SWE621
Project Reflection

Group:

Venkata Krishna Chaitanya Chirravuri **(G01336659)**

Sai Prashanth Reddy Kethiri **(G01322333)**

Reference System – VisTrails

Additional Systems – Visualization Tool Kit (VTK), matplotlib

1. Very briefly state the purpose of your 3 systems. What is the domain of these systems, and what are the key requirements each seek to meet?

a. **VisTrails**

(Purpose) VisTrails is a scientific workflow and provenance management system created to aid in the process of scientific discovery. VisTrails features a user-centered architecture, extensive provenance infrastructure, and exceptional support for data analysis and visualization. VisTrails is used in the **domains** of computer graphics, human science, medicine, and Earth science research to manage scientific process and provenance. Researchers have utilized VisTrails to assist them in developing workflows that specify the processes in the scientific method and serve as a basis for repeatability, transparency, and software reuse. **Key Requirements** met by VisTrails are,

- **Flexible Provenance Architecture:** VisTrails logs changes made to processes in a transparent manner, including all the steps taken during the exploration.
- **Querying and Re-using History:** The provenance information is stored in a structured way which can be queried and reused for later purposes.
- **Support for collaborative exploration:** The system can be configured with a database backend that can be used as a shared repository.
- **Extensibility:** VisTrails provides a very simple plugin functionality that can be used to dynamically add packages and libraries.
- **Scalable Derivation of Data Products and Parameter Exploration:** VisTrails supports a series of operations for the simultaneous generation

of multiple data products, including an interface that allows you to specify sets of values for different parameters in a workflow.

- **Task Creation by Analogy:** Analogies are supported as first-class operations to guide semi-automated changes to multiple workflows, without requiring you to directly manipulate or edit the workflow specifications.

b. VTK

(Purpose) The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, modeling, image processing, volume rendering, scientific visualization, and 2D plotting. It supports a wide variety of visualization algorithms and advanced modeling techniques, and it takes advantage of both threaded and distributed memory parallel processing for speed and scalability, respectively. VTK is used in different **domains** like Insight Segmentation and Registration Toolkit, medical research, visualization, medical image analysis, preclinical animal studies, surgical planning and guidance, medical robot control, population studies, Weather Research and Forecasting. **Key Requirements** met by this system are,

- Integrates seamlessly with other windowing systems.
- **Data Interaction:** Supports a variety of interaction styles. In VTK, 3D widgets, interactors, and interfaces to 2D widget libraries like Qt enable you to add comprehensive user interaction to your programs.
- Includes an extensive set of 3D widgets including point, line, plane, implicit plane, box, sphere, scalar bar, etc.
- **Filters:** VTK applications manipulate data with filters. Each filter inspects the data it receives and produces derived data.
- **Graphics System:** VTK adds a rendering abstraction layer over the underlying graphics library
- **Data Model:** VTK's core data model has the ability to represent almost any real-world problem related to physical science. The fundamental data structures are particularly well-suited to medical imaging and engineering work that involves finite difference and finite element solutions.
- **2D Plots and Charts:** VTK has a full set of 2D plot and chart types for tabular data. VTK's picking and selection capabilities help you interactively query your data.
- **Parallel Processing:** VTK has excellent support for scalable distributed-memory parallel processing under MPI.

c. **matplotlib**

(Purpose) Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. **Domains** of matplotlib system include, Data Science, Artificial Intelligence, Business Analytics, Data Analytics. **Key Requirements** met by the system are,

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

2. Drawing on your past HW assignments, briefly summarize 3 differences and 3 similarities between each of the project's design decisions, abstractions, architectural styles, design patterns, and programming styles.

Similarities

I. First similarity that we found in all the three systems is an **architectural decision** to cache the results.
II. Another similarity that we observed among all the three systems was Querying, which is considered as an **abstraction** in all the three systems.
III. Third similarity that we found between all the three systems was the implementation part of GUI where all the systems followed the same **programming style** i.e, Hollywood / Inversion of Control.

Differences

I. The first difference that was found between our main system and the other two reference systems was in the **architectural decision** about the provenance infrastructure i.e, about the provenance infrastructure that refers to the source or origin of the objects or data.
II. **Architecture Styles** of the systems was the second difference between that we observed, where VisTrails followed flexible provenance architecture along with layered architecture and pipeline architecture, VTK followed pipeline architecture and matplotlib followed layered architecture.

