GNUstep: Architectural Enhancement

April 4th, 2025

Group 17: Architecture Addicted Aficionados

Group Members:

Sungmoon Choi	22sc@queensu.ca
Dayna Corman	21drc13@queensu.ca
Michael Cox	21mwc3@queensu.ca
Aydan Macgregor	21ajm32@queensu.ca
Samantha Mak	21yssm@queensu.ca
Kylie Wong	21kycw@queensu.ca

Abstract

In this report, we propose a new enhancement for the GNUstep framework. One serious functionality that GNUstep lacks is 3D support. This is a common functionality in modern-day applications and expected of most up-to-date GUI frameworks. We believe this is a critical flaw that GNUstep lacks. This report will lay out a detailed analysis of our proposed enhancement, the NFRs and stakeholders involved, how we can implement the enhancement into the GNUstep architecture, and perform a SAAM analysis on ways to realize our enhancement. The enhancement will be split into five main components that will each handle a specific service for 3D visuals. These components can easily be mapped to existing directories in the codebase. This allows for ease of creation for the enhancement and its tools. As we have the community at heart when proposing such an enhancement, we have identified four segments of the community as key stakeholders in specific sections of the enhancement and have analyzed their major NFRs. An impact report was made to show any increased complexities, risks, or improvements to GNUstep. We have also analyzed the desired quality of requirements and such tasks that will need to occur in testing for the desired quality. Finally, to realize the enhancement, we proposed two architectural styles and assessed their impact on the stakeholders and NFRs. From those two realizations, we determined which approach would be best for our enhancement.

1. Proposed Enhancement

We propose to enhance GNUstep with native three-dimensional graphics support with hardware accelerated rendering (3D-HAR, – in short as abbreviation in this report).

1.1 Three-Dimensional (3D) Computer Graphics

In today's computing standards, 3D support is a basic and core feature in computer graphics. It is obvious in 3D systems like architecture design, product design, medical applications, film and animation, video games, virtual reality and augmented reality, etc. It is also embedded in stylish and aesthetic 2D graphics generation from 3D models, which is used in any graphical user interfaces.

We represent a 3D object or world by some 3D models in computers, which can only be visualized or viewed on a computer display by human eyes after a 3D model is converted or rendered into a 2D image. This rendering task takes significant computing power and is commonly accelerated by specialized hardware like graphical processor units (GPU) on top of central processor units (CPU).

1.2 GNUstep's Current State

The current version of GNUstep does not support 3D graphics at all. There was an attempt to support such in the 3DKit[1] project to provide an object-oriented graphics API using GNUstep or Cocoa. But the project was marked as either unmaintained or unknown status since Nov 2012.

1.3 Values and Benefits

Nowadays, 3D is a basic feature for the majority of graphical application developments. 3D-HAR enhancement allows developers to create 3D graphics, and it modernizes and extends the GNUstep framework to today's stylish graphical computing world.

The 3D-HAR should support GNUstep application developers in developing platform-independent graphical systems, as mentioned above and value-added 3D graphics software like those listed in Wiki [2]. At the same time, the application developers do not need to take care of fast-changing hardware technologies, which should speed up the application development cycle and make development investment more sustainable. Application developers just need to focus on their own application's functionalities with the GNUstep framework.

1.4 Functional Partitioning

The 3D-HAR component can be functionally partitioned into 5 components, namely Core Model Services (CMS), Transformer Services (TS), Scene Services (SS), Virtual Rendering Services (VRS) and Physical Rendering Services (PRS) as below,

1. **Core Model Services (CMS)** – This provides services to support core 3D models, including 3D objects consisting of algorithms and data structure / format, as well as data conversation among supported formats. A 3D object's characteristics include shapes, textures, colors, materials, etc.
2. **Transformer Services (TS)** – This provides services to transform 3D objects from one state to another state. It supports an object to change its characteristics by the object's actions, from simple actions like changing property and scale in size to complex actions like motions. It also provides services for interactions between two objects. An object can interact with another by combining, intersecting, or subtracting the other. In addition, this service provides utility to automatically generate intermediate objects during complete transformation of the object into the other, just like a transformer in movies.
3. **Scene Services (SS)** – This provides services for a 3D space with one or more 3D objects there, as well as all environment factors of the space, like lighting from different angles. The service allows movement and rotation of an object within the 3D space, and event services when an object needs to interact with another object due to their positions in the space, i.e. overlapping.
4. **Virtual Rendering Services (VRS)** – This service provides the generation of 2D graphics from 3D objects by supporting viewpoints on a scene. A viewpoint is a 2D view on a 3D space through a camera from a position at a particular angle. A camera supports features like focal length, aperture setting, resolutions, etc. It is also equipped with various visual accessories like lenses and filters. As a result, the camera delivers various visual effects like focus point, field of depth, fisheye, color schema, etc. GNUstep application developers can develop platform-independent applications by using VRS,

