

POLITECNICO DI MILANO
COMO CAMPUS



Master of Science
In
COMPUTER SCIENCE AND ENGINEERING

IPTV implementation for Television Broadcaster

Prof. Sara Comai
Department of Electronics, Information and Bioengineering

THESIS By
Arunkumar Muthusamy

Academic year 2016/2017

Acknowledgement

First and Foremost I would like to thank my **Prof. Sara Comai** Department of Electronics, Information and Bioengineering. Without her assistance, dedicated involvement and guides in every step throughout the process, this application would have never been accomplished and everyone from Politecnico Di Milano who helped to accomplish this application.

I am very much thankful to my external supervisors **Mr. Francesco Ciacca** Software Architect SkillBill S.R.L and **Mr. Antonio Castaldi** IT Professionist SkillBill S.R.L with their deep knowledge in IT technologies and their involvement to make me to understand the basic knowledge of this application and flexible to implement my own ideas make me to complete this application. And thanks to **Mr. Matteo Borace** Senior IOS developer Viacom S.R.L for my documentation translation

Abstract

The thesis is about up gradation of the television broadcasting from satellite to IPTV broadcasting. This proposal is conceded by a leading television broadcasting company in Italy. Its implementation has to keep in mind about two major requirements: Analysis of the middleware implementation in order to implement the EPG application, the content management system of television broadcaster.

We went through a paid training session in order to understand the middleware implementation and develop the EPG proposal with minimal modifications from the existing system, trained by who already well vest in developing and maintaining the middleware applications for the television broadcasters. Through this training I got familiar with middleware framework WWAF, on top of which we implemented the IPTV functionalities for EPG. The IPTV functionalities of EPG application are extended and implemented from WWAF.

WWAF is a framework implemented in JavaScript technologies and easily extendable as web application. The EPG application is very much similar to the development of modern mobile applications where the connections between the web layer and the underlying native shell are made by means of REST API calls. In other words all the application protocols are based on http. So the entire EPG application implemented as the single page web application development with dynamic contents

Content management system of television broadcaster, provides the data through Rest API calls with secure parameters. I implemented the Rest API calls for some peculiar EPG pages. The thesis describes singularities of the application architecture and major design changes in EPG level with some fine extensibilities of the framework where I gave my main contribution.

Sommario

La tesi riguarda il cambiamento dell'aggiornamento di informazioni delle trasmissioni televisive dal sistema Satellitare a IPTV. Questo progetto è stato concesso da una società di radiodiffusione televisiva Italiana che è leader nel settore. Per l'implementazione del progetto si è tenuto conto di due fattori fondamentali: l'analisi dell'implementazione del middleware esistente (al fine di implementare la nuova applicazione EPG) e il sistema di gestione dei contenuti dell'azienda.

Per comprendere il funzionamento del middleware e capire come sviluppare l'applicazione EPG con poche modifiche è stata fatta una sessione di formazione a pagamento presso l'azienda che ha sviluppato e che attualmente mantiene l'applicazione middleware. Attraverso la formazione è stata presa familiarità con il middleware WWAF sul quale successivamente è stata implementata la funzionalità dell'EPG per l'IPTV. L'applicazione EPG è stata implementata come estensione del middleware WWAF.

Il WWAF è un framework implementato con tecnologia JavaScript e facilmente estendibile come applicazione web. L'applicazione EPG è molto simile allo sviluppo di applicazioni mobile moderne, dove le connessioni tra lo strato web e l'applicazione nativa sono realizzate per mezzo di chiamate REST API. In altre parole tutti i protocolli applicativi sono basati su http. L'intera applicazione EPG è stata implementata come una single page application con contenuti dinamici.

Il sistema di gestione dei contenuti rende accessibili i dati attraverso uno strato di API sicure. Ho implementato le API per alcune nuove funzionalità EPG. La tesi descrive le singolarità dell'architettura dell'applicazione e le principali modifiche di progettazione a livello di EPG mostrando alcune delle estensioni del framework nelle quali ho dato il mio maggior contributo.

Table of Contents

1	Introduction	1
2	Pre-Requesting	2
2.1	GENBOX [3]	3
2.2	Application Framework	3
2.3	Standard Rest API.....	4
2.4	Inter Process Communication.....	4
2.5	Middleware Architecture for IPTV	5
2.5.1	Content handling: ecosystem	5
2.5.2	PVR: Ecosystem	6
2.5.3	Playback: ecosystem.....	7
2.5.4	Stand By: Ecosystem.....	7
2.5.5	Platform adaptation (platform).....	7
3	Technologies.....	8
3.1	Lodash	8
3.2	Stapes.js.....	8
3.3	promise.js	9
4	Wyplay Web Application Framework	10
4.1	Note about model updates.....	12
5	EPG Application Development [9]	13
5.1	Code conventions and best practices adopted in the WWAF framework and EPG APP	13
5.2	Code organization, modularity and extensibility of the WWAF framework and EPG APP	14
5.3	GUI Building using WWAF	14
5.4	EPG App high level components	15
5.5	Integration of Graphics and Video	15
5.6	Banner component	16
5.7	Grid Component	17
5.8	DTT Mutex [7]	17
5.9	Interactive Apps framework.....	18
5.10	Recording within WWAF and EPG APP.....	18

5.11	Reminders within WWAF and the EPG App	19
5.12	Developing within WWAF and EPG APP.....	20
5.12.1	Introduction on WWAF and EPG APP development	20
5.12.2	Required skills for WWAF and EPG APP development.....	20
5.12.3	Setup of the development environment	20
5.12.4	Deploying and testing on the development environment	22
6	User Interface Implementation for IPTV.....	24
6.1	Boot with First install	24
6.2	Boot without First install	24
6.3	TV stream menu (live full screen)	24
6.4	Banner Component.....	25
6.4.1	Channel List.....	25
6.4.2	Program List	25
6.4.3	Audio subs.....	26
6.4.4	Ribbon.....	26
6.4.5	Synopsis	27
6.4.6	Footer Pin Component	27
6.4.7	Banner Pin Component.....	28
6.4.8	Zap.....	28
6.4.9	Timeshift action.....	29
6.4.10	IP Recording (single/series).....	30
6.4.11	DTT Recording	31
6.5	Main Menu	31
6.5.1	Opzioni Menu.....	31
6.5.2	Configura menu.....	33
6.5.3	GuidaTV menu and subcategories.....	34
6.5.4	MyTV menu and subcategories.....	36
6.5.5	On Demand and subcategories	38
6.5.6	Ricerca	42
6.5.7	Parental control.....	42
6.5.8	Altre impostazioni	43
6.5.9	Modifica PIN.....	43

6.5.10	Interattivi	43
6.5.11	Contatta Tim	43
6.5.12	Contatta SKY.....	43
7	Conclusion.....	44
8	References.....	45

LIST OF ABBREVIATIONS

API	Application program interface
APP	Application
AVIO	Audio Video Input Output
CMS	Content Management System
CDS	Common Directory Service
CMDC	Content Metadata Discovery Component
DVB	Digital Video Broadcasting
DOM	Document Object Model
GUI	Graphical User Interface
HLS	Http Live Streaming
IPC	Inter Process Communication
IPPV	Impulse Pay per View
IPTV	Internet Protocol Television
IAPP	Interactive Application
MVC	Model View Controller
PPV	Pay per View
PVR	Playback Video Recorder
REST	Representational state transfer
STB	Set Top Box
SVOD	Subscription Video on Demand
TVOD	Transactional Video on Demand
OTT	Over The Top
UI	User Interface
VOD	Video On Demand
WWAF	Wyplay Web Application Framework
WyRest	Wyplay Rest
STB	Set Top Box

1 Introduction

This entire thesis is about upgrading the satellite version of television broadcasting to IPTV broadcasting. In order to achieve this proposal, first of all had a keen knowledge about satellite television broadcasting functionalities and its limitations. At the same time I gathered knowledge about IPTV functionalities and its features. From these analyses the thesis implementation needs strong middleware accessible applications related to television broadcasting.

The higher-level structure of the thesis is providing a light weighted application to the end users, which contains CMS, IPTV features and User intractable interface design. Our main focus is towards extending those middleware IPTV features and access via web application.

While implementing the application I concentrated on the architecture, which consists of the CMS, middleware accessibility and web application communication with middleware and user interface design. All these modules act as independently accessible application. CMS can access only through Rest API calls for all the information fetch. So no need of local storage, no need of security firms and easy to access the data. Middleware has been specially implemented for the IPTV purposes like schedulers, recorders, live content playback and developed with the mixture of C/C++ and python. It makes use of dbus for inter process communication and http REST for client API. In fact from the service to the end user perspective the application can be implemented as an identical fusion solution except that there is no need for a connection to a satellite dish, but broadband connection to Internet is used instead.

EPG application, the most presentable part where all the CMS and middleware works embedded in STB kind of Linux machine that runs only this EPG application. A fine extendable, rewritable with lot of flexible JavaScript framework technologies are used to develop this web application part, Even though its UI application, the framework itself contains MVC structure in order to maintain the Rest API calls, user interface designs and dynamic content handling everything. So in a whole the combination of CMS, middleware and web application will make this IPTV broadcasting as a well defined, future extendable and most user accessible application.

2 Pre-Requesting

Before developing the EPG allocation for IPV we have to know about the architecture of the whole IPTV, Which consists of different layers with different components. The understanding of the communication between the layers and functionalities of the components will lead us to develop the application in efficient way with the help of Installed Genbox, Frog prebuilt, REST principles and web development (JavaScript, HTML5, and CSS3). In the below architecture diagram will explain everything clearly.

Overall Architecture[1]

The overall architecture is illustrated in the following picture

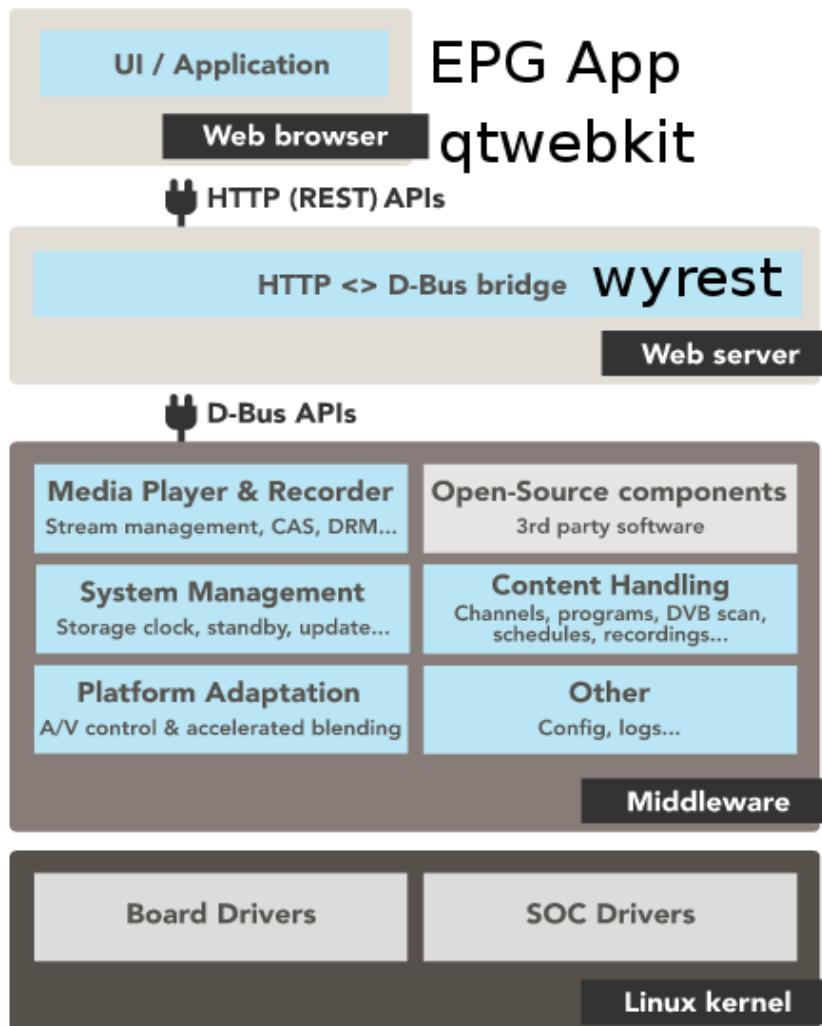


Figure 1

In Figure 1, the bottom Linux kernel is mandatory for this application.

2.1 GENBOX [3]

Genbox is the build environment for EPG applications. It consists of profiling, target and overlay. Profiling required components like name, version and options. Targets, Root file system profile build result. Overlay is component store. In Figure 2, illustrate how to resolve the dependencies and convert source code converted to target app.

