# Architecture and Social Context of Glide

Team_Costco: Duo Chai, Soobin Choi, Marc Andrada

## Glide Architecture

### Context

The landing page for documentation related to the Glide system is on github (https://bumptech.github.io/glide/). The website provides documentation for Glide components that can be aggregated into Glide's major features such as efficient image loading, processing, and caching. Since the system architecture is not explicitly discussed, we attempt to document and produce the system architecture based on available resources such as this documentation, system source code, and the UML diagram.

### Architecture Recovering Process

We applied the Model-View-Controller architecture pattern, as shown in Figure 1, in the attempt to delineate Glide's system architecture. Glide is a library that provides a smooth way for applications to display images.

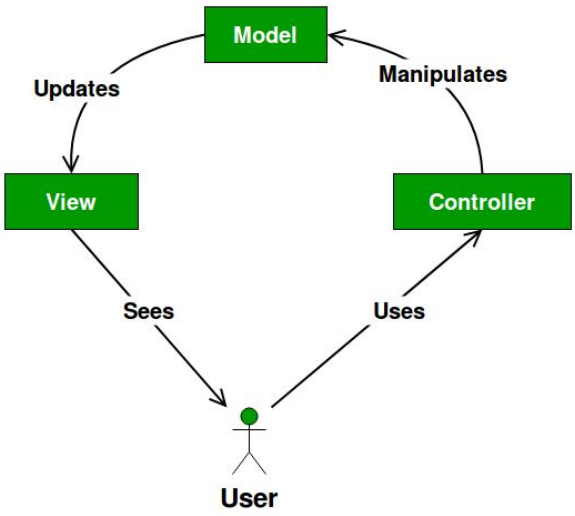



Figure 1: MVC Architecture Pattern (Image Source: GeeksforGeeks.org)

Since Glide is a library, the View of the system would be the UI of the external applications that utilize the Glide api to deal with image load, display, and caches. The Model object in the architecture contains the request data that essentially gets built into the Controller that controls how the data gets displayed based on the end-user's interactions with the external application.

In class, we applied a bottom-up comprehension method to derive the architecture diagram for JPacman4. Since the Glide system is so large, such comprehension strategy poses various challenges; so instead, we recovered the architecture of Glide through top-down comprehension of the system. We took a look at Glide's website and used the documentation to extract meaningful features of the system at high-level. Then we tried to figure out the relationships between and behaviors among these meaningful features while confirming our hypotheses by referring to the source code and folder hierarchies. After iterations of confirming our hypotheses

while revising our architecture, we drew the final version of our architecture as shown in Figure 2.

