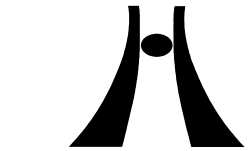
Beteckning:_____



HÖGSKOLAN I GÄVLE

Institutionen för matematik, natur- och datavetenskap

Interoperabilitet mellan DVB-HTML och DVB-J

Lars Djerf
juni 2007

Examensarbete, 10 poäng, C
Datavetenskap

Datavetenskapliga programmet
Examinator/handledare: Fredrik Bökman
Medbedömare: Anders Jackson

Interoperabilitet mellan DVB-HTML och DVB-J

av

Lars Djerf

Institutionen för matematik, natur- och datavetenskap
Högskolan i Gävle

S-801 76 Gävle, Sweden

Email:
kp02ldf@student.hig.se

Abstrakt

Vid sändning av digital television är det möjligt att utöver ljud och bild även sända och erbjuda olika typer av tjänster och applikationer till mottagare. Vid utveckling av applikationer för digital television kan man välja att utveckla i antingen DVB-J (som är en bantad variant av Java för exekvering i Set-Top-Boxar) eller i DVB-HTML (som till stor del överensstämmer med XHTML). Syftet med detta arbete är att utreda hur DVB-J och DVB-HTML kan kombineras i en och samma applikation och hur interoperabilitet kan uppnås mellan DVB-J och DVB-HTML.
Nyckelord: DVB, MHP, DVB-J, DVB-HTML, interoperabilitet

Innehåll

1 Inledning

Vid sändning av digital television så är det möjligt att utöver ljud och bild även sända och erbjuda olika typer av tjänster och applikationer till mottagare. Dessa tjänster och applikationer kan erbjuda användaren olika grader av interaktivitet och funktionalitet. Det är till exempel möjligt att utveckla och erbjuda tjänster som tillåter användare att interagera genom att sända tillbaks information till sändaren så som exempelvis en tjänst för röstning i samband med ett TV-program.

För leverans av interaktiva medier finns en uppsättning öppna globala standarder som tillhandahålls av ett marknadslett konsortium vid namn Digital Video Broadcast (DVB). I de standarder som DVB specificerar så ingår bland annat standarder för teknik som möjliggör sändning och mottagning av digitala medier. DVB har även designat ett öppet middleware för de terminaler, Set-Top-Boxar, som tar emot och exekverar interaktiva tjänster. Detta öppna middleware kallas för Multimedia Home Platform (MHP).

Interaktiva tjänster erbjuds genom att sända eller på annat sätt distribuera MHP-applikationer till mottagares Set-Top-Boxar. Interaktiva tjänster, MHP-applikationer, utvecklas främst i DVB-J, som är en bantad variant av Java för exekvering i Set-Top-Boxar, men även i DVB-HTML, som till stor del överensstämmer med XHTML.

1.1 Problem

Vid utveckling av interaktiva tjänster för TV kan man välja att utveckla endera i DVB-HTML eller i DVB-J.

I vissa sammanhang kan det vara intressant att kombinera båda dessa tekniker i en och samma applikation. Detta gör att standardiserade sätt att kontrollera, exekvera och förmedla information mellan applikationer måste hittas så att interoperabilitet mellan applikationer och/eller funktionella enheter uppnås.

1.2 Syfte

Syftet med detta arbete är att på uppdrag av ITVARENA undersöka på vilka sätt interoperabilitet mellan DVB-J- och DVB-HTML-applikationer kan uppnås samt att utveckla en applikationshanterare (launcher). Applikationshanteraren ska på ett standardiserat sätt kontrollera både DVB-J- och DVB-HTML-applikationer och vid behov förmedla information till och mellan dessa.

1.3 Frågeställningar

- Vilka olika sätt att uppnå interoperabilitet mellan DVB-J- och DVB-HTML-applikationer finns det?
- Vilka för- och nackdelar finns det med de olika sätten att uppnå interoperabilitet mellan DVB-J och DVB-HTML?

2 Teoretisk bakgrund

2.1 Digital Video Broadcast (DVB)

Digital Video Broadcasting Project (DVB) grundades i september 1993 och är ett marknadslett konsortium bestående av över 260 organisationer från ifrån fler än 35 länder. DVB har som mål att designa och tillhandahålla öppna globala standarder för leverans av digitala medier [1].

DVB tillhandahåller främst tre öppna standarder för överföring och leverans av digitala medier:

- **DVB-S (Sattelite)** – signaler sänds via satellit till mottagare.
- **DVB-C (Cable)** – signaler sänds via kabelnät till mottagare.
- **DVB-T (Terrestial)** – signaler sänds via antenner till mottagare.

Utöver dessa standarder tillhandahåller DVB även en rad andra standarder så som för bland annat IP-baserad television (DVB-IPTV) och för television för batteridrivna handhållna enheter (DVB-H).

2.2 Multimedia Home Platform (MHP)

Multimedia Home Platform (MHP) är ett öppet middleware designat av DVB. Det definierar ett generiskt gränssnitt mellan applikationer för interaktiv television och de terminaler som dessa exekverar på. Gränssnittet gör att applikationer kan skapas och exekveras utan att behöva ta hänsyn till specifika hård- och/eller mjukvaruimplementationer i terminaler [2].

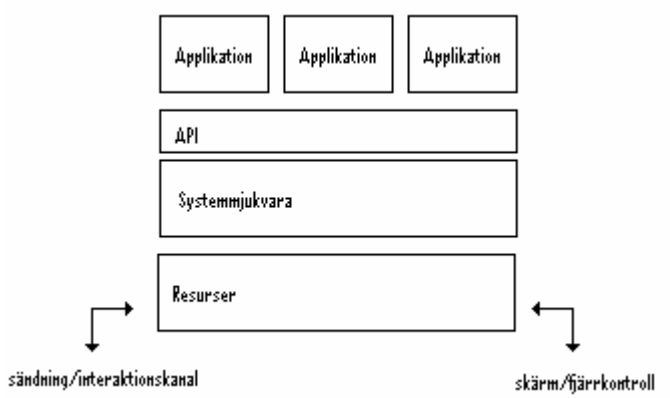
MHP är baserat kring en plattform som kallas DVB-J, eller DVB-Java, och inkluderar en virtuell maskin efter specifikation av Sun Microsystems. Ett generiskt Application Program Interface (API) tillhandahålls med hjälp av olika mjukvarupaket och applikationer får igenom detta API tillgång till MHP-plattformen och dess resurser [2]. MHP-applikationer är interaktiva applikationer skrivna i DVB-J eller i DVB-HTML och de körs ovanpå MHPs middleware.

MHP finns i ett flertal versioner och den funktionalitet som finns tillgänglig i MHP-mottagare avgörs av vilken version av MHP som mottagaren implementerar. Jag har här främst koncentrerat mig på MHP version 1.1 och framåt.

2.2.1 Arkitektur

Multimedia Home Platforms arkitektur delas in i följande lager (se figur 1):

- **Applikationer** – De applikationer som utgör det interaktiva innehållet och möjliggör de tjänster som tillhandahålls av leverantören.
- **Systemmjukvara** – Systemmjukvara som hanterar och abstraherar systemets resurser för applikationer. I systemmjukvaran ingår även en applikationshanterare för att kontrollera plattformen och de applikationer som körs på den.
- **Resurser** – De systemresurser som finns så som I/O-enheter, CPU, minne, grafiksystem och bearbetning av MPEG-strömmar.



