public Iterator getNames();
    public String getValue( String name );
}
```

A.1.2 Fişierul RssService.java

```
package com.monad.homerun.pkg.rss;

/**
 * RssService describes the services offered by the rss system
 */

public interface RssService
{
/**
 * Poll the current source for rss data
 *
 */
public void pollSource();

/**
 * Returns current rss report
 *
 * @return report the current rss report
 */
    public Report getReport();
}
```

A.1.3 Fişierul Report.java

```
package com.monad.homerun.pkg.rss;
```

```
import java.io.Writer;
import java.io.BufferedReader;
import java.io.Serializable;
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

/**
 * RssReport este un container pentru informatiile celectate de sursa rss.
 * Each source need not maintain the full complement of values.
 */

public class Report implements Serializable
{
private static final long serialVersionUID = -4807153946185413175L;
// name identifying the source of the data
    private String source = null;
    // timestamp for this rss data
    private long reportTime = 0L;
    // rss values
    private Map<String, String> valMap = null;

    public Report( String source )
    {
        this.source = source;
        reportTime = System.currentTimeMillis();
        valMap = new HashMap<String, String>();
    }

    // getters & setters
    public String getSource()
    {
        return source;
    }

    public long getReportTime()
    {
        return reportTime;
    }

    public String getValue( String valName )
    {
     return valMap.get( valName );
    }

    public void setValue( String valName, String value )
    {
     valMap.put( valName, value );
    }

    public void setValue( String valName, Observation obs )
    {
     setValue( valName, obs.getValue( valName ) );
    }

    public void setValues( Observation obs )
    {
     Iterator iter = obs.getNames();
     while ( iter.hasNext() )
     {
     String name = (String)iter.next();
     // check for & suppress no data
     String value = obs.getValue( name );
     if ( value != null && ! "NA".equals( value ) )
     {
     setValue( name, value );
     }
     }
    }

    public boolean isSet( String valName )
    {
     return ( valMap.get( valName ) != null );
    }

    public void writeReport( Writer out ) throws IOException
    {
        // start with name of source
        out.write( "Source:" + source + '\n');
        for ( String name : valMap.keySet() )
        {
         out.write( name + ":" + valMap.get( name ) + '\n' );
```

```
        }
        out.flush();
    }

    public static Report readReport( BufferedReader in ) throws IOException
    {
        String line = in.readLine();
        // first line must have source identifier
        if ( line == null || ! line.startsWith("Source:") )
        {
            return null;
        }

        Report report = new Report( line.substring( line.indexOf(":") + 1 ) );
        // read rest of lines
        while ( true )
        {
            line = in.readLine();
            // test for end of data
            if ( line == null || line.length() == 0 )
            {
                break;
            }
            String[] pair = line.split( ":" );
            report.setValue( pair[0], pair[1] );
        }

        return report;
    }
}
```

A.1.4 Fişierul Source.java

```
package com.monad.homerun.pkg.rss;

import java.util.Properties;
import java.util.logging.Logger;


/**
 *  IRssSource is the interface implemented by providers of rss data
 */

public interface Source
{
    /**
     * Perform any initialization required to operate, possibly based on passed
     * properties. Note that if no properties are identified/required, 'props'
     * may be empty. init() may be called repeatedly after the source has been
     * instantiated, with any calls susequent to the first having the sense of
     * a 'reset' operation. The mechanics of the initialization/reset are
     * source-dependent. After init() has been called the source is considered
     * to be functional.
     */

    public boolean init( Logger logger, Properties props );

    /**
     * Returns a descriptive name for this source
     * @return name a descriptive name
     */
    public String getName();

    /**
     * Return an integer that expresses - in minutes - the currency of rss
     * data from the source. This is used by the consumer to determine how often
     * to poll the source and whether the source is operating correctly. For
     * example, if the source provides new data only every 60 minutes, but the
     * last report from it is 2 hours old, the consumer may judge that the
     * source has failed, and reinitialize it, or revert to an alternate source.
     */

    public int getUpdateMins();

    /**
     * Return the the most recent rss data.
     */

    public Report reportRss();

    /**
     * Shutdown the source. The mechanics of shutdown are also source-dependent,
```

```
     * but a source that has been shutdown has permanently released any
     * resources it held, and may not be reset or restarted. Any calls
     * subsequent to shutdown are ignored. A source should be shutdown before
     * it is deallocated.
     */

    public void shutdown();

