Fall 2023 SSW-345 Homework

by

Anthony Rizzuto

Stevens.edu

November 13, 2023

Fall 2023 SSW-345 Homework

Anthony Rizzuto
Stevens.edu

This document provides the requirements and design details of Fall 2023 SSW-345 Homework. The following table (Table 1) should be updated by authors whenever changes are made to the architecture design or new components are added.

Table 1: Document Update History

Date	**Updates**
01/30/2023	Initials: • A list of important updates to the document. • Added the introduction (Chapter 1).
09/14/2023	Overleaf Introduction: • Added a section in the introduction (Section 1). • Added a Team Chapter and a section about myself (Chapter 2).
09/21/2023	UML Class Modeling: • Added a chapter, UML Class Modeling (Chapter 4).
09/28/2023	Advanced Class Modeling: • Added a chapter, Advanced Class Modeling (Chapter 5).
10/12/2023	Software Execution Model: • Added a chapter, Software Execution Modeling (Chapter 6).

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

– Anthony Rizzuto

Hello

This manual contains all my completed homework assignments from this semester. It will be updated frequently.

Chapter 2

Team

– Anthony Rizzuto

2.1 About Me

Hello. My name is Anthony Rizzuto. I am a third year Software Engineer major and also am pursuing a minor in Computer Science at Stevens Institute of Technology in Hoboken, NJ. I do not have any prior experience in the field, however I have some personal projects I created. I worked with a couple programming languages, such as C++, Python, and Java. I also have experience in web development languages HTML and CSS.

Chapter 3

Git Homework

– Anthony Rizzuto

3.1 Learning Git

Below is a screenshot of me demonstrating that I have completed all the levels of: https://learngitbranching.js.org:

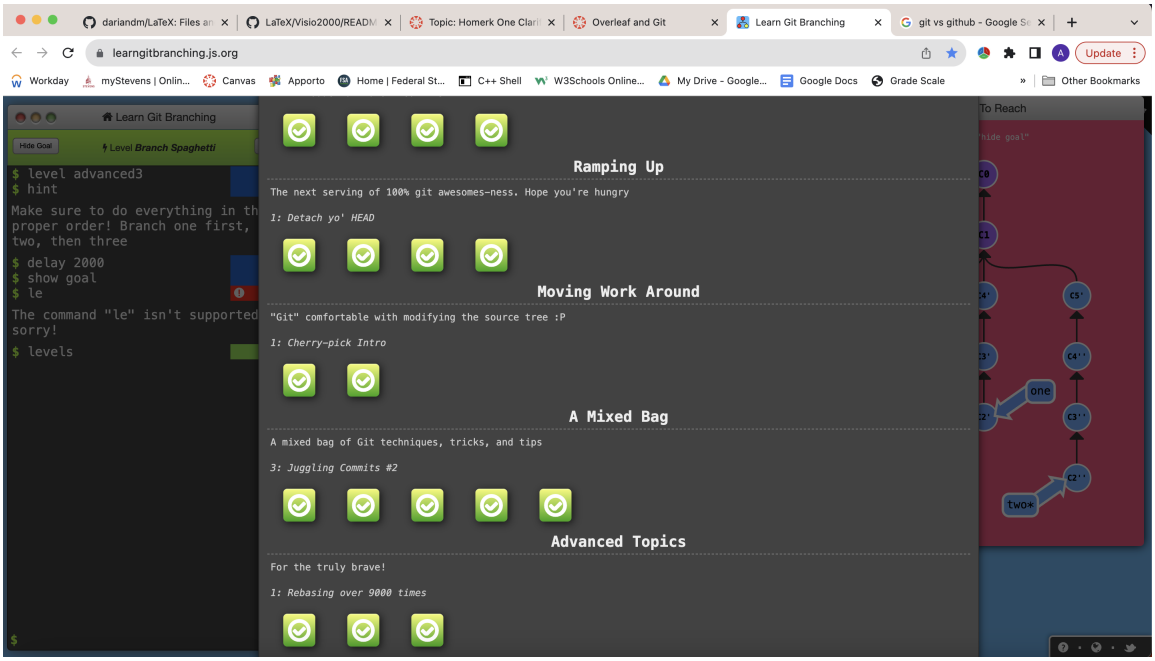


Figure 3.1: Screenshot of Level Completion

Chapter 4

UML Class Modeling

– Anthony Rizzuto

4.1 Exercise 2.1

Below is the class model I used to describe the sample undirected graph provided in the figure below:

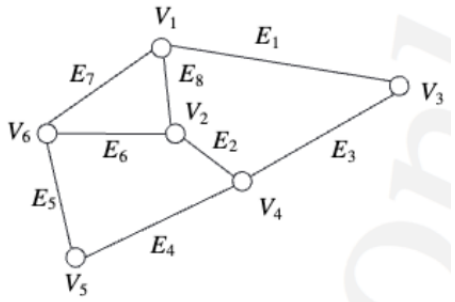


Figure 4.1: Figure of the proviced sample undirected graph.

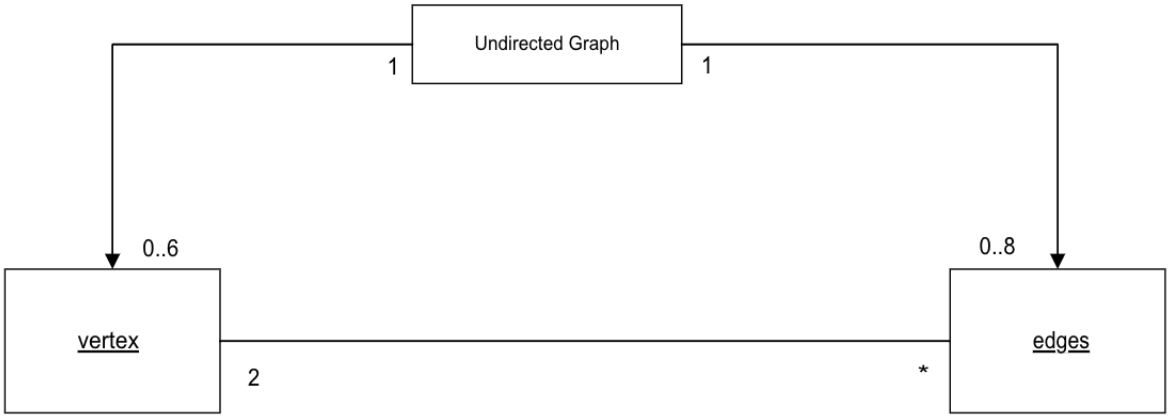


Figure 4.2: The class model to describe the sample undirected graph.

4.2 Exercise 2.2

Below is another class model I used to describe the sample directed graph provided below:

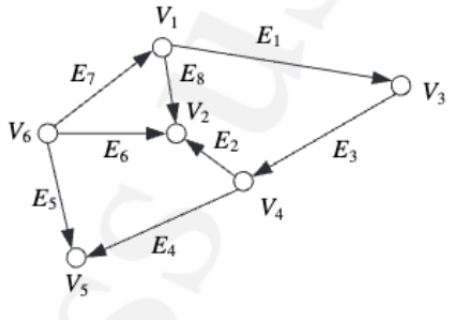


Figure 4.3: Figure of the proviced sample directed graph.