which in turn hides Physical Rendering Services for platform-specific optimization like hardware acceleration.

5. **Physical Rendering Services (PRS)** – This service helps to achieve platform-independent GNUstep application development together with VRS. PRS acts as a translator from the virtual world of VRS to platform-specific rendering services by translating related objects and APIs. PRS can make use of platform-specific APIs, libraries, drivers and utilities for hardware-accelerated rendering. PRS can leverage the latest hardware and system technologies like multiple cores, multi-threading, task distribution among CPU (central processing unit) vs GPU (graphical processing unit), and even cloud rendering farms.

1.5 Interaction with Other Features

The 3D-HAR should be delivered as APIs in GNUstep. Its 3D objects/classes should extend from GNUstep's root class NSObject. A 3D object can be viewable or non-viewable after a 3D object is created or a 3D class is instantiated.

When a 3D object is viewable, it is viewed via one of its viewpoints and displayed as a 2D image on the display by its rendering services. After the display, the 3D object should be able to support the same functions as a 2D image object in GNUstep and interact with other GNUstep objects and features, too. This includes capabilities to receive user input events like mouse and keyboard.

However, a 3D object can behave differently from the GNUstep image object when responding to an event or to implement a feature on it. The 3D-HAR needs to provide additional 3D specific functions & features via APIs, methods and events. Optionally 3D-HAR may come with additional user input events like multi-touch gestures commonly used in today's user interaction with 3D graphics objects.

Within the 3D-HAR project, Gorm should be enhanced to support the new 3D classes and objects for GNUstep application development. Gorm can provide additional user interface templates handy for user interactions on 3D objects like rotation, selection of an object within a scene, full screen mode, etc.

1.6 Major Stakeholders

The complete cycle of 3D-HAR enhancement project includes at least the following 4 major stakeholders,

1. **GNUstep project community** –Tthe owner of existing GNUstep core libraries, incl. libs-corebase, libs-base, libs-back, libs-gui and apps-gorm
2. **Enhancement contributors** – Those contributing to this enhancement of

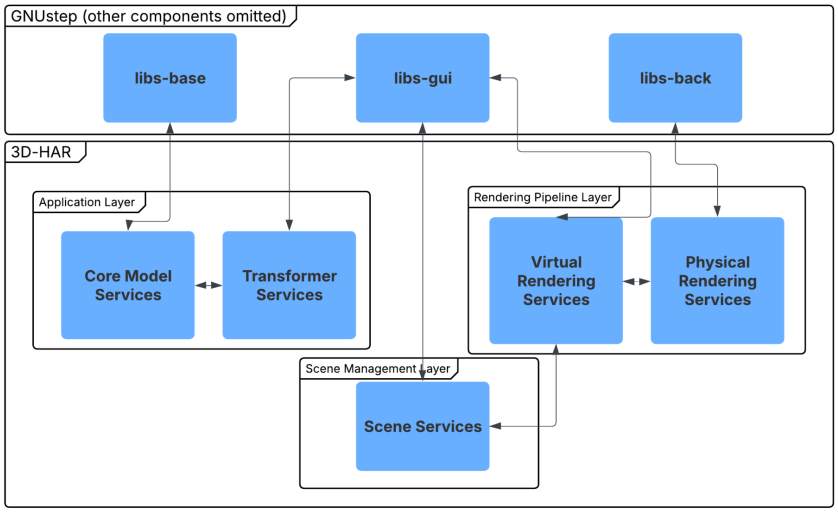
3. **GNUstep application developers** – Developers, independent software vendors (ISV) or organizations using the enhancement to further develop their own platform-independent applications or software
4. **GNUstep application users** – End users of GNUstep applications with this enhancement, e.g. graphics designers, game users, architecture designers, animation movie viewers, etc.

