






/ A Pluggable Player Business Framework

/ Zhang Dawei

Outline

-  Introduction of the background
-  Design of the framework
-  Choices

Introduction of the background

Too many playback related businesses exists in a single activity, and most businesses are logically independent.



Introduction of the background

Some businesses are different in presentation only, and lots of playback activities contains the same businesses.



剧集列表



C63 coupe-TVC

Introduction of the background

It's hard to manage the Z-order and interactions between views from different controller.

```
void showViewOfA() {
    dismissViewOfB();
    dismissViewOfC();
    dismissViewOfD();
    ...

    viewOfA.setVisibility(View.VISIBLE);
}




void showViewOfB() {
    ...
}
```

Introduction of the background

According to the factor above, we want our new architecture can archive the following goals:

- All businesses is isolated both in logics and storage, and insensitive to other businesses.
- Has a linear increment of complexity when business grows
- Separate views and logics as much as possible.
- Find a way to manage the Z-order and the interaction of views easily.
- Businesses can be easily reused in other playback activities.

Outline

-  Introduction of the background
-  Design of the framework
-  Choices

Design of the framework

Runtime Container of Businesses

Video Advertisement

Subtitles

Danmaku

Play list

Playback statistic reporting

Trail playback

Voice control

...

Video Player
Implementation

View
Management
Implementation

Activity Life Cycle
& Event
Dispatching

Business Related
Dependencies

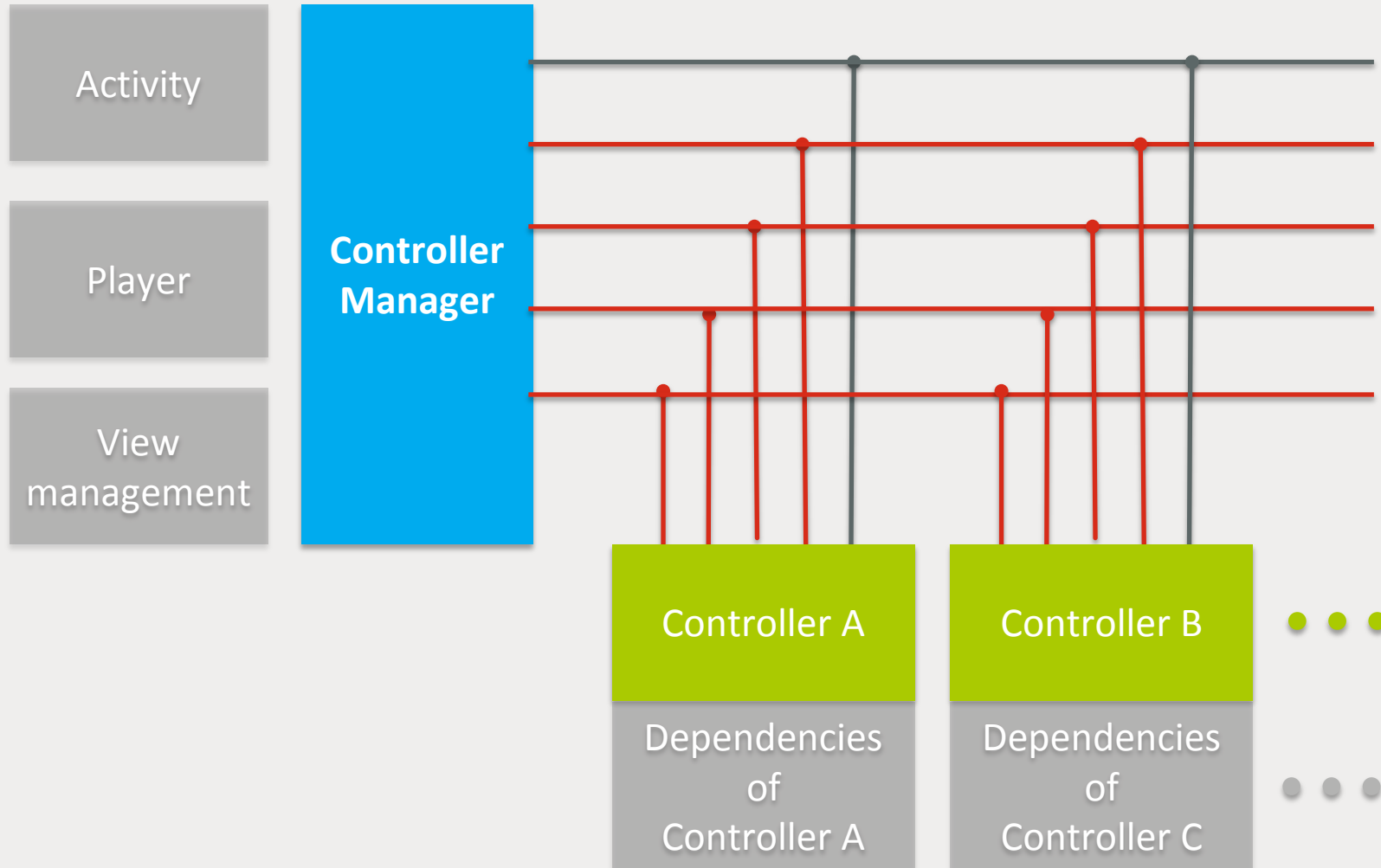
/ Design of the framework

Each business exists as an implementation of **IController** interface.

<< interface >> IController	
+ onInit() + onRelease()	Lifecycle of a controller it self.
+ onPrepared() ...	Event callbacks from the player.
+ onActivityResuming() ...	Lifecycle callbacks from the host activity or fragment.
+ onViewDismissed() ...	Event callbacks from view management.
+ onDispatchKeyEvent() ...	Key input event handling.

