01 - Visual Process Manager Final Report
Dan Bye
Dan Brill
James Draper

---

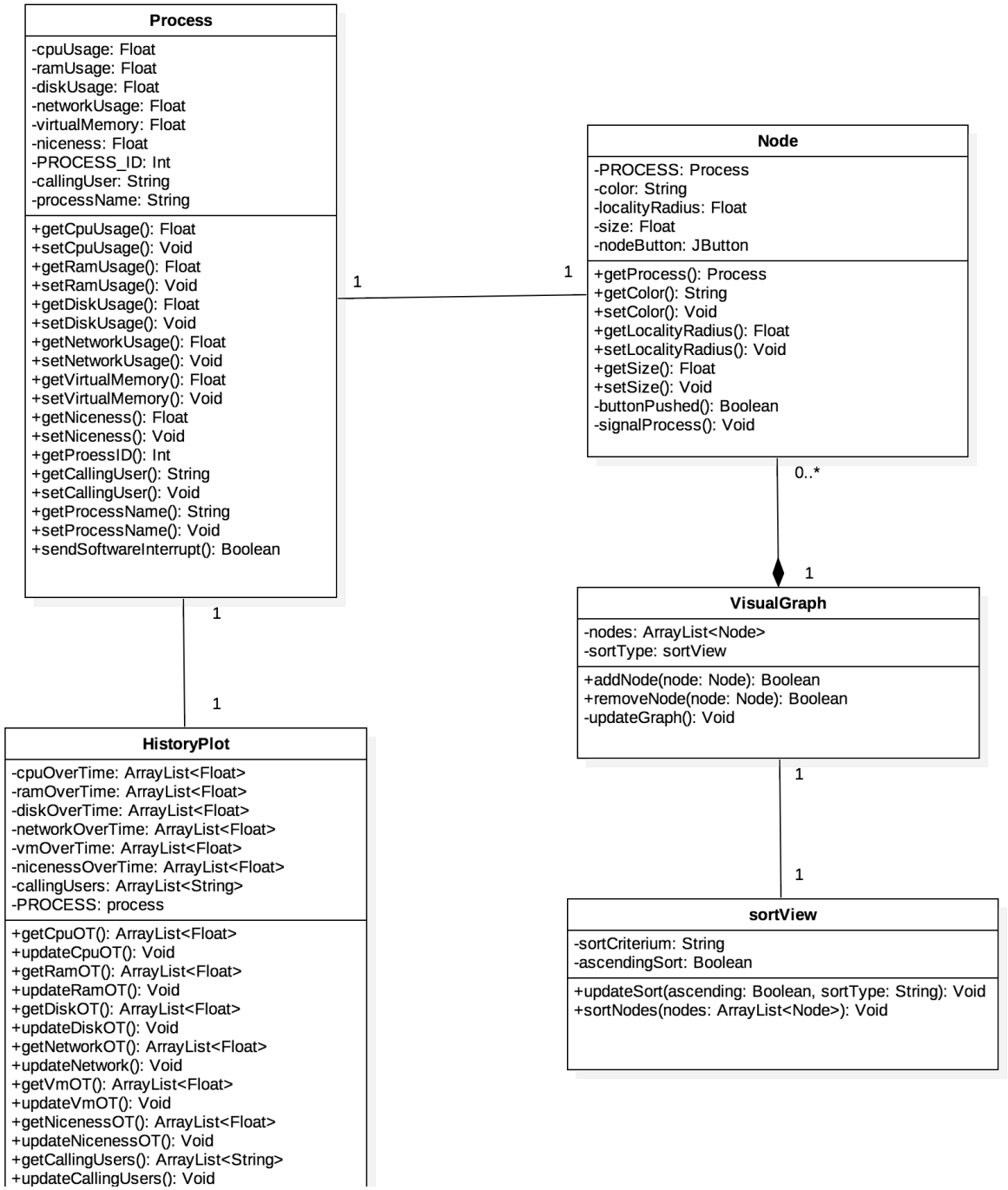
1. Features Implemented - Based on Part 2 Tables (Edits made to adhere to feedback)

ID	Requirement	Topic Area	Priority
US-01	I want to be able to review my processes visually	Visualization	Critical
US-02	I want to manage my processes by interacting with the visualization	Process Management	High
US-04	I want to be able to sort my processes by different metrics	Visualization	Medium
FR-01	Processes must be identified by PID	Process Management	Critical
FR-02	Processes must be presented visually	Visualization	High
FR-03	Visual process instances change appearance based on activity	Visualization	High
FR-04	Processes can be managed by sending signals	Management	High
FR-06	Process data can be stored in a database	Recording	Medium
NF-04	Color and representation of nodes should show process information	Visualization	Medium
NF-05	Data should be updated periodically	Performance	Low