III. Third difference that we observed in all the three systems was how the system was designed to make users to backtrack their changes while working on a workflow. This comes under a difference in **design pattern.** We observed that VisTrails made use of memento behavioral design pattern to implement this, whereas VTK used factory method to solve their problem, and matplotlib particularly did not follow any sort of design pattern. Instead they made use of cache memory to achieve their goal.

3. Considering all of these together, what forces do you think might have led to these differences and similarities? To what extent were these differences and similarities due to differences in requirements or differences in approaches to meeting the same requirements?

Similarity in architectural decision

There is no difference in the requirement of all the systems, their **main goal** was only to make the system more efficient by simply retrieving results of a task that is already computed instead of recomputing the same task again. But we can see it is **implemented differently** in each system. VisTrails makes use of VisTrails Cache Manager (VCM), VTK uses classvtkTemporalDataSetCache for cache management, whereas matplotlib uses a widget to store the results in cache. Both VisTrails and VTK have an inbuilt feature for cache management and matplotlib itself being a package, uses a third-party widget for implementing the same feature. The **difference** in implementation is due to the language they are coded in and the architecture of each system.

Similarity in abstraction

Querying in each system is implemented in different ways. VisTrails uses two methods to query the data, first one being Query by Example interface which allows you to build query workflows and search for those with similar structures and parameters and second one is a textual interface with straightforward syntax. VTK takes a SQL query as an input for vtkSQLQuery class. In Matplotlib, querying mainly focuses on the creation, updating and retrieving of the object parameters and minute details of the graphs and subplots. The **difference** of implementation in each system is because, VisTrails consists of a user interface to query the data making it easy for the users to query, VTK makes use of SQL database and is coded in C++ language making it to implement using vtkSQLQuery class. Whereas in matplotlib, querying is implemented at a lower level with only to retrieve previous plots and extract subplots from the main plot.

Similarity in Programming Style

All the systems follow this programming style because all of them provide an interface to interact with the users but there is a **difference** in implementation because of the features they offer, fields they are supposed to work in, languages they are coded in and the requirements. Like VisTrails and VTK used mostly in medical and scientific fields whereas matplotlib is mostly used in the data analytics field.

Difference in architectural decision

(Reason) VisTrails provides a comprehensive provenance infrastructure that maintains detailed history information about the steps followed and data derived in the course of an exploratory task.

The **reason** for having standalone rendering infrastructure instead of provenance infrastructure in other systems is, the entire point is to be lightweight which includes the ability to output 1D, 2D, and 3D plots, with basic axes and annotations, including text. The ability to output polygons/line segments as either vector primitives to these EPS files with clipping, or to use a raster renderer (GL) and output the pixels within the viewport while still using vector annotations. Another capability implemented was that multiple windows could render the same scene, and you could switch renderers without having to rebuild the scene.

Difference in architectural style

Reason for using flexible provenance architecture along with layered architecture and pipeline architecture in vistrails is (Provenance architecture style) it tracks run-time information about the execution of workflows (*e.g.,* who executed a module, on which machine, elapsed time *etc.*). (Layered architecture style) One of the key components of any system supporting provenance is the serialization and storage of data. For this they have added a db layer. (Pipeline architecture style) Workflow systems support the creation of pipelines (workflows) that combine multiple tools. As such, they enable the automation of repetitive tasks and result reproducibility.

VTK uses pipeline architecture **because** visualization data that would not normally fit into memory can be processed and can be run with a smaller memory footprint resulting in higher cache hits, and little or no swapping to disk. **Another reason** for using this architecture style is that it allows us to control the size of the data through the pipeline, and configure the algorithms.

(Reason) Matplotlib makes use of layered architecture in order to solve one of the core architectural tasks by implementing a framework for representing and manipulating the Figure that is segregated from the act of rendering the Figure to a user interface window or hardcopy. This style also brings a great flexibility to the developers to share the script with other developers in the artist layer and for non-programmatic users, the scripting layer automates the process of defining the FigureCanvas and artist instance, which makes it easy to use for quick exploratory analysis.

Difference in design pattern

The **reason** for choosing the memento behavioral design pattern in VisTrails is there was a requirement to make changes to a workflow and to keep track of all updated information on the objects involved in the operations. Keeping the logs of all the states, properties and the objects is a wiser decision to achieve complete provenance infrastructure.