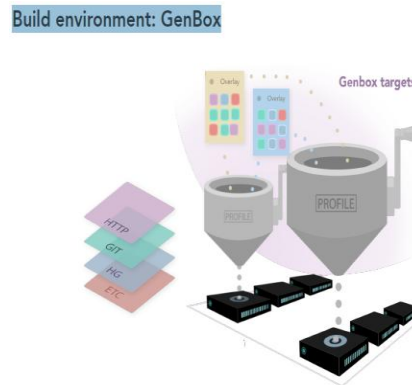


Figure 2

Genbox, general benefits are application distribution and integration tool: Source & binary packaging system, Flexible build options. EPG Genbox benefits are Multi-target, Native transparent cross-compiling, Reproducible compiles (configure once, compile anywhere).

2.2 Application Framework

Figure 3 illustrates, the entire EPG app has been developed as HTML5 app running in WEBKIT and interfacing the rest of the middleware by means of REST-API exposed by wyrest component. For the HTML5 part a new framework has been developed (name WWAF). The WWAF is reusable JavaScript framework for HTML5 UI. It has been started for the EPG project, with the goal to be re-usable in future projects.

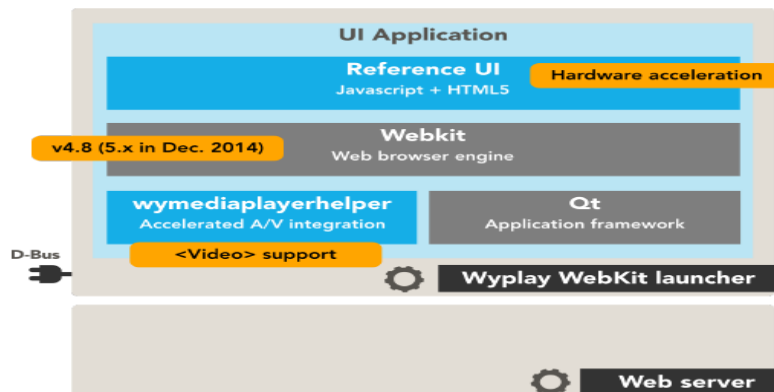


Figure 3

2.3 Standard Rest API

Figure 4, Explains how the Rest API communicate through middleware to web server. In the entire application framework the reset api access as Server Side Events signals. It ensures the blazing fast web server and companion device friendly.

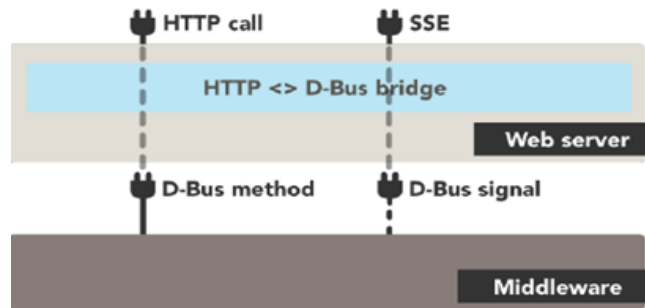


Figure 4

2.4 Inter Process Communication

In Figure 5, the entire application how connected with other services like recording , time shifting etc. The internal communication takes place through D-Bus (or the daemon language), it is not directly but used by the UI (wyrest application wraps it). Because of Widespread usage & many tools, Optimized APIs, Simple, complete, Secured (ACL based), IDL for constants, proxies & adapters, documentation (XML files), it is used for the inter process communication moreover in that especially we are using Shared D bus, for unified Exchanges data & alerts between middleware components, message-based, method call (sync or async) or signals (pure async), object-oriented, services expose object hierarchy, implementing interfaces

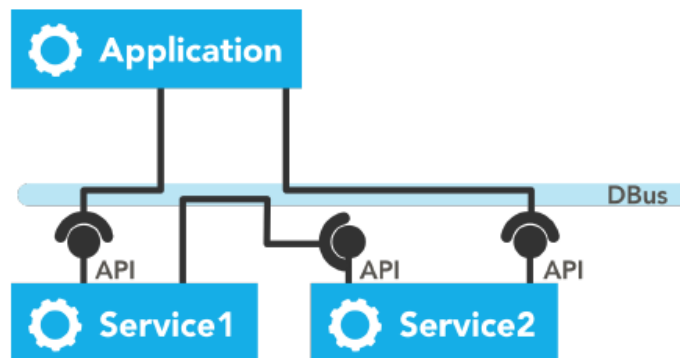


Figure 5

2.5 Middleware Architecture for IPTV

Middleware plays a vital role in EPG application development. Middleware as a platform for the Set top boxes (STB) running Linux. The middleware developed with the mixture of C/C++ and python. It makes use of dbus for inter process communication and http REST for client API. Hardware related components are built in C/C++, while business objects and server part is built in python. A python component, wyrest, acts as a server for the http Rest API, publishing through it the interfaces of the others middleware components. Middleware consists of Native Linux Open source, scripting languages for UI, C & C++ for middleware, Multi-process + IPC. The typical structure of the application consists Minimal porting: boot the target UI: modify/adapt Chipset & hardware: porting additional drivers Middleware: extend & customize Conditional access: integrate & certify Firmware update: integrate backend & boot loader hook.

Below the explanation about the middleware components with its communication between the EPG application and the sequence flow with each component towards application framework.

2.5.1 Content handling: ecosystem

Content handling plays a vital role under the wyrest API; it is used for retrieving program and channel data. It received in many forms: as IP HLS stream for live channels, as DVB stream coming through the DTT RF tuner, as on demand content as HLS IP stream. The content basically deals with Options, Post, Get and Delete request methods. So each and every request before going to perform the request call, the option request method will be executed and cross check with the server whether the request URL has the following request options like (Post/Get/Delete). In figure 6, illustrates the content handling ecosystem's position where it takes place in the overall application.

Wyrest deals with three different types of content handling,

1. Content for single selection
2. Content for multi selection
3. Content item for both single and multi selection

In content handling, mainly focused on TV scan and browsing with respect to UI and MW sequence.

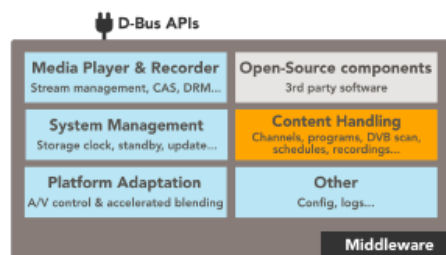


Figure 6

2.5.3 Playback: ecosystem

This playback component will take care of playing the scheduled and recorded events from the STB which is demonstrated in Figure 8. Basically these data are stored in the STB itself. And listed in an appropriate page to play or to do other operations and the WWAF, provides functions to access this playback option from UI.

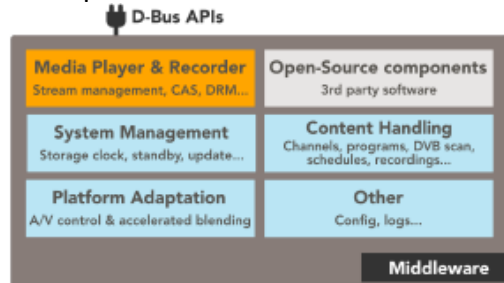


Figure 8

2.5.4 Stand By: Ecosystem

The standby functionality developed in this application, the purpose of this component will explain in the application development face and the figure 9 explains where this standby ecosystem comes to the picture. Moreover it also extended as web application function call through WWAF.

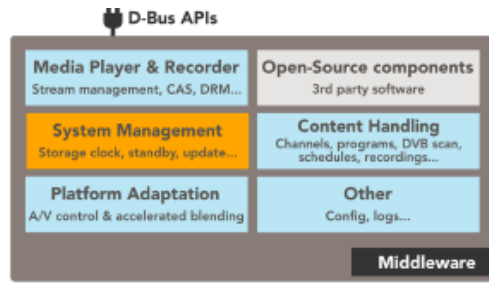


Figure 9

2.5.5 Platform adaptation (platform)

The figure 9 shows the platform adaptation position and its communication flow. It controls both audio and video input and output functionalities – AVIO. Moreover it composes of graphic layers (video, UI, subtitles, etc...)- Wyrender accesses its own backend/middleware/service with custom code- Custom adaptation plug-in etc.

3 Technologies

The below software third parties libraries are involved in the development of the EPG application through WWAF framework

1. Lodash [4],
2. Stapes.js [5],
3. promise.js [6]

3.1 Lodash

Lodash is a utility library with many useful objects and methods. It is used for data structure manipulations, performing lookups, string manipulation, object checking. Array and hash manipulations are wrapper for useful method chaining among the others. The inclusion of this library can be considered a smart choice: it is the result of a good compromise between features and size, and its usage standardizes a lot of common task across the whole code base. IT is pretty similar to the better known underscore.js. However Lodash fits better in STB for its compactness and consistency.

3.2 Stapes.js

Stapes.js is “the little JavaScript framework that does just enough”. In fact it is aimed to provide a limited set of constructs and language integrations so, although it is quite powerful, it is a good choice for acting as building block for a more complete framework. In WWAF it is used exactly in this way: the WWAF framework itself is based on stapes objects, using its Class structure as base building block for WWAF.

Stapes in WWAF provides:

1. Class definition and inheritance
 - a. MVC base classes are stapes Class
 - b. In WWAF MVC, model objects are stapes object. WWAF in particular relies on stapes notifications, view updates at model changes are triggered as stapes notification
 - c. Class inheritance using stapes is widely adopted in WWAF and in our application, all the framework hierarchy is built on it
2. Dynamic data methods and attributes
 - a. the common practice adopted in WWAF /EPG API about data transfer object is to wrap this data in a Stapes class
 - b. method such **getAll()**, **each()** and **filter()** of a Stapes object is used extensively in order to make attribute navigation and modification flexible and dynamic
3. Events
 - a. Events generated by data object play a big role in Wwaf/Polka, the model updates events are emitted as stapes events since model are Stapes objects
 - b. The simple event registration model make the Wwaf/Polka more readable and less lengthy
 - c. the custom events mechanism is also widely adopted when the update has to be mediated or aggregated

3.3 promise.js

A light weighted JavaScript implementation. It provides an alternative to callback-passing. Asynchronous functions return a Promise object onto which callbacks can be attached. Callbacks are attached using the **.then** (callback) method. They will be called when the promise is resolved.

```
var p = asyncfoo(a, b, c);
p.then(function(error, result) {
  if (error) return;
  alert(result);
});
```

Asynchronous functions must resolve the promise with the **.done()** method when their task is done. This invokes the promise callback(s) with the same arguments that were passed to **.done()**.

```
function asyncfoo() {
  var p = new promise.Promise(); /* (1) create a Promise */
  setTimeout(function() {
    p.done(null, "O hai!"); /* (3) resolve it when ready */
  }, 1000);
  return p; /* (2) return it */
}
```

The entire application having enormous asynchronous requests and responses sure these lead to parallel process data loss or functionality exceptions. So these JavaScript technologies will take care all these issues.

4 Wyplay Web Application Framework

The EPG APP realizes the GUI of STB as a single page application adopting an extended variant of the MVC (“Model-View-Controller”) paradigm. Adopting this schema in an HTML5 app makes the development substantially different from what is the common html GUI development. The html file itself is absolutely minimalistic while all the GUI part are built programmatically by dynamically including components and widgets which alters the underlying DOM and event handling according to their view properties and controls. A lot of widely adopted JavaScript frameworks share the MVC paradigm, Backbone.js, Angular.js, Ember.js to name a few. Those frameworks are general purpose and not always ideal for an embedded environment. Furthermore in the EPG app case, handling of lists (list of channels, programs, schedules...) is absolutely critical and the underlying middleware component (CdsSource) has already built-in functionalities for listing data, with perfecting, cursor indexes and pagination. Because of this, the EPG APP instead of lying on some other well known frameworks, it uses its own MVC framework, the so-called WWAF (“Wyplay Web Application Framework”).

The WWAF framework, in respect to any other MVC framework, has the following peculiarities:

- The view gathers data from the model always as data updates (“Observer Pattern”). Even initial loading is treated as an update
- The data is exchanged between Model and View always through the Controller, letting the Controller to be always able to enrich, transform or apply any transformation to data flowing through it. This is useful since the data model exposed by the middleware can differ a lot from the one used in the View

The common workflow of WWAF MVC implementation is:

1. Controller tells Model what should be displayed.
2. Model gives Controller data/delta.
3. Controller adapts data then gives them to the View.

This Controller mediated data exchange is the base mechanism for make the Controller adapting the data to the view. The data source behind the Model is accessed using Proxies. These proxies are useful to reuse as well the fetched data can be shared by multiple consumer. It must be said that this indirect mechanism makes the data flow more obscure and unintuitive. Also proxies are often hacked and abused making the code more complicated to understand.

MVC implementation in WAS adopts the common workflow of WWAF MVC schema. This configuration is a cascading effect because a change on the Model induces a change on the Controller, and a change on the Controller induces a change on the View.

However, the Controller is still useful. It is more than just a proxy between Model and View. It can add context-related information to data (like ongoing selection in the listing, or managing states) and be used as an adapter to process the Model data before sending them to the view.