2. Mapping of Functional partitioning onto architecture's structural decomposition

To integrate 3D-HAR enhancement into GNUstep, we followed the Software Architecture Analysis Method (SAAM) to map the functional partitioning of its core components to current architecture.

2.1 Mapping of Functional Partitioning

Mapping the five functional partitioning (CMS, TS, SS, VRS, PRS) would look like the following:



2.2 Changes Required for Enhancement

To support 3D-HAR, existing GNUstep architecture must be extended to integrate new components into its framework. New 3D classes should derive from NSObject to ensure compatibility with GNUstep's existing object-oriented framework with a new rendering engine that must be integrated to support 3D visualization (such as OpenGL, Metal, Vulkan, etc.). To edit the 3D objects in Gorm, new UI elements must be interactable with a mouse and keyboard. Lastly, the scene object hierarchy system is used to handle multiple 3D objects and lighting.

2.3 Component and Interface Changes

Pre-existing components in GNUstep must be changed or extended to accommodate the introduction of new components.

- Core Model Services (CMS): Changed directory/file: libs-base/3DModel - new classes to represent 3D object representation

- Transformer Services (TS): Changed directory/file: libs-gui/3DTransform - functions for scaling, rotating, and interaction
- Scene Services (SS): Changed directory/file: libs-gui/3DScene - management of objects and lighting in 3D space
- Virtual Rendering Services (VRS): Changed directory/file: libs-gui/Rendering - integration for 2D viewport
- Physical Rendering Services (PRS): Changed directory/file: libs-back/3DRendering - integration for GPU-bound rendering work
- Gorm: Changed directory/file: apps-gorm/3DObjects - UI implementation for 3D object manipulation

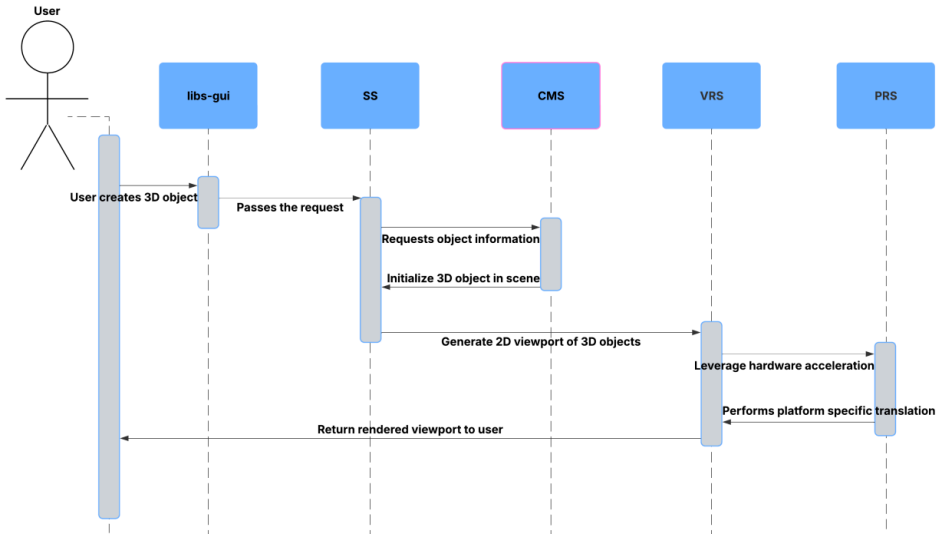
2.4 Impact on Conceptual Architecture

The addition of 3D-HAR changes both the high-level and low-level conceptual architectures of GNUstep and Gorm, as its existing architecture is designed for 2D graphics. Some of the impacts on conceptual architectures include:

- High-level
  - The existing GNUstep object model is expanded to accommodate 3D elements.
  - A layered system is introduced for logical rendering (VRS) and hardware rendering (PRS).
  - Gorm must support 3D scene editing and UI for 3D object interactions.
- Low-level.
  - NSResponder and related input handlers are extended to accommodate 3D input events.
  - New API/communication system for interacting with the rendering engine (OpenGL, Metal, Vulkan, etc.).
  - Hierarchical scene representation for managing 3D elements.