    /**
     * Enumerated state values for managing the semantics implied above
     */

     public static final int UNALLOCATED = 0;
     public static final int ALLOCATED = 1;
     public static final int DEALLOCATED = 2;
}
```

A.2 rss.impl

A.2.1 Fişierul Activator.java

```
package com.monad.homerun.pkg.rss.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.pkg.rss.RssService;
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 RssService rssSvc;

// 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 );

     if ( GlobalProps.DEBUG )
     {
     // DEbug
     System.out.println( "Rss activator start: " + bc );
     }

        // if I am not yet installed, 'pre-install' config file,
     // since config parameters needed in RssManager initialization
     // RLR TODO review this logic
     Installer.installConf( context.getBundle() );

     try
{
     // register our service
     RssManager mgr = new RssManager();
     // start him up
     mgr.init( true );
     rssSvc = mgr;
     Dictionary props = new Hashtable();
     bc.registerService( RssService.class.getName(),rssSvc, props );

}
     catch ( Throwable t )
{
         if ( GlobalProps.DEBUG )
         {
         t.printStackTrace();
         System.out.println( "Caught Exception: " + t.toString() );
         }
}
     if ( GlobalProps.DEBUG )
     {
     System.out.println( "Rss 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( "Rss 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;
    }
}
```

A.2.2 Fişierul RssManager.java

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

import java.util.Date;
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.modelmgt.ModelInformer;
import com.monad.homerun.pkg.rss.RssService;
import com.monad.homerun.pkg.rss.Source;
import com.monad.homerun.pkg.rss.Report;

/**
 * RssManager gathers information by polling the
 * configured 'rss source', which may be a web rss site,
 * an internet rss service providing news for the category selected.
 * If the first source fails to report timely data, this manager will attempt
 * (if configured) to establish 'fall-back' (alternate) sources. The primary
 * source may be re-established by issuing a 'reset' to the manager.
 * It is a ModelInformer, and so can drive (push) data to models.
 */

public class RssManager implements RssService, ModelInformer
{
    // the source for rss 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 rss 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 RssManager()
    {
        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( "RssPoll" );
        }
        // get config info
        ConfigContext chanCtx = Activator.getContext( "channels/@" + rank );
        // intantiate the class configured to be the rss 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( "RssPoll", 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( "FirstRssPoll", PollSourceJob.class,
                                  "date", new Date( firstPoll ) );

        // adjust rank
        sourceRank = rank.equals( "primary" ) ? 1 : 2;

        return true;
    }

    public void pollSource()
    {
        curReport = source.reportRss();

        if ( GlobalProps.DEBUG && curReport != null )
        {
            logger.log( Level.FINE, "Temp: " + curReport.getValue( "OutTemp" ) );
        }

        checkSanity();

        // NB: we always notify models, even if the rss 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( "WethM informModel - report lacks data for: " +
        infModel.type );
        }
        }
    }
    else
    {
           if ( GlobalProps.DEBUG )
    {
        System.out.println( "WethM informModel - lacks report for: " +
        infModel.type );
    }
       }
   }

   private String convertValue( String valStr, String type )
   {
   /* if ( "OutTemp".equals( type ) ||
     "OutHumidity".equals( type ) ||
"WindDirection".equals( type ) )
    {
    // String[] parti = valStr.split( "\\." );
    // return Integer.parseInt( parti[0] ) ;
    // return Integer.parseInt( valStr ) ;
    return valStr;
    }
    else if ( "WindSpeed".equals( type ) ||
      "Pressure".equals( type ) )
    {
    // Decimal - round for now
    String[] parts = valStr.split( "\\." );
    int val = -1;
    if ( parts.length > 1 )
    {
    val = Integer.parseInt( parts[0] );
    int frac = Integer.parseInt( parts[1] );
    if ( parts[1].length() == 1 )
    {
    if ( frac > 5 )
    {
    ++val;
    }
```

```
    }
    else if ( parts[1].length() == 2 )
    {
    if ( frac > 50 )
    {
    ++val;
    }
    }
    }
    else
    {
    val = Integer.parseInt( valStr );
    }
    return "" + val;
    }

    return -1 + ""; */

    // Commentariu pentru Debuging
   logger.log(Level.WARNING, valStr);
    return valStr;
   }

   public void reset()
   {
    // cancel polling job
    Activator.timingSvc.killJob( "RssPoll" );
       // unregister with ModelManager
       Activator.modelSvc.unregisterInformer( this );

       // now redo init
       init( true );
       logger.log( Level.INFO, "RssManager reset" );
   }

   // save any unsaved state data and release all resources
   public void shutdown()
   {
       // forward command to rss source
       if ( source != null )
       {
           source.shutdown();
       }

    // cancel polling job
    Activator.timingSvc.killJob( "RssPoll" );

       // unregister with ModelManager
       Activator.modelSvc.unregisterInformer( this );

       logger.log( Level.INFO, "RssManager shutdown" );
   }

   // End IStatMonitor methods

   // ModelInformer methods
   public boolean addModel( String type, String domain,
     String objectName, Model model )
   {
    if ( GlobalProps.DEBUG )
    {
    System.out.println( "WETHM: addmdel: " + model.getModelName() );
    }
    InformModel infModel = new InformModel( type, domain, objectName, model );
    // inform model now
    informModel( infModel, source.reportRss() );
    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();
     // RLR TODO - typing should be encoded in Rss Mgr config data
     if ( "degrees F".equals( vtName ) )
     {
     EventDesc[] points = new EventDesc[1];
     points[0] = new EventDesc( "temp is", "temp", null );
     emitter = new Emitter( "rss: " + source.getName(), points, 1 );
     }
     }
     return emitter;
    }

    public String getInformerName()
    {
     return "RssManager";
    }

    // Monitor may also be queried for the last retrieved rss 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 rss 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( "Titlu" ) )
            {
                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;
    }
}
}
```