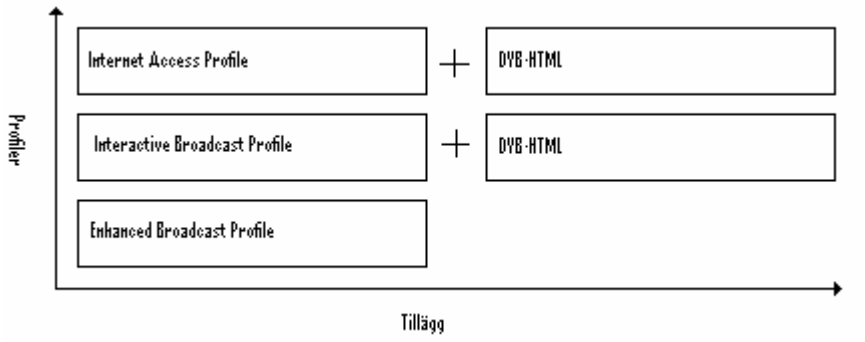
Figur 1. Multimedia Home Platform – Arkitektur.

2.2.2 Profiler

Multimedia Home Platform version 1.1 delas in i, och definierar, tre profiler för tjänster och applikationer:

- **Enhanced Broadcast Profile** – Profil som kombinerar digitala sändningar av ljud/video med applikationer som laddas ner. Denna profil tillåter lokal interaktivitet.
- **Interactive Broadcast Profile** – Profil som utökar Enhanced Brodcast Profile genom att tillåta applikationer att sända och ta emot information genom en interaktionskanal. Profilen utökar interaktivitet ifrån enbart lokal interaktivitet till interaktivitet mellan till exempel sändare och mottagare av en applikation.
- **Internet Access Profile** – Profil för Internet-baserade tjänster. Inkluderar även länkar mellan Internet-baserade tjänster och broadcast-tjänster.

Varje profil specificerar ett antal funktioner och egenskaper som måste implementeras och uppfyllas av Set-Top-Boxen i dess hård- och mjukvara. Utöver dessa tre profiler finns även ett frivilligt tillägg som kallas DVB-HTML. Figur 2 visar profilerna och det frivilliga tillägget.



Figur 2. Multimedia Home Platform – Profiler (efter [4]).

2.2.3 DVB-J

Applikationer för MHP skrivs främst i DVB-Java. DVB-Java eller DVB-J är en

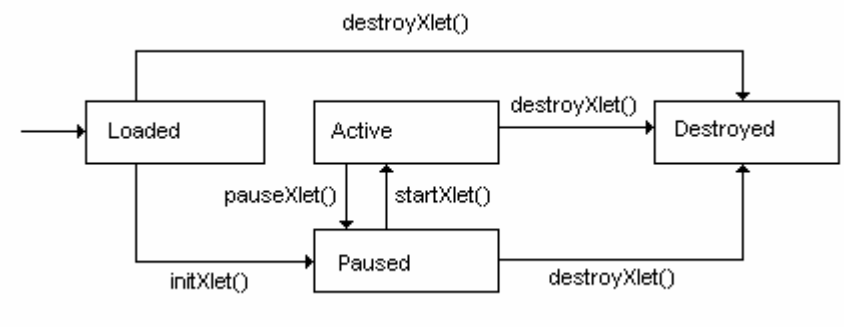
version av Java, baserad på Personal Java 1.2, som är speciellt framtagen för digitala sändningsmedier. I DVB-J har funktionalitet som inte är relevant i en televisionskontext tagits bort medan relevant funktionalitet lagts till [4].

2.2.3.1 Xlet

En DVB-J-applikation kallas för Xlet. Flera Xlets kan exekvera samtidigt i samma virtuella maskin och dela på de resurser som finns tillgängliga. Detta gör att en Set-Top-Box och dess middleware måste kunna kontrollera och styra varje individuell Xlets livscykel. En Xlet implementerar därför ett gränssnitt som definierar fyra metoder som gör det möjligt kontrollera dess livscykel:

- `initXlet()`
- `startXlet()`
- `pauseXlet()`
- `destroyXlet()`

En Xlet kan huvudsakligen befinna sig i fem olika tillstånd *Not Loaded*, *Loaded*, *Paused*, *Active* och *Destroyed* (se figur 3).



Figur 3. DVB-J – tillståndsdiagram för livscykel.

Utöver dessa fem huvudsakliga tillståndet kan en Xlet även befinna sig i tillståndet *Invalid* vilket sker när applikationen inte kan exekvera och inte ännu hanterats av garbage collector [5].

2.2.4 DVB-HTML

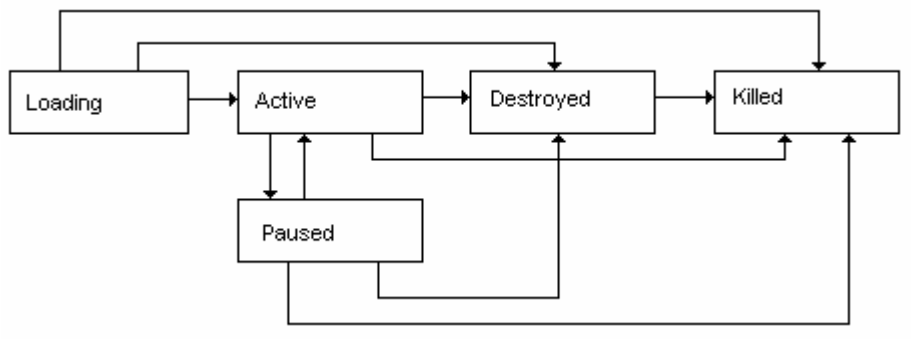
DVB-HTML är ett applikationsformat som tillkommit i och med version 1.1 av MHP-specifikationen. Stöd för applikationsformatet är frivilligt och applikationsformatet ses som ett tillägg utöver de tre profiler som MHP 1.1 specificerar.

DVB-HTML baseras på en del utav de moduler som XHTML 1.0 specificerar och är därför främst baserat på XML 1.0 [3]. Utöver XHTML 1.0 så bygger DVB-HTML på Cascading Style Sheets 2 (CSS 2), Document Object Model 2 (DOM 2) och ECMAScript (JavaScript).

DVB-HTML har stöd för DOM 2 vilket innebär att det via ECMAScript, eller ifrån andra applikationer, går att dynamiskt modifiera innehållet av en sida. DVB-HTML stödjer inte alla de moduler som definieras av DOM 2 och utöver de moduler ifrån DOM 2 som stöds så definierar DVB-HTML även några egna moduler som måste stödjas av användaragenten. De egna moduler som DVB-HTML tillför är moduler som är relevanta i en televisionskontext och i dem ingår bland annat moduler för att hantera olika sorters händelser så som knapptryckningar och signalering av förändring i applikationers tillstånd.

En DVB-HTML-applikation definieras som en samling av DVB-HTML-dokument, det vill säga dokument av typ XHTML 1.0, CSS 2, och/eller ECMAScript. Vilka dokument som ingår i en DVB-HTML-applikation anges med hjälp av reguljära uttryck vilket medför att ett dokument kan ingå i flera DVB-HTML-applikationer samtidigt. Möjligheten att ett dokument ingår i flera applikationer kan utnyttjas av applikationsutvecklare för att spara resurser så som minnesförbrukning och för att minska laddningstider [5].

Precis som för DVB-J-applikationer så är DVB-HTML-applikationers livscykel väl definierad för att applikationen skall kunna kontrolleras och styras av en Set-Top-Box och dess middleware. DVB-HTML-applikationers livscykel skiljer sig dock något ifrån DVB-J-applikationers livscykel. De tillstånd en DVB-HTML-applikation kan befinna sig i åskådliggörs i figur 4 nedan.