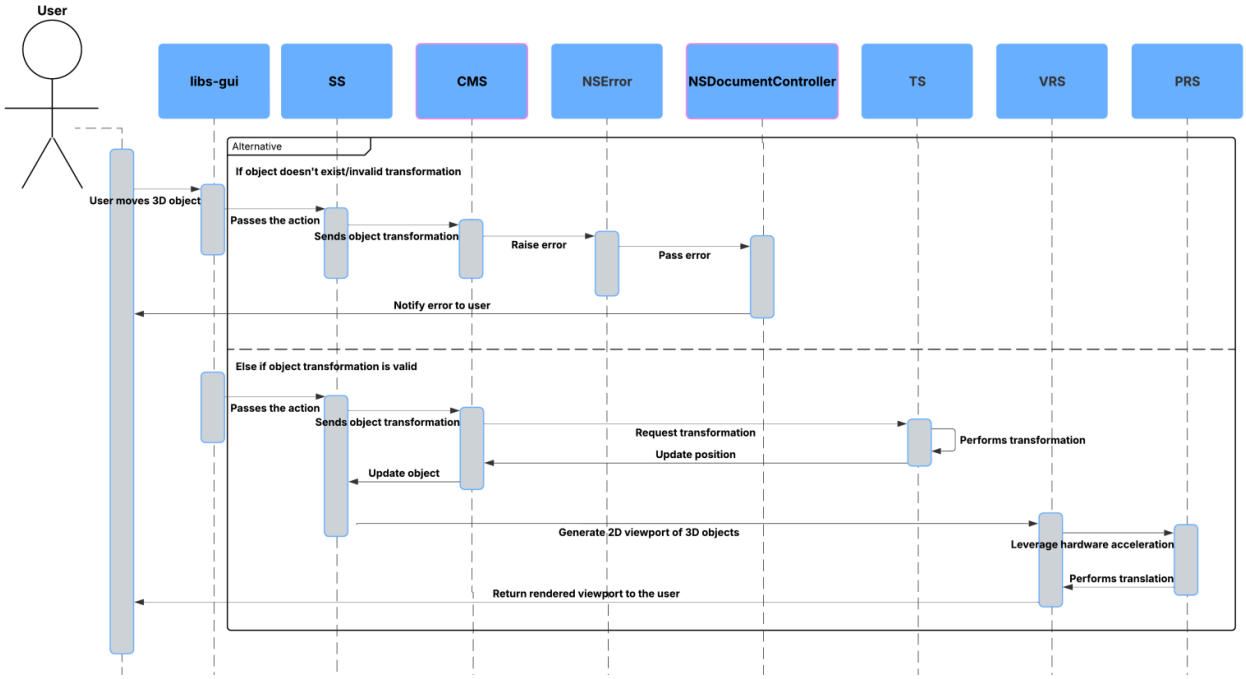
2.5 Use Case Sequence Diagrams

Use case 1: User creates a 3D object



This use case follows a user creating a new 3D object in Gorm. libs-gui, upon receiving the user input, will request scene services to create and store a 3D object. Scene services will pass the request core model services to retrieve and initialize the 3D object's information. Scene services then will request virtual rendering services for the viewport of the object and utilize hardware acceleration with physical rendering services performing platform-specific translation, it displays the viewport back to libs-gui.

Use case 2: Applying transformation to 3D object



The following diagram follows a use case of applying transforming to a 3D object. libs-gui, upon receiving the user input, will pass the input, which will be validated by core model services to see if the object exists and the transformation is valid; if not, NSError (in libs-base) is raised. If the transformation is valid, the core model services requests the transformation to transformer services and updates the object information. As usual, VRS and PRS are called to display the object to the user.

Legend:

3. Non-Functional Requirements (NFRs)

To ensure the successful integration of 3D-HAR into the GNUstep framework, we apply the Software Architecture Analysis Method (SAAM) to assess how the enhancement affects the key quality attributes. Based on the proposed enhancement, we will focus on some key

non-functional requirements including performance, portability, reusability, modifiability, maintainability, usability, and availability.

3.1 Key Quality Attributes (NFRs) for each Stakeholder

1. Key NFRs for GNUstep Project Community
   - **Modifiability**: The 3D-HAR system must allow easy updates and improvements without disrupting the core functionalities of GNUstep.
   - **Maintainability**: The system should be well-documented, structured, and easy to debug to long-term support and sustainability.
   - **Reusability**: The 3D rendering components should be modular and reusable across different parts of the GNUstep ecosystem to maximize development efficiency.
2. Key NFRs for Enhancement Contributors
   - **Modifiability**: The architecture should follow a modular design to allow future enhancements with minimum changes.
   - **Maintainability**: The framework must be well-structured and documented to facilitate debugging, testing, and continuous improvement.
   - **Portability:** The 3D-HAR framework must be designed to run on multiple operating systems without requiring major modifications. It also should abstract platform-specific graphics APIs, such as OpenGL and Metal, to maintain compatibility across different hardware environments [3].
3. Key NFRs for GNUstep application developers
   - **Performance**: The system must efficiently utilize hardware acceleration to provide smooth rendering and responsiveness.
   - **Portability**: The rendering system must support multiple GPU architectures, ensuring optimized performance across various devices.
   - **Reusability**: The rendering components should be designed as independent modules so that they can be integrated into various applications without requiring significant modifications.
4. Key NFRs for GNUstep application users
   - **Performance**: The rendering must be fast, smooth, and responsive, ensuring a seamless user experience.
   - **Usability**: The system must offer an intuitive and user-friendly interface for interacting with 3D objects.
   - **Availability**: The enhancement should be stable and reliable, with 99% annual durability.

3.2 Impact of the Enhancement and Potential Risks

The 3D-HAR enhancement introduces significant improvements to GNUstep's graphical capabilities while also presenting various challenges. We will discuss the impact of the enhancement with respect to the key quality attributes and outline the potential risks associated with the changes.

Impact on GNUstep

The introduction of new 3D classes, rendering services, and APIs increase the complexity of the codebase, which it will be important to maintain proper documentation, modular architecture, and clear API designs to ensure long-term maintainability. From a performance point of view, hardware acceleration will significantly enhance rendering speed. However, it introduces additional hardware dependency that would require careful adaptation as it may create performance inconsistencies across different platforms. GNUstep would need to advance their memory management strategies to be able to handle large 3D models and high-resolution textures without degrading system performance. To implement the enhancement, GNUstep developers need to ensure consistent rendering results across different OS environments and GPU drives, which requires extensive and expensive compatibility testing.

Potential Risks

The security risks associated with the 3D-HAR enhancement arise from the increased reliance on GPU-based rendering, which can introduce potential attack vectors such as buffer overflow exploits, memory leak, and unauthorized memory access. As highlighted in the research paper "Protected Graphics Processing Units: A Survey of Secure 3D Graphics Rendering" from Stanford University, modern GPUs are susceptible to side-channel attacks, privilege escalation, and unauthorized access to sensitive data stored in graphics memory. These risks arise due to the lack of strong memory isolation mechanisms and the shared execution environment between GPU and CPU processes [4]. To mitigate these risks, the implementation of 3D-HAR must incorporate secure memory management practices.

Performance risks are also a major concern. Some legacy systems may not have sufficient GPU support, potentially causing degraded performance or rendering failures. It is a challenge to balance CPU-GPU workload and poor optimization may lead to bottlenecks and slow rendering times. In terms of maintainability, the expansion of the codebase due to the enchantment could increase technical debt if it is not well-structured and monitored.

4. Testing Plan

For the seven main NFRs discussed in section 3, we will highlight some possible concrete tasks that could help 1 to test the desired quality requirements. These tasks have to be objective, the result must not change depending on who is testing the system.

Maintainability: The idea of maintainability can be estimated with a "maintainability index", which defines a formula that combines several other code metrics. The three metrics that are most commonly combined to make one unified score are as follows. Halstead volume is a calculation of the "size" of a fragment of code, calculated based on counts of operations in the code. Lines of code is just a metric of how many code statements are introduced, however, this metric usually includes white space and brackets that enhance ease of understanding. Finally,

cyclomatic complexity measures the number of paths that execution can take through a module of code.