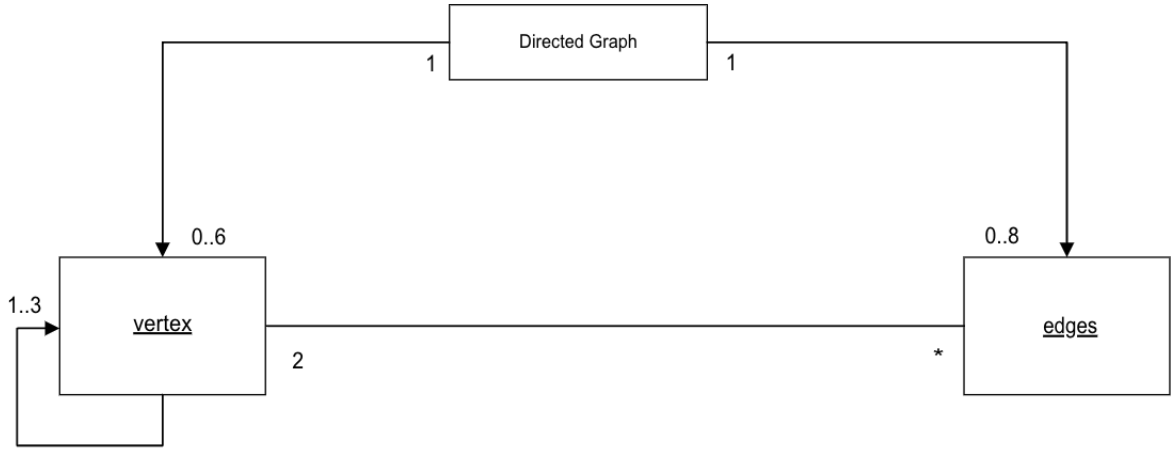


Figure 4.4: The class model to describe the sample directed graph.

4.3 Exercise 2.3

Below is all the associations of the window system provided in the assignment (Figure 5.5):

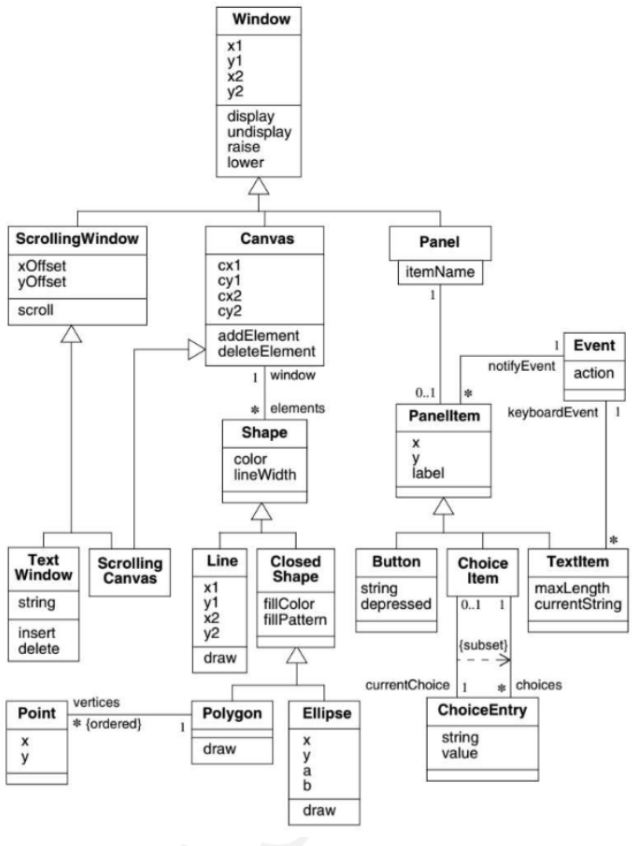


Figure 4.5: Window System.

Window: The window displays the contents of the system in the box depending on coordinates x1, y1, x2, y2. Display and undisplay allow the user to show and hide the window.

Scrolling Window: Scrolling window allows the user to see more contents that cannot fit on the window itself, and with xOffset and yOffset, the user can scroll through the canvas of the window.

Canvas: The canvas association is an object used to display graphics, such as text, images.

Panel: Panel is displayed above the canvas in the window and is used for controls inside the window.

Shape: Shape allows shapes to be inserted in the canvas.

PanelItem: Panel Item is where objects are displayed inside the Panel.

Event: An event is an action that notifies the windowing system when a user is using the system.

TextWindow: TextWindow displays text into the ScrollingWindow or ScrollingCanvas, which goes to Canvas.

ScrollingCanvas: ScrollingCanvas is associated with graphical objects.

ClosedShape: This allows any shape to be filled, thus explains fillColor and fillPattern.

Button: A button is a ClosedShape that when clicked, it starts an action.

ChoiceItem: ChoiceItem allows shapes to be selected by the user.

TextItem: This association allows the shapes to have text.

Point: A point is associated with polygon, which is associated with shape.

Polygon: Polygon is associated with ClosedShape, which is a shape on canvas.

Ellipse: Ellipse is a type of closed shape, which follows the same concept of polygon.

ChoiceEntry: This is a type of panel item that the user can select.