A.2.3 Fişierul NWSXmlObservation.java

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

import java.io.ByteArrayInputStream;
import java.util.Iterator;
import java.util.Properties;
import java.util.logging.Level;
import java.util.regex.Pattern;

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;

import com.monad.homerun.core.GlobalProps;
import com.monad.homerun.pkg.rss.Observation;

/**
 * NWSXmlObservation encapsulates the coding specifics of an XML observation
 * record used on the web by the BBC Rss Service.
 **/

public class NWSXmlObservation implements Observation
{
// map of value names to doc xpaths
private static Properties pathProps = null;
// condition state values - break out to a config file
private static String[] tokens = { "Fair", "Cloudy", "Overcast", "Rain", "Storm", "Snow" };
// as regex patterns
private static Pattern[] patterns = null;
// the observation record as a string
private String rawObs = null;
// the parsed observation document
private Document docObs = null;

static
{
pathProps = Activator.getConfProperties( "rssnwsxml" );
patterns = new Pattern[tokens.length];
for ( int i = 0; i < patterns.length; i++ )
{
patterns[i] = Pattern.compile( tokens[i] );
}
}

public NWSXmlObservation()
{
}

public NWSXmlObservation( String rawObs )
{
this.rawObs = rawObs;
}

public String getValue( String name )
{
if ( docObs == null )
{
// lazy parse - only if a value requested
   docObs = Activator.cfgSvc.getDocFromStream(
    new ByteArrayInputStream( rawObs.getBytes() ) );
}
return getNormalizedValue( name );
}
```

```
public Iterator getNames()
{
return pathProps.keySet().iterator();
}

// for debuggin mostly
   public String getRawObs()
   {
   return rawObs;
   }

   private String getNormalizedValue( String name )
   {
   String obsValue = getObsValue( name );
   if ( obsValue != null )
   {
   if ( ! "Cover".equals( name ) )
   {
   return obsValue;
   }
   // RLR TODO - generalize & break out to config file
   // look for specific tokens in the value, which is a free-form
   // description of rss conditions
   for ( int i = 0; i < tokens.length; i++ )
   {
   if ( patterns[i].matcher( obsValue ).matches() )
   {
   return tokens[i];
   }
   }
   }

   return "NA";
   }

   private String getObsValue( String name )
   {
   // use XPath find values
   String val = null;
   if ( docObs != null )
   {
   XPath xpath = XPathFactory.newInstance().newXPath();
   String expr = pathProps.getProperty( name ) + "/text()";
if ( GlobalProps.DEBUG )
   {
   System.out.println( "getVal name: " + name + " Xpath: " + expr );
   }
   try
{
   Node node = (Node)xpath.evaluate( expr, docObs,
     XPathConstants.NODE );
   val = node.getNodeValue();
   if ( GlobalProps.DEBUG )
   {
   System.out.println( "getVal name: " + name + " val: " + val );
   }
}
   catch ( Exception e )
{
   Activator.logSvc.getLogger().log( Level.SEVERE,
         "Exception evaluating path: " + expr );
}
   }
   return val;
   }
}
```

A.2.4 Fişierul PollSourceJob.java

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

import org.quartz.Job;
import org.quartz.JobExecutionContext;

/**
 * Quartz wrapper for polling current rss source
 */
public class PollSourceJob implements Job
{
    public PollSourceJob()
    {
```

```
    }

    public void execute( JobExecutionContext ctx )
    {
     Activator.rssSvc.pollSource();
    }
}
```