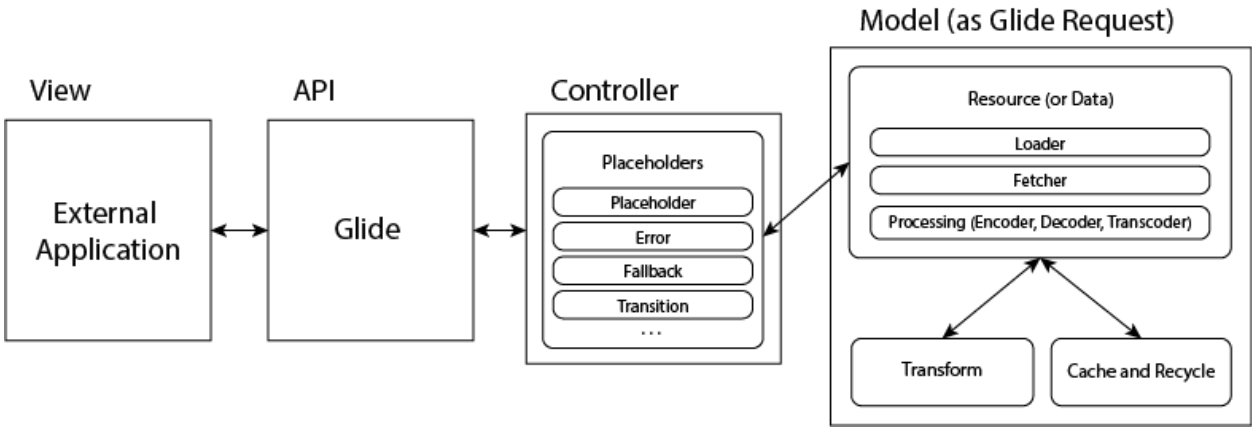



Figure 2: Glide Architecture

**Architecture Result and Description**

In the previous section, we described the method in which we recovered the architecture of Glide. To verify the accuracy of this architecture in code, we examined classes and their behaviors to consolidate each aspect of the MVC pattern.

Model

For the Model object, which we defined as a Glide request, we are specifically dealing with resources of different data types (such as bitmap, bytes, drawable, file, gif, etc.). The operations that deal with these resources, which we interpret as “Processing” as shown in Figure 2, are stored in the Glide/src/library/load/resource/ folder. Prior to resource processing, the Fetcher (implemented in Glide/src/library/load/data/ folder) fetches data from URI pointing to a local resource. The Loader(implemented in Glide/src/library/load/model/ folder) translates a specific model into a data type that can be decoded into a resource. Then, through a series of resource processing operations, such as encoding, decoding, and transcoding, the system makes a resource of any data type available for building a request.

In addition to processing resources, the system also caches resource data. We observed a bidirectional relationship between resources that are processed and operations that generate cache keys from original resource data (can be either remote or local) to either load data from cache or skip caching based on the caching strategy. Caching and recycling of the program provides the ability for Glide to retrieve resources faster and more efficiently by checking different layers of caches before starting a new request for a resource such as image. The classes and interfaces related to caching and recycling are stored in the Glide/src/library/load/engine/ folder.

The transformation component also maintains a bidirectional relationship with resources. The operation will take a resource, mutate it if possible, and return the mutated (or un-mutated) resource, which can also be used to create cache keys through the system’s cachers and

recyclers. The classes related to resource transformations are accessible throughout the Glide/src/library/load/ folder.

Controller

The Controller object, which we defined as “Placeholders”, provides different methods to external applications to control how the different types of resources are being loaded and displayed into their View object. We chose this as the essential feature of this program and discussed it in homework 1 and 2. To localize this component, we dove deep into the Glide/src/library/ folder. Glide/src/library/request/target/ folder stores different target options, which are responsible for both displaying placeholders and loaded resources and determining the appropriate dimensions to ensure that images are loaded properly into the views for each request.

Another component that affects how “Placeholders” behave is through the way transitions are defined within the library. Transition is a controller object that exists between the user interface of the external application and the resources that are processed through Glide. Glide/src/library/request/transition/ folder stores different transition options. The different transition options, which can be adjusted in transition factories, determine how a drawable transitions from a placeholder into a new image view.

The remaining files in Glide/src/library/request/ folder and the methods defined in Glide/src/library/ folders contribute to the implementation of other Placeholders, or Drawables that are shown while a request is in progress, such as an error, fallback, and placeholder drawables.

Through thorough examination of all relevant files and folders in the Glide/src/library/ folder, which is the main src of our program, we were able to consolidate the MVC architecture components above that cover all major aspects of the Glide system.