VTK made use of factory method of creational design pattern to implement the same feature as in VisTrails **because** VTK provides an interface in superclass to create the objects but the changes to that object can only be made in the subclass.

Coming to matplotlib, there is no such feature as backward compatibility. The **reason** for this, if a user wants to make changes to a plot then they would simply make changes to the code at the programming level or simply change the parameters at the UI level. **Reason** for not implementing this is because matplotlib does not make use of flexible provenance infrastructure, where a user or developer can make changes to their already existing workflows.

4. Are there ideas used in one system which could be beneficially applied to another system? Either explain at least 3 examples of where this could be the case or clearly explain why the differences between the systems makes this impossible.

There are quite ideas and motives which are purely domain specific and have elevated the essence of using the particular systems.

- **One** is having a dedicated User Interface, VisTrails has made it more user friendly whereas matplotlib has to be imported as a library or it could be accessed with libraries like python GUI libraries on top of it requiring the user to have an ample amount of coding experience to use it. Similarly VTK though it can be coupled as a library and can also be accessed from a User Interface

giving little flexibility for the user has many backdraws for not having a dedicated interactive user interface as VisTrails. A dedicated interactive User Interface with dashboards for workflows, figures in case of matplotlib and VTK would benefit more and have been more user friendly as the domain of scientific computation is widely used by all domains.

- **Second** is VisTrails has a provenance infrastructure where all the workflows could be stored and can be retrieved at every step, having such features will certainly elevate the user experience and help them throughout using the system. Matplotlib and VTK certainly do not support this feature completely, as the rendering figures and their properties can be still retrieved in case of matplotlib and VTK pipelines can be retrieved as a whole as they are cached.
- **Third** being the architectural style, mainly because of the modularity and reusability. Matplotlib follows layered architecture and VisTrails follows layered architecture alongside pipeline architecture, whereas VTK follows only pipeline architecture. Following layered architecture provides the user or the developer enough flexibility to update/modify the functionalities easily in the layers itself. Following the pipeline architecture does provide the flexibility but still has some dependencies that have to be dealt with. Moreover following such architectures, systems are easy to maintain, the overhead costs are low to maintain and the testings are pretty simple as they are separated as components

5. Reflect on your experiences reverse engineering the design and architecture from these systems. What strategies did you use to understand the design and architecture? How well did these strategies work, and what sometimes made them hard to follow? What resources were most helpful in understanding the design and architecture of these systems? In what ways could these systems improve the documentation of their design and architecture to make it easier to understand?

(Experience) Initially, we were perplexed about choosing the main systems and their reference systems for the assignments and the project. We had changed our three systems after the initial proposal following extensive research along with inputs from professor and TA on the availability of the resources for all the additional systems.

The **strategy** we have followed is the following textbook for the VisTrails documentation to secure a basic outline of how the system is designed, later we have gone through the documentation, Github about their architectural implementation. With introduction to newer terms about the architectural designs, design patterns, abstraction and programming styles we have spent enormous time learning about the various details of how they are chosen, when they are chosen, how they are implemented and what would the consequences be? Later, we did a literature survey on how different systems

in the domain we choose are designed, developed as reference and analyzed how VisTrails, Visualization Tool Kit and Matplotlib are designed and analyzed. We did face difficulties with very few availability of resources about the Visualization tool kit .

Our most helpful **resources** for reverse engineering are reading the documentation alongside the github code base with literature surveys in the current market about the techniques. **(Improvement)** We felt including documenting changes , motives to changes , and their consequences explicitly in the documentation with the details in their github profile would make it more user friendly.

Resources

- **VisTrails Documentation -** https://www.vistrails.org//index.php/Main_Page
- **VisTrails AOSA Book -** http://aosabook.org/en/vistrails.html
- **VisTrails GitHub -** https://github.com/VisTrails/VisTrails
- **VTK Documentation -** https://vtk.org/Wiki/Main_Page
- **VTK AOSA Book -** http://aosabook.org/en/vtk.html
- **VTK GitHub -** https://github.com/Kitware/VTK
- **matplotlib Documentation -** http://aosabook.org/en/matplotlib.html
- **matplotlib AOSA Book -** https://matplotlib.org/3.5.3/index.html
- **matplotlib GitHub -** https://github.com/matplotlib/matplotlib
- **Design Patterns Reference -** https://refactoring.guru