4.4 Exercise 2.4

Below is a description of every association of the credit card system provided (Figure 5.6):

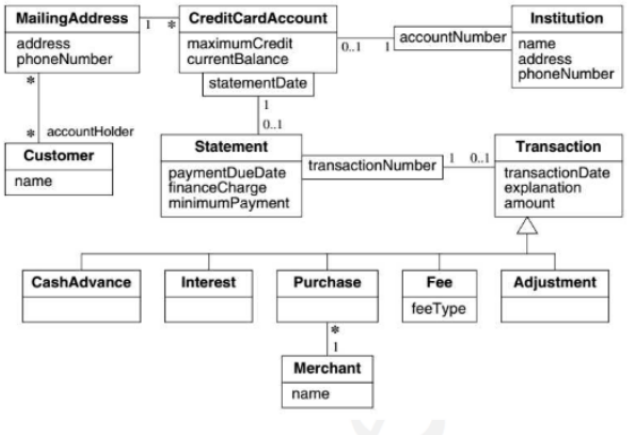


Figure 4.6: Credit card system.

Customer: The customer, or user, provides their name for an order.

MailingAddress: MailingAddress pulls the customers name and links it to their address and phoneNumber, which the customer provides.

CreditCardAccount: The customer provides their credit card information in CreditCardAccount, and this links it to their statement and sends the information to the card Institution as well.

Institution: The institution is the credit card company that the customer uses, it receives the customer's accountNumber and receives their name, address and phoneNumber.

Statement: The Statement receives the transaction information and sends it to the CreditCardAccount.

Transaction: The Transaction then creates all the transaction information (with a transactionNumber) as a Statement.

Merchant: The merchant totals the cost (CashAdvance, Interest, Purchase, Fee, and Adjustment) and creates a transaction.

Chapter 5

Advanced Class Modeling

– Anthony Rizzuto

5.1 Exercise 3.4

1. Below is the CRC card solution for the user story:
"John and Maria are two students at Stevens Institute of Technology, a local university. John gets to his classes by riding his bike and Maria gets to her class by putting on her shoes and walking to class."

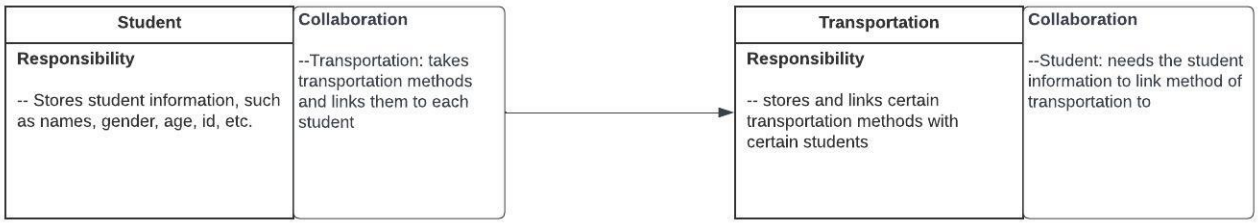


Figure 5.1: CRC card solution.

I split this scenario (or system) into two classes (or cards). One being the Student class, so John or Maria, and the other being the method of Transportation they take to get to school, bike riding or walking. Student stores the information of the student, like name/age/gender/id and transportation stores the method of transportation, bike-riding/walking. These cards collaborate with each other, as they need information from each other.