These three metrics are typically combined with particular coefficients and result in values between 0 and 100 when properly condensed into a formula. We can then define a concrete requirement, such as: “The system must have an average maintainability index of 90”. This would provide an objective way of measuring the maintainability of the software.

Modifiability: Modifiability is a difficult idea to define objectively, however, maintainability indices, as discussed above, also describe ease of modification, so we can group these two NFRs into one test. We can also utilize tests for the next NFR to satisfy both this and the next NFR as well.

Reusability: Reusability is another NFR that cannot be tested as is and must be analyzed through sub-criteria. In particular, we will focus on Modularity. Modularity can be assessed using unit testing and integration testing. Unit tests are code that is run against single modules /functions/classes, or “units”, and ensure that said unit functions as designed. As a concrete measure, we can specify a “coverage” requirement, which states that a certain percentage of the source code has at least one accompanying unit test to ensure it is functioning correctly.

Integration testing is similar to unit testing, except for the fact that it tests the functioning of multiple units working together. So you could test whether a GUI element functions correctly with the functionality defined in libs-back and renders with the functionality defined in libs-gui.

All in all, we could state that “Unit testing must be fully passed with at least 90% test coverage” as a concrete task to test these attributes.

Usability: Usability is very difficult to quantify objectively, so the best way that we have to test it in a concrete way, is to do user testing. This is where you let many users test the software to ensure that they are easily able to perform intended tasks with as little issue as possible, and collate their experiences to understand whether the software is user-friendly or not.

Availability: Availability is a bit more difficult to quantify for GNUstep then most other applications where availability is tested, as GNUstep is an open-source repository where in theory stable releases should always be available, and there is no central server to crash or have issues. The main way to test for availability is to just make sure that all tests pass before release, and record if and when a major issue is reported by users, and how fast the mean developer response is. The hope is that this response is quick and efficient to allow users to be able to use the programs that they need as soon as possible.

Portability: Portability can be tested, again with a kind of integration testing, where tests are run on different kinds of hardware/OS/windowing systems, such that proper functionality can be assured regardless of the environment the end-user is using. In the case of GNUstep, we could define that “For all specified operating systems and hardware architectures” Which would be specified somewhere in the testing requirements, “Integration testing should pass successfully”. This test would ensure that the proper functionality is maintained across multiple different environments, ensuring portability is maintained.

Performance: The test for performance is easy to define concretely. Performance metrics can be defined, depending on some minimum hardware requirements. As an example, “A minimum of 60 FPS must be achieved while displaying 1 million triangles on screen, on a device with the defined minimum specifications”. These minimum specifications can be defined in terms of the processor, graphics processing unit, and more.

Finally, Unit and Integration Testing can be used once again to ensure that the 3D-HAR enhancement does not negatively affect the functionality of the rest of the library by making sure that the tests defined before the addition of the 3D-HAR enhancement still pass. While also making extra tests that ensure that the 3D-HAR enhancement is able to integrate properly with the modules that have been defined previously.

5. Architectural Enhancement Realization Comparison

5.1 Layered Architectural Style

In the current state of GNUstep, there are various dependencies and functional partitions, such as libs-base, libs-gui, and libs-back, that could benefit from the implementation of a layered architecture before introducing the proposed enhancement, 3D-HAR. A layered architecture style would help separate logical components, allowing for clear and independent layers. This would allow each component to be independently developed, tested, and optimized while avoiding the risk of manipulating other layers. This architectural style would also be compatible with the proposed architecture for its functional partitions (Section 1.4). Not only would it allow for the separation of 3D component concerns, but also for modular and scalable integration with the 3D features. Hardware is constantly evolving, and the physical rendering services layer found in the 3D-HAR partitions will be required to evolve too. By adapting a layered architecture style, layers that require updates or new 3D features can be upgraded without running the risk of modifying other layers.

5.1.1 NFR Impact of Layered Architectural Style

- **Modifiability:** The layered architectural style will reduce the need for interdependencies by organizing the functional partitions into layers. This implementation will allow for isolated modifications to be made to specific layers. Updates will become easier since

each layer will have specific responsibilities. This allows for future enhancements on hardware or 3D features to be modified on specific layers.