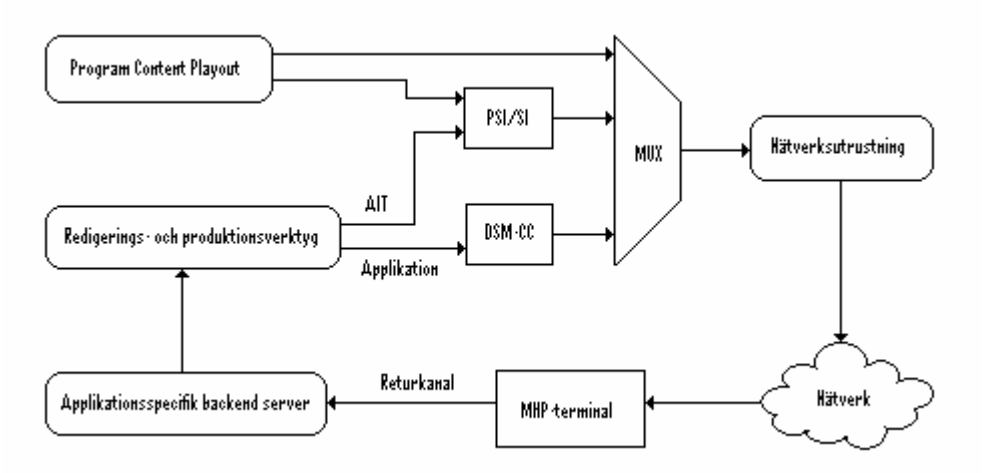


Figur 4. DVB-HTML – tillståndsdiagram för livscykel (efter [3]).

2.3 Den interaktiva sändningskedjan

För att sända ut en applikation till mottagare så måste applikationen inkluderas vid sidan av bild och ljud i transportströmmen och för att klasser och datafiler ska kunna laddas så behövs någon sorts filsystem. För att ge tillgång till ett filsystem så använder sig MHP av en standard vid namn Digital Storage Media, Command and Control (DSM-CC). DSM-CC är en del av Moving Picture Expert Group (MPEG) standard för video, MPEG-2 [4].

DSM-CC använder sig av ett koncept som kallas objektkarusell där kataloger och filer delas upp i en eller flera moduler. Objektkarusellen kombineras med hjälp av en multiplexer med Application Information Table (AIT), som beskriver applikationer, Program Content Playout, som innehåller undertexter, ljud- och bildmaterial, och Program Specific Information/Service Information (PSI/SI) för att forma en gemensam transportström som sedan sänds ut till mottagare [4]. PSI/SI innehåller tabeller och information som beskriver data i transportströmmen så att de olika delarna kan identifieras. I figur 5 nedan syns hela den interaktiva sändningskedjan.



Figur 5. Den interaktiva sändningskedjan (efter [4]).

Modulerna innehållandes applikationer och data sänds ut en modul åt gången i transportströmmen. När den sista modulen ifrån objektkarusellen lagts till transportströmmen och sänts ut så börjar modulerna åter igen läggas till transportströmmen ifrån den första modulen och framåt. Detta medför att applikationer, filer och kataloger sänds ut kontinuerligt med ett visst intervall som varierar beroende på karusellens konfiguration. En mottagare som missat en modul innehållandes en del av en applikation kan därför vänta tills den missade modulen återkommer igen i transportströmmen och därmed komplettera överföringen av applikationen [7].

Returkanalen, som ger MHP-terminalen möjlighet att interagera med sändare via en backend server, är bara tillgänglig förutsatt att MHP-terminalen har stöd för en returkanal och att den implementerar Interactive Broadcast Profile.

2.3.1 Application Information Table (AIT)

AIT är en datastruktur som innehåller information om de appliaktioner som sänds ut till mottagare. I AIT finns bland annat information om följande:

- Applikationens namn
- Applikationens version
- Applikationens prioritet
- Applikationens ID och organisations ID
- Applikationens status
- Applikationens typ (DVB-J eller DVB-HTML)
- Applikationens huvudklass eller HTML-fil

Den status som ges applikationen via AIT kan påverka dess livscykel. I status beskrivs till exempel om en applikation ska autostartas, om den kan ses av andra applikationer och om den är bunden till en viss tjänst.

Det är applikationshanterarens uppgift att hämta och tolka informationen i AIT [8]. En applikationshanterare kan utifrån informationen i AIT presentera, kontrollera och styra applikationer på ett korrekt sätt.

2.4 Interoperabilitet

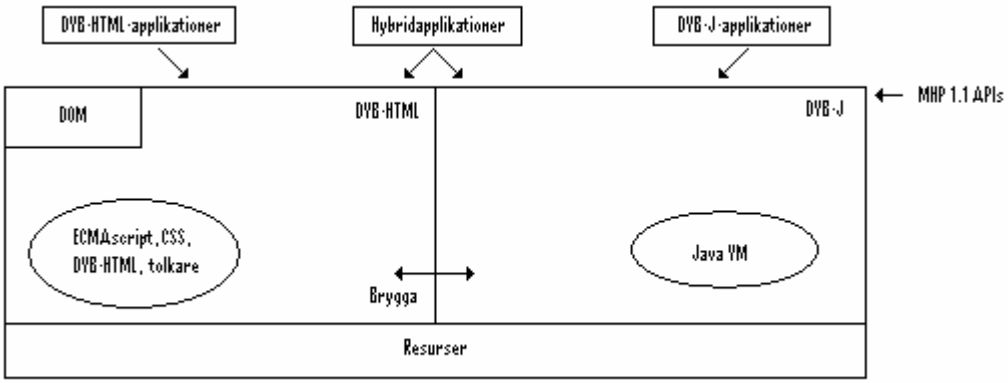
Med interoperabilitet menas möjligheten att mellan olika program, eller funktionella enheter, på ett standardiserat sätt kunna kommunicera, dela information och exekvera

program [9]. Med interoperabilitet avses inte här någon sorts portabilitet det vill säga att applikationerna skall fungera likvärdigt på olika typer av Set-Top-Boxar.

2.5 Metoder för interoperabilitet mellan DVB-J och DVB-HTML

Från och med version 1.1 av MHP är det utöver DVB-J- och DVB-HTML-applikationer även möjligt att skapa hybridappliaktioner. Med hybridapplikationer menas applikationer som kombinerar både DVB-J och DVB-HTML i en och samma applikation. Detta förutsätter dock att Set-Top-Boxen är kraftfull nog att kunna exekvera båda en virtuell maskin för Java och en DVB-HTML användaragent samtidigt [5].

För att bättre åskådliggöra vad hybridapplikationer är och hur de fungerar så har arkitekturen ifrån figur 2 utökats med mer detaljer i figur 6 nedan.



Figur 6. Multimedia Home Platform 1.1 – Grundläggande arkitektur (efter [4]).

Skapandet av fungerande hybridapplikationer kräver ett eller flera standardiserade sätt att kommunicera, exekvera och dela information på mellan DVB-J och DVB-HTML. Interoperabiltet mellan DVB-J och DVB-HTML är därmed en förutsättning för hybdriapplikationer.

Nedan följer utvalda sätt för att uppnå interoperabilitet.

2.5.1 Inre applikationer

Det är möjligt att bädda in DVB-J-applikationer i DVB-HTML-applikationer och tvärtom vilket kallas för inre applikationer. Applikationer är dock begränsade till att enbart kunna ha inre applikationer av någon annan typ än sig själva. En DVB-HTML-applikation kan alltså ha en inre applikationer av typen DVB-J men inte av typen DVB-HTML.

Funktionalitet för att inkludera applikationer inom varandra finns tillgänglig i två paket i MHP 1.1:s API, `org.dvb.dom.inner` för inre DVB-HTML-applikationer och `org.dvb.application.inner` för inre DVB-J-applikationer [5].

2.5.2 ECMAScript-Java brygga

Alla publika DVB-J API är tillgängliga att användas av ECMAScript via paketet `Packages`. Detta gör att det via ECMAScript direkt går att använda de olika publika API som finns i DVB-J och skapa och manipulera Javaobjekt [5].