2. Below is a use case diagram, depicting the story above:

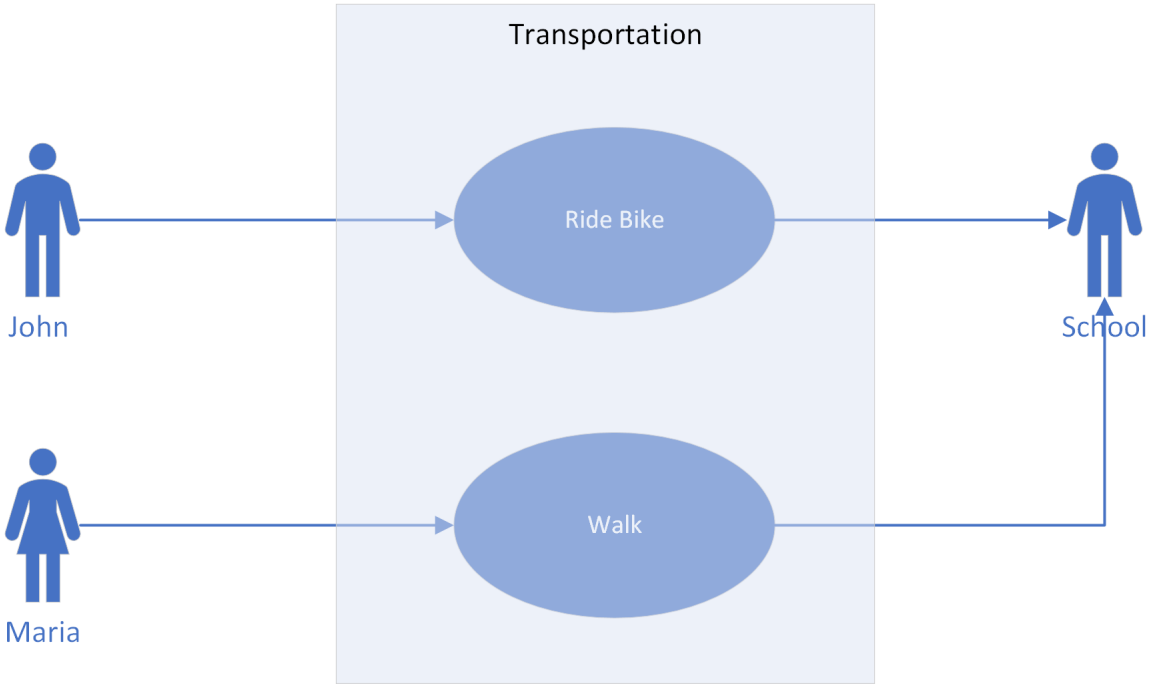


Figure 5.2: Use case diagram.

Here, John and Maria are actors. John is associated to the useCase of ”Ride Bike” as according to the user story. Maria is associated to the useCase of ”Walk” once again according to the user story. These two useCases are in the system boundary ”Transportation” and are associated to School, which is where John and Maria are going.

3. Here is an object diagram depicting the exact user story scenario:

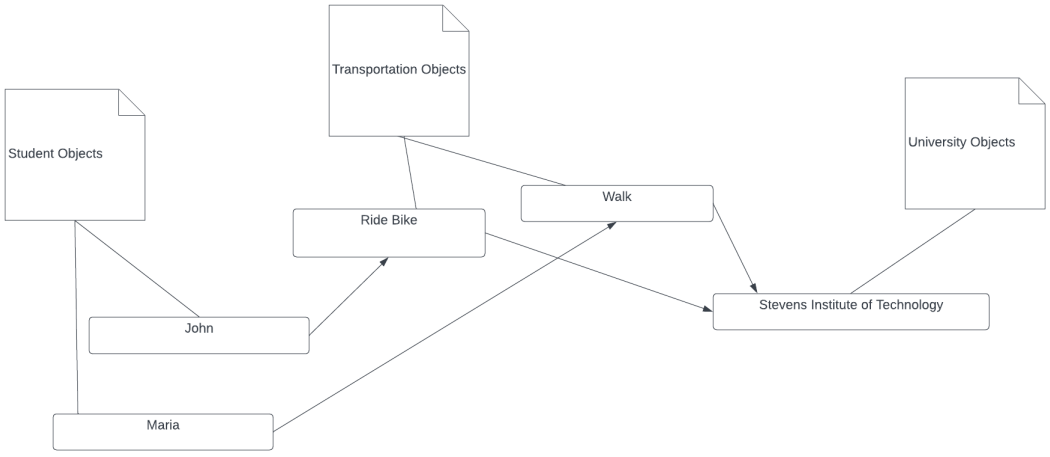


Figure 5.3: Object diagram.

In this object diagram, John and Maria are instances in the Student class, with their name being an attribute. The same goes for Ride Bike and Walk, they are instances of the Transportation class. I decided to create a new class, University, to make the diagram more accurate and straightforward. This would also make it easier to expand the universities in the future, expanding this user story system, if needed.

4. Here is the class diagram I formed from the object diagram:

