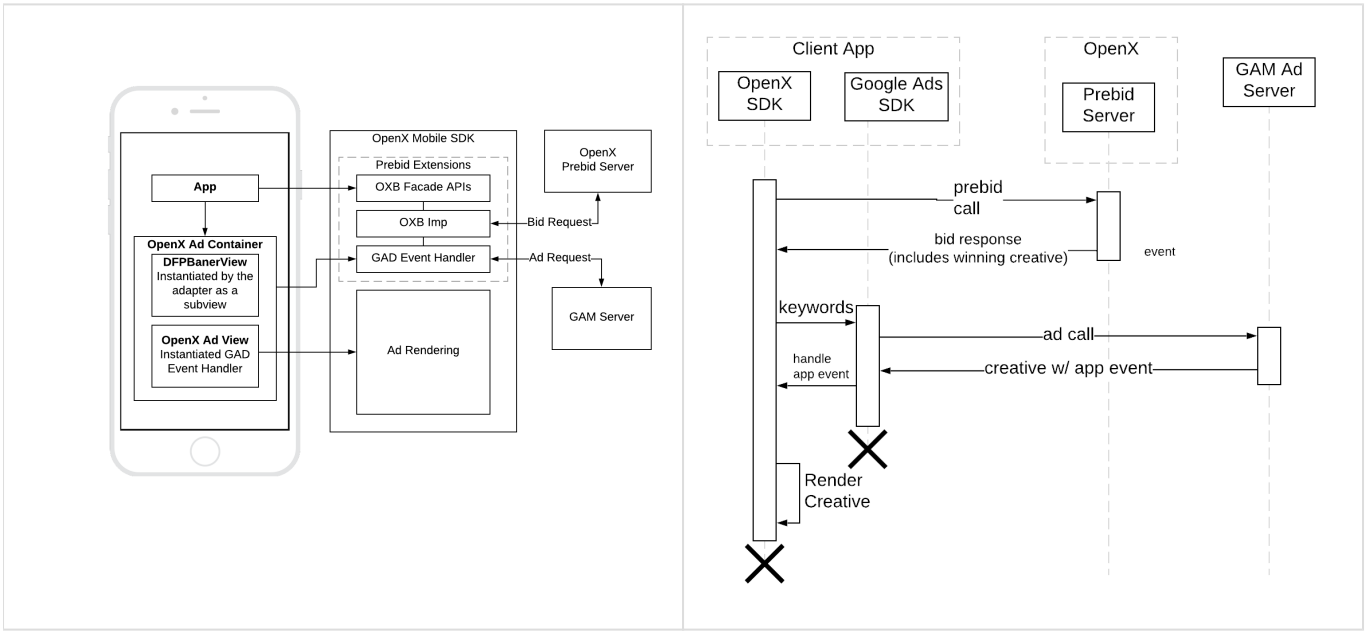
[EXTERNAL EXPORT] GAM Facade

The support of Google Ad Manager (GAM) as a Primary Ad Server is implemented using the App Events feature. GAM's line item contains information about the prebid extension. On receiving the app event SDK instantiates the view for rendering the winning bid. Otherwise, the native GAM Ad View will be displayed.



Banner

The GAM Banner Facade is a set of classes that provides the ability to bind the Rendering Module with Google Ad Manager SDK (GMA SDK).

BannerView - the view object that serves as a view container and as a loading manager for a certain ad view. The BannerView should be built into the UI like any other view. The publisher is able to customize the view itself and the loading properties. To start the loading process the publisher have to explicitly invoke the **load()** method.

Responsibilities:

- Be a placeholder in the UI
- Configure a bid request
- Make a bid request
- Pass the bid response to the `GAMBannerEventHandler`
- Inform the delegate about the flow's stages

BannerEvent - the protocol (interface) which serves as a bridge between SDK and independent class GAMBannerEventHandler. BannerView depends on this protocol but not on a particular event handler, and as a result not on GAM classes.