2.6 Inter-Xlet Communication (IXC)

Av säkerhetsskäl så hålls Xlets helt separerade ifrån varandra när de exekverar i en Set-Top-Box virtuella Java-maskin. MHP åstadkommer denna separation av Xlets genom att specificera att klasser som tillhör olika applikationer skall laddas av olika klassladdare [13]. Detta gör att Xlets i teorin kan exekvera i helt skilda virtuella Java-maskiner förutsatt att detta implementeras av en Set-Top-Box [5].

Användning av separata klassladdare för olika applikationer medför att två applikationer inte kan referera till en och samma klass och därför inte heller kan passa objekt mellan varandra eller använda sig av statiska fält eller metoder för att manipulera samma objekt.

För att kommunicera med varandra måste Xlets använda sig av något som kallas för Inter-Xlet Communication (IXC) och som liknar, och återanvänder delar av, Remote Method Invocation (RMI) [11]. RMI gör det möjligt att definiera och skapa speciella objekt vars metoder kan anropas ifrån Xlets som laddats av andra klassladdare än objektet.

Objekt som tillhandahåller metoder som andra Xlets kan anropa skapas igenom att implementera gränssnittet `java.rmi.Remote` och därefter definiera de metoder som skall finnas tillgängliga att anropa [10]. Objektet exporteras sedan och binds till ett register varifrån Xlets kan begära och få tillgång till en stub-klass av det exporterade objektet. Stub-klassen agerar proxy för det faktiska objektet och dess metoder [10]. Till skillnad ifrån konventionell RMI, där en IP-adress ingår i identifieringen av objekt, så använder sig IXC av ett godtyckligt namn kombinerat med de två fält i AIT som anger varje applikations organisations- och applikations-ID för att namnge och identifiera exporterade objekt.

2.7 Applikationshanterare

En applikationshanterare är en applikation som hanterar de applikationer som sänds ut till eller på annat sätt är tillgängliga för den Set-Top-Box som applikationshanteraren körs på. En applikationshanterares huvudsakliga uppgift är signalera och ta hand om applikationers tillståndsförändringar och kontrollera deras livscykler [8]. Det är även applikationshanterarens uppgift att hantera tillgängliga resurser och ta hand om eventuella undantag som kastas av applikationer.

Information om vilka applikationer som är tillgängliga, av vilken typ de är, och annan nödvändig information hämtas av applikationshanteraren ifrån AIT.

Utöver detta är det även applikationshanterarens uppgift att hantera inmatning ifrån användare och att ge en återkoppling till användaren igenom att presentera information på ett korrekt sätt, det vill säga att utforma och presentera informationen på ett sådant sätt att den lämpar sig för television.

3 Metod och genomförande

Jag har genomfört en litteraturstudie där jag hämtat information om olika sätt att uppnå interoperabiltet mellan DVB-J och DVB-HTML. Litteraturstudien har kombinerats med utvecklandet av en applikation, en applikationshanterare, som via ett standardiserat sätt startar, kontrollerar och avslutar andra MHP-applikationer och deras livscykler. Detta fungerar väl i praktiken för DVB-J-applikationer och i teorin för DVB-HTML-applikationer.

3.1 Miljö

Jag har använt mig av en Set-Top-Box av modell ADB T.75 /MHP DEV. Set-Top-Boxen är en så kallad development box vilket innebär att den lämpar sig för och är speciellt framtagen för utveckling av MHP-applikationer.

Tekniska data för ADB T.75 /MHP DEV Set-Top-Box:

CPU: STi5517 166MHz
RAM: 72 MB
Flashminne: 16 MB
EEPROM: 32 kB
API: MHP 1.0.2

Applikationer för Set-Top-Boxen har utvecklats främst i DVB-J och all kod har därför skrivits och kompilerats speciellt för Java 1.2. För att föra över applikationer till Set-Top-Boxen har jag använt mig av seriellkabel och en proxyserver-programvara vid namn stbproxy.

Den Set-Top-Box jag haft tillgång till och som jag utvecklat för saknar stöd för MHP 1.1 och för DVB-HTML vilket har lett till att jag inte kunnat testa så kallade inre applikationer. Jag har dock med hjälp av den kommersiella webbläsaren Sofia Backstage® Browser [6] kunnat testa att via IXC kommunicera med och kontrollera webbläsaren.

4 Resultat

4.1 Interoperabilitet mellan DVB-J och DVB-HTML

De främsta metoderna för att uppnå interoperabilitet mellan DVB-J och DVB-HTML är inre applikationer och/eller att via ECMAScript nyttja publika DVB-J-klasser. För att kommunicera mellan applikationer kan Inter-Xlet Communication (IXC) användas.

4.1.1 Inre applikationer

Inre applikationer är DVB-J-applikationer inbäddade i DVB-HTML-applikationer eller tvärtom. En inbäddad inre applikation måste dock vara av annan typ än den applikation som den bäddas in i. En DVB-HTML-applikation kan alltså ha inre applikationer av typen DVB-J men inte av typen DVB-HTML.

4.1.2 Inre DVB-J-applikationer

Att inkludera DVB-J-applikationer inom DVB-HTML-applikationer kan liknas vid inbäddning av applets i webbsidor [5]. Parametrar kan skickas med vid inkludering ifrån DVB-HTML-applikationen och tas emot av den inbäddade DVB-J-applikationen. Här följer ett exempel på hur en Xlet kan inkluderas i en DVB-HTML-applikation:

```
<object
type = "application/dvbj"
id = "minXlet"
codetype = "application/javatv-xlet"
classid = "minXlet.class"
codebase = "minXlet/"
height = "320"
width = "256">

<param name = "arg_0" value = "parameter1"/>
<param name = "arg_1" value = "parameter2"/>

<embed height="320"
width="256"
arg_0="parameter1"
arg_1="parameter2"
src="parameter1">
</embed>
</object>
```

4.1.2.1 Fördelar

- Applikationer kan enkelt bäddas in utan att behöva vara speciellt utvecklade för detta syfte. Applikationer behöver alltså inte modifieras för att kunna inkluderas

4.1.2.2 Nackdelar

- Klassladdare för en applikationen och dess inre applikation skiljer sig åt vilket medför att IXC måste användas för att kommunicera mellan

applikationerna. Detta gör att kommunikation mellan applikationerna blir komplicerad.

4.1.3 Inre DVB-HTML-applikationer

För att inkludera en inre DVB-HTML-applikation i en DVB-J-applikation så finns klassen `HTMLApplication` tillgänglig i paketet `org.dvb.application.inner`. Klassen `HTMLApplication` implementerar gränssnittet `org.dvb.application.inner.InnerApplication` som definierar två konstruktörer som tar emot de parametrar som krävs för att kunna starta en DVB-HTML-applikation [13]. Parametrarna inkluderar sökväg till applikationen, applikationens första sida, de argument som skall passas till applikationen och/eller reguljära uttryck som anger vilka dokument som ingår i applikationen.

DVB-HTML ger igenom DOM även applikationer möjlighet att ändra innehållet på en sida, möjligheten att skapa dynamiska sidor. Även DVB-J-applikationer kan ha tillgång till DOM via paketet `org.dvb.dom` [5]. Eftersom både DVB-J och DVB-HTML har tillgång till DOM så kan applikationer utbyta information mellan sig igenom att modifiera och läsa samma delar av en sida. Att ifrån en DVB-J-applikation modifiera DOM är enbart möjligt om DVB-J-applikationen har en inre DVB-HTML-applikation.

4.1.3.1 Fördelar

- Applikationer kan på ett standardiserat sätt byta och dela information igenom att modifiera DOM.

4.1.3.2 Nackdelar

- Det finns inget sätt att explicit stoppa eller avsluta en inre DVB-HTML-applikation annat än att radera alla referenser till applikationen.