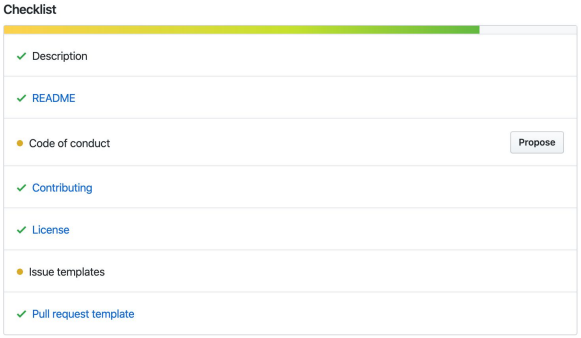
View

The Glide API facilitates the communication between the Controller object and the View object. The View object is represented as external applications interacting with the API. Depending on the requirements of the developers that are adopting Glide, the library can be utilized to fulfill specific and customized functionality for their system. Reliable and effective communication between these two components is essential for Glide to achieve it’s end goal of providing a “flexible API”, which allows:

1) Scrolling any kind of a list of images to be as smooth and fast as possible
2) Effective fetching, displaying, caching of both remote and local images
3) Capabilities for plugging it in to almost any network stack.

**Social Context of Glide System**

1. Project State: Still active
    - Level of usage: It has been forked by 5.2k, watched by 1.1k and starred by 28.5k users compared to Picasso, a similar image loading/displaying library, which has 4k, 921, 17.3k respectively.
    - Last release date: Last release date was on Jan 8, 2020. The last PR was merged on Feb 25, 2020.
    - Release roadmap: According to https://github.com/bumptech/glide/releases releases do not follow a development roadmap that are based on system updates determined by Glide's development team. Instead, they seem to push releases twice a year after enough commits have been added to the system. Glide evolves through weekly PR merges, observing from their activities on merging PR and closing issues.
    - #commits as of late: There have been 2472 commits as of Feb 25, 2020. Observing from the commit history, there are a few new PRs being merged weekly.
    - #open issues as of late: 144 open issues and 3484 closed issues as of Feb 25, 2020. We observed that maintainers deal with and close open issues once a week.
    - #active developers: Sjudd appears to be the only active maintainer for now. But there are many developers, since a lot of people contribute by creating new PRs: mkj-gram, ouattararomuald, landicefu, and etc.
2. Github Standards: PRs must include a description, motivation and context; however, they don't have a template established for issues.




3. Process for Contribution: In general, all contributions are welcome as mentioned here (https://github.com/bumptech/glide/blob/master/CONTRIBUTING.md). Since Github is the only way to contribute code changes, contributors must sign the Google's individual contributor license agreement before contributing. Although they have a defined code style in their IntelliJ code files in the repo, contributors who aren't able to pass the code styles are encouraged to submit a pull request anyway as the team will provide support where necessary. The lifecycle of submitting codes is not exactly the same as what we discussed in class since they won't assign a developer to fix a problem and will only upgrade Glide by merging PRs. By observing the lifecycle of how maintainers merge PRs, we discovered that they practice the following:
    - Find a bug/improvement
    - Create a new PR describing the problem, motivation, and context

- Test the PRs if maintainers think it is meaningful and the method used to solve the problem can be maintained from a long term perspective
- Merge PRs that pass the test, otherwise close the PR.

4. Support Tools:
    - By observing the PR lifecycle, we found they use Bots such as googlebot, stalebot (Automatically add CLA tags and close stale Issues and Pull Requests that tend to accumulate during a project).
    - They also use labels to categorize pull requests and issues (enhancements, bug, etc.).
    - Since they use github as the main platform to share debugging and code changes for all aspects of libraries, we can assume they also use version control to maintain the system on their end.
    - They use TravisCI to test project build.
    - Checkstyle: To check coding styles.
    - They use Maven central to check Maven dependencies in the local repository, and will search in the Maven central repository if dependency cannot be found locally.

**5 Interesting Pull Requests**

1) Update ByteBufferStream read() to return byte values from 0-255 or -1 if EOF reached
   https://github.com/bumptech/glide/pull/3887
   → Adjusted ByteBufferStream read(), by adding a bitmap mask '0xff'. After masking by '& 0xff', it will leave the value in the last 8 bits of the bit stream, promising the accuracy of bit computing.