Design of the framework

The runtime container of controllers is called **ControllerManager**.



/ Design of the framework

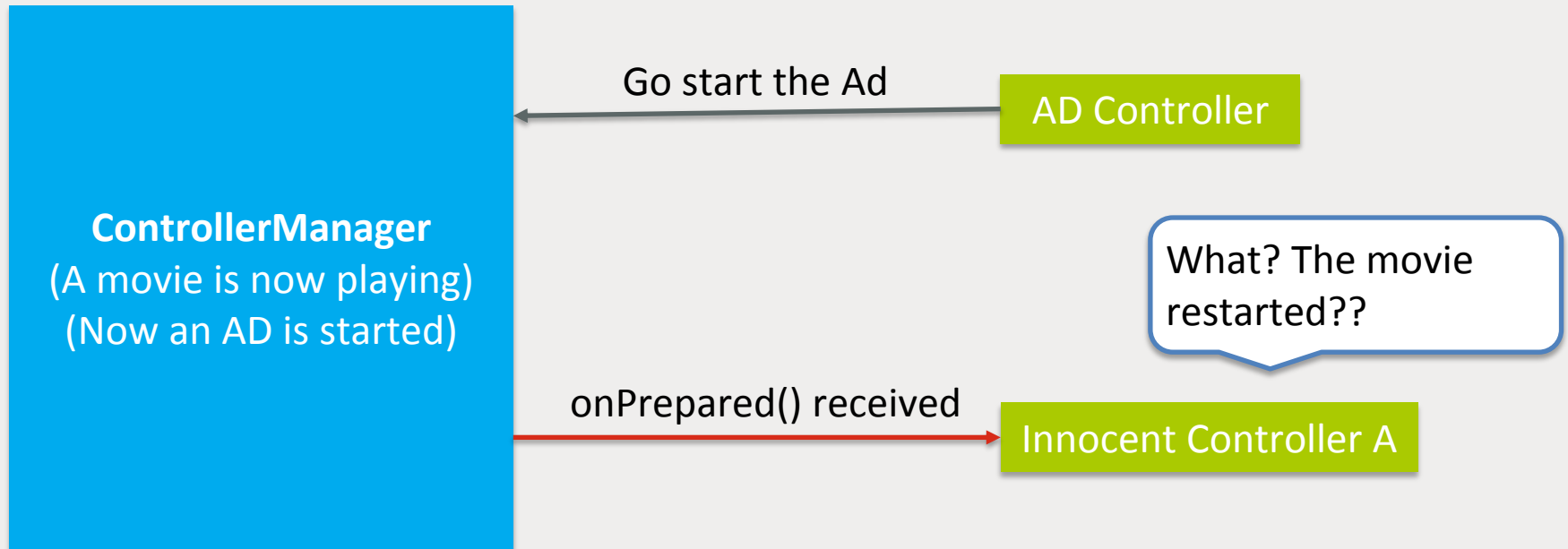
To build a new playback activity, just add all controllers you need into **ControllerManager**, and then, everything is started.

```
void onCreate() {  
    controllerManager.addControllers(new IController[] {  
        new ControllerA(),  
        new ControllerB(),  
        ...  
        new ControllerZ()  
    });  
  
    controllerManager.start();  
}
```

Design of the framework

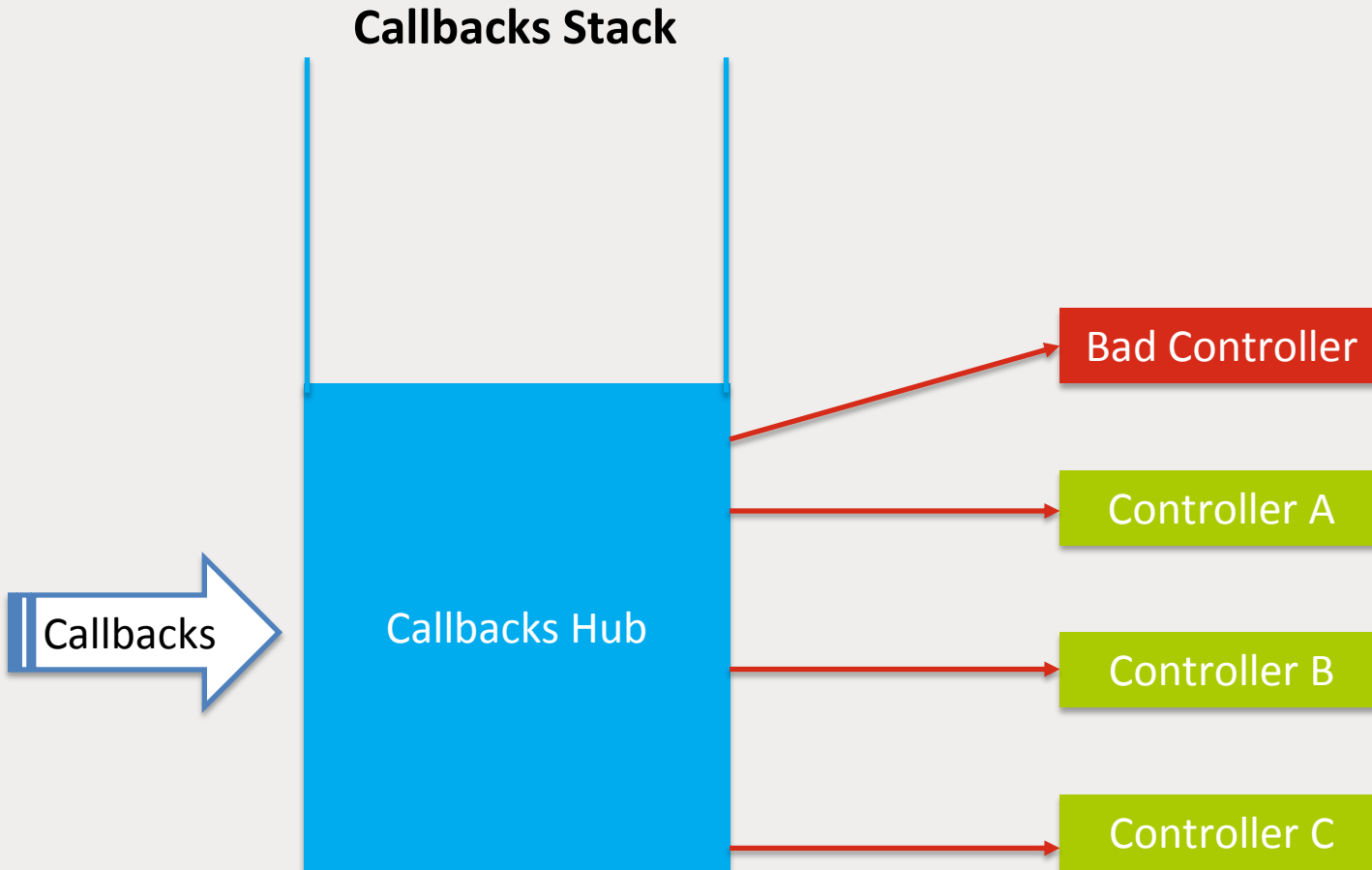
Controllers still can sense the existence of other controllers.