A.2.5 Fişierul RssServer.java

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

import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.net.ServerSocket;
import java.util.Properties;
import java.util.TimerTask;
import java.util.logging.ConsoleHandler;
import java.util.logging.Logger;
import java.util.logging.Level;

import com.monad.homerun.core.GlobalProps;
import com.monad.homerun.pkg.rss.Source;
import com.monad.homerun.pkg.rss.Report;
import com.monad.homerun.util.HRTimer;
//import com.monad.homerun.log.LogMgr;

/**
 * RssServer is a container for a RssSource that allows it to
 * operate on a host remote to the server and communicate with the server via
 * a SourceProxy. The server listens on the configured port, and then is
 * passed a source class and properties to load. When loaded, it forwards the
 * source's rss data when requested.
 */

public class RssServer
{
    // default port we are listening on for proxy requests
    private static final int defaultServerPort = 8182;
    // the rss source we are hosting
    private Source source = null;
    // System timer facility
    private HRTimer statTimer = null;
    // task for polling the source
    private TimerTask pollTask = null;
    // polling frequency
    private long pollPeriod = 0L;
    // system logger
    private Logger logger = null;
    // the rss data from source
    private Report curReport = null;

    public RssServer()
    {
    }

    public void init( String[] args )
    {
        int serverPort = defaultServerPort;

        // parse command line options
        if ( args != null )
        {
            int i = 0;
            while ( i < args.length )
            {
                if ( args[i].startsWith( "-?" ) || args[i].startsWith( "-help" ) )
                {
                    System.err.println( "Usage: Rss Server port <portNum> [log] ");
                    System.exit( 0 );
                }
                // determine the port to listen on
                else if ( args[i].startsWith( "port" ) && args[i+1] != null )
                {
                    try
                    {
                        serverPort = Integer.parseInt( args[i+1] );
                        ++i;
```

```
                }
                catch ( NumberFormatException e )
                {
                    System.err.println( "Invalid port number: " + args[i+1] );
                    System.exit( 0 );
                }
            }
            else if ( args[i].startsWith( "log" ) )
            {
                //logger = LogMgr.getLogger();
            }

            ++i;
        }
    }

    // if 'log' option not specified on command line, create a simple
    // console logger for serious errors
    if ( logger == null )
    {
        logger = Logger.getAnonymousLogger();
        logger.addHandler( new ConsoleHandler() );
        logger.setLevel( Level.SEVERE );
    }

    // get system timer
    statTimer = HRTimer.getInstance();

    // create a stub rss report in case we get a request before the
    // source reports to us
    curReport = new Report( "Server" );

    // now start a thread to listen on server socket for requests
    try
    {
        ServerSocket serverSocket = new ServerSocket( serverPort );
        new ServerThread( serverSocket ).start();
    }
    catch ( Exception e )
    {
        logger.log( Level.SEVERE, "Cannot create server socket" );
        System.exit( 0 );
    }

    System.out.println( "RssServer listening on port " + serverPort );
}

private boolean initSource( String sourceDesc )
{
    // if a source is currently instantiated, shut it down
    if ( source != null )
    {
        source.shutdown();
        source = null;
        // stop the poll
        if ( pollTask != null )
        {
            pollTask.cancel();
            pollTask = null;
        }
    }

    // repackage the sourceDesc string as a Properties object
    Properties sourceProps = new Properties();
    String[] nvPairs = sourceDesc.split( ";" );
    for ( int i = 0; i < nvPairs.length; i++ )
    {
     String token = nvPairs[i];
        int split = token.indexOf( "=" );
        String name = token.substring( 0 , split );
        String value = token.substring( split + 1 );
        //System.out.println( "adding: " + name + " : " + value );
        sourceProps.put( name, value );
    }

    // lookup the source class & instantiate it
    String sourceClass = (String)sourceProps.get( "class" );

    // create a rss source
    if ( sourceClass != null )
    {
     try
{
```

```
         source = (Source)Class.forName( sourceClass ).newInstance();
         }
         catch (Exception e )
{
         //LogMgr.getLogger().log( Level.SEVERE, "caught exception loading class '" +
         // sourceClass + "': " + e.getMessage() );
         //e.printStackTrace();
         source = null;
         }

            if ( source != null )
            {
                //source.init( sourceProps );

                // get the updateTime to set up the poll of the source
                int freq = source.getUpdateMins();
                if ( freq > 0 )
                {
                    pollPeriod = freq * 60 * 1000;
                    pollTask = new TimerTask() { @Override
public void run() { pollSource(); } };
                    // delay the first poll to let the source initialize & gather data
                    statTimer.addDelayedTask( pollTask, pollPeriod, pollPeriod );
                    return true;
                }
            }
        }

        return false;
    }

    private void pollSource()
    {
        Report report = source.reportRss();
        if ( report != null )
        {
            curReport = report;
        }
    }

    private class ServerThread extends Thread
    {
        private ServerSocket serverSocket = null;
        private boolean stop = false;

        public ServerThread( ServerSocket serverSocket )
        {
            this.serverSocket = serverSocket;
        }

        @Override
public void run()
        {
            try
            {
                while ( ! stop ) // RLR - won't work
                {
                    Socket socket = serverSocket.accept();
                    // hand off to a request thread for processing
                    new RequestThread( socket ).start();
                }
            }
            catch ( Exception e )
            {
                logger.log( Level.SEVERE, "Caught exception: " + e.toString() );
            }
        }

        public void stopServer()
        {
            stop = true;
        }
    }

    private class RequestThread extends Thread
    {
        private Socket socket = null;

        public RequestThread( Socket socket )
        {
            this.socket = socket;
        }
```

```
        @Override
public void run()
        {
            BufferedReader in = null;
            BufferedWriter out = null;

            try
            {
                in = new BufferedReader( new InputStreamReader(
                                         socket.getInputStream() ) );
                out = new BufferedWriter( new OutputStreamWriter(
                                          socket.getOutputStream() ) );

                String cmdStr = in.readLine();
                if ( GlobalProps.DEBUG )
                {
                 System.out.println( "get Cmd: " + cmdStr );
                }

                if ( cmdStr.equals( "report" ) )
                {
                    curReport.writeReport( out );
                }
                else if ( cmdStr.startsWith( "load:" ) )
                {
                    boolean OK = initSource( cmdStr.substring( cmdStr.indexOf( ":" ) + 1 ) );
                    if ( OK )
                        out.write( "OK" );
                    else
                        out.write( "Fail" );
                    out.flush();
                }
                else if ( cmdStr.equals( "poll" ) )
                {
                    out.write( source.getUpdateMins() );
                    out.flush();
                }
                else
                {
                    out.write( "Error - unknown command" );
                    out.flush();
                }
            }
            catch ( IOException e )
            {
                logger.log( Level.SEVERE, "Caught exception: " + e.toString() );
            }
            finally
            {
                try
                {
                    if ( in != null )
                        in.close();
                    if ( out != null )
                        out.close();

                    socket.close();
                }
                catch ( Exception e )
                {
                    logger.log( Level.SEVERE, "Exception in finally: " + e.toString() );
                }
            }
        }
    }

    public static void main( String[] args )
    {
        new RssServer().init( args );
    }
}
```

A.3 rss.source

A.3.1 Fişierul WebSource.java

```
package com.monad.homerun.pkg.rss.source;

import java.io.BufferedReader;
import java.io.InputStreamReader;
```

```
import java.net.URL;
import java.util.Properties;
import java.util.logging.Logger;
import java.util.logging.Level;

import com.monad.homerun.core.GlobalProps;
import com.monad.homerun.pkg.rss.Observation;
import com.monad.homerun.pkg.rss.Source;
import com.monad.homerun.pkg.rss.Report;
import com.monad.homerun.pkg.rss.impl.NWSXmlObservation;

