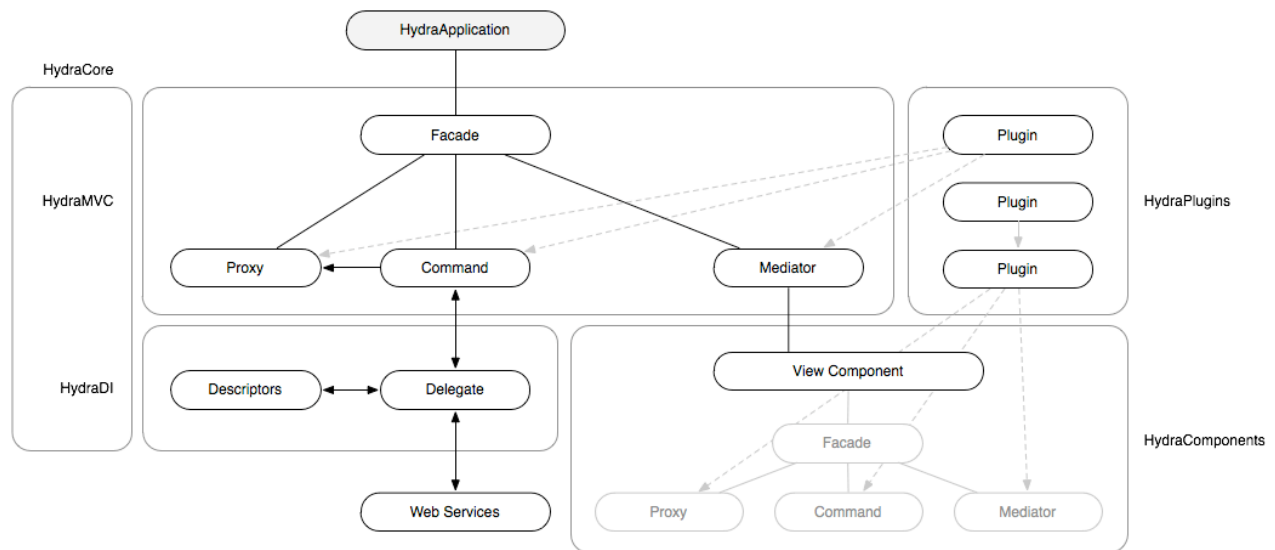


## HydraFramework



### Object Type

- **component**
  - **controller**
    - Command.as
  
- **data**
  - **descriptors**
    - ActiveDescriptor.as
  - **delegates**
    - Delegate.as
  - **interfaces**
    - IInterface.as
  
- **model**
  - **descriptors**
    - SimpleDescriptor.as
  - Proxy.as
  
- **view**
  - **components**
    - complexComponent\*
    - SimpleComponent.mxml
  - **events**
    - Event.as
  - Mediator.as
  - Component.mxml
  - ComponentFacade.as

### Purpose

**Commands** are state-and-context-agnostic implementations of units of application logic. They are executed by the **Facade** when their mapped **Notification** is sent from any registered actor. In contrast to calling a method directly, commands allow an object to represent and encapsulate the information needed to call a method (typically on a **Proxy**) irrespective of state or context.

**Active Descriptors (Hydra DI)** are strictly-typed, complex objects that represent stored objects in the system, and are used in client-server interactions.

**Delegates (Hydra DI)** defines how the system interacts with remote data, typically via a RemoteObject, webservice, or a simulated server interaction in the case of **Mock Delegates**.

**Interfaces (Hydra DI)** allow the implementation of AS3 representation of data objects to be decoupled from the system through abstraction.

**Simple Descriptors** are strictly-typed, complex objects used for the purposes of moving data around in the view, but do not represent stored data nor need to interact with web services or other remote data.

**Proxies** maintain data and state for the component, and are responsible for sending **Notifications** that reflect their state.

**Recursively follow this pattern.**

**Simple Components** are a collection of view components that are self-contained.

**Events** are dispatched from **Component**, and observed either internally by the **Component** or by the **Mediator**.

**Mediators** define how the view should react to **Notifications** via the view's public API.

**Components** are a composition of view components that provides an interface to the user.

**Facades** provide a comprehensive map of the application logic (by defining **Notification** names and mapping them to **Commands** via `.registerCommand()`) and registered actors (by registering core actors via `.register{Actor}()`).