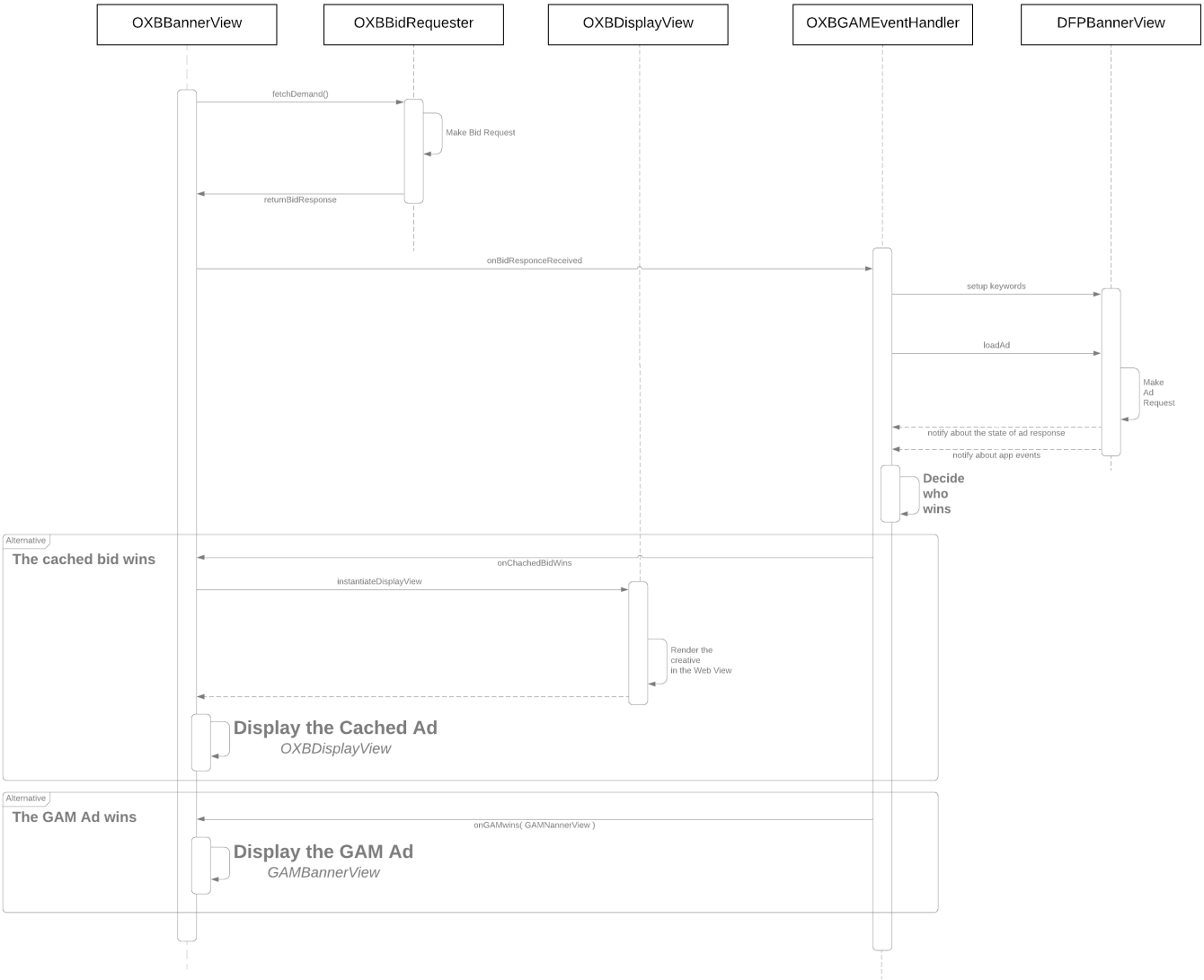
GAMBannerEventHandler - the wrapper that manages the work with GMA SDK classes according to the in-app bidding flow.

Responsibilities:

- Create a GMA SDK Ad View
- Configure the GMA SDK Ad View respectively to the bid response
- Make an ad request to the GAM Server
- Notify the BannerView about the waterfall winner
- Provide the GMA SDK view for displaying if the Ad Server wins
- Handle the inactive Ad View on switching the won ad source

BannerViewDelegate - the interface that a delegate should implement to handle events from **BannerView**.