Moreover, for Containers, Controller tells the Model what Items have to be exposed. Here is a simplified diagram to explain the global workflow for a listing of items. Figure 10 explains the flow between the front end MVC framework.

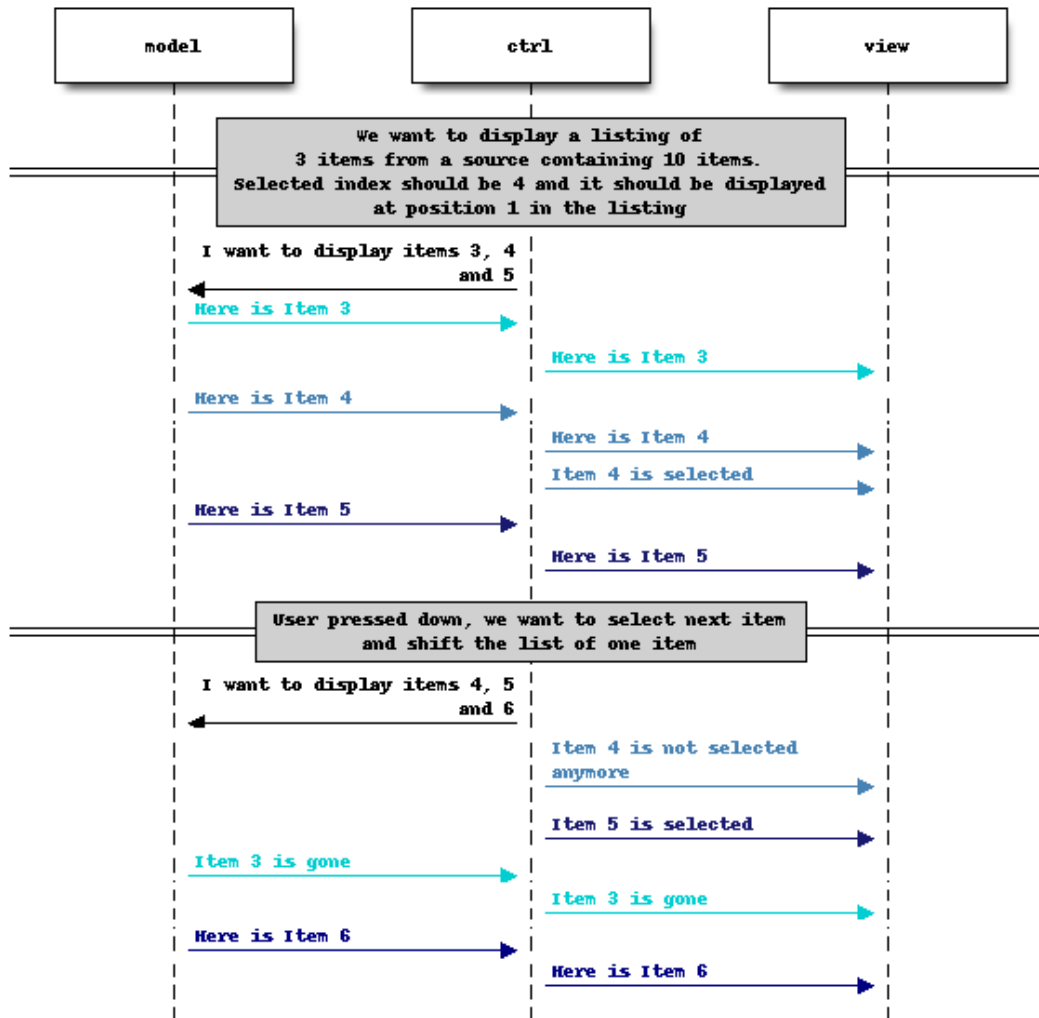


Figure 10

Our Model and Controller are not “pure-MVC” because the listings logic is shared between both (when it should be a pure Controller thing). **Controller manages navigation**-logic, and **Model manages source**-logic.

As simple as it can be: Container contains Items. For each type of Container (Model, View or Controller), there is an associated type of Item. The workflow between types of Containers is the same that between the type of Items, i.e. They are linked through Stapes signals. Container content is a collection of Items, and Item content is a collection of metadata.

Here is the organization of the classes.

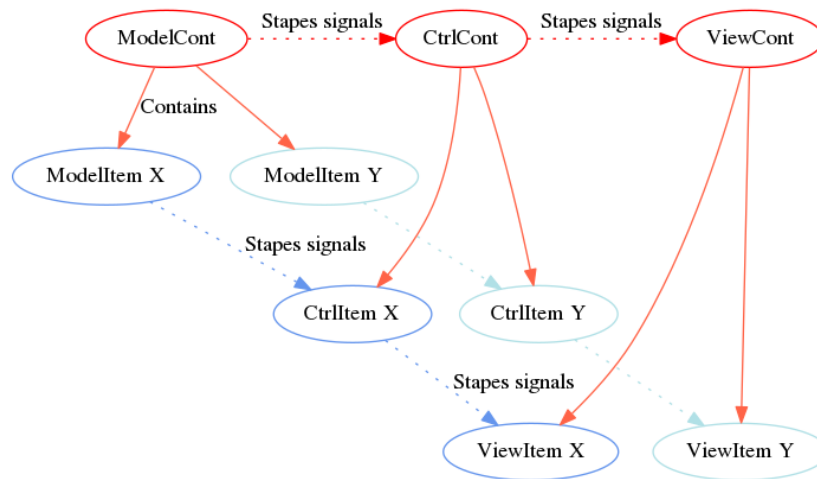


Figure 11

Figure 11, explains the MVC processing flow is not that simple, because for each event on Container (i.e. an item has to be added or removed), a “sub MVC processing flow” is created for the corresponding item. A lot of the work in EPG is about lists navigation. List navigation occurs when user navigates through channels, and through programs belonging to a channel. Those navigations are made against lists with potentially a large amount of items each one carrying a large set of attributes. Those lists also are susceptible to be updated while the user is navigating the dataset. The common pattern in EPG app for such tasks is to use a Proxy instance between the current content query (producer) and the object using it (consumer). The proxy allows asynchronous operations decoupling the consumer requesting data and the producer retrieving it.

Furthermore the proxy registers itself for any Server Side Event which may invalidate the query results since they are changed meanwhile. Once received this event, the model is outdated, the controller receives the model outdated event and reloads the data updating the model which updates the view.

4.1 Note about model updates

This events chain is widely adopted in EPG app. Programs list in GuidaTV and in the banner are handled in this way. All the information about program and channels is retrieved with a single query (paginated) using many tables aggregating truly EPG data with user’s data like recordings, schedules and reminders.

This has some drawbacks: when the user inserts or delete of these items triggers the event chain for update, which can take a while, these results in slow updated of the GUI signaling for example that a new reminder is set. Some workaround is in place for this, for example the recording or reminder icon is forced to be shown immediately without waiting for the update chain, and however the BIL is still updated lately, so the resulting responsiveness is not excellent

5 EPG Application Development [9]

In application development some coding standards are followed in order to adopt with WWAF framework.

5.1 Code conventions and best practices adopted in the WWAF framework and EPG APP

1. JavaScript “**strict modes**” is mandatory
2. Module dependencies is expressed using **node.js**’s require construct, with browserify used to pack all the required libraries in the final JavaScript file used in the browser
3. Modules exports only specified objects and functions, namespace pollution risk is highly mitigate
4. All the framework sub-modules are exported under the wwaf namespace (wwaf.ajax, wwaf.input etc...)
5. All I/O operations are performed asynchronously. All the functions performing such operations are supposed to return a Promise. This is absolutely mandatory and the usage is almost ubiquitous in Wwaf/Polka. Promise helps developers to chain asynchronous actions in a cleaner and more readable way in respect of just using callbacks. All returned Promises have to respect the function (error, result) signature.
6. Fake promises used to respect the interface must be instantiated using `wwaf.tools.getDonePromise(false, settings)`. Fake promises are useful when for some reason a method expected to make an asynchronous call does not need it anymore.
7. All the texts must be mapped under trad.json and screen.json, and used only through key-value mapping on those JSON files.
8. Private functions belonging to a class should start with the prefix “_” (underscore)
9. In Wwaf/Polka there is no the concept of routing as you can find in many single page framework such as AngularJs, instead it’s up to the application code to switch between application contexts or navigate through menu items
10. Mixing video and graphics is simple: the default browser background is fully transparent, showing the video content under it. Transparency is realized setting the opacity of the html elements
11. User events dispatching is pretty simple: GUI components are susceptible to receive such events, for example if a component should handle a key (says the “Sky” button), it is enough to add the function `onRcuEventShortSky`

12. OSD messages are controlled and spawned through a common infrastructure, for example adding new error messages is implemented just by specifying its error message and basic behavior. These OSD can appear both in full screen and within the mini TV (over the video pip window)

CSS files are built using the Stylus preprocessor (<https://learnboost.github.io/stylus/>), the stylus file format is more compact and flexible: in CSS clause with stylus you can embed functions, import and extend other declarations, make use of variables and mixes. Also background images can be in-lined as Base64 encoded images. In EPG app background images are typically embedded in such way.

5.2 Code organization, modularity and extensibility of the WWAF framework and EPG APP

The code organization of the EPG app projects is built across a strong separation between code belonging to the WWAF framework and code belonging just to the EPG App. Even the **GIT** source code repositories are separated. The idea behind for Wyplay is to endorse as much as possible the WWAF framework, making it available to all the Frog community. With this target in mind, Wyplay team kept separated what of EPG app is generic and reusable and what it is specific to the EPG app project. It must be said that this separation worked not as much as expected: since WWAF wasn't developed together with its reference TV application, most of the code base relies on the EPG app tree, while the WWAF codebase is strictly bound to the building blocks. The result is very constrained code base for the WWAF framework and a large code base for the EPG App, which may be not helpful for a quick adoption of the WWAF framework in other Frog projects. A large adoption of the WWAF framework could lead to a better supported platform in the mid-terms, so not succeeding in it may affects EPG app in the future.

5.3 GUI Building using WWAF

We already illustrated the basis for the MVC model in Wwaf. However a triplet Model View Controller has to be contained in a visual component by which the app can control the effective visualization and disposal of the View, specifying the wiring with its model and its controller, and the interaction with the end user and reactions to SSE events. In Wwaf such object is the so called "**Component**" (and its subclasses). The Component composes the MVC triples and specifies data binding between the model and the view.

There a bunch of components for all needed purposes, including container components which are actually responsible of displaying page composing inner components as sub-page elements. For the View the Components specifies its options including its **template**. Templates are basically chunks of html. CSS class names are used to identify elements in each template,

following a simple convention for these names give simple way to specify data-binding between the model and the view. If for example a template has to be named reminder-banner, the CSS class of the containing div must be reminder-banner (adding other classes is still possible), while all then contained elements have to have a CSS class name “reminder-banner-XXXX” and must be unique. The XXXX will correspond to model fields, and in this way the data binding is implemented.

5.4 EPG App high level components

The EPG App, as said, is a single page application. The single html file includes elements acting as placeholder for all the main graphical components of the GUI.

The **first install** element is used for displaying any message related to the first boot of the STB.

The **full screen** element is used when the application needs to display content while the video is displayed in full screen. The main usage for this element is for displaying the banner. Its sub elements are the placeholder for the iapp button and the fingerprint.

The **app** element is used to display all the part of application displayed while the video is not full screen, GuidaTV, My TV, and so on. When this app is shown, the application cover all the screen, eventually display the video content in the mini TV. Its sub elements are the top-level menu, the ddm, the title, the main content, the mini TV, the clock.

The **iapp** elements are used for the visualization of the iapps. Internally it has an iframe used to be loaded with the iapp content, plus a set placeholders used for the visualization of some content over the iapp itself: OSD, reminder banner and fingerprint have all specific placeholder for this.

The listed elements correspond to the possible contexts of the application. Switching between contexts is implemented acting on the display CSS attribute of those elements, only one element at time is not hidden.

5.5 Integration of Graphics and Video

The integration of video and graphics in EPG is quite straightforward. The main idea is that the web kit is placed on top of the video window, with its default background set to fully transparent. Furthermore the opacity CSS attribute is fully supported also in respect of the underlying video content. In this way, if the application is need to display the main app in full screen mode, the app element display attribute is set to visible. Its background is blue and opaque so the video is fully covered. If instead only the banner has to be displayed, the only show context element must be full screen, displaying only the banner with partially cover the

video. The opacity of the container element's value between 0 and 1, in this way the video under the banner is partially visible.

The AVIO middleware component can be controlled in order to resize and move the video window. In case of the miniTV for example, the box where the video has to be placed is left empty by the graphic, while by means of the AVIO controls the video is resized and moved exactly in the empty place. The AVIO routes control separately the video output for analog (RGB) and digital (HDMI) output, so two REST api calls are needed for that purposes.

5.6 Banner component

The banner components are the part of the GUI where the user accesses information about current channel, current program (or vod or pvr content). The banner components are also fully navigable: user can navigate through channels and through programs (current and futures). The banner also gives end user access to other functionalities: switching audio and subtitle language, inserting PIN for protected content, zapping to channel, buy ippv content, set/unset recordings and reminders.

The banner components are displayed under the full screen context.

The banner component source file is located in `app/screen/fullscreen/contexts/tv/banner.js` (beside the TV case other two similar components are present, one for the vod and one for the pvr). The data source for the model is mediated through a proxy (multiselect proxy). This solution allows sharing the same data source (the proxy itself) between the subcomponent navigating the channels and the subcomponent navigating through programs within the channel.

The multiselect query used by the proxy carries several metadata including user generated data, such recordings and reminders. This information is shown by icons disposed on the box containing the title. User can also toggle this information, setting/unsetting on a program a schedule recording or a reminder. Furthermore these actions can be set on single program or on the entire TV series belonging to. In that case a popup selection is shown to the user for selecting the desired option.

When for example the user set a recording, the component invokes directly the `wwaf` call for setting such schedule. This implies a modification on the results of the multiselect query originating the data in the model, this make the model outdated. The controller receives the model outdated event and reloads the data updating the model which updates the view. After that the view is updated with the proper recording icon set. Since this chain can be time consuming, the introduced delay is too big so a workaround is in place: when the component set the recording it also updates the model forcing the icon to appear.

However this workaround is not fully operational: the ribbon component (the component displaying the action currently available with their button) is not updated until the chain is

completed. This drawback has not been fixed since it implies a complete refactoring of the solution, since it is not addressable by just a workaround. The result is that although the user can see quickly, the result of his set action (the icon appears) meanwhile he can't unset it until a refresh is completed. The duration of this incoherent visualization is about 2-3 seconds, however it may have a longer duration under heavier load, and in case of future developments SKY must take care of it since they can introduce longer delays with the duration this phase becoming too big to ignore. The zapping code is located in the banner component. Furthermore, other part of the application which needs to zap at the end will call zap functions of the banner. It is evident that in this case a GUI component acts as a service provider for the rest of the application; this is really a poor design and a source of issues.

5.7 Grid Component

The grid component is the component responsible of the implementation of the GuidaTV page. The grid component source file is located under `app/screen/guidatv/grid.js`. Although it has much functionality in common with the banner, it shares almost nothing with it. This is a big issue, since most of the code must be replicated on the two files, without any factorization of logic or GUI operations between the two. Using the GuidaTV user can to browse TV guide up to 7 days, schedule record, set reminders, or buy PPV.

The grid then has its own proxy, again with its multiselect query susceptible to be updated whenever a modification occurs, whether it is due to and EPG update coming from the backend or from a local set/unset of a schedule/reminder or it is due to the time progress.

In fact the grid shows the data in a tabular way where the x coordinate represents the time, starting from now and ending 7 days later maximum. As the time pass by, the grid content is left shifted accordingly. Along the y coordinate the user selects the channel. Both the channels and the time are scrollable, according the pagination the query is re-executed with proper results bound.

5.8 DTT Mutex [7]

One of the critical objects handled by the banner and by the grid is the DTT Mutex, it is used in order to manage resource contention about the DTT tuner. Keeping in mind that the STB may have (now or in the future) active recording on a DTT channel. Since the DTT tuner in the STB is just one, there is no chance to zap to another DTT channel while the DTT tuner is currently used for recording. The DTT Mutex controls the access to the DTT tuner letting the use zap a DTT channel only when dtt recording is not active. The inner status of the DTT Mutex is kept synchronized by using listener for the "schedule" event type: if an event for an active recording is received, the DTT Mutex is raised.

Not only the grid and the banner should respect the DTT Mutex whenever a DTT zap is needed, the same constraint must be respected by all the code implementing channel zapping: for instance the reminder popup raised to inform the user that a program is going to start, if the user want to switch to that channel, the DTT Mutex rule must be used. In order to do so the make Zap of the banner.js is going to be called. Again, this result in a poor design where there is no service layer, and GUI object enriched with a lot of logic are used instead.

5.9 Interactive Apps framework

The EPG app solution includes a way to enrich channel content with interactive application which can be loaded on demand and can run on top of the EPG UI. From the application UI perspective, interactive apps are HTML5 applications loaded inside the iframe under the iapp context element.

The iapps have some capabilities specific for the Sky STB, as purchasing content, zapping and so on. These capabilities are enabled by means of a JavaScript library which the iapp can refer to, acting as a bridge between the iapp and the underlying middleware. Also, when closing the iapp, the iapp developer can choose which menu and sub action activate on the EPG app.

From the EPG app perspective, the iapp introduce a new particular context to manage. For the app and the full screen case, all the GUI components which are supposed to be shown to the user over the rest of the app, such OSD messages and reminder banner, have a common infrastructure easy to maintain. However some OSD message must be shown also over the iapp. This requirement led to a specific handling for this GUI OSD over the iapp, which is displayed in a context where the app GUI is hidden so a specific management is needed. This is a non optimal solution, because the solution is iapp specific and must be put in place for every OSD needed.

The key of this solution is the enrichment of the iapp context element adding a pair of elements useful for this purpose, letting OSD html content to be displayed their thus covering the iframe content which contains the iapp. However all this management is specific for each OSD and doesn't share almost anything with the common infrastructure of the OSD

5.10 Recording within Wwaf and EPG APP

Recordings in Wwaf are the results of the combination of the pvr infrastructure in Frog and the SKY requirements for pvr functionalities. The middleware has a precise statuses workflow for recording, as other parts of the platform Frog follows as much as possible the UPnP standard part which covers scheduled recordings. The following picture illustrates the workflow. Some of

these statuses have to be handled directly by the Epg App, user actions and information depends upon those.

Recording are almost handled in GuidaTV (Grid), full screen (Banner) and MyTV. Schedule can be inserted in the Grid and in the Banner. However the code does not shared and results from a mix of copy and paste between the two plus some specific adjustment in each case.

The retention policy is implemented in middleware, however from the MyTV page user can “protect” the content tagging it to preserved as much as possible. Scheduled recordings may have several conflicts. There are possible conflicts between overlapping schedules (with different constraints for DTT and IPTV channels) and conflicts between DTT recording and zapping. These conflicts are generated by the middleware once user has inserted conflicting items, because of that conflicts displaying to the user happen asynchronously. In fact the middleware sends a SSE event signaling the conflict, then the EPG App show a proper alert to the user, with the full details of the conflicting item giving user the choice about how to resolve those conflicts. Most of the code concerning the recording is located under the Wwaf framework.

5.11 Reminders within WWAF and the EPG App

Reminders are managed by the middleware as scheduled recording, with just few minor exceptions. This keeps uniform the code managing as well as the interfaces of the middleware. While this could be considered a big advantage, the solution adopted is quite poor when focusing on how to distinguish reminders from schedules: if any of the program fields 'srs_taskstate_pendingerrors' or 'srs_record_task_errors' contains, among the others, the value “154” then that schedule is actually a reminder.

1. Furthermore reusing the same middleware infrastructure of the scheduled recordings added some non-trivial bug during development. For example conflict management have to be different in case of program started in the past: while a conflict of a new schedule recording with a running program previously scheduled still makes sense, the same behavior is almost no sense in case of reminder. Since this special case was not considered, this bug raised up during development. Even once a full QA has been implemented, the potential risk of new issues like that is, in our opinion, non negligible, since a recording has several business rule associated and, most important, a reminder is not a recording. This force identity is then a potential source.

Most of the code of the reminders, as opposite as for recordings, is located under the EPG app app (app/tools/reminder.js, app/screen/fullscreen//contexts/tv/reminder-banner.js). Under

the Wwaf framework there is only one file `wwaf/reminders.js` which just implements simple CRUD operations on reminder.

5.12 Developing within Wwaf and EPG APP

5.12.1 Introduction on Wwaf and EPG APP development

Development on Wwaf and EPG App is mainly an HTML5 programming tasks, very much similar to the development of hybrid mobile app, where the connection between the web layer and the underlying native shell is made by means of REST API call. In other words all the protocols used are http based so the nature of this development can be considered very similar to modern web development of a single page application.

5.12.2 Required skills for Wwaf and EPG APP development

Since is an HTML5 single page application, JavaScript is an absolutely mandatory skill for anyone willing develop EPG App. To be more detailed, the developer has to be confident with single page application, grunt, css3, and functional programming with JavaScript. Besides these skills, a specific background in STB development is need too, since it is needed to be familiar to many IPTV/OTT/STB concepts like HLS live and vod streaming, video mixing, interaction by means of the remote, DTT scanning and zapping, Conditional Access and pvr functionalities

5.12.3 Setup of the development environment

As prerequisite, the developer must have the setup with a working genbox suited for EPG app. Furthermore, a serial connection from the developer PC to the STB must be present.

As stated by Wyplay, for the development of the EPG App a special STB setup must be arranged. Each developer must be able to quickly apply new modification to the HTML5 app and immediately check the result. In order to do this the setup has to be as the following:

STB must be configured to boot kernel (`vmlinux`) from TFTP server, server must be the developer PC

As counterpart, the developer PC must have TFTP server running and serving the `vmlinux` file. The file must be the one contained in the desired target of the genbox

The STB must mount root file system through NFS, server must be the developer PC

As counterpart, the developer PC must have NFS server running and serving the desired target of the Genbox. Besides these mandatory requirements, a real development environment has to support immediate web files deployments. In order to do this

The STB must serve html5 files from a directory mounted through NFS, server must be the developer PC

As counterpart, the developer PC must have NFS server running and serving the directory containing the web files

In order to have this setup in the boot-loader shell the following commands are required:

```
$ setenv -p ETH0_HWADDR "d8:25:22:41:e3:54"  
$ setenv -p STARTUP "boot -tftp -nz -elf 10.11.11.109:vmlinux 'eth=d8-25-22-41-e3-54  
root=/dev/nfs nfsroot=10.11.11.109:/home/toto/workspace-  
sky/genbox/targets/current/root,tcp,vers=3 nfsrootdebug ip=dhcp"
```

Where:

d8:25:22:41:e3:54 must be replaced with the real H/W Mac Address of the STB

10.11.11.109 must be replaced with the real IP address of the developer PC
/home/toto/workspace-sky/genbox/targets/current/root must be replaced with the real nfs exported path pointing to the root of the target on the developer PC

Regarding the setup for serving HTML5 files through the NFS share on the developer PC, a script must be executed in order to setup such config.

First of all, on the genbox, the desired target must be downloaded, i.e.:

```
$ USE=prebuilt xtarget --verbose --create --dir polka-3.2.10 --arch alan =product-targets/polka-  
3.2.10
```

Where:

polka-3.2.10 must be replaced with the real target version number (the list of the available targets can be downloaded with `xtarget --sync; xtarget -p -v polka`

Once installed the desired target, issue the following commands:

```
$ cd  
$ git clone git@gitlab.wyplay.com:polka/polka-tools.git  
$ vi .polka-dev-setup.rc
```

In the `.polka-dev-setup.rc` set ,

```
HOST_IP=10.11.11.109  
WORKSPACE_DIR=/home/franz/polkaworkspace  
MOUNTPOINT=/workspace/
```

Where:

10.11.11.109 must be replaced with the real IP address of the developer PC

/home/user_folder/polkaworkspace must be replaced with the root of the EPG project (the directory containing `polka-ui` and `polka-ui-framework`).

This directory must be exported through NFS. Finally, issue the following commands:

```
$ cd  
$ ./polka-tools/ui/polka-dev-setup-2.sh
```

After this setup (the so called “polka tools”) the STB will take HTML files directly from the user workspace, shortening the code/deploy circle.

5.12.4 Deploying and testing on the development environment

The EPG App JavaScript files are using browserify in order to fully support node.js packaging systems. When the developer wants to deploy the solution a specific grunt task is needed. Furthermore the CSS files are built using stylus, which needs too another grunt task execution whenever the developer wants apply some change to style sheets. This means that every time the developers makes a change on the source a grunt execution is needed prior to update the app.

The EPG app tools described in the previous chapter must be setup, so the html5 file are taken from user workspace, which should contain the following directories:

1. polka-ui
2. polka-ui-framework

This layout must be preserved (the directory is called the workspace).

Once the user has performed all the changes he should issue the grunt execution by means of “grunt” command.

After that, on STB side, issue the following command to reload the web files:

- www

Sometimes the previous command doesn’t work. In that case a restart of the web kit must be performed. The described procedure allows to deploy & test running in the STB. If the developer needs to access the browser developer’s console, he can connect from his PC browser to the <ip_of_the_STB>:8080. In this way the remote console is available with the same tools available as built-in in the normal chrome browser.

This procedure can be time consuming since for each development cycle the following steps are needed after each modification grunt is mandatory, which also merges all the JavaScript files in a single file. This single file is compressed so debugging it becomes almost impossible. In order to supersede this limit and speed up development times, developers can install the “beefy” tool (<http://didact.us/beefy/>).

This tool embeds a web server which serves the JavaScript files built using browserify as they are (without the merge process). With this tool is possible to execute the full EPG app in the developer browser, as long as the following requirements are satisfied:

1. beefy is installed and
2. file `app/rcu.js` in `polka-ui` dir is changed with a version useful for sending remote keys from the browser
3. file `index.html` is modified in order to reference `app/main.js` instead of `app.min.js`
4. beefy is executed with `"beefy app/main.js"` in `polka-ui` dir
5. the browser is set to open `http://127.0.0.1:9966`

With this setup all the EPG app is running in the desktop browser (Chrome). Of course video is still displayed to the STB output, but the app is shown in the browser. Debug is fully available, and changes to any JS file are applied on the fly, without reassemble the app by means of grunt.

6 User Interface Implementation for IPTV

It's the most challenging face of the application accessing the middleware, through that accessing the content management system and achieving the IPTV features. Here everything explained from the scratch how to setup the machine to develop the application to implementation of each functionalities and final application usability.

6.1 Boot with First install

First install is a process where STB needs to be authorized on Sky network and get user's requested certificates.

This process is executed just once and it concerns only the very first boot of the STB on Sky network. The process can be found in `app/main.js:~185` where the first install status is checked (is already executed or not) and in `app/first_install.js` where the complete process can be found. At the end of first install process, a welcome screen was displayed.

6.2 Boot without First install

If the STB has already executed the first install process, it needs to authenticate to the backend and get the authorizations. This process can be found in `app/main.js:~202`

6.3 TV stream menu (live full screen)

TV context (`app/screen/fullscreen/contexts/tv/index.js`) is the place holder for all visual components in relation with TV universe. This full screen contains very few logic, except for the iApps green button enabling. It mostly relies under the components beneath such as:

- **bannerComponent** : `app/screen/fullscreen/contexts/tv/banner.js`
- **restartTVComponent** : `app/screen/fullscreen/contexts/tv/restart-tv.js`
- **IPPV** : `app/screen/fullscreen/contexts/tv/ippv.js`
- **timeshiftTV** : `app/screen/fullscreen/contexts/tv/timeshift.js`
- **BMailController** : `app/screen/fullscreen/contexts/tv/bmail.js`

With the above component the following functionalities are handled in this application. Like,

- `zapToNext`
- `zapToPrevious`
- `zapTo`
- `zapToHref`
- `zapToLcn`
- `zapToHomeChannel`
- `zapToFirstChannel`
- `zapToFirstDTTChannel`

6.4 Banner Component

The banner is the main TV component allowing the user to zap between channels, set audio-sub preferences, record TV programs, purchase PPV events, set a reminder on a program, display program information. The banner holds the hybrid channel list mixing IP & DTT channels, and displays a program list. It is also used to unlock a parental locked stream.

6.4.1 Channel List

This component is a `wwaf.joins.containerComponent` lists all the channels available for playback. Its view allows to see the current channel selected. Its items are retrieved from a `wwaf.joins.JoinProxy` relying on a `cds multiselect` query (watch out `getMultiselectProxy` to see how the query is built). It allows to get both channels (DTT & IP) and their respective programs at once. In Figure 12 highlighted the channel list.

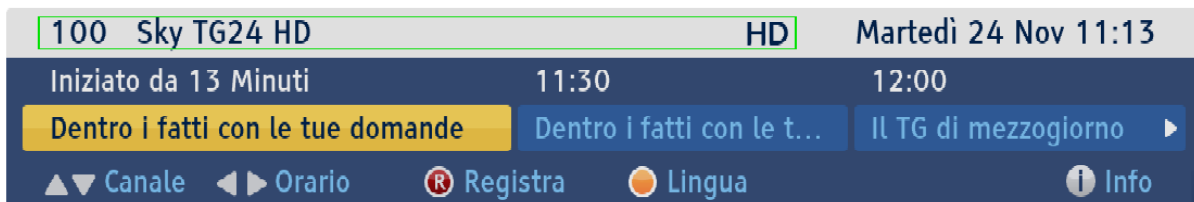


Figure 12

6.4.2 Program List

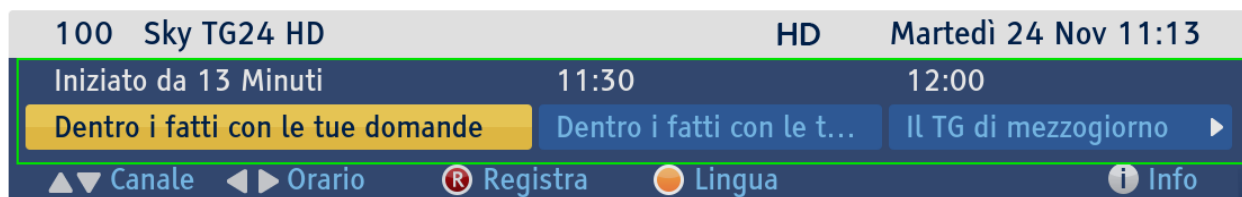


Figure 13

This component is a `wwaf.joins.containerComponent` which view allows displaying past, current or future TV programs on the current selected channel. The proxy containing the program items is extracted via the current channel item `>joins>lut` controlled metadata, so it relies on the same `cds` query as the channel List. (Watch out `channelSelected` function where it happens). Figure 13 mentioned the panel where the program list placed.

6.4.3 Audio subs

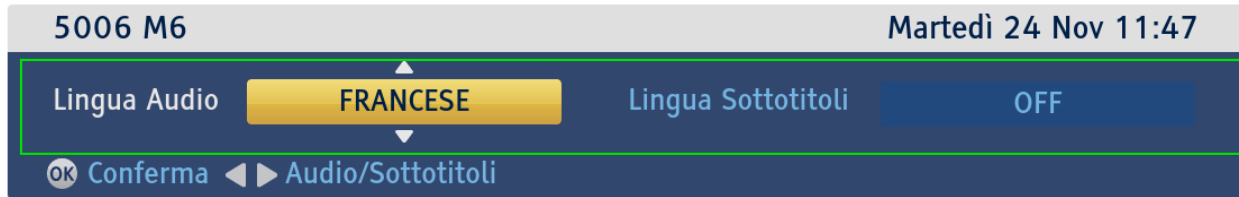


Figure 14

This component allows to configure subtitle and audiosub tracks on the current channel in playback. Its defined in file `:app/screen/fullscreen/helpers/audiosubs.js`. It contains two spinners `app/components/options/components.js::SpinnerComponent` allowing to see the current subtitles and audio tracks detected on the current channel in playback. A callback is combined on `mutate:>selectedIdx` event from each spinner and modifies **audio subs** settings according to the selected option. In figure 14, mentioned way the available languages are listed. And the player REST APIs used by this component are:

- get subtitle tracks
- set subtitle track
- get audio tracks
- set audio track

6.4.4 Ribbon

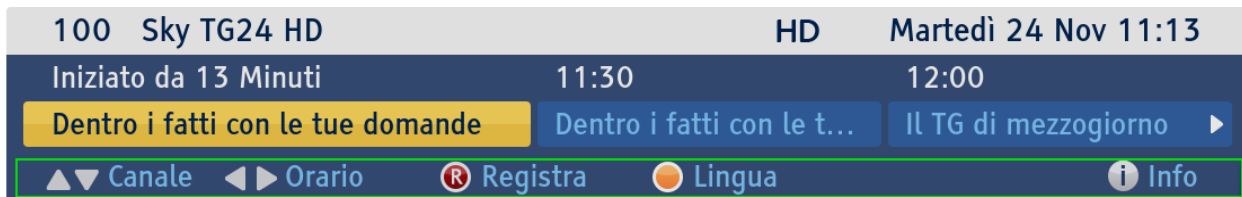


Figure 15

This component allows to bind specific user actions on the colored bullets at the bottom of the banner or simply to display pictograms. Watch out `app/components/ribbon.js` for the component definition. Figure 15 clearly illustrate the ribbon panel and its components too.

The ribbon items are updated after each program or channel selection (among other use cases), in reaction to a `mutate:>selectedIdx` event from **channel List** or **program List** controllers. The function which computes the ribbon buttons is `fetchRibbonItems` from banner. Be warned that it's used both in the banner and in the app menus, so it switches between dom elements according to the focused context.

6.4.5 Synopsis

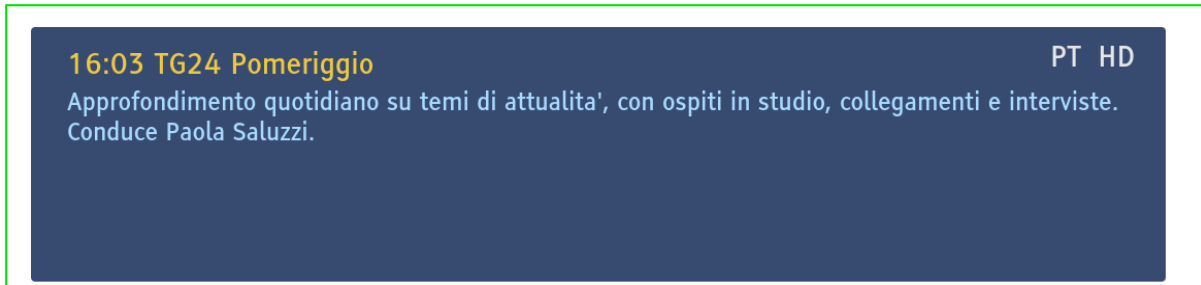


Figure 16

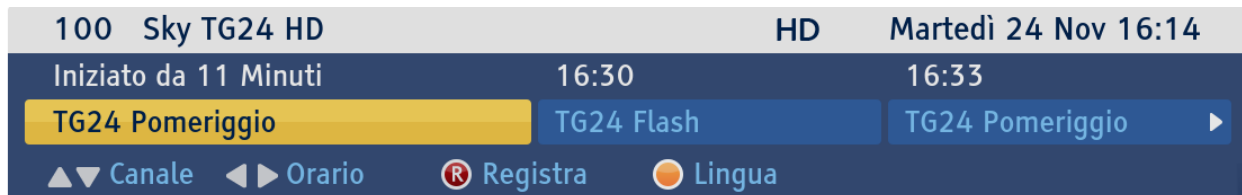


Figure 17

This component is raised when RCU event “info” is caught and allow displaying a short summary of the current selected program. It’s defined in file `app/components/synopsisfullscreen.js`. Which is illustrate in figure 16 and figure 17.

6.4.6 Footer Pin Component

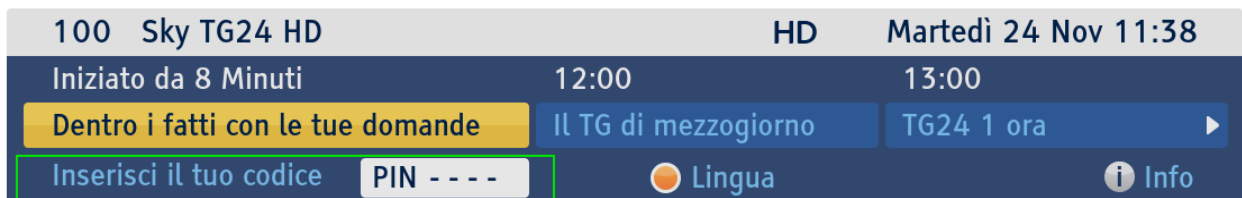


Figure 18

This component is related to parental control and positioned at the bottom of the banner when an event is locked (And the end user has not made any attempt to unlock it yet).Its defined in file `app/components/pincode.js`. Footer pin component is clearly highlighted in figure 18.

The entered pin code as for all others pin related components is checked using function `checkPincode` from `app/pin_manager.js::pincodeManager`. It’s hidden when the program is successfully unlocked or if the pin verification failed. In the latest case the **bannerPinComponent** will be raised instead. Watch out `app/verifier.js` to see all the UI CAS APIs.

6.4.7 Banner Pin Component

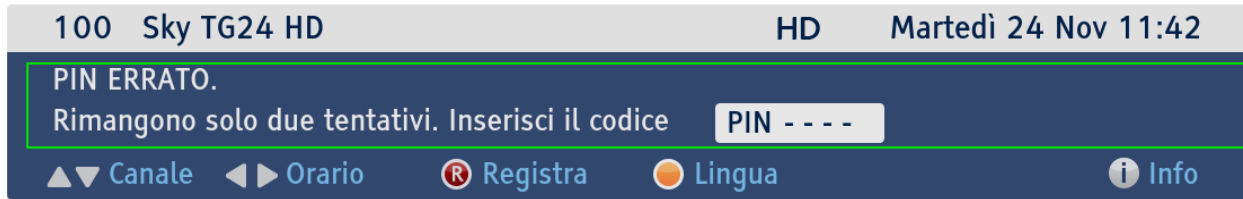


Figure 19

This component is also related to parental control and displayed in the center of the banner when the end user has entered an erroneous pin code. It is also defined in file `app/components/pincode.js`. It reacts to `pincodeManager` Stapes event `remainingTriesUpdate` so that its texts are updated according to remaining pincode tries left. Referred in figure 19.

6.4.8 Zap

There are three distinct ways to zap, some of which can be restricted. Check TV banner select function for navigation rules between channels. Each zap is made via function `makeZap` from TV banner.

6.4.8.1 Zap Numeric

The numeric zap over TV is handled by controller **TVZapNumComponent** described in `app/screen/fullscreen/contexts/helpers/zapnum.js` inheriting from component **ZapNumInputComponent** in `app/components/zapnum.js`. Basically it intercepts RCU numeric key events via handler `onRcuEventNum`. After validating the input it zaps to the channel number entered. Numerical zap suffers no restrictions, each channel of the channel list is accessible.

6.4.8.2 Zap P+/P-

Using P+/P- from RCU, it's possible to zap one channel up/down from current lcn. Some channels are restricted to P+/P- navigation. It concerns channels whose metadata `nav_program` equals 0. In this case, zap will be attempted on the next channel and so on until one channel allows it.

6.4.8.3 Zap via banner

Zap via banner is the zap bind to "OK" button from RCU (when focus is on the first item in the program list) while browsing the banner channel list using "Up" and "Down" keys.

As per **Zap P+/P-** some channels are restricted to vertical navigation. It concerns channels whose metadata `nav_banner` equals 0.

6.4.9 Timeshift action

Timeshift actions logic is mostly embedded in file `app/screen/fullscreen/contexts/tv/timeshift.js`. This file contains the controller catching RCU events related to timeshift over TV. Basically 6 user actions can be used in timeshift:

- **Rewind** -Uses the player speed api with negative speed value (ex: -2).
- **Forward**-Uses the player speed api with positive speed value (ex : 2).
- **Pause**-Uses the player speed api with speed value set to 0.
- **Play**-Uses the player speed api with speed value set to 1.
- **Full Rewind**-Uses the player seek api with following parameters :
 - **seekTarget** : 0 (unit is seconds)
 - **seekMode** : `from_start`
- **Back to live**-Uses the player seek api with following parameters :
 - **seekTarget** : 0 (unit is seconds)
 - **seekMode** : `from_end`

When using **trick modes** in Timeshift a clock widget is displayed at the left bottom of the screen. It shows the current position inside the buffer compared to the live and the current playback speed. The view responsible this widget, i.e. `clockwiseView` is located in file `app/screen/fullscreen/helpers/clockwise.js`. This view also manages the front panel ring led. When using **trick modes** or in pause, to display an accurate position, it's necessary to poll the current position in order to refresh it. It achieved using the `player position` API, with parameter **pos_type** set to 1, i.e. `POS_RELATIVE_TIME`.

For more information about player rest APIs: seek, speed, position.

For more information about UI framework APIs: `wwaf.player`.

To enable Timeshift for TV channel playback make sure to correctly use **json** parameter **transport_mode** with value: `mode_timeshift` along with other parameters for the **ajax PLAY request**. The UI logic ensures whether a TV channel is allowed for timeshift or not. And it is embedded in wrapper object `playWrapper` from the file `app/tools/playwrapper.js`. The maximum size (duration) allowed for timeshift buffer is fetched from backend, and stored STB side in the file `/etc/params/stores/mediarenderer.cfg`.

6.4.10 IP Recording (single/series)

IP recordings via banner are coerced to RCU event “record”. There are several IP recordings use cases:

6.4.10.1 Regular program event recordings

Regular program events are tv programs that do not require subscription or purchase to be viewed or recorded. This concerns programs whose metadata `is_ppv` is `false`. For those kinds of events you can distinguish 2 use cases:

Series program event recording

This concerns programs whose metadata `series_id` is not null. This means future events of the same tv serie could be broadcasted in future. In this case the end user is given the possibility to record either a single episode (the one selected) or the whole serie (the current selected + all the future broadcasts sharing same `series_id`). It's achieved by raising a small popup in the middle of the screen over the banner, with the two options. See `app/components/verticalList.js VerticalListComponent`. It is clearly designed in figure 20.



Figure 20

Single program event recording

This concerns programs whose metadata `series_id` is null. In this case the STB will only attempt to record the current program selected.

Linear PPV event recordings

PPV events need to be purchased before being viewed and/or recorded, this concerns programs whose metadata `is_ppv` is `true`. There is specific treatment in order to purchase this kind of events before finally being able to record them.

The entry point for any IP recording is the function `_programRecord` from `bannerComponent`. It verifies if the end user has the necessary entitlements for recording, and which is the current use case from the ones described above. The library `app/tools/records`, exposes the functions which are actually used to create the schedules for the recording in the end :

- `records.createProgramRecord` wrapper for `www.pvr.createProgramSchedule`
- `records.createSerieRecord` wrapper for [www.pvr.createSerieSchedule](#)

6.4.11 DTT Recording

DTT recordings via banner are also covered to RCU event “record”. Unlike IP recordings, DTT schedules are not bounded by programs start_date & end_date because the validity of the EPG over EIT p/f over dvb cannot be guaranteed. As a result, the end user needs to manually configure the date and boundaries of its schedule.

In TV banner this is done via function showRecordBanner, which appends the recordBannerComponent to TV banner in place of the usual program list. The component recordBannerComponent is defined in file app/screen/fullscreen/helpers/recordbanner.js. It presents 4 spinners allowing getting the parameters needed to create the schedule using **wwaf framework pvr module** createChannelSchedulemethod, i.e. its date, repetition, and duration.

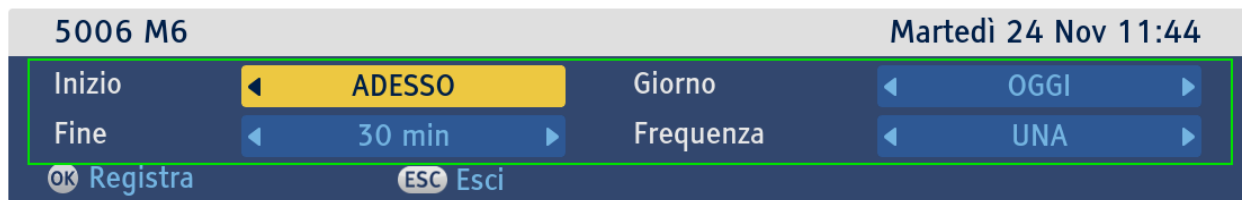


Figure 21

There is a specific behavior to know about spinner Inizio. There are 2 possible options, either a predefined value ADESSO or an empty time input field to fill. It is possible to record video from timeshift buffer, in this case ADESSO option will pick the time determined from the position being played in timeshift and not the current time (the live). It is also possible to set manually a date in the past, if it is included in the timeshift buffer, otherwise an error will be raised and an OSD presented to the end user.

Note: As the STB possess a single tuner it is not possible to record and watch 2 different DTT channels at the same time.

6.5 Main Menu

Main menu, it is app page, contains all the menu items, sub menu items and components everything.

6.5.1 Opzioni Menu

6.5.1.1 Reg Manuale DTT

User can record what he wants through this menu. He needs to choose the channel number, enter a title for the record program, the day, a start time and end time and a frequency. The component ManualRecord (app/screen/opzioni/regist_manuale.js) works with **wwaf pvr component**

6.5.1.2 Digitale Terrestre

This menu (`app/screen/opzioni/digitale_terrestre.js`) attempt that user choose between two sub-menu items.

- **Ricerca `app/screen/opzioni/digitale_terrestre.js`**

Search for some DDT channel The UI use `dvb_scan` from waff to start the scan DTT channel.

- **Scans. periodica `app/screen/opzioni/periodic_scan.js`**

Settings the auto search of DTT channel Scans periodic is just a reminders created on STB.

6.5.1.3 Diagnostica

User can access to Diagnostica through the Green button from Opzioni menu. (Defined in `app/component/menu.js` file on `setRibbon()` method. When user is on this menu, an OSD invite the user to process of the test through an OSD. (`app/screen/assistenza/diagnostica.js`)

6.5.1.3.1 Test Canale DTT

Through Diagnostica menu, user can access to Test canale DTT with the Red button. It's defined in `app/screen/assistenza/diagnostica.js` file on `onshow()` method

6.5.1.3.2 Test banda

Through Diagnostica menu, user can access to Test banda with the Green button. It's defined in `app/screen/assistenza/diagnostica.js` file on `onshow()` method

6.5.1.3.3 Test disco

Through Diagnostica menu, user can access to Test disco with the Yellow button. It's defined in `app/screen/assistenza/diagnostica.js` file on `onshow()` method

6.5.1.3.4 Test Smart Card

Through Diagnostica menu, user can access to Test Smart Card with the Blue button. It's defined in `app/screen/assistenza/diagnostica.js` file on `onshow()` method

6.5.1.4 Dati Decoder

User can access to Dati Decoder through the Yellow button from Opzioni menu. (Defined in `app/component/menu.js` file on `setRibbon()` method. All informations about decoder are displaying here. This page is defined on `app/screen/assistenza/product_info.js` file.

6.5.1.5 Aggiorna SW

Through Dati Decoder menu, user can access to Aggiorna SW with the Yellow button. It's defined in `app/screen/assistenza/product_info.js` file on `onshowDatiDelDecoder()` method

6.5.1.6 *Licenze SW*

Through Dati Decoder menu, user can access to Licenze SW with the Blue button. It's defined in `app/screen/assistenza/product_info.js` file on `showDatiDelDecoder()` method

6.5.2 **Configura menu**

6.5.2.1 *Televisore*

Televisore (`app/screen/configura/televisore.js`) menu manage video settings. Those settings are saved inside the **config store** with helpers. Located in this file (`app/tools`):

- `tools.getVideoSettings`
- `tools.setVideoSettings`

6.5.2.2 *Lingua e sottotitoli*

Lingua e sottotitoli (`app/screen/configura/lingua_e_sottot.js`) menu manage global preference for language and subtitles. Those settings are saved inside the **config store**. Those settings are given to **wwaf.playerConfig** to set the preference globally

6.5.2.3 *Linea Dati*

Linea Dati (`app/screen/configura/linea_dati.js`) menu manage the network with or without the **skylink**. Two components are used:

- `linea_data_manual_wifi` (`app/screen/configura/linea_dati_manuale_wifi.js`) used for manuale wifi ip configuration
- `linea_data_network_list` (`app/screen/configura/linea_dati_network_list.js`) used for the listing of available networks.

For **skylink** management check `app/tools/skylink.js`

6.5.2.4 *Imposta my SKY*

Imposta my Sky (`app/screen/configura/risp_energetico.js`) menu manage the behavior of the standby. Those settings are saved inside the **config store**. Three options are available:

- Spegnimento automatico: STB goes in standby after a period of inactivity, and goes in suspend every night, the STB will reboot after suspend period
- Stand-by automatico: STB goes in standby after a period of inactivity, and reboot every night
- Stand-by Manuale: STB goes never in standby, but reboot every night

For more information about standby management information check `app/power_manager.js`

6.5.2.5 Impianto

Impianto (`app/screen/configura/impianto.js`) menu is used to set adsl/fiber or satellite. The STB after will work with reduce functionalities. It uses `set_mode` from `app/backend_manager.js`. The STB software will be change at the next update.

6.5.2.6 Audio

Audio (`app/screen/configura/Audio.js`) menu manage audio settings. Those settings are saved inside the `config_store` with helpers. Located in this file (`app/tools`):

- `tools.getAudioSettings` get stb audio parameters with wyrest
- `tools.setAudioSettings` set stb audio parameter(s) with wyrest

6.5.3 GuidaTV menu and subcategories

This menu displays a TV program grid allowing to browse tv guide up to 7 days, schedule record, set reminders, or buy ppv. Related files are:

- `app/screen/guida/grid.js`
- `app/screen/guida/nowline.js`
- `app/screen/guida/radio.js`

Grid program items are extracted from a `wwaf.joins.JoinsProxy`.

See `app/screen/guida/grid.js :: GridScreen._getNowProxy`. Each sub-menu share a common `cds multiselect query` json template,

```
{
  "selections": [
    {"selection": "channels"},
    {"selection":
      "broadcasted_programs", "relation_type": "join_left",
      "related_selection": "channels",
      "criteria": [{"metadata": "service_id", "operator": "=", "related_metadata": "service_id"}]
    },
    {"source_type": {"channels": {"any_tv_source": ""}, "broadcasted_programs": {"any_tv_source": ""}},
    "object_type": {"channels": ["tv_service"], "broadcasted_programs": ["tv_epg_event"]},
    "metadata_filter": {
      "channels": channelsFilter,
      "broadcasted_programs": "end_date > " + start + " AND start_date < " + (start + duration) + ""
    }
  ],
}
```

```

"sort_params": {"channels": ["lcn"], "broadcasted_programs": ["start_date"]},
"options": {"broadcasted_programs": ["update_filter_on_get_results"]}
}

```

Where channelFilters differs for the different channel categories:

- **intrattenimento** : "genre == 'Intrattenimento'"
- **sport** : "genre == 'Sport'"
- **calcio** : "genre == 'Calcio'"
- **cinema** : "genre == 'Cinema'"
- **doc e lifestyle** : "genre == 'Doc e Lifestyle'"
- **news** : "genre == 'News'"
- **bambini** : "genre == 'Bambini'"
- **musica** : "genre == 'Musica'"
- **sky music e radio** : "category == 'SKY_CHANNEL_AUDIO'"
- **digitale terrestre** : "category IN {DVB_CHANNEL_TV, DVB_CHANNEL_AUDIO}"
- **hd e 3d** : "resolution == 'Video HD'"

And start, duration adjusted when scrolling right or left in the grid. Here in figure 22 is an illustration to locate main grid objects and how components are imbricate:

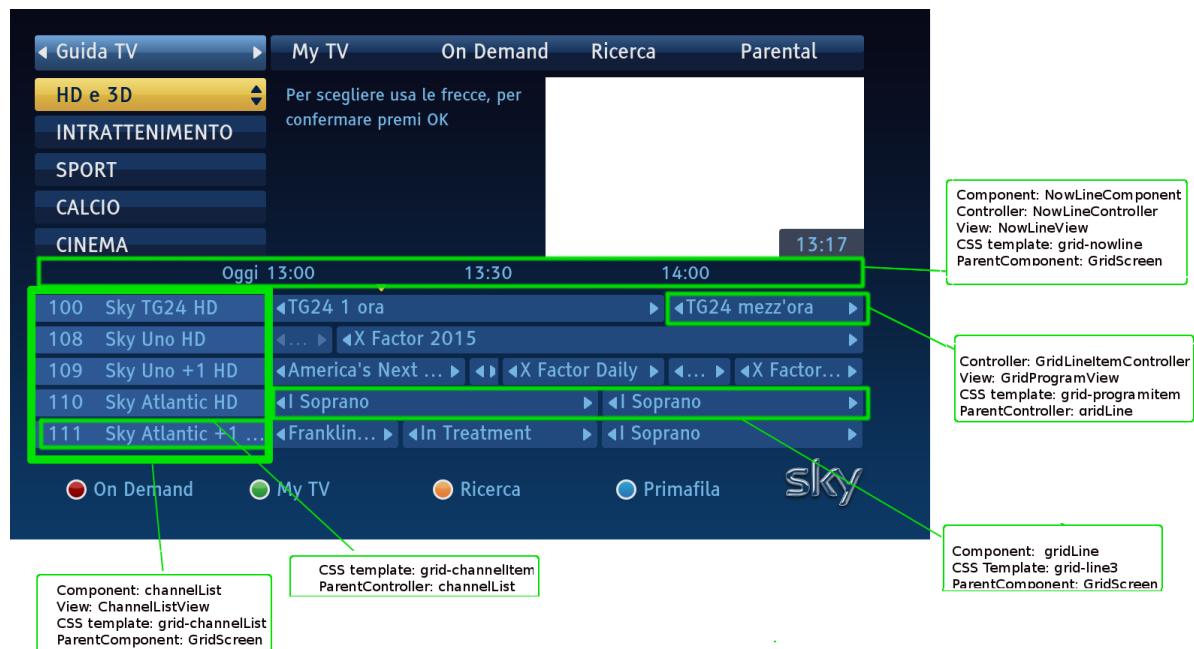


Figure 22

6.5.4 MyTV menu and subcategories

This menu is used to list and manage PVRs, owned VODs, reminders. Related files are :

- `app/screen/my_tv/my_tv_generic.js`
- `app/components/my_tv/list.js`
- `app/components/my_tv/items/*`

The architecture of the menu is similar to tv grid.

For each submenu items the cds multiselect query is changed to fetch the content to feed the list. Check out `app/records/cdspvr.js` `cdspvrProxyFactory` to look how the query is generated.

There are 4 basic item types in the list:

- PVR : `app/components/my_tv/items/records.js`
- Schedules + Reminders : `app/components/my_tv/items/schedules.js`
- VOD : `app/components/my_tv/items/vod.js`

We also have a notion of container items, ie an item built to group a set of basic items: its children. Containers or grouping items are recognized via metadata `child_count` whose value is greater than 1. There are several containers types:

- Record serie containers allow to list all pvr records related to a serie.
- dtt record containers allow to list all pvr records related to a same schedule which has been fragmented for any reason (the dtt cable removed or reboot during the schedule time).

Container items use the same item classes than basic PVR items:

`app/components/my_tv/items/records.js`. When an item needs to be rendered on screen, the `onReady` method from `MyTVRecordController` is called. For items which are containers (`child_count > 1`) , its needed to pass through method `_fetchGroupMeta` to be able to distinguish between series and single grouping item, looking on which metadata the grouping is done.

6.5.4.1 Grouping series

Series containers record episodes are grouped via metadata `series_id`. For such items controlled Boolean metadata `>isSerieContainer` is set to true. It allows to append the CSS class `is-group` to the item view `MyTVRecordView`, toggling the display of div elements specific to series containers.

Following figure 23 illustration shows how they are rendered,

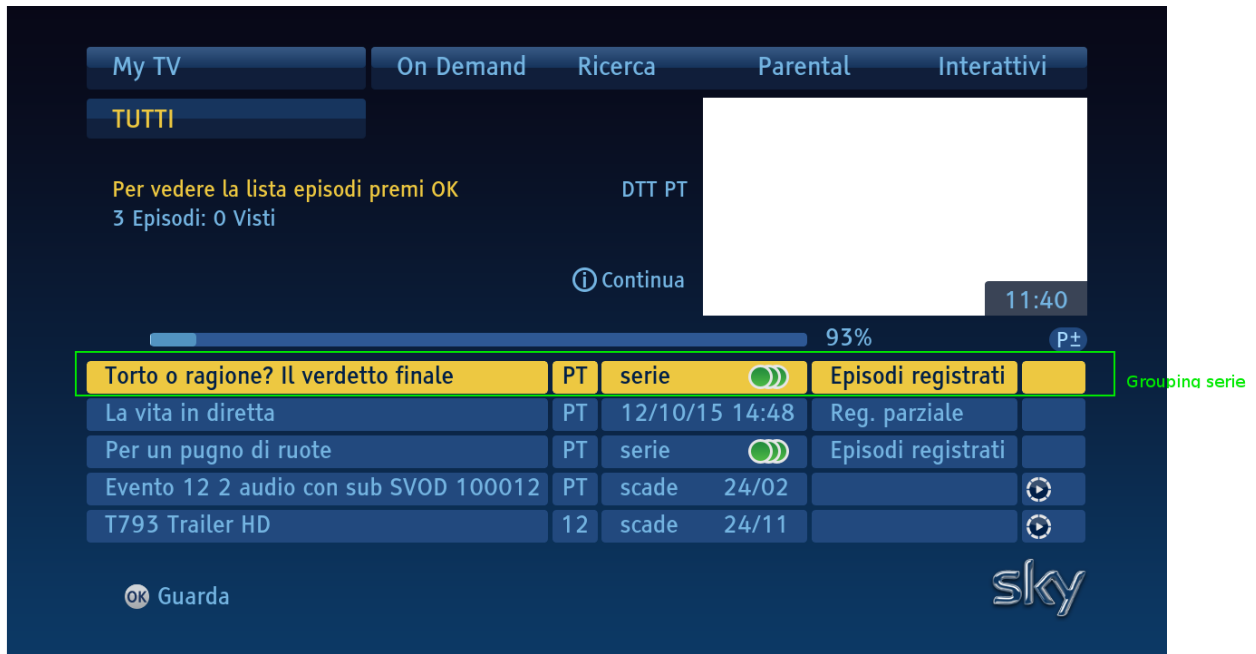


Figure 23

6.5.4.2 Grouping one item (after a reboot for example)

Single item containers records tracks are grouped via metadata srs_id. Visually speaking there are no differences with a basic PVR item.

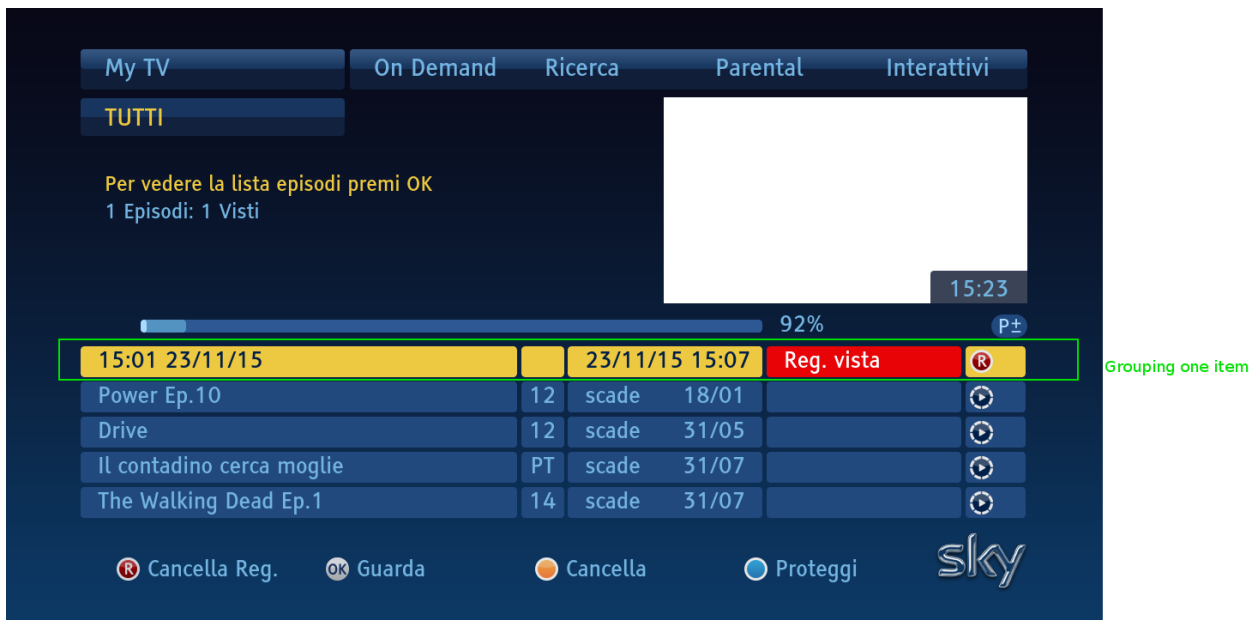


Figure 24

Here in the figure 24 the mentioned container allows together the different segments of the recording to be able to play them sequentially. When a recording has been spitted, the `retrieveGroupedRecord` function from `app/tools/records.js` is applied to gather all the record parts which are collected in the item metadata `playlist_info`.

The `playlist_info` is then used by pvr context (`app/screen/fullscreen/contexts/pvr/index.js`) to play in a row each of the record parts. Between each parts of the record a popup is displayed as transition to the playback of the next segment:

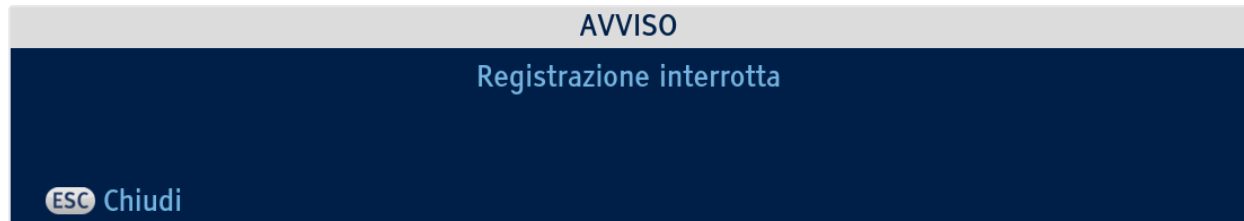


Figure 25

Pressing “ESC” the next part of the record is then played.

6.5.5 On Demand and subcategories

The VOD section of the UI is special in the sense that it is the only section that is not handled through the Wyrest API [8].

The VOD catalog is retrieved through asynchronous HTTP calls to CISCO servers named **CMDC** (Content Metadata Discovery Component, ex NDS). It provides a web centric (REST-like) API which can be used by clients to access the catalogue, contents, products, etc. The results returned by the API are JSON encoded.

You can download an usage guide which provides some useful tips: **CMDC Usage Guidance**.

The “**On Demand**” menu sub-items are retrieved through the **CMDC** server so when the authentication cannot, for some reason, be performed or when there are Smart card errors, there is a hardcoded “**On Demand**” menu sub-items list that is used as a fallback (see `app/components/menu.js`).

Paths to involved files (abstraction layer):

- `app/vod/index.js` : Takes care of authentication to CMDC servers.
- `app/vod/cmdcds.js` : Takes care of VOD resources information locally stored through CDS.
- `app/vod/request.js` : Takes care of building specific needed CMDC requests.
- `app/vod/classification.js` : Takes care of retrieving data for VOD classifications (understand VOD sections and subsections).
- `app/vod/asset.js` : Provides functions for actual VOD resources (Assets) CMDC requests.

Paths to involved files (UI specific)

- `app/components/vod/list.js` : Component that manages the browsing of the catalog.
- `app/components/vod/voditem.js` : View class of a VOD item.
- `app/components/vod/vodribbon.js` : Component that manages the ribbon's different states when browsing the catalog.
- `app/components/vod/season.js` : Component that manages the specific Series items behaviors (series containers).
- `app/components/vod/assetdetails.js` : Component that manages the PLAYBACK and PURCHASE screens behavior of a VOD item.
- `app/components/vod/manager/*` : files that manage the catalog's browsing history. So that the application is able to handle the back browsing correctly (Esc. button actions etc ...)
- `app/components/screen/on_demand/explorer.js` : the screen file component that references all of the components above and makes use of them.

The **list** component (`app/components/vod/list.js`) handles the browsing of the catalog. It manages three different types of display:

- The **Button display mode** which actually displays the sub-classifications (understand sub-categories) of a classification.



Figure 26

- The **Assets display mode** displays actual VOD items in a “poster” mode.

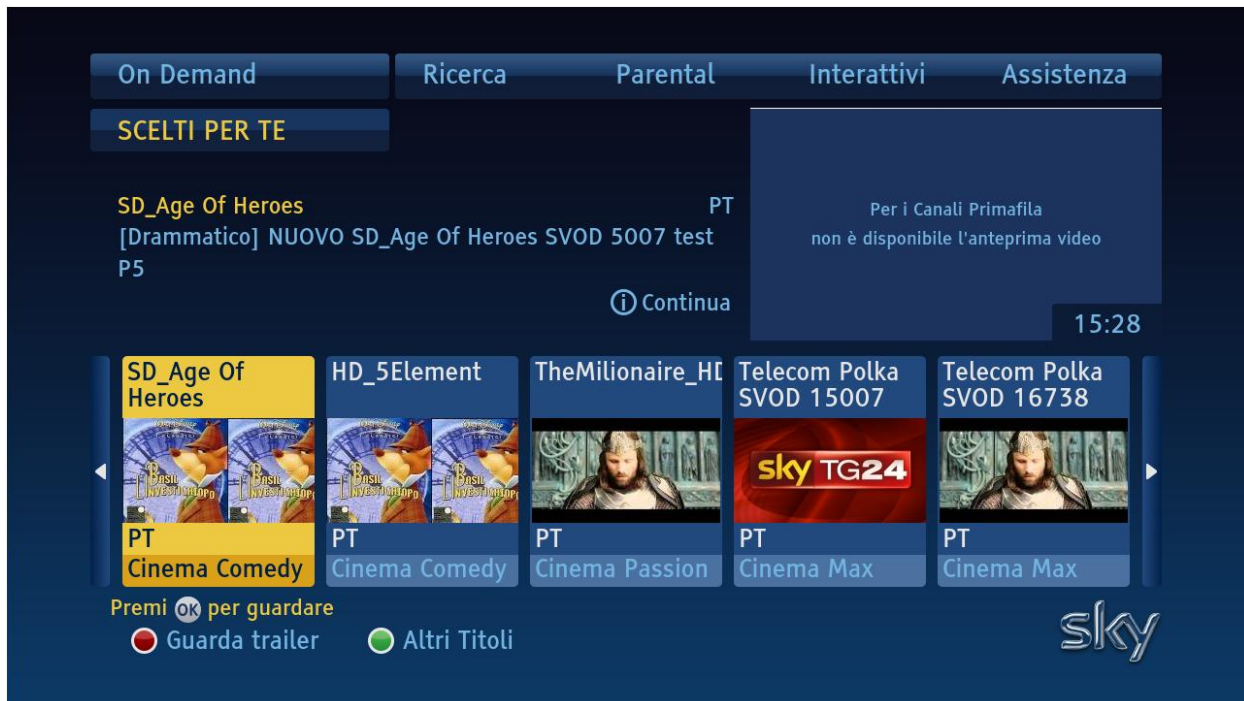


Figure 27

- The **lines display mode** displays actual VOD items in a “line” mode.

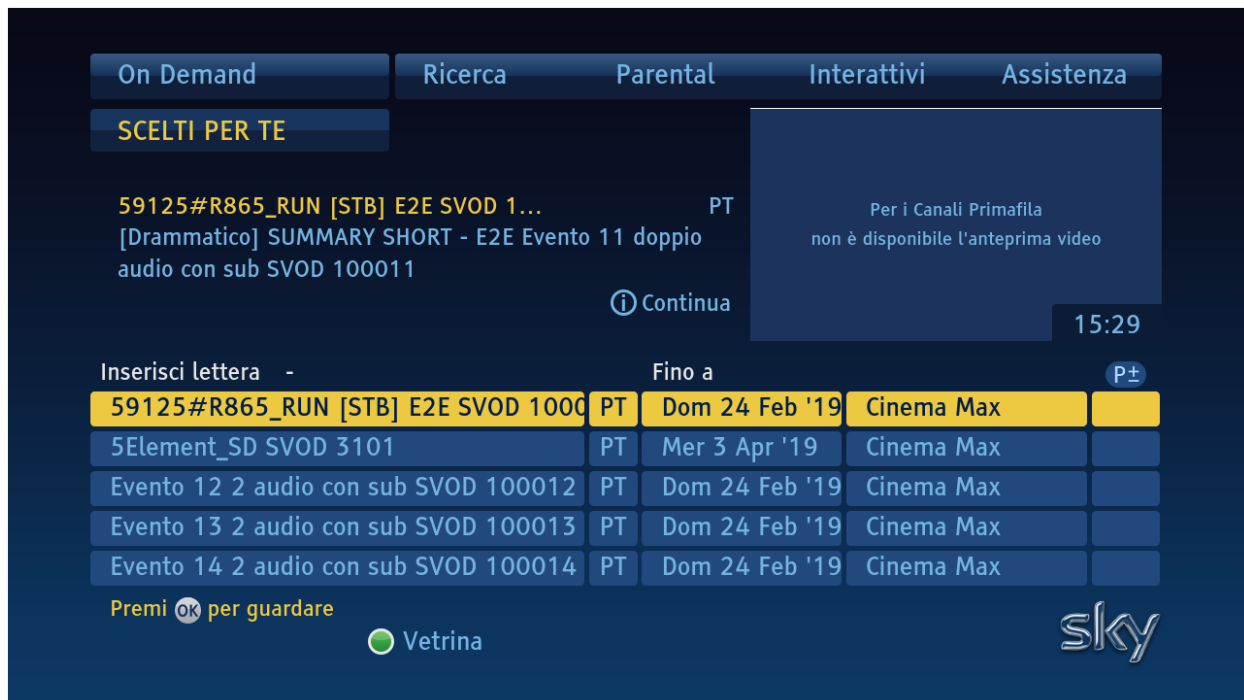


Figure 28

The Season component (app/components/vod/season.js) manages the series containers. A season container is displayed in **lines** display mode. When in **lines** mode a **collapsed** CMDC request is performed. As mentioned in figure 29.



Figure 29

Series Containers (collapsed)



Figure 30

Series container's content (uncollapsed) are clearly pictured in figure 30

When selecting an actual VOD item and pressing the **OK button**, the “**Playback/Purchase screen**” is displayed. The component involved is `app/components/vod/assetdetails.js`.

- **Playback screen** is displayed if the item is a **SVOD** item (**Subscription VOD**). The user has subscription to the item. See `showSVOD()` method.
- **Purchase screen** is displayed if the item is a **TVOD** item (**Transactional VOD**). The user has to pay for the item. See `showTVOD()` method.

It is important to notice that when you launch the playback of a VOD item, a switch from the **App context** to the **VOD fullscreen context** is performed passing the **noReset** parameter. Doing so, the state of the app context is not reset and we are able to go back to the app in its previous state. (See `app/components/vod/assetdetails.js` function `_launchPlayback`)

```
ui.contexts.global.switchToFullScreen('vod', {}, true) ...
```

When playing a VOD item for the first time, an entry is inserted in CDS store so that we locally have an occurrence and allows us to fill the **MyTV** menu section (check `app/vod/cmdcds.js` for detailed involved metadata). When pressing **Esc button** or **Stop button** while streaming a VOD item, the current playback position is memorized alongside the according CDS VOD item occurrence.

6.5.6 Ricerca

This menu is used to search content by title in 3 different universes, TV, VOD & MYTV (records & purchased vod).

`app/screen/search/programmi.js` and its dependencies:

- `app/screen/search/tools/epg.js`
- `app/screen/search/tools/vod.js`
- `app/screen/search/tools/mytv.js`

The menu is composed of 4 options component:

- **Titolo**- A text input empty field allowing to type text with RCU
- **Genere**- A spinner allowing to filter results on a certain gender
- **Sottogenere**- A spinner is allowing to refine the search more accurately with a sub-gender. Its content is updated according to the selection on the previous spinner.
- **Cerca in**- A spinner is allowing to search content either in: TV, VOD, MYTV universe.

6.5.7 Parental control

This menu provides the functionality which user can lock the channel with credentials, it leads the user can controller their kids and younger family member s watchable of the channels

6.5.7.1 Blocca programma

This menu is used to personalize the parental control rating. Before accessing this menu item the pin code of user will be asked for security reason. Unless or otherwise the user from other parental control menu item.

app/screen/parental_control/blocca_program.js

6.5.8 Altre impostazioni

This menu is used to personalize the restriction of purchase events/VOD item. Before accessing on this menu, the pin code of user is asked for security reason. Unless user come from other parental control menu app/screen/parental_control/altre_impostazioni.js

6.5.9 Modifica PIN

This menu is used to personalize the pin code. Before accessing on this menu, the pin code of user is asked for security reason. Unless user come from other parental control menu

app/screen/parental_control/modifica_pin.js

6.5.10 Interattivi

This menu is used to list iapps available and allows to launch them manually from the EPG. app/screen/interattivi/tutti.js IApp objects retrieved via getIApps function from module app/iapps.js are listed in this menu if

- inImm metadata is true
- its bind and active (i.e. status is enable) on a channel or its not bind to any channel

getIApps relies on route GET /backend_manager/request/iapp/ to fetch iapp catalog from back-end and builds a map in which iapp objects are indexed by id.

6.5.11 Contatta Tim

This menu is used to display contact information such as telephone number, web URL to contact sky customer services in relation with user subscription management. app/screen/assistenza/servizio_clienti_tim.js

6.5.12 Contatta SKY

This menu is used to display contact information like telephone, SMS, fax numbers, web address to contact sky customer services in relation with the STB or EPG usage. app/screen/assistenza/servizio_clienti_sky.js

So in this way we implemented the IPTV functionalities for the television broadcasting company. We implemented some more features like network traffic analyzer in order identify the customers desire with that the television broadcasting company can offer those contents and that helps a lot in revenue perspective.

7 Conclusion

Through EPG application development I gathered wide knowledge of television broadcasting functionalities, Structuring and Querying of CMS, middleware IPTV implementation and its extendibility, Front-end application development with HTML5 and JavaScript Frameworks.

For content management system I explored MongoDB and querying functionalities. CMS contains various levels of accesses. In that those who are all provide contents to CMS are known as content providers, the next level of access is those who are all retrieving the content according to application needs are known as API team and I am one among them. The CMS contains data from different regions and different channels. So in order to retrieve efficiently the API team has the structured implementation which contains complex logics, API have to provide the data URLs (http links) to application developers which is in the JSON format. The application developers are belongs to web and mobile applications for both Android and IOS platforms, in this I involved in web application development a single page application.

The whole application will be meaningful because of this middleware implementation, which gives the whole ideology in the form of WAAF framework to both the web and mobile applications by using and extending WAAF in order to achieve the IPTV features. In this entire implementation the single page web application required lot of logical and technical knowledge in order to enhance the middleware implementation, handling the CMS data and implementation of the user interface.

Now this EPG application has lot dependences with middleware framework and CMS. In feature this application should be an independent application so that the application can able to reusable for various television broadcasters, with this achievement the leading television broadcasting tool.

8 References

- [1] Documented for all components used in middleware in detail: <https://frogbywyplay.com/>
- [2] Available CMS contents: <http://www.sky.it/>
- [3] Genbox implementation <https://portal.frogbywyplay.com/docs/wytv/featured/frog-genbox/toc-index/>
- [4] Lodash: <https://lodash.com/>
- [5] Stapes JavaScript framework: <https://hay.github.io/stapes/>
- [6] Promise JS: <https://github.com/stackp/promisejs>
- [7] D-Bus Mutex: <https://portal.frogbywyplay.com/docs/wytv/featured/guide-dbus/toc-index/>
- [8] WYRest: <https://portal.frogbywyplay.com/docs/wytv/featured/components/appframeworks-wyrest/overview/>
- [9] EPG guide: <https://portal.frogbywyplay.com/docs/wytv/featured/guide-epg/toc-index/>