/**
 * WebSource obtains a rss report from a web site, for instance the BBC
 * (BBC Rss Service) for the configured category. It is essentially a
 * screen-scraper for various formats.
 */

public class WebSource implements Source
{
    // the operating state of this source
    private int state = UNALLOCATED;
    // properties for use by source
    private Properties webProps = null;
    // update frequency in minutes
    private int updateMins = 0;
    // system logger
    private Logger logger = null;
    // current observation
    private Observation obs = null;

    public WebSource()
    {
    }

    public boolean init( Logger logger, Properties srcProps )
    {
     this.logger = logger;
        if ( GlobalProps.DEBUG )
        {
            logger.log( Level.FINE, "WebSource initializing" );
        }

        if ( state == DEALLOCATED )
        {
            return false;
        }

        if ( state == ALLOCATED )
        {
            // this is a 'reset'
            // RLR TD - must stop existing session thread
            ;
        }

        state = UNALLOCATED;
        webProps = srcProps;

        if ( webProps != null )
        {
            String updateMinsStr = getProperty( "updateMins" );
            if ( updateMinsStr != null && updateMinsStr.length() > 0 )
            {
                try
                {
                    updateMins = Integer.parseInt( updateMinsStr );
                    // convert to milliseconds
                    long sleepTime = updateMins * 60 * 1000;
                    // start a session thread to manage the web session
                    new SessionThread( sleepTime ).start();
                    state = ALLOCATED;
                    return true;
                }
                catch ( NumberFormatException e )
                {
                    logger.log( Level.SEVERE, "Update interval: '" + updateMinsStr
                                + "' is not a number." );
                }
            }
            else
            {
                logger.log( Level.SEVERE, "No update interval property" );
            }
        }
```

```
        else
        {
            logger.log( Level.SEVERE, "No source properties" );
        }

        return false;
    }

    public String getName()
    {
     return getProperty( "tag" );
    }

    // Return the update frequency in minutes
    public int getUpdateMins()
    {
        return updateMins;
    }

    // Return the most current rss data in a RssReport
    public Report reportRss()
    {
        if ( state == ALLOCATED && obs != null )
        {
            Report report = new Report( "Web" );
            report.setValues( obs );
            return report;
        }
        return null;
    }

    public void shutdown()
    {
        // RLR - TD stop threads
        state = DEALLOCATED;
    }

    private String getProperty( String name )
    {
        return webProps.getProperty( name );
    }

    // SessionThread manages the web session that retrieves the rss data
    private class SessionThread extends Thread
    {
        private long sleepTime = 0L;
        private boolean stayAlive = true;

        public SessionThread( long sleep )
        {
            sleepTime = sleep;
        }

        public void stopThread()
        {
         this.interrupt();
        }

        @Override
public void run()
        {
            while ( stayAlive )
            {
                // get the encoded observation record from web site
                retrieveData();
                // sleep until next update
                try
                {
                    sleep( sleepTime );
                }
                catch( InterruptedException e )
                {
                 stayAlive = false;
                }
            }
        }
    }

    // get current rss conditions in an observation via an Http get
    private void retrieveData()
    {
     StringBuffer obsBuf = new StringBuffer();
        try
```

```
        {
         URL url = new URL( getProperty( "sourceURL" ) );
         BufferedReader in = new BufferedReader(
         new InputStreamReader( url.openStream() ) );
         String lineIn = null;
         while(  ( lineIn = in.readLine() ) != null )
         {
                if ( GlobalProps.DEBUG )
                {
                    logger.log( Level.FINE, "WebSource retrieveData: " + lineIn );
                }
         obsBuf.append( lineIn );
         }
         // stuff into an observation based on format
            String fmt = getProperty( "dataFormat" );
         // debug only
         if ( GlobalProps.DEBUG )
         {
         logger.log( Level.FINE, "Raw: " + obsBuf.toString() );
         }
            if ( "BBC NWS XML".equals( fmt ) )
            {
             obs = new NWSXmlObservation( obsBuf.toString() );
            }
            /* add other formats here */
        }
        catch ( Exception e )
        {
            logger.log( Level.SEVERE, "Can't connect to: " +
             getProperty( "sourceURL" ) );
            if ( GlobalProps.DEBUG )
            {
             e.printStackTrace();
            }
        }
    }
}
```

A.3.2 Fişierul ProxySource.java

```
package com.monad.homerun.pkg.rss.source;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.util.Properties;
import java.util.Iterator;
import java.util.TimerTask;
import java.util.logging.Logger;
import java.util.logging.Level;

import com.monad.homerun.core.GlobalProps;
import com.monad.homerun.pkg.rss.Source;
import com.monad.homerun.pkg.rss.Report;
import com.monad.homerun.config.ConfigContext;
import com.monad.homerun.util.HRTimer;
//import com.monad.homerun.log.LogMgr;