- **Maintainability:** Each layer has a focused functionality of the system, for example, the UI or rendering. This can simplify debugging, allowing for each layer to be tested and debugged independently of the others. Layers can be updated and fixed independently, leading to faster maintenance.
- **Reusability:** With clear and defined layers, components of the functionality can be reused independently. With layers designed to abstract platform-specific details, we can reuse the code across different platforms. As new platforms emerge, we can reuse code to integrate new platform-specific layers that can be added, swapped, or removed.

5.1.2 Stakeholder Impact of Layered Architectural Style

1. **GNUstep Project Community:** A layered architecture provides the community with clearer transparency and decision-making. With layers focusing on specific portions of the project, it becomes easier to establish feedback loops and make decisions to improve a certain layer.
2. **Enhancement Contributors:** Separating the concerns in our layered architecture makes it easier for the contributors to understand the project. The style allows for easier collaboration, where contributors can work on separate sections of the project without overlapping or overwriting other sections.
3. **GNUstep Application Developers:** Since layers are well defined, developers can focus on their specific application logic without the requirement of knowledge on the inner workings of lower level services. Additionally, when new features are added, developers will not need to rework their entire application, only their specified layer.
4. **GNUstep Application Users:** With a layered architecture, changes made to the underlying system will not affect the UI, resulting in stable and predictable behaviour across applications. Since the UI and the core logic are separated, the rendering and hardware-specific components have better performance.

5.2 Event-Driven Style

GNUstep incorporates a large portion of the event-driven architectural style through its use of UI updates and file system events; however, this can also extend to 3D-HAR. Event-driven architecture allows for real-time updates while ensuring non-blocking interactions, which helps to make the applications run smoothly for users. This architecture approach ensures that GNUstep components only react when they need to, to reduce processing. 3D-HAR can also adapt this architecture through asynchronous processing of 3D graphics and objects. Instead of continuously rendering, this architecture approach allows for rendering only when triggered by an event to reduce CPU and GPU load to reduce unnecessary overhead. This architecture style also allows for modular and extensible 3D components. This approach lets developers extend the functionality of GNUstep without modifying the core rendering engine.

5.2.1 NFR Impact of Event-Driven Style

- **Modifiability:** Since our components interact through events instead of dependencies, modifying one component does not require a change to the other components. Additionally, event-driven object behaviours will be more flexible, allowing for adjustments without the modification of the core rendering pipeline.
- **Maintainability:** If components interact through event triggers, then it becomes easier to track the flow of events instead of dealing with complicated dependencies. As time goes on and GNUstep expands, it will become easier to maintain an event-driven system since developers just need to update specific event handlers.
- **Reusability:** Common 3D operations such as object transformation or input handling can be made with independent event listeners that can be reused across different GNUstep applications. Developers can reuse the event-driven components.

5.2.2 Stakeholder Impact of Event-Driven Style

1. **GNUstep Project Community:** Event-driven architecture provides a well-defined event flow, making the project easy to understand. There is also a faster adoption of new features for the community, allowing new enhancements incrementally by adding new events instead of refactoring the whole system.
2. **Enhancement Contributors:** Contributors can work on isolated event-driven modules without worrying about breaking other parts of the system. Since we don't need to modify existing components, we can extend the functionality without any deep integration challenges from other components or functionalities.
3. **GNUstep Application Developers:** The applications are designed to only react when necessary, which reduces the amount of unnecessary computations. Developers can also focus solely on handling specific events instead of dealing with complex dependencies**.**
4. **GNUstep Application Users:** Since event-driven architecture reduces the processing overload, applications will feel faster and more interactive, where users will benefit from real-time updates. By processing only when necessary, we can avoid lag, leading to better performance

5.3 Preferred Realization of Proposed Enhancement

Layered Style Benefits	Event-Driven Style Benefits
• Clear and independent layers separate logic • Independent component development, testing, and debugging • Compatible with proposed function partities • Modular layers are easily scalable • Easier updates to components	• Responsive real-time updates • Reactive components reduce processing requirements • Modular 3D components • Independent component development, testing, and debugging • Easier to maintain and learn codebase through triggers

• Reuse of sections of the codebase • More transparency and better understanding of the project • Errors made will be contained in the section with bugs • The separation of logic and rendering allows for optimized performance	• Reusable components • Increased performance through event-driven processing