4.1.4 ECMAScript-Java brygga

Eftersom alla publika DVB-J API är tillgängliga för DVB-HTML-applikationer via ECMAScript och det globala objektet `Packages` så är det möjligt att skapa nya Javaobjekt med hjälp av operatorn `new [13].` Det kan till exempel göras på följande sätt:

```
<script>
new Packages.java.lang.String(“Mitt nya javaobjekt”);
</script>
```

Det går att med hjälp av ett metaelement vid namn codebase att ange en sökväg till klasser och därmed komma åt även egna publika klasser.

```
<meta name="codebase" content="http://exempel.se/klasser/">
```

Den sökväg som angetts med hjälp av metaelementet codebase används när ECMAScript via `Packages` refererar till en klass som inte är laddad vilket resulterar i att klassen söks i och laddas ifrån den angivna sökvägen om klassen är tillgänglig.

Det är dock bara huvuddokumentet i DVB-HTML-applikationen som kan ange sökvägar.

Det går även ifrån ECMAScript med hjälp av operatorn `Subtype` att definiera subklasser av javaklasser och skapa instanser av dessa med hjälp av operatorn `new` [13]. På detta sätt kan man till exempel skapa en lyssnare som nedan.

```
<script>
function hanteraHandelse (x) {
// gör något
}

function chatKonstruktor() {
this.receiveUserPreferenceChangeEvent = hanteraHandelse;
}

chatListenerType                    =                    new
Subtype("org.dvb.user.UserPreferenceChangeEvent",
chatKonstruktor);
chatListener = new chatListenerType();
Packages.org.dvb.user.UserPreferences.addUserPreferenceChange
Listener(chatListener);
</script>
```

4.1.4.1 Fördelar

- Enkelt sätt att separera DVB-HTML och DVB-J och av dem välja att använda det som bäst lämpar sig för specifika delar av programmet.
- DVB-J-applikationer ger DVB-HTML-applikationer utökad funktionalitet och tillgång till innehåll som inte annars kan kommas åt av DVB-HTML-applikationer.

4.1.4.2 Nackdelar

- Inställningen till typkonvertering skiljer sig mellan Java och ECMAScript. ECMAScript använder sig av så kallad weak typing medan Java använder sig av strong typing. Detta påverkar hur anrop till Javametoder ifrån ECMAScript görs eftersom typerna kan skilja sig åt vid anropet och anrop därför kan matcha flera olika signaturer.
- DVB-J-klasser bör utvecklas med hur de ska användas ifrån DVB-HTML-applikationer i åtanke.

4.1.5 Inter-Xlet Communication (IXC)

IXC (Se 2.6) är något som kan vara viktigt för att kunna uppnå interoperabiltet mellan DVB-J- och DVB-HTML-applikationer. I vissa fall kan IXC användas för att få till ett mer meningsfullt och flexibelt sätt att kommunicera på mellan applikationer. Vid till exempel inre DVB-J-applikationer kan IXC användas för att kommunicera utöver de parametrar som skickas in till applikationen när den startas. Det är möjligt att ifrån DVB-HTML-applikationen och via ECMAScript få tillgång till och anropa metoder som den inre DVB-J applikationen tillhandahåller via ett exporterat objekt och därmed utbyta data mellan applikationerna. I detta avsnitt tar jag upp några exempel på hur IXC kan implementeras.

De applikationer som vill kunna kommunicera med och erbjuda metoder åt andra applikationer måste definiera och exportera speciella objekt till ett register. Ifrån detta

register kan applikationer få tillgång till stub-klasser av de exporterade objekten och via stub-klasserna komma åt de metoder som den exporterande applikationen erbjuder.

För att lista alla objekt som exporterats till registret kan man till exempel göra på följande sätt:

```
String[] foo = org.dvb.io.ixc.IxcRegistry.list(context);
for(int i=0;i<foo.length;i++) System.out.println(foo[i]);
```

Den applikation som vill lista exporterade objekt skall se till att `context` i koden ovan refererar till den Xlet context som passats till metoden `initXlet()` i applikationen ifrån Set-Top-Boxens middleware. Alla exporterade objekt kan importeras och deras metoder anropas och användas.

4.1.5.1 Importera och använda objekt

För att ge exempel på hur ett objekt importeras och hur dess metoder anropas så använder jag mig av kod som importerar och använder ett objekt vid namn control som exporterats av den kommersiella användaragenten Sofia Backstage® Browser [6]. Koden för att importera ett objekt kan se ut såhär:

```
java.rmi.Remote remObject=null;

try{
  remObject =
    org.dvb.io.ixc.IxcRegistry.lookup(context,
    "/8/1/control");
} catch(Exception e){
  System.out.println(e);
}
```

Det som är specifikt för det exporterade objekt man vill importera och som finns i koden ovan är det exporterade objektets namn, applikations ID och organisations ID. I koden ovan är namnet på objektet control, applikations ID är 1 och organisations ID är 8. Applikations ID och organisations ID måste stämma med de värden som signaleras i AIT för den applikation som exporterat objektet (se 2.3.1). Objektets namn bestäms när objektet exporteras (se 4.1.5.2).

För att sedan anropa och använda objektets metoder kan man exempelvis göra såhär:

```
if(remObject != null)  {
  fi.sfd.SofiaRemoteObject s =
    (fi.sfd.SofiaRemoteObject) remObject;
  try {
    s.showXletPage(
      "http://www.student.hig.se/~kp02ldf/index.xhtml",
      true);
    }
    catch (java.rmi.RemoteException e){
      System.out.println(e);
    }
  }
else{
  System.out.println("Sofia control-object not present!");
}
```

I koden ovan anropas en metod vid namn `showXletPage()` som får Sofia Backstage® Browser [6] att ladda och visa den sida som skickas in till metoden som parameter.

Det är viktigt att det importerade objektet konverteras till rätt typ och att man känner till namn på de metoder man vill anropa.

4.1.5.2 Definiera och exportera objekt

För att skapa egna objekt som andra applikationer kan importera och använda för att kommunicera med applikationen via så måste objekten implementera gränssnittet `java.rmi.Remote.` Genom att implementera gränssnittet `java.rmi.Remote` så blir alla publika metoder som finns i objektet tillgängliga att anropa ifrån de applikationer som importerar objektet. Alla metoder i objektet deklareras fritt men måste dock kasta undantaget `java.rmi.RemoteException` så som metoden nedan gör.

```
public void minIXCMetod() throws java.rmi.RemoteException {
// gör något
}
```

Objekt har samma applikations ID och organisations ID som den applikation som exporterar dem. För att exportera egna objekt till registret kan man göra på följande sätt:

```
if (exportedObject == null) {
  mittObjekt exportedObject = new mittObjekt();
}

try {
  org.dvb.io.ixc.IxcRegistry.bind(
  context, "mittObjekt", exportedObject);
}
catch (Exception e) {
  System.out.println(e);
}
```

4.1.5.3 Fördelar

- IXC gör det möjligt för en applikation att tillhandahålla metoder som kan anropas av andra applikationer. Detta medför att det går att fritt kommunicera och utbyta information mellan applikationer.

4.1.5.4 Nackdelar

- Det finns inga metoder som måste implementeras av exporterade objekt och inget sätt att lista tillgängliga metoder.
- Det kan uppstå konflikter om flera applikationer kommunicerar med en och samma applikation. Två applikationer kan till exempel försöka att ändra och läsa samma värde igenom att anropa samma metoder.

4.2 Applikationshanterare

Jag har utvecklat en applikationshanterare som ersätter den inbyggda applikationshanteraren i en Set-Top-Box. Det går med hjälp av den utvecklade

applikationshanteraren att kontrollera vilka applikationer och tjänster som finns tillgängliga för användaren och att via en lista välja vilka av dem som skall visas och/eller köras.