/**
 * ProxySource is an Source implementation run locally that relays
 * rss data from a remote RssServer. At the specified polling interval,
 * it simply opens a socket to the remote rss server, receives the data,
 * and stores it for the local RssManager.
 */

public class ProxySource implements Source
{
    // the state of the proxy
    private int state = UNALLOCATED;

    // Address & port for remote rss server
    private String remHost = null;
    private int remPort = 0;

    // system logger
    private Logger logger = null;

    // System timer facility
    private HRTimer statTimer = null;
```

```
// task for polling the source
private TimerTask pollTask = null;

// poll period in minutes
private int pollPeriod = 0;

// last reported rss
private Report curReport = null;

// flag for remote source creation
private boolean remSourceLoaded = false;

// descriptive tag for source
private String tag = null;

public ProxySource()
{
}

public boolean init( Logger logger, Properties srcProps )
{
 this.logger = logger;
    if ( GlobalProps.DEBUG )
        logger.log( Level.FINE, "ProxySource initializing" );

    if ( state == DEALLOCATED )
    {
        logger.log( Level.SEVERE, "ProxySource deallocated" );
        return false;
    }
    // no 'reset' to perform, so skip test: state == ALLOCATED
    state = UNALLOCATED;

    if ( srcProps == null )
    {
        logger.log( Level.SEVERE, "No source properties found" );
        return false;
    }

    // get required rss server info from srcProps
    remHost = srcProps.getProperty( "remoteHost" );
    if ( remHost == null || remHost.length() == 0 )
    {
        logger.log( Level.SEVERE, "Missing/invalid remoteHost" );
        return false;
    }

    String port = srcProps.getProperty( "remotePort" );
    if ( port == null || port.length() == 0 )
    {
        logger.log( Level.SEVERE, "Missing/invalid remotePort" );
        return false;
    }

    try
    {
        remPort = Integer.parseInt( port );
    }
    catch ( NumberFormatException e )
    {
        logger.log( Level.SEVERE, "remotePort: "+ port +" is not a number." );
        return false;
    }

    // create a stub rss report, to have if reuqested before source
    // has one ready
    curReport = new Report( "SourceProxy" );

    // attempt to load the source on the remote server
    startRemoteSource( srcProps.getProperty( "remoteSource" ) );
    if ( ! remSourceLoaded )
    {
        logger.log( Level.SEVERE, "Unable to load remoteSource: " +
                    srcProps.getProperty( "remoteSource" )  );
        return false;
    }

    // contact the rss server to get his poll time
    pollPeriod = 0;
    remoteCommand( "poll" );
    if ( pollPeriod == 0 )
    {
        logger.log( Level.SEVERE, "Unable to get poll period" );
```

```
            return false;
        }

        // set up the poll
        statTimer = HRTimer.getInstance();
        long pollTime = pollPeriod * 60 * 1000;
        pollTask = new TimerTask() { @Override
public void run() { pollServer(); } };
        statTimer.addTask( pollTask, pollTime );

        state = ALLOCATED;
        tag = srcProps.getProperty( "tag" );
        return true;
    }

    public String getName()
    {
     return tag;
    }

    private void startRemoteSource( String sourceName )
    {
        ConfigContext ctx = null; //Config.getContext( "server/managers/@rss/sources" );

        // get the properties for the source, including class name,
        // and construct a command string with them
        Properties props = ctx.getFeature( sourceName ).getProperties();

        if ( props.size() == 0 )
        {
            logger.log( Level.SEVERE, "found no properties for source: " + sourceName );
            return;
        }

        Iterator iter = props.keySet().iterator();
        StringBuffer cmdBuf = new StringBuffer();
        cmdBuf.append( "load:" );

        while ( iter.hasNext() )
        {
            String propName = (String)iter.next();
            cmdBuf.append( propName );
            cmdBuf.append( "=" );
            cmdBuf.append( (String)props.get( propName ) );
            cmdBuf.append( ";" );
        }