The sequence diagram for fetching the bids and displaying the ads combining the GAM banner and DisplayView:



Native

The GAM Native Facade is a set of classes that provides the ability to bind the In-App bidding with GAM integration.

NativeAdUnit - the object that serves as a bid requester which enriches the GAMAdLoader with targeting keywords. The publisher is able to customize the bid request properties. Also, the publisher should invoke the code for loading the GAM ad explicitly in the callback of the bid request. To start the loading process the publisher has to invoke the **fetchDemand()** method explicitly.

Responsibilities:

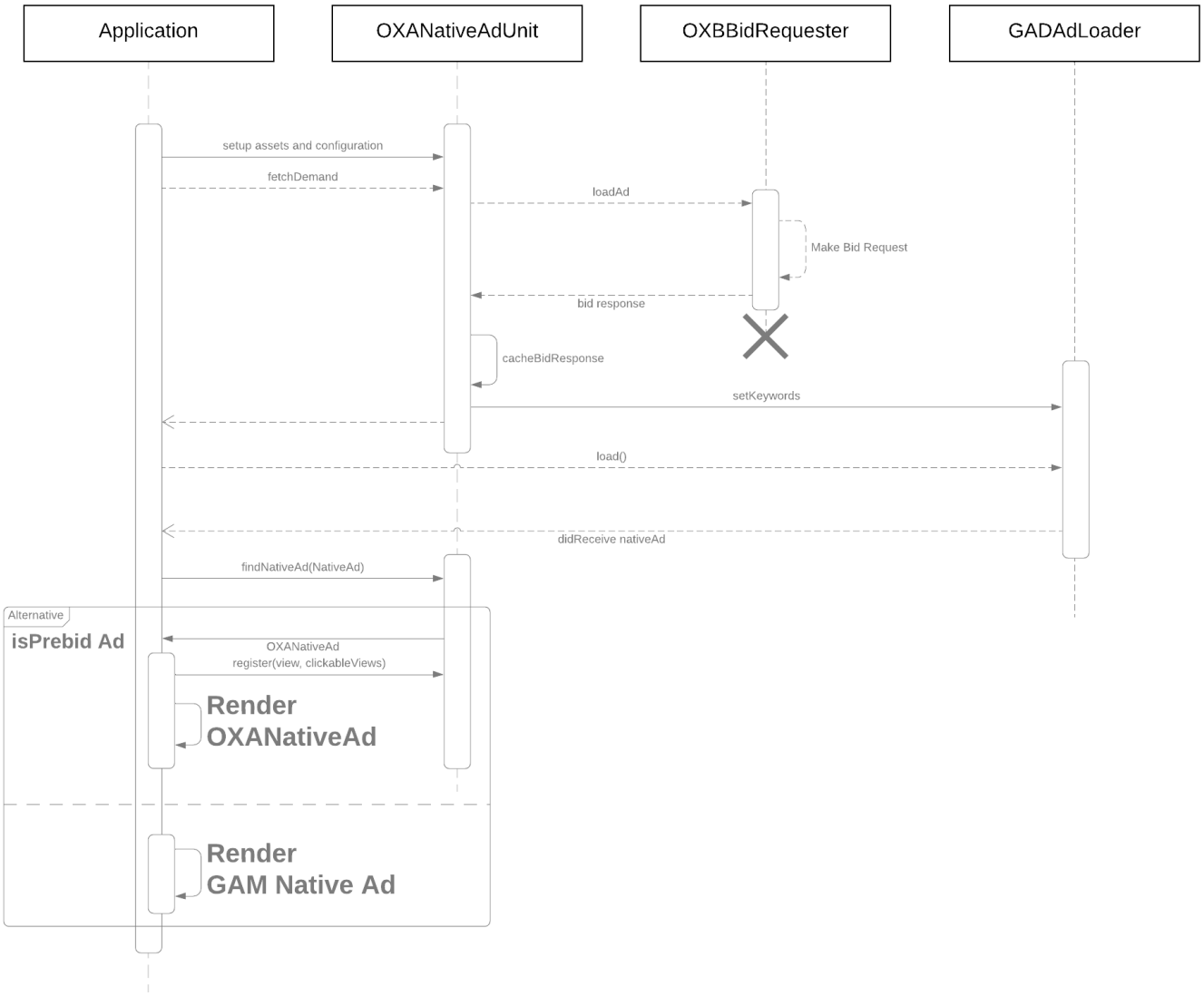
- Configure a bid request
- Make the bid request
- Enrich the GAMAdLoader with targeting keywords.
- Invoke a callback with the status of the bidding and NativeAd object if Apollo's ad source won.