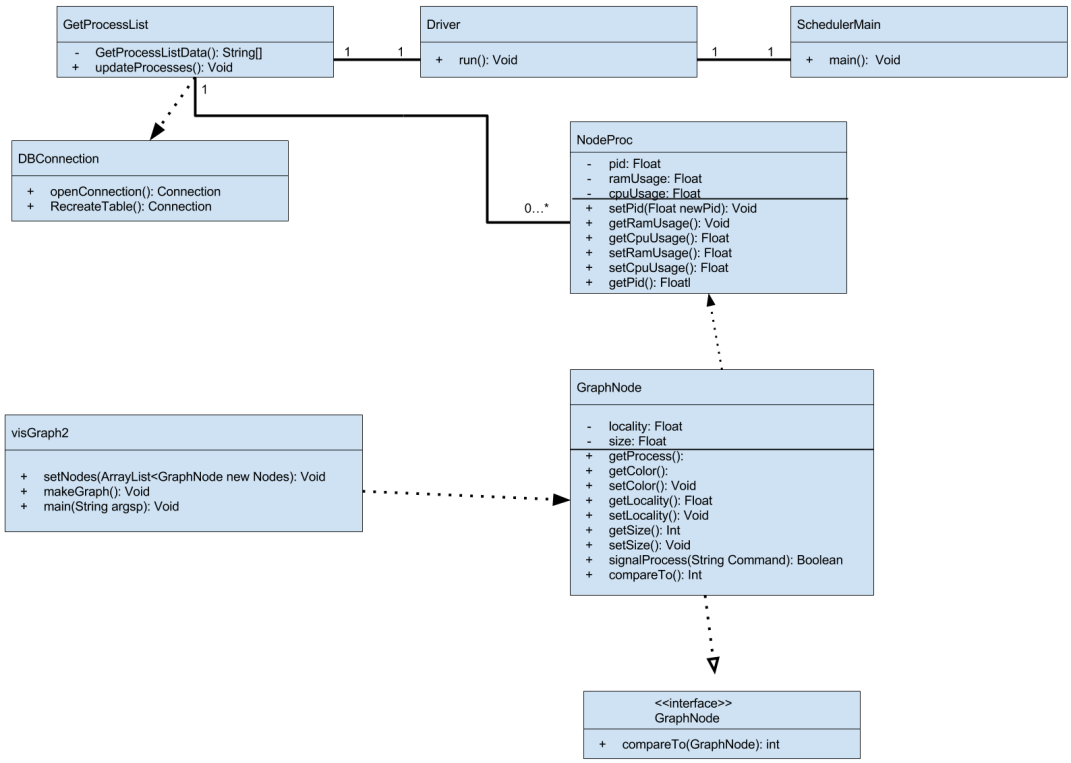
2. Features Not Implemented - Based On Part 2 Tables (Edits made to adhere to feedback)

ID	Requirement	Topic Area	Priority
US-03	I want to see a history of my usage	Recording	Medium
FR-05	Processes can be sorted by category/metric	Management	Medium
FR-07	Database data can be shown over time	Visualization	Medium
FR-08	Process manager should not be able to kill itself	Management	Low
NF-01	Visualization should be sleek and intuitive	Visualization	Low
NF-02	Process privilege between users should be maintained	Security	Medium

3. Class diagram comparison



Part 2 Class Diagram



Part 5 Class Diagram

Some fundamental ideas about the class diagram remain, however, most of the structural components changed significantly. This can be attributed to our team's limited experience with Java relative to other languages, as well as keeping the program intuitive for our purposes. There were certain architectural and data-structural nuances that didn't make themselves known as flawed until we actually began implementing everything. While we moved towards more design patterns as we got more comfortable, it was hard to do so initially without knowing more about how Java behaves. The initial planning and class diagram were still useful because it gave us various specific directions to pursue, however, the paths we took to get there were not as straightforward as we expected.

---

4. Design patterns

There were a few key areas where we recognized the possibility for implementing different design patterns. The most obvious of which was the flyweight and observer patterns. We could've used the flyweight pattern to make a lightweight class that represented every process running on the system or stored in the database. Because we are constantly destroying and instantiating these objects, sometimes hundreds of them at a time, it is a prime opportunity for implementing the flyweight pattern. We also recognized the possibility to use the observer pattern to monitor the state of the database and call the visualization class to regenerate the graph anytime the database changes. We also could've used the FactoryMethod design pattern in order to make our NodeProc objects automatically generate their associated GraphNode objects. Finally, while it isn't really all that worth mentioning our DBConnect follows the structure of a singleton as only one instance can exist at a time.

---

5. Things learned about the process of analysis and design.

Regardless of the changes between the initial planning process and the presentable product, the planning process helped the group maintain direction and divide the work as equally as possible. The design process is useful in this sense, but we overestimated the amount of work we could get done, given the time we could commit to the project. Creating and implementing the system was difficult, but allowed us to get used to Java and discover some of the language intricacies relative to the project. One thing that helped drive our project home was that despite changing architectural components the core values, detailed by use cases and requirements, never changed. This allowed us to refactor our software stack as much as we wanted to without compromising on the core ideas behind the project.