For example, playing an video advertisement is done by a controller, and then other controllers can receive the playback callbacks belongs to the advertisement. So they are confused!



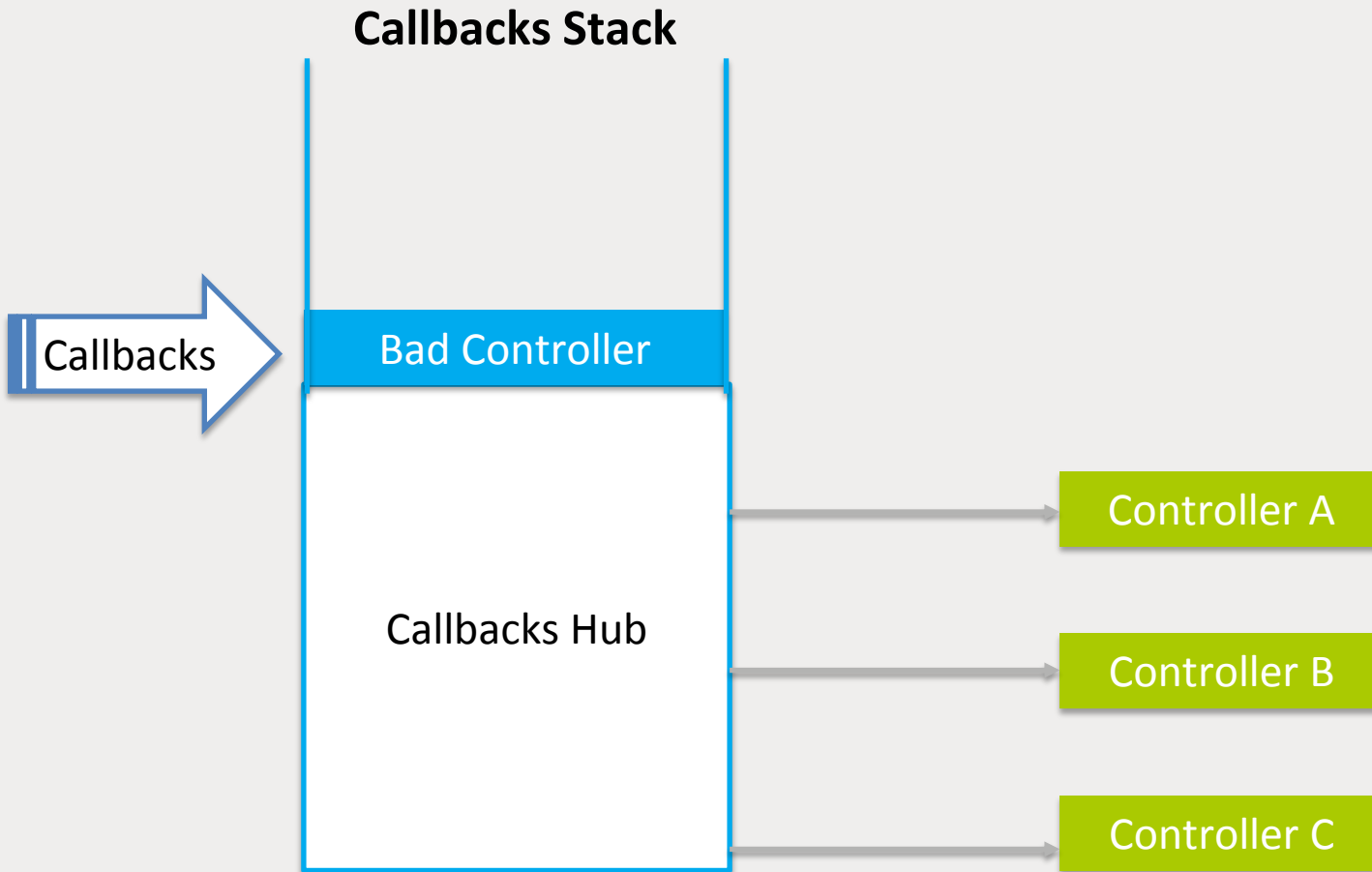
Design of the framework

There is a lot of way to prevent this issue, better than checking if a AD is playing or a movie is playing.



Design of the framework

How a callback stack solves this issue?