2) Add a VideoDecoder for ByteBuffers
   https://github.com/bumptech/glide/pull/4033
   → Resolved an issue with VideoDecoder, where if the given DiskCacheStrategy: RESOURCE or NONE, video files would not be properly read and converted by the decoder. The pull request appears to resolve this by implementing a byte buffer to the VideoDecoder and was merged. This pull request also directly addresses an issue opened earlier here: https://github.com/bumptech/glide/issues/4021.
3) Improve handling of EOF in DefaultImageHeaderParser.Reader
   https://github.com/bumptech/glide/pull/3952/files
   → Original code in DefalutImageHeaderParser.java does not support EOF exception handling. The PR handles additional failures on any that reads or parses byte array data. Seems like an oversight at the time of initial development since the system deals with various data types and error handling is an important aspect.

4) https://github.com/bumptech/glide/pull/3880
   → Updated SDK to meet with Google Play's API requirements. The conversion didn't affect the build of the system per the Travis CI tests. This PR is interesting because it

shows possible best practices when it comes to keeping a system up to date. The developer seems to be considering the future of the system by updating to the new SDK.

5) Improve comments and consistency of sampling in DownsampleStrategies.
https://github.com/bumptech/glide/pull/3703/files
→ Good comments in a large system like Glide will help users tremendously in regards to what components play a role and how they are related. This PR not only improves downsampling the dimensions for image resources but also add helpful comments detailing how the downsample strategy changes when the system is developed and run on KitKat API.

**5 Interesting Issues**

1) Glide.with(Context) very very slow (takes 1 sec)
https://github.com/bumptech/glide/issues/3876
→ This is an issue with performance. Apparently, the initialization of Glide takes “too long” (1 second). One of the maintainers mentioned that they recognized this issue and proposed a soft solution of allowing glide to be “lazily initialized”. By this, we believe the maintainer means to initialize the Glide class as close as possible to when it will be utilized for example when loading an image. Other comments appear interested in a more formal solution to addressing the performance.

2) Certain GIFs display red rectangles after 3.7
https://github.com/bumptech/glide/issues/3319
→ Issues with certain gif files after 3.7 update. “Red rectangles” appear in the frame of the loaded gif. There rises the question of whether it is just bad files or something with how the gif file is being loaded using glide. Currently no PR has been attempted, but ideas are being sought after.

3) In a weak network (3G), the local picture is displayed only after the network picture fails to load.
https://github.com/bumptech/glide/issues/2998
→ There appears to be a lot of discussion on this issue, which is why we wanted to look into it further. The main issue appears to be difficulty loading an image depending on the type of network utilized. The networks considered were GSM and UMTS. These are types that can be determined in the android emulator settings. A potential workaround for this has been proposed, which would involve “reading image bytes into memory and wrap it with the ByteArrayInputStream”. The user who proposed this noted that it would take away from a currently standing method in the system: setPriority.

4) Glide's `trimMemory()` implementation doesn't trim when foregrounded
https://github.com/bumptech/glide/issues/2810
→ This issue revolves around problems with memory “trimming” and management depending on whether the application is in the foreground or background. The outside

developer noted that when the application is in the foreground, a call to Glide's method: `android.content.ComponentCallbacks2.TRIM_MEMORY_MODERATE` does not return a high enough integer to trim memory. On the other hand, when the application is in the background, the same call returns a value large enough to trigger memory trimming.

5) [Question] How to prevent clearing an ImageView before the load is complete?
https://github.com/bumptech/glide/issues/2505
→ The issue is mainly requesting help with locating useful features within current documentation involving their concern with image load transitions. A user was wondering whether they could prevent Glide from clearing their image from an image view before loading is complete. Everytime he refreshed the image, it would flicker white for a second before it was completely loaded, unless it was currently present in memory cache. The user was looking for updated documentation for a method called dontAnimate() hoping this could disable the transition issue he was having. According to the maintainers, this transition is necessary to prevent potential memory leaks from occurring due to frequent refresh calls. They suggested utilizing a ViewSwitcher class, which can display an image and wait for the next to load before transitioning.