Applikationshanteraren använder sig av AIT (Se 2.6) för att ta reda på vilka tjänster och applikationer som finns tillgängliga. Klassen `org.dvb.application.AppsDatabase` används av applikationshanteraren för att få tillgång till en abstrakt vy över AIT och igenom att implementera gränssnittet `org.dvb.application.AppsDatabaseEventListener` hanteras förändringar av AIT. Applikationshanteraren bygger en lista över tillgängliga applikationer och tjänster varefter en menyn ritas och presenteras för användaren (se figur 7). När informationen i AIT förändras så uppdateras applikationshanterarens lista över applikationer och tjänster automatiskt tillsammans med menyn som presenteras för användaren.

Figur 7. Den utvecklade applikationshanteraren och dess GUI[1] under körning.

Via den lista av applikationer som applikationshanteraren bygger, de objekt den innehåller och gränssnittet `org.dvb.application.AppAttributes` är det möjligt att få tillgång till information om applikationer och tjänster så som namn, typ, prioritet och annat. Med hjälp av denna information kan applikationshanteraren få tillgång till AppProxy-objekt. AppProxy-objekt implementerar gränssnittet `org.dvb.application.AppProxy` och fungerar som en proxy till en applikation. Via AppProxy-objekt och deras metoder kan applikationshanteraren sedan ta reda på vilket tillstånd applikationer befinner sig i. Det går även att vi AppProxy-objekt att kontrollera applikationers livscykler igenom att be Set-Top-Boxens middleware att starta, stoppa, pausa, eller återuppta körning av ett pausade

[1] Graphical User Interface

applikationer. Applikationshanteraren använder sig av `org.dvb.application.AppStateChangeEventListener` för att bevaka och hantera applikationers tillståndsförändringar.

All kontroll över applikationers livscykler sker alltså på ett standardiserat sätt via MHPs API och med hjälp av funktionalitet som återfinns i paketet `org.dvb.application`. För vidare information kring applikationshanterarens implementation se källkoden i bilaga 1.

Användaren kan med hjälp av Set-Top-Boxens fjärrkontroll navigera i applikationshanterarens meny och bläddra mellan tillgängliga applikationer och tjänster. Via menyn kan användaren växla till och starta en applikation som är pausad, ladda och starta en ny applikation eller avsluta en oönskad laddad applikation. Användaren kan välja att när som helst pausa en applikation som körs och få upp applikationshanteraren och dess meny. Det går även att pausa alla laddade program och att dölja applikationshanteraren tills användaren åter har behov av applikationshanteraren och/eller de program som laddats.

Applikationshanteraren har stöd för och ger användaren möjlighet att ha flera applikationer laddade samtidigt och att växla mellan dessa, något som den inbyggda applikationshanteraren inte klarar av. Detta leder till minskade väntetider då applikationer inte behöver laddas om när en användare växlar mellan dem och till att applikationer kan kommunicera med varandra via IXC.

Att kontrollera andra MHP-applikationer fungerar väl i praktiken för DVB-J-applikationer och i teorin för DVB-HTML-applikationer.

Applikationshanteraren har även tillgång till den kommersiella användaragenten Sofia Backstage® Browser [6] och implementerar IXC för att kommunicera med agenten. För att implementera IXC använder sig applikationshanteraren av paketet `org.dvb.io.ixc` och importerar det objekt som Sofia Backstage® Browser [6] exporterar. Detta medför att en användare igenom en knapptryckning kan navigera användaragenten till en eller flera fördefinierade URL:er.

5 Diskussion

Att hitta litteratur som tar upp interoperabilitet mellan DVB-J och DVB-HTML har visat sig vara svårt. Interoperabilitet omnämns i flera källor men syftar då oftast till portabilitet mellan Set-Top-Boxar och de gånger någon källa använder interoperabilitet på samma sätt som jag gör i mitt arbete så är informationen ofullständig eller väldigt kortfattad. De källorna som tar upp ämnet konstaterar att interoperabilitet mellan DVB-J och DVB-HTML är ett komplext ämne och refererar till MHP 1.1 specifikationen [13] för mer information. Jag tycker att det över lag har varit svårt att hitta sammanhängande information om utveckling för Set-Top-Boxar och interaktiv television. Det har tagit mycket tid att sätta sig in i och sammanställa den information jag hittat och det har varit långt ifrån att bara sätta sig ner och utveckla och testa programkod.

Jag har tyvärr inte haft tillgång till någon Set-Top-Box som implementerar MHP 1.1 och DVB-HTML. Detta har medfört att jag helt fått förlita mig på och utgå ifrån den litteratur jag hittat utan att kunna implementera och testa interoperabilitet mellan DVB-J och DVB-HTML fullt ut i praktiken. Resultat av mitt arbete hade helt klart blivit bättre om jag haft möjlighet att utföra praktiska tester.

Applikationshanteraren som jag utvecklat visar dock att det i praktiken går att på ett standardiserat sätt exekvera MHP-applikationer och att med hjälp av IXC kommunicera mellan applikationer.

Vad gäller MHP och framtiden för DVB-HTML så är den osäker. Stödet för DVB-HTML är i dagsläget lågt eftersom många utvecklare av Set-Top-Boxar anser att det är för komplext, krävande och svårt att implementera [4]. Det är dessutom inget som tyder på att DVB-HTML blir mindre komplext i kommande versioner av MHP.

Om stödet för DVB-HTML ökar så kommer varje utvecklare av Set-Top-Boxar att ha en egen implementation av DVB-HTML. Detta kommer med största sannolikhet att leda till samma problematik som för webbläsare på persondatorer, det vill säga att sidor inte ser lika ut i alla webbläsare eftersom World Wide Web Consortiums (W3C) rekommendationer tolkas olika av olika utvecklare. På grund av dessa problem finns det en möjlighet att DVB-HTML försvinner helt och/eller ersätt av någon annan standard [5].

Det finns i dagsläget dessutom webbläsare som kan distribueras till Set-Top-Boxar på samma sätt som vanliga MHP-applikationer och som helt eller delvis kan ersätta behovet av att implementera och stödja DVB-HTML.

6 Slutsatser

Jag har valt att här kort försöka besvara de frågeställningar som presenterades i uppsatsen och att dra slutsatser utifrån det arbete som utförts.

6.1 Interoperabilitet

- *Vilka olika sätt att uppnå interoperabilitet mellan DVB-J- och DVB-HTML-applikationer finns det?*

De främsta sätten att uppnå interoperabilitet mellan DVB-J och DVB-HTML-applikationer är att inkludera applikationer i varandra, så kallade inre applikationer, eller att via ECMAScript komma åt, skapa och manipulera instanser av publika klasser. För att kommunicera mellan program eller funktionella enheter så kan IXC användas.

- *Vilka för- och nackdelar finns det med de olika sätten att uppnå interoperabilitet mellan DVB-J och DVB-HTML?*

Fördelarna med inre DVB-J-applikationer är bland annat att de är lätta att inkludera i DVB-HTML-applikationer utan att de applikationer som inkluderas speciellt behöver utvecklas för detta syfte. Applikationer som fungerar fristående kan alltså inkluderas utan att behöva modifieras.

Nackdelar med inre DVB-J-applikationer är främst att inre DVB-J-applikationer inte laddas av samma klassladdare som DVB-HTML-applikationen som inkluderar dem. Detta leder till att de måste använda IXC för att kommunicera och byta information med varandra utöver den information som kan skickas in som parametrar vid inkludering.

Fördelar med inre DVB-HTML-applikationer är främst att både den inkluderande applikationen och den inre applikationen kan modifiera DOM och därmed skapa dynamiska sidor. Inre DVB-HTML-applikationer och inkluderade DVB-J-applikationer kan därför utbyta och dela information med varandra igenom att modifiera och läsa samma delar av en sida.