        // now contact the rss server to get him to load this source
        remoteCommand( cmdBuf.toString() );
    }

    private void remoteCommand( String cmdStr )
    {
        Thread reqThread = new RequestThread ( cmdStr );
        reqThread.start();
        try
        {
            reqThread.join();
        }
        catch( InterruptedException ie )
        {
            logger.log( Level.SEVERE, "Request Thread interrupted" );
        }
    }

    public int getUpdateMins()
    {
        return pollPeriod;
    }

    public Report reportRss()
    {
        if ( state == ALLOCATED )
        {
            return curReport;
        }

        return null;
    }

    public void shutdown()
    {
        // no resources to free, just set state
```

```
        state = DEALLOCATED;
    }

    private void pollServer()
    {
        new RequestThread( "report" ).start();
    }

    private class RequestThread extends Thread
    {
        private String cmd = null;

        public RequestThread( String cmd )
        {
            this.cmd = cmd;
        }

        @Override
public void run()
        {
            if ( GlobalProps.DEBUG )
            {
             System.out.println( "Starting reqt host: " + remHost + " port: " + remPort + " cmd: " + cmd );
            }
            Socket socket = null;
            BufferedReader in = null;
            BufferedWriter out = null;

            try
            {
                socket = new Socket( remHost, remPort );

                in = new BufferedReader( new InputStreamReader(
                                         socket.getInputStream() ) );
                out = new BufferedWriter( new OutputStreamWriter(
                                          socket.getOutputStream() ) );

                // write cmd to socket, then get response
                out.write( cmd + '\n' );
                out.flush();

                if ( cmd.equals( "report" ) )
                {
                    Report report = Report.readReport( in );
                    if ( report != null )
                    {
                     if ( GlobalProps.DEBUG )
                     {
                     System.out.println(" got rpt: " + report.getSource() );
                     }
                        curReport = report;
                    }
                    else
                    {
                        logger.log( Level.WARNING, "Unable to obtain report" );
                    }
                }
                else if ( cmd.equals( "poll" ) )
                {
                    pollPeriod = in.read();
                 if ( GlobalProps.DEBUG )
                 {
                 System.out.println(" got poll: " + pollPeriod );
                 }
                }
                else if ( cmd.startsWith( "load:" ) )
                {
                    String loadResult = in.readLine();
                    remSourceLoaded = loadResult.equals( "OK" );
                }

            }
            catch ( Exception e )
            {
                logger.log( Level.SEVERE, "Caught exception " + e.toString() );
            }
            finally
            {
                try
                {
                    if ( in != null )
                        in.close();
                    if ( out != null )
```

```
                    out.close();
                if ( socket != null )
                    socket.close();
            }
            catch ( Exception e )
            {
                logger.log( Level.SEVERE, "finally: "
                            + "caught exception " + e.toString() );
            }
        }

    }
  }
}
```

Appendix B

Tabela de Bibliografie

Pagini oficiale ale dezvoltatorilor
- Pagina oficială a tehnologiei OSGi: http://www.osgi.org/
- Pagina oficială a produsului Equinox: http://www.eclipse.org/equinox/
- Pagina oficială a produsului Knopflerfish: http://www.knopflerfish.org/
- Pagina oficială a produsului Felix: http://felix.apache.org/
- Pagina oficială a produsului Concierge: http://concierge.sourceforge.net/

Produse bazate pe tehnologia OSGi
- Aplicaţia Eclipse - http://www.eclipse.org/
- Pagina oficială Spring-OSGi: http://www.springsource.org/osgi/
- Business Intelligence and Reporting Tool - http://en.wikipedia.org/wiki/BIRT_Project/
- Glassfish, Open source application server - https://glassfish.dev.java.net/
- Aplicaţia Open Source - HomeRun http://sourceforge.net/projects/homerun/

Articole web online (făceţi clic pe lincurile de mai jos)
- Build and deploy OSGi bundles using Apache Felix
- Build and deploy OSGi as Spring bundles using Felix
- Service Management Framework bundle with WebSphere Studio Device Developer
- Understanding how Eclipse plug-ins work with OSGi
- Managed mobile clients with OSGi: Managed smart clients
- Explore Eclipse's OSGi console
- Learn the basics of Eclipse plug-in development
- Learn the basics about plug-in development and rich-client applications
- Rich Ajax Platform, Part 1: An introduction

- Rich Ajax Platform, Part 2: Developing applications
- Wikipedia about OSGi

Resurse pentru programatorii OSGi

- OSGi DZone
- Resurse OSGi pe Eclipse
- OSGi Users' Forums
- International - OSGi Users' Forums Web site
- Forum de discutii: Knopflerfish OSGi
- Forum dedicat tehnologiei OSGi

Lista Figurilor

List of Tables