We believe that the best approach to realizing our proposed enhancement is the layered architectural style. This style would provide the best architecture for all aspects of our NFRs and stakeholders. The independence of each layer allows for a clear separation of logical components. This makes almost all aspects of the development cycle much easier. Maintaining and updating the codebase would be much easier with this style, as 3D rendering will continue to evolve in the coming years and constant updates will be needed. Being able to modify or reuse sections of the codebase will be an important aspect of updating and maintaining the codebase, which a layered style would be best at. Furthermore, with layered styles and clear delineation between different components, it makes it easier for the community to know where improvements need to be made and improves the overall understanding of the project. The contributors and developers will be able to focus on their own sections without needing to understand the whole project or even other components within a certain architecture. And the users that end up creating these 3D renders in GNUstep will also be shielded by the layered style when updates occur, as the rest of GNUstep will be in their own layers. This means that if bugs or errors occur with the implementation of this enhancement, it will have minimal impact on the original function of GNUstep.

Event-driven architecture does provide similar upsides to layered architecture, as it's already in use with some of the components within GNUstep. Its real-time updates and asynchronous processing would help create a more responsive application for users. However, that's where most of its benefits end. Many of the upsides that event-driven architecture has are also present within the layered style. The main upside for stakeholders is that it is more responsive, with less compute time, and incremental enhancements will be easier as new 3D features become available.

Compared to layered, which has the same upside as understandability, the move to event-driven architecture for faster compute time does not work. Although having a faster compute time would help on some slower machines or with complex 3D work that some may want, that upside is more relevant to the fringe of users. Most of the 3D work will be simpler and run on modern machines that can handle extra computations. This is not worth the trade-off compared to layered. Extra optimization can be made with the layered style as the separation of logic and

rendering can optimize performance and compute time. Overall, the stakeholders, which include the GNUsteps development community and users, would have a more refined experience with a layered style than event-driven for the development of our enhancement. Therefore, layered architecture would be better overall for our enhancement.

6. Conclusion

With modern applications, 3D support is a basic and core feature. Many GUIs of applications either utilize 3D graphics to enhance the aesthetics of their application or use it as a core functionality, like with CAD software, animation, and video games. Being able to provide that functionality is crucial for any software framework that deals with creating GUIs in the modern age. Therefore, we believe that adding 3D rendering support to GNUstep is a crucial enhancement for the framework and would help advance its user base and community. A previous attempt with 3DKit was made, but a serious modernized attempt should be made to create this feature. The project would be split into five main components, each handling a specific service for the creation of 3D visuals within GNUstep. It can easily be mapped to the already existing part of the GNUstep codebase, and new files/directories only need to be created in already existing directories. This allows for ease of creation for the enhancement and its tools. The community would be the most important aspect of the creation of such a tool, therefore, we have identified four segments of the community as key stakeholders who hold an interest in specific sections of the enhancement and have analyzed their major NFRs. An impact report was made to show any increased complexities, risks, or improvements to GNUstep, with its main takeaway being the increased complexity in the codebase and security vulnerabilities. We have also analyzed the concrete tasks that will need to take place to help test the desired quality of requirements. Finally, to realize the enhancement, we proposed two architectural styles and assessed their impact on the stakeholders and NFRs. The decision was made to go ahead with the layered architectural style as it has the most potential for maintainability, performance boosts, and separation of components, which benefits all stakeholders.

Overall, we have learned much as a group while researching GNUstep, not only for this report but all previous reports. It takes the whole team to research, develop, and test ideas for this project, just like it does with GNUstep developers. This process has taught us many valuable lessons and techniques about architectural analysis that we will carry forward into our careers.

References

[1] GNUstep's 3DKit Wiki page: https://mediawiki.gnustep.org/index.php/3DKit
[2] List of 3D graphics software: https://en.wikipedia.org/wiki/List_of_3D_computer_graphics_software
[3] Joseak. (2023). *Understanding Graphics APIs*. Medium. https://medium.com/@joseak.tech/understanding-graphics-apis-0ae8a5a7cbdf.

[4] Govindaraju, N. K., Buck, I., Gupta, A., & Mark, W. R. (2006). *Protected Graphics Processing Units: A Survey of Secure 3D Graphics Rendering*. Stanford University. https://graphics.stanford.edu/papers/protected/protected.pdf.