Fördelar med att via ECMAScript använda publika DVB-J-klasser är att applikationsutvecklare via DVB-J ges möjlighet att tillföra funktionalitet som saknas i DVB-HTML. Detta medför bland annat att DVB-HTML-applikationer kan komma åt information som inte annars är tillgänglig för dem.

Nackdelar med att via ECMAScript använda publika klasser är bland annat att ECMAScript och Java använder sig av weak respektive strong typing vilket kan leda till att metodanrop matchar fler än en metodsignatur.

6.2 Applikationshanteraren

Jag har utvecklat en fungerande applikationshanterare som på ett standardiserat sätt startar och kontrollerar andra applikationer och deras livscykler. Detta fungerar väl i praktiken för DVB-J-applikationer och i teorin för DVB-HTML-applikationer. Applikationshanteraren tillåter även användare att ha flera applikationer laddade samtidigt och att växla mellan dessa, något som den inbyggda applikationshanteraren inte klarar av. Detta leder till minskade väntetider då applikationer inte behöver laddas om när en användare vill växla mellan dem och till att applikationer kan kommunicera

med varandra via IXC.

Utöver ett standardiserat sätt att lista och exekvera program på så implementerar applikationshanteraren IXC för kommunikation och kontroll av en kommersiell användaragent.

6.3 Tack

Jag vill tacka min handledare Fredrik Bökman för synpunktrer och hjälp med uppsatsen. Jag vill även tacka Fransisco Fraile, Anders Willmes och alla andra på ITVARENA för stöd och för att jag fått möjligheten att utföra mitt examensarbete på ITVARENA.

Referenser

[1] *"Introduction to the DVB Project"*, URL: http://www.dvb.org/technology/fact_sheets/DVB%20Project%20Fact%20Sheet.0307.pdf (2007-04-18)

[2] *"Multimedia Home Platform"*, URL: http://www.mhp.org/about_mhp/WP02%20(MHP).pdf (2007-05-02)

[3] Perrot, P. "*DVB-HTML – an optional declarative language within MHP 1.1"*, URL: http://www.mhp.org/documents/mhp_perrot-dvb-html.pdf

[4] *"The MHP-guide"*, URL: http://www.mhp-knowledgebase.org/publ/mhp-guide.pdf (2007-05-23)

[5] Morris, S., Smith-Chagneau, A. *"Interactive TV Standards"*, Elsevier INC, USA, 2005

[6] *Sofia Backstage® Browser Platform*, URL: http://www.sofiadigital.com/content/view/33/117/ (2007-05-44)

[7] Whitaker, J. *"Interactive Television Demystified"*, The McGraw-Hill Companies, 2001

[8] Peng, C., Vuorimaa P. *"Digital Television Application Manager"*, IEEE International Conference on Multimedia and Expo 2001, Tokyo, Japan, 22-25 augusti, 2001, http://users.tkk.fi/~pcy/P133.pdf (2007-05-28)

[9] URL: http://en.wikipedia.org/wiki/Interoperability (2007-05-25)

[10] URL: http://java.sun.com/j2se/1.4.2/docs/api/java/rmi/Remote.html (2007-05-28)

[11] Giguere, E. *"Getting Started with Inter-Xlet Communication (IXC)"*, URL: http://developers.sun.com/techtopics/mobility/personal/ttips/ixc/ (2007-05-28)

[12] Peng, C., Lugmayr, A., Vuorimaa, P. *"A Digital Television Navigator,"* the International Journal of Multimedia Tools and Applications, 17 (1) 2002, 129-149.

[13] Europen Broadcasting Union, *"Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.1.1"*, URL: http://www.mhp.org/documents/Ts102812.V1.2.1.zip (2007-05-30)