NativeAd - the object that represents the native bid response. Publishers should use this object to initialize UI elements of the ad. In addition, the NativeAd is responsible for processing native event trackers, received in the bid response, impression trackers, and click trackers. So publishers should register the Ad View and clickable UI components in the NativeAd before adding the ad to the UI.

Responsibilities:

- Provide an access to the deserialized native bid response.
- Track native ad events like impression, click, viewability status, and others.

The sequence diagram for fetching the bids and displaying the ads combining the app's UI, GAM Ad Loader, and NativeAdUnit:



Interstitial

The GAM Interstitial Facade is a set of classes that provides the ability to bind the In-App bidding with GAM integration.

InterstitialAdUnit - the controller that manages the bid requests, ad requests, and displays the interstitial ads. The InterstitialAdUnit is a regular object that should be instantiated and used in the part of the app (controller or activity) where the interstitial should appear. The publisher is able to customize the bid request properties. To start the loading process the publisher have to explicitly invoke the **load()** method.

Responsibilities:

- Configure a bid request
- Make a bid request
- Delegate the ad request to the event handler.
- Remember who wins in the waterfall
- Delegate opportunity to show the Interstitial to the win object - GAM or interstitial
- Inform the delegate about the flow's stages

InterstitialEvent - the protocol (interface) which serves as a bridge between SDK and independent class GAMInterstitialEventHandler. InterstitialAdUnit depends on this protocol but not on a particular event handler, and as a result not on GAM classes.

Responsibilities:

- Provide a communication interface between the ad unit and the GAM wrapper.

InterstitialEventHandler - the wrapper that manages the work with GAM classes according to the in-app bidding flow.

Responsibilities:

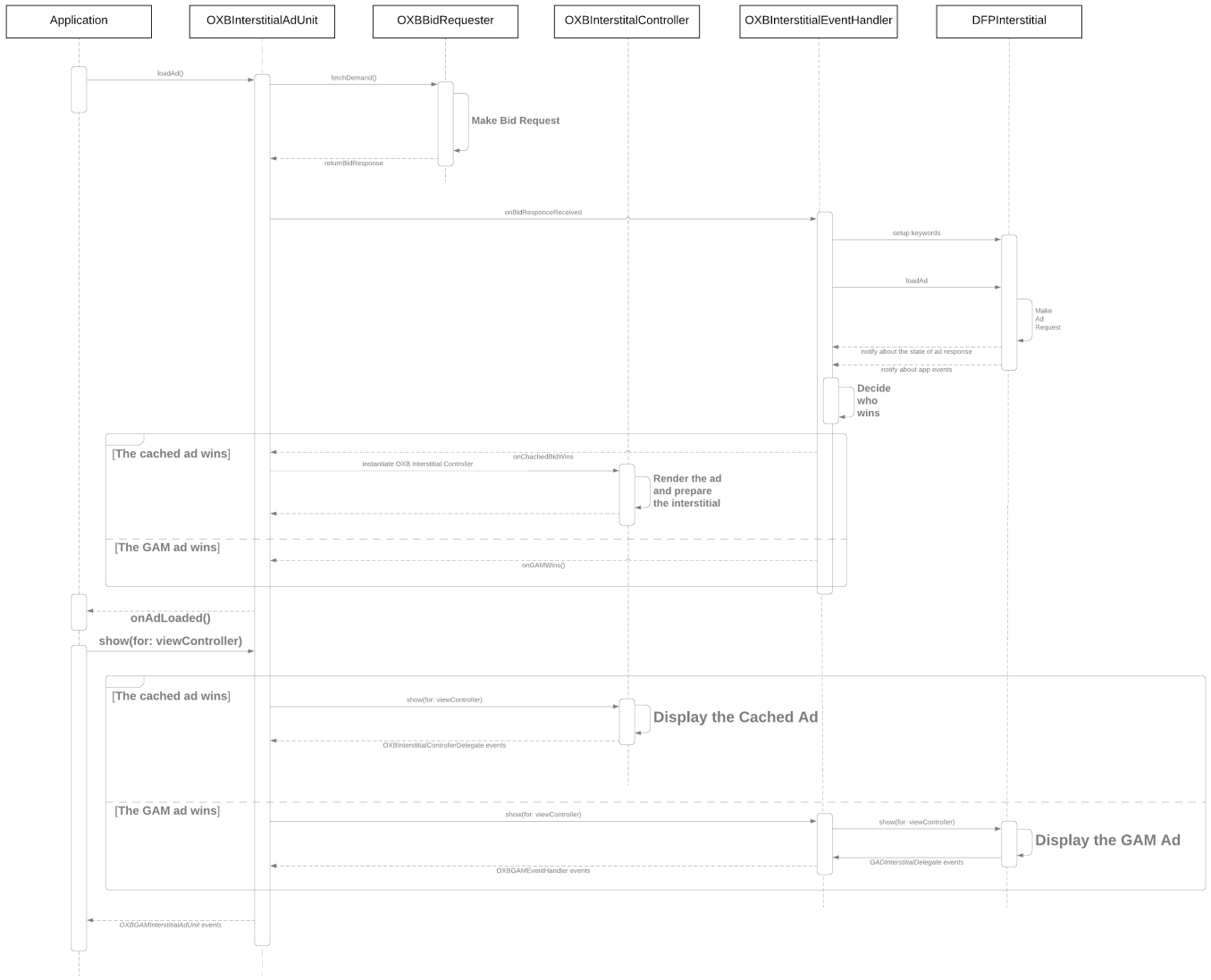
- Create a GMA SDK Ad View
- Configure the GMA SDKAd View respectively to the bid response
- Make an ad request to the GAM Server
- Basing on the ad response decide who wins - the In-App Bidding ad source or the primary ad server ad source.
- Notify the GAMBannerView about the waterfall winner
- Provide the GAM's interstitial controller for displaying if the Ad Server wins
- Handle the inactive Ad Viewinterstitial controller on switching the won ad source

InterstitialAdUnitDelegate - the interface that a delegate should implement to handle events from **InterstitialAdUnit**.

Responsibilities:

- Provide a communication interface between the app code and the interstitial ad unit.

The sequence diagram for fetching the bids and displaying the ads combining the GAM banner and DisplayView:



Rewarded

The API for Rewarded ads is based on the Interstitial API. The flow of loading and showing the ads is totally the same.

The difference between those two ad formats is additional methods for rewarding the user at some point. So the `RewardedAdUnitDelegate` has an additional callback `userDidEarnReward` to inform the app that the reward has been received. The argument of this callback is `NSObject` on iOS and `Object` on Android. Publishers have to cast it to `GADAdReward`.

RewardedAdUnit - totally the same as `InterstitialAdUnit`.

RewardedAdEvent - the same protocol as `InterstitialEvent` with *additional responsibility* - passing the Reward object to the `RewardedAdUnit`.

GAMRewardedAdEventHandler - the same wrapper as `InterstitialEventHandler` which manages the `GADRewardedAd` with *additional responsibility* - passing the Reward object to the `RewardedAdUnit`.

RewardedAdUnitDelegate - the same protocol as `InterstitialAdUnitDelegate`with *additional responsibility* - passing the Reward object to the delegate class in the app.