/ Design of the framework

View of a controller exists as an implementation of **IControllerView** interface

<< interface >>
IControllerView

+ getView(): View
+ getType(): int

- getView(): To obtain the actual view to be displayed.
- getType(): To indicate the type of this view.

<< interface >>
IPlayInfoView

+ setVideoInfo(info: VideoInfo)
+ updateProgress(int progress)
...

AbsUserActionView

+ setActionListener(I:UserActionListener)
+ triggerAction(actionCode: int, T params)
...

/ Design of the framework

Controller can get an instance of `IControllerView` and display it via **ViewManager**.

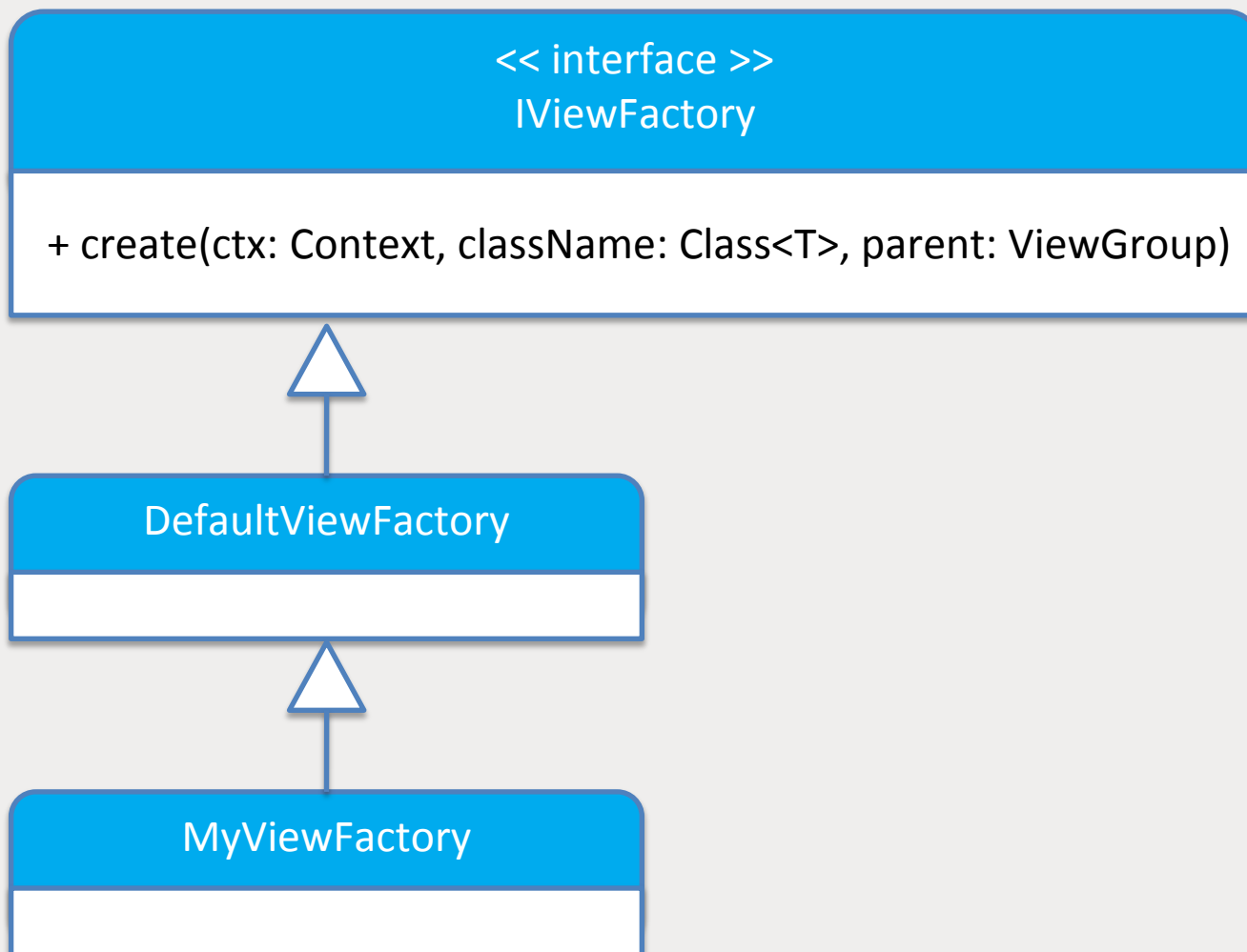
```
// To get an instance which implements the given interface  
<T extends IControllerView> T getView(IController controller, Class<T> clazzName);  
  
// To display an IControllerView  
boolean showView(IController controller, Class viewClass, IControllerView view);
```

Controller only knows the interface of the view but don't know what the view actually is.

For showing a view, controller do not care where the view is displayed, nor who is the parent of the view, nor what happened to other views if this view is shown.

/ Design of the framework

IViewFactory will take care of the creation of IControllerView.



Design of the framework

Different implementation of IViewFactory brings different feeling for the same controller

淋漓，尽致

梅赛德斯-AMG C 63 S轿跑车限量特别版



☆ 点击收藏

正在播放



C 63 coupe limited special edition

TopicFactory



AMG



☆ 点击收藏

2016魔幻大片盘点
人兽混战 特效炸裂

TimeLineTopicFactory



此“神”一出谁与争锋

以爱之名，与神为敌！

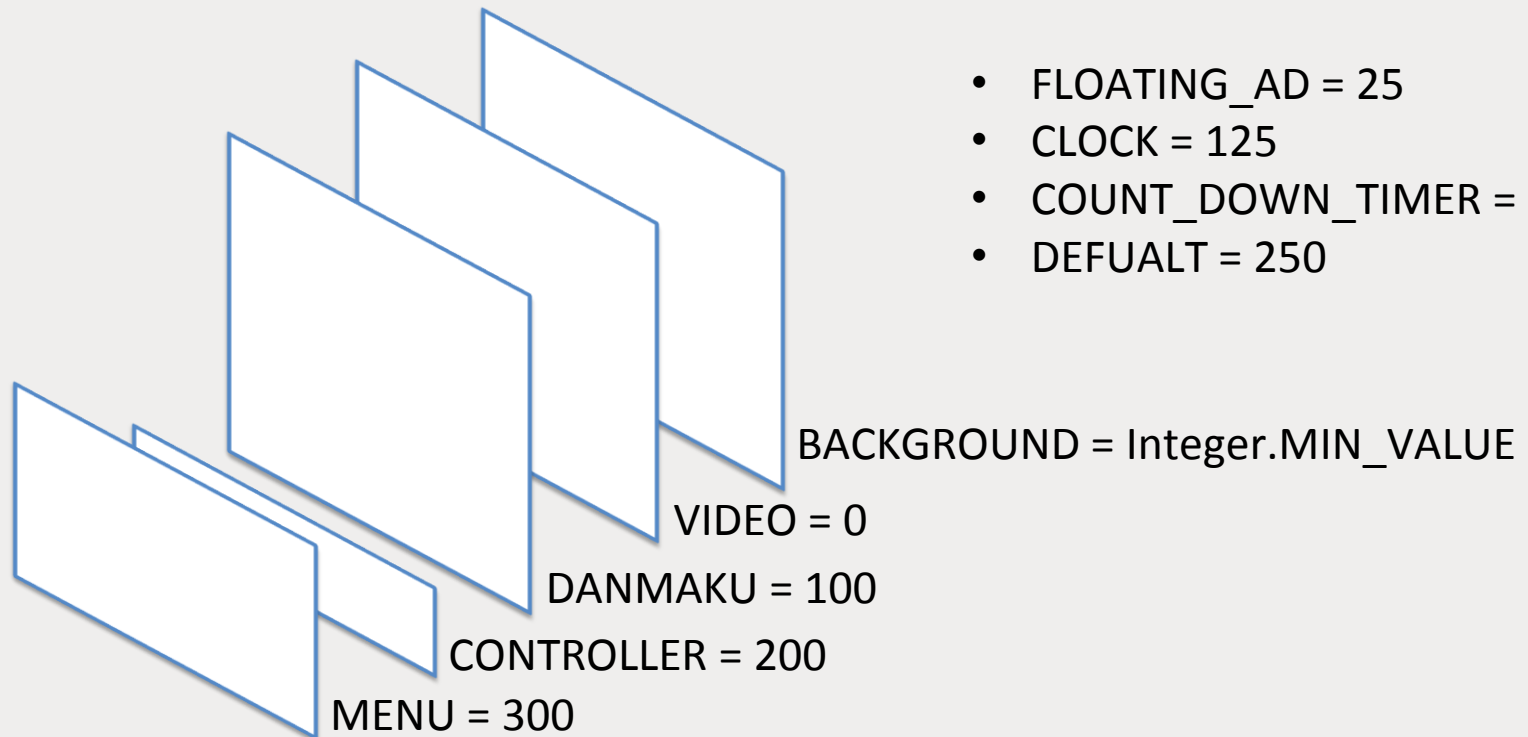
cd、搞笑、基情一个也... 范冰冰陈学冬携众星CP...

邓超林允演绎

Design of the framework

IViewPolicy manages the Z-Order of the views, and what will happen to other views if certain view is just displayed.

IViewPolicy do this according to the view type.



- FLOATING_AD = 25
- CLOCK = 125
- COUNT_DOWN_TIMER = 175
- DEFAULT = 250

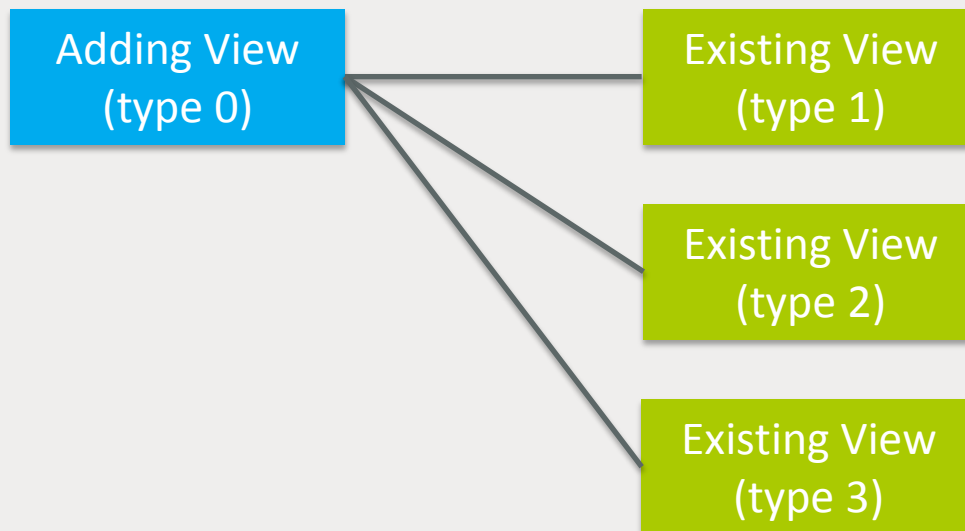
Design of the framework

What happened if a view is requesting to be displayed by a controller?

- Refuse the request - **AddViewStrategy.REFUSE**
- Dismiss one or more other views - **AddViewStrategy.REMOVE_EXISTING**
- Just display - **AddViewStrategy.KEEP_BOTH**

// To get an instance which implements the given interface

AddViewStrategy **getAddViewStrategy**(IControllerView *adding*, IControllerView *existing*)



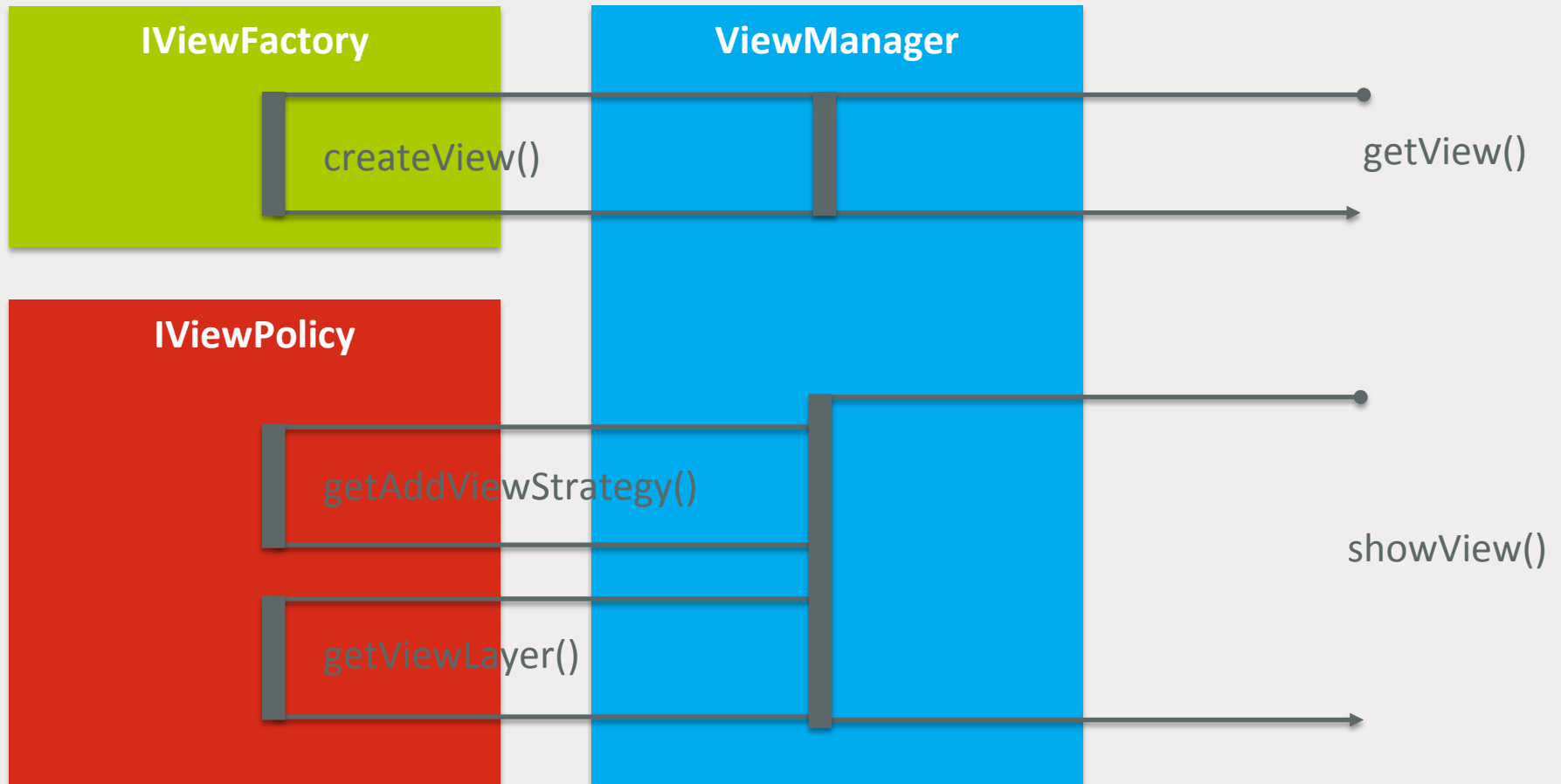
/ Design of the framework

Strategies of the return value of `getAddViewStrategy()`:

- **AddViewStrategy.REFUSE:** Stop the traversal and refuse the view to be added.
- **AddViewStrategy.REMOVE_EXISTING:** Existing views with this strategy will be collected, and they will be removed later.
- **AddViewStrategy.KEEP_BOTH:** No effects for existing views with this strategy.

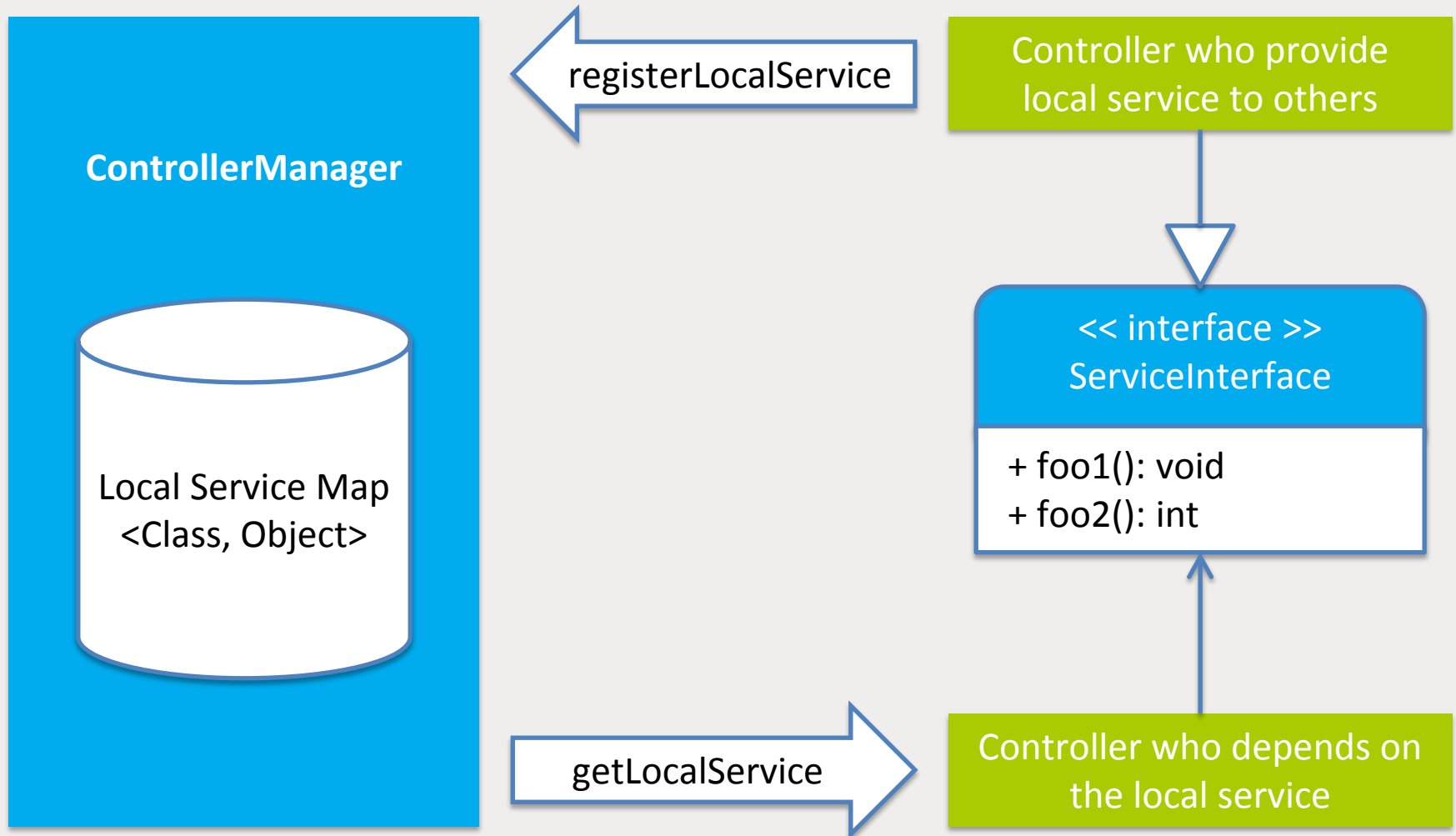
Design of the framework

Overview of ViewManager, IViewFactory & IViewPolicy.



Design of the framework

How a controller to communicate with other controllers?



/ Design of the framework

Thank to the local service, we can provide lots of common functionalities by providing certain controllers.

- Playback statistics reporting
- Video time line management
- Heartbeat scheduler
- Settings framework
- Play list
- Trail playback
- ...

We can also provide Extension Packs which contains a set of controllers that is suitable for certain business domain.

- VOD extension packs
- Live streaming extension packs

/ Design of the framework

Overview of this framework:

- How a business nested in: **IController**
- ControllerManager
- Various callbacks

- How a business presents it self to the user: **IControllerView**
- ViewManager
- ViewFactory & ViewPolicy

- Communications between controllers: **Local Service**

ViewFactory is an abstract factory to create view for each controller. If you need a different presentation of a controller, just inherit an existing implementation of this interface, and return your customized View.

AddViewStrategy tells ViewManager how to deal with the exclusion of Views.
*REFUSE what the *HIDE_E and want *KEEP_E will be shown.

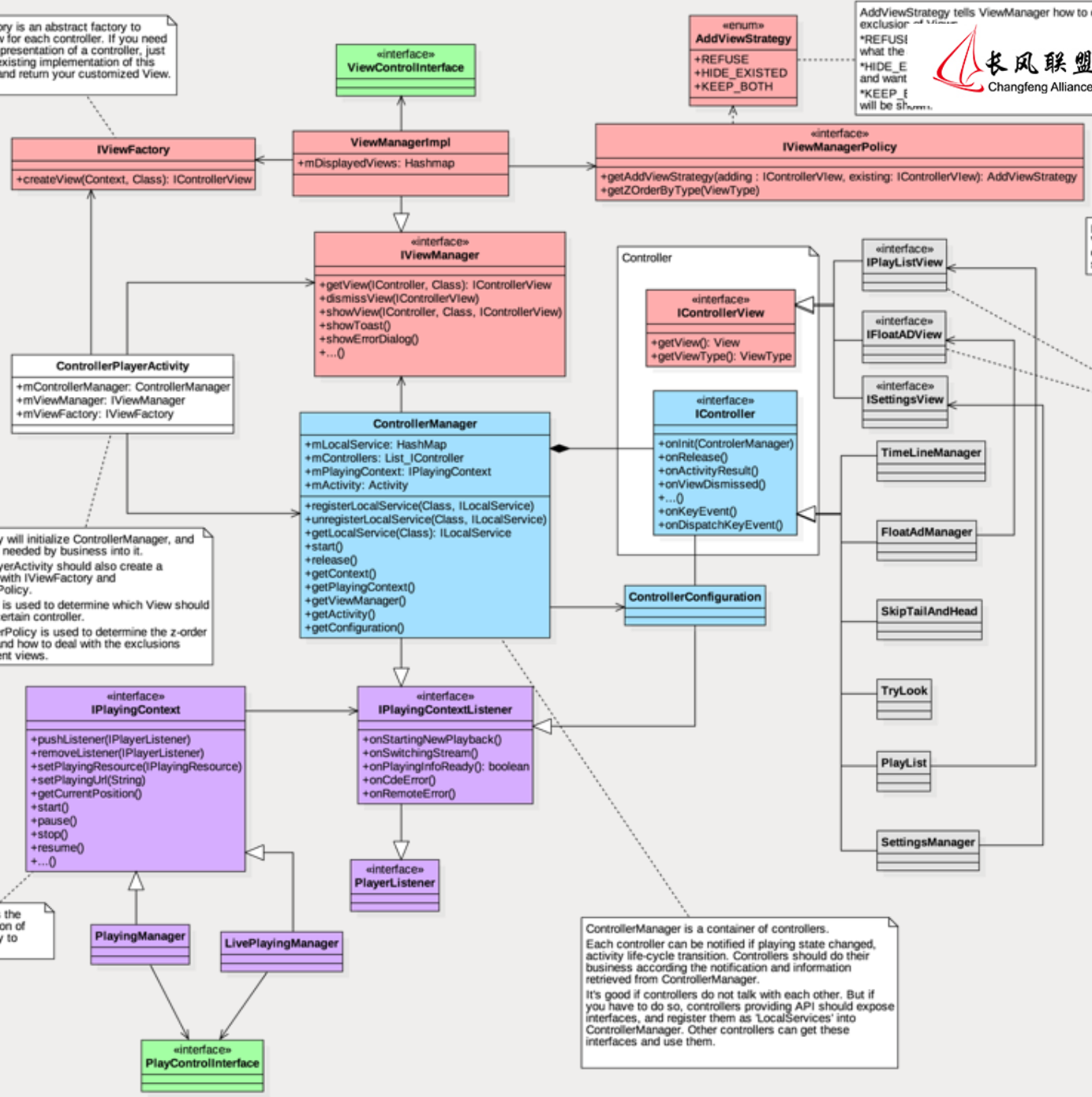
Each controller view should have an viewType, to be a factor for calculating the z-order if the Views shown together.

For controllers with complex logics, we'd better to use interface to represent the view in these controllers. It will make the controller testable (unit test)




* PlayerActivity will initialize ControllerManager, and add controllers needed by business into it.
* Besides, PlayerActivity should also create a IViewManager with IViewFactory and IViewManagerPolicy.
* IViewFactory is used to determine which View should be used for a certain controller.
* IViewManagerPolicy is used to determine the z-order of the Views, and how to deal with the exclusions between different views.

IPlayingContext gives the ability to get information of playing, and the ability to control the playing.

ControllerManager is a container of controllers. Each controller can be notified if playing state changed, activity life-cycle transition. Controllers should do their business according to the notification and information retrieved from ControllerManager. It's good if controllers do not talk with each other. But if you have to do so, controllers providing API should expose interfaces, and register them as 'LocalServices' into ControllerManager. Other controllers can get these interfaces and use them.



Outline

-  Introduction of the background
-  Design of the framework
-  Choices

Choices

Shell we supports hot-plugging of controllers, which means we can add or remove controllers at runtime?

NO.

Who has the rights to add or remove the controller?

Effort is needed for ensure the stability.

There are lots of alternative ways to support enable/disable a controller during runtime.

Choices

Is Message better than LocalService?

Message is lightweight than callbacks, and is very easy to use for events. It also provide the weakest dependency between sender and receiver.

But it is a disaster for complex communications, but usually abused because it is too easy to send or receive a message (~~cause we are lazy~~).

We provided a controller named **MessageCenter** for message delivery.

Choices

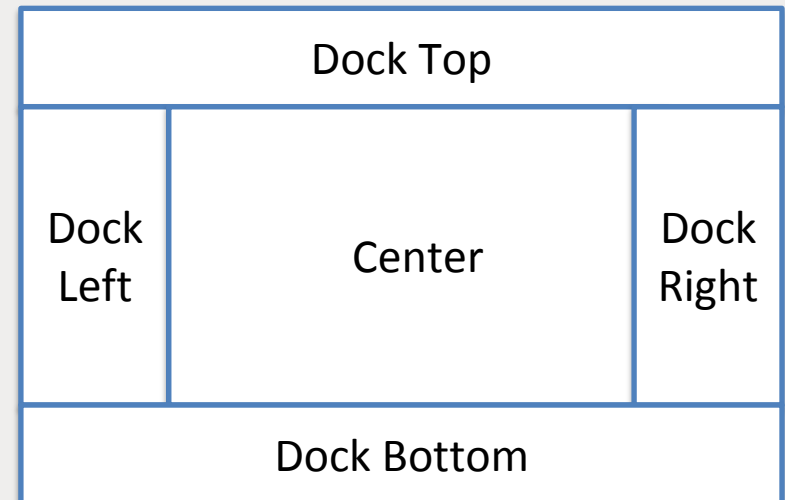
What about to provide a layout template? Just like BorderLayout in Swing? So showing a view could be like this:

```
// To get an instance which implements the given interface  
// To display an IControllerView  
boolean showView(IController controller, Class viewClass, IControllerView view, int  
contract);
```

NO.

We don't want the controller knows any thing about how the view layout is arranged.

But doing this in the implementation of IViewFactory sounds good.



Choices

Using an matrix to determine the strategies when adding a view seems simpler than using an method of IViewPolicy.

This matrix works well for most cases, but it does not contains enough information for solving complexed cases.

This is why we didn't include this matrix into our framework.

But it is good implement IViewPolicy based on this matrix, and write extra code to solve complexed cases.

existing

	A	B	C	D
A	0	-1	0	1
B	1	0	1	1
C	1	1	0	1
D	1	-1	1	0

Adding



Thank you!