Bilagor

Bilaga 1. Launcher.java

```
/**
* Launcher
*/

import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;
import java.util.Enumeration;

import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;
import javax.tv.xlet.XletStateChangeException;

import org.dvb.application.AppAttributes;
import org.dvb.application.AppProxy;
import org.dvb.application.AppStateChangeEventListener;
import org.dvb.application.AppsDatabase;
import org.dvb.application.AppsDatabaseEventListener;
import org.dvb.application.AppsDatabaseFilter;
import org.dvb.application.CurrentServiceFilter;
import org.dvb.event.EventManager;
import org.dvb.event.UserEventListener;
import org.dvb.event.UserEventRepository;
import org.dvb.ui.DVBColor;
import org.havi.ui.HComponent;
import org.havi.ui.HScene;
import org.havi.ui.HSceneFactory;
import org.havi.ui.HSceneTemplate;
import org.havi.ui.event.HRcEvent;

public  class  Launcher  extends  HComponent  implements  Xlet,  KeyListener,
UserEventListener, AppStateChangeEventListener, AppsDatabaseEventListener{

  private HScene scene;

  private ArrayList appList;
  private AppsDatabase db;
  private AppProxy proxy;
  private String textString="loading ...";
  private int index=0;
  private XletContext context;


/**
 * Constructor
 */
public Launcher(){
}


/**
 * initXlet()
 */
public void initXlet(XletContext context) throws XletStateChangeException {
System.out.println("Debug: initXlet() running....");

  this.context = context;

  HSceneFactory factory = HSceneFactory.getInstance();
  HSceneTemplate hst = new HSceneTemplate();
```

```
  hst.setPreference(
  HSceneTemplate.SCENE_SCREEN_DIMENSION,  //set dimension of screen
  new org.havi.ui.HScreenDimension(1,1), //set full screen
  HSceneTemplate.REQUIRED);               //priority
  hst.setPreference(
  HSceneTemplate.SCENE_SCREEN_LOCATION, //set location of screen
  new org.havi.ui.HScreenPoint(0,0),  //starting at top left corner
  HSceneTemplate.REQUIRED);             //priority

  scene = factory.getBestScene(hst);
  Rectangle rect = scene.getBounds();
  setBounds(rect);

  EventManager manager = EventManager.getInstance();
  UserEventRepository repository = new UserEventRepository("Blue key");
  //take exclusive control of blue button
  repository.addKey(org.havi.ui.event.HRcEvent.VK_COLORED_KEY_3);
  manager.addUserEventListener( this, repository);

  scene.add(this);
  this.requestFocus();
  scene.setVisible(true);

  db = AppsDatabase.getAppsDatabase();
  db.addListener(this);
  addKeyListener(this);
  buildAppList();

  System.out.println("Debug:               Enhanced               broadcast:
"+System.getProperty("mhp.profile.enhanced_broadcast"));
  System.out.println("Debug:             Interactive             broadcast:
"+System.getProperty("mhp.profile.interactive_broadcast"));
  System.out.println("Debug:                 Internet                access:
"+System.getProperty("mhp.profile.internet_access"));
  System.out.println("Debug:    Enhanced    broadcast    profile    version:
"+System.getProperty("mhp.eb.version.major
")+"."+System.getProperty("mhp.eb.version.minor")+System.getProperty("mhp.eb
.version.micro"));
  System.out.println("Debug:    Interactive    broadcast    profile    version:
"+System.getProperty("mhp.ib.version.major")+System.getProperty("mhp.ib.vers
ion.minor ")+System.getProperty("mhp.ib.version.micro"));
  System.out.println("Debug:     Internet     access     profile     version:
"+System.getProperty("mhp.ia.version.major")+System.getProperty("mhp.ia.vers
ion.minor ")+System.getProperty("mhp.ia.version.micro"));
  System.out.println("Debug:                 DVB-HTML                status:
"+System.getProperty("mhp.option.dvb.html"));
  System.out.println("Debug:                 OpenType                status:
"+System.getProperty("mhp.option.opentype"));
}


/**
 * startXlet()
 */
public void startXlet() throws XletStateChangeException {
  System.out.println("Debug: startXlet() running....");
  this.requestFocus();
  scene.setVisible(true);  // show ourself
  repaint();
}


/**
 * pauseXlet()
 */
public void pauseXlet(){
  System.out.println("Debug: pauseXlet() running....");
  textString="Paused ...";
  repaint();
}


/**
 * destroyXlet()
```

```
 */
public       void       destroyXlet(boolean       unconditional)       throws
XletStateChangeException {
  if (unconditional) {
    scene.dispose(); // free resources
    appList=null;
    db=null;
    proxy=null;
    textString=null;
    System.out.println("Debug: death!");
    context.notifyDestroyed();
  }
  else {
    System.out.println("Debug: trying to stay alive!");
    throw new XletStateChangeException("I want to live!");
  }
}


/**
 * buildAppList()
 */
public void buildAppList(){
  index=0;
  appList = new ArrayList();

  AppsDatabaseFilter filter = new CurrentServiceFilter();
  Enumeration attributes = db.getAppAttributes(filter);

  while(attributes.hasMoreElements()) {
    AppAttributes info = (AppAttributes)attributes.nextElement();
    if (info.getName().equals("Launcher")) {
      continue;
    }
    appList.add(info);
  }
  if (appList.size()<=0){
    textString="No applications!";
  }
  else {
    textString="Use up/down arrows!";
  }
}


/**
 * sofiaIXC() - dummymethod to test IXC with Sofia Browser
 */
public void sofiaIXC(){
  java.rmi.Remote remObject=null;

  try{
    // org-ID/app-ID/name
   remObject = org.dvb.io.ixc.IxcRegistry.lookup(context, "/8/1/control");
  } catch(Exception e){
  System.out.println("Debug: "+e);
  }

  if(remObject != null) {
    fi.sfd.SofiaRemoteObject s = (fi.sfd.SofiaRemoteObject) remObject;
    try {
      s.showXletPage("http://www.student.hig.se/~kp02ldf/index.xhtml",
true);
    }
    catch (java.rmi.RemoteException e){
      System.out.println("Debug: "+e);
    }
  }
  else{
    System.out.println("Debug: Sofia control-object not present!");
  }
}
```

```
/**
 * Method to implement KeyListener
 */
public void keyPressed(KeyEvent key){
  if(appList.size() > 0){  // skip if there are no applications
    AppAttributes info=(AppAttributes)appList.get(index);
    proxy  = db.getAppProxy(info.getIdentifier());

    switch(key.getKeyCode()){
      case KeyEvent.VK_UP:
      if(index > 0) {
        index--;
      }
      System.out.println("Debug: Up!");
      repaint();
      break;

      case KeyEvent.VK_DOWN:
      if(index<appList.size()-1) {
        index++;
      }
      System.out.println("Debug: Down!");
      repaint();
      break;

      case KeyEvent.VK_ENTER:
      textString="launching "+info.getName()+" ...";
      repaint();
      // check if application is paused
      if (proxy.getState()== AppProxy.PAUSED){
        System.out.println("Debug: RESUME!");
        try{
          proxy.resume();
        }
        catch(SecurityException e){
        System.out.println("Debug: "+e);
        }
      }
      else{
        try{
          proxy.addAppStateChangeEventListener(this);
          proxy.start();
        }
        catch(SecurityException e){
          System.out.println("Debug: "+e);
          proxy.removeAppStateChangeEventListener(this);
        }
      }
      context.notifyPaused(); // let's pause
      break;

      case 403: // RED KEY
      info=(AppAttributes)appList.get(index);
      if   ((proxy.getState()   ==   AppProxy.STARTED)||(proxy.getState()   ==
AppProxy.PAUSED)){
        if (proxy.getState()!= AppProxy.DESTROYED){
          try{
            proxy.stop(true);
          }
          catch(Exception ex){
            System.out.println("Debug: "+ex);
          }
        }
      }
      break;

      case 405: // YELLOW KEY
      sofiaIXC();
      break;

      default:
      repaint();
      break;
    }
```

```
  }
}

/**
 * Method to implement KeyListener
 */
public void keyTyped(KeyEvent key){
}


/**
 * Method to implement KeyListener
 */
public void keyReleased(KeyEvent key){
}

public void userEventReceived(org.dvb.event.UserEvent e){
  if(e.getType() == KeyEvent.KEY_PRESSED){
    if(e.getCode() == HRcEvent.VK_COLORED_KEY_3){
      System.out.println("Debug: Blue!");

      if(scene.isVisible()){
        scene.setVisible(false);
        System.out.println("Debug: Hide!");
        context.notifyPaused();
      }
      else {
        try{
          proxy.pause();
        }
        catch(SecurityException ex){
          System.out.println("Debug: "+ex);
        }
        scene.setVisible(true);
        System.out.println("Debug: Show!");
        context.resumeRequest();
        this.requestFocus();
        repaint();
      }
    }
  }
}


/**
 * Method to implement AppStateChangeListener
 */
public        void        stateChange(org.dvb.application.AppStateChangeEvent
appStateChangeEvent) {
  if (proxy.getState() == (AppProxy.NOT_LOADED)){
    System.out.println("Debug: application not loaded!");
    textString="application not loaded!";
    scene.setVisible(true);  // hide ourself
    repaint();
  }
  if (proxy.getState() == (AppProxy.DESTROYED)){
    System.out.println("Debug: application killed!");
    proxy.removeAppStateChangeEventListener(this);
    textString="application killed!";
    context.resumeRequest();  // start ourself
    repaint();
  }
  if (proxy.getState() == (AppProxy.PAUSED)){
    System.out.println("Debug: application paused!");
    //textString="application paused";
    scene.setVisible(true);  // show ourself
    //repaint();
  }
  if (proxy.getState() == (AppProxy.STARTED)){
    System.out.println("Debug: application started!");
    //textString="application started";
    scene.setVisible(false);  // hide ourself
    //repaint();
  }
```

```
}


/**
 * Method to implement AppsDatabaseEventListener
 */
public        void        entryChanged(org.dvb.application.AppsDatabaseEvent
appsDatabaseEvent){
  System.out.println("Debug: entryChanged");
}


/**
 * Method to implement AppsDatabaseEventListener
 */
public         void         entryAdded(org.dvb.application.AppsDatabaseEvent
appsDatabaseEvent) {
  System.out.println("Debug: entryAdded");
  buildAppList();
  repaint();
}


/**
 * Method to implement AppsDatabaseEventListener
 */
public        void        entryRemoved(org.dvb.application.AppsDatabaseEvent
appsDatabaseEvent) {
  System.out.println("Debug: entryRemoved");
}


/**
 * Method to implement AppsDatabaseEventListener
 */
public         void         newDatabase(org.dvb.application.AppsDatabaseEvent
appsDatabaseEvent) {
  System.out.println("Debug: newDatabase");
  buildAppList();
  repaint();
}


/**
 * Paint "pretty" pictures - dummylook
 */
public void paint(Graphics g) {

  Dimension size = getSize();

  g.setColor(new DVBColor(255,0,0,128));  //color and alpha
  g.fillRect(100,120,500, 40);  //draw rectangle

  g.fillRect(100,170,500,appList.size()*40); //draw rectangle


  g.setFont(new Font("Tiresias", Font.PLAIN, 32));
  g.setColor(new DVBColor(255,255,255,255));

  try{
    g.drawString(textString, 120, 150);
  }catch(Throwable t){
  }

 for(int i=0;i<appList.size();i++){
   AppAttributes info=(AppAttributes)appList.get(i);
   String menuString=info.getName();

   if(index==i){
     g.setColor(new DVBColor(255,255,255,255));
   }
   else{
     g.setColor(new DVBColor(0,0,0,255));
   }
```

```
   try{
     g.drawString(menuString, 120, 202+(i*32));
   }catch(Throwable t){
   // Catch any errors that get thrown.
   t.printStackTrace();
   }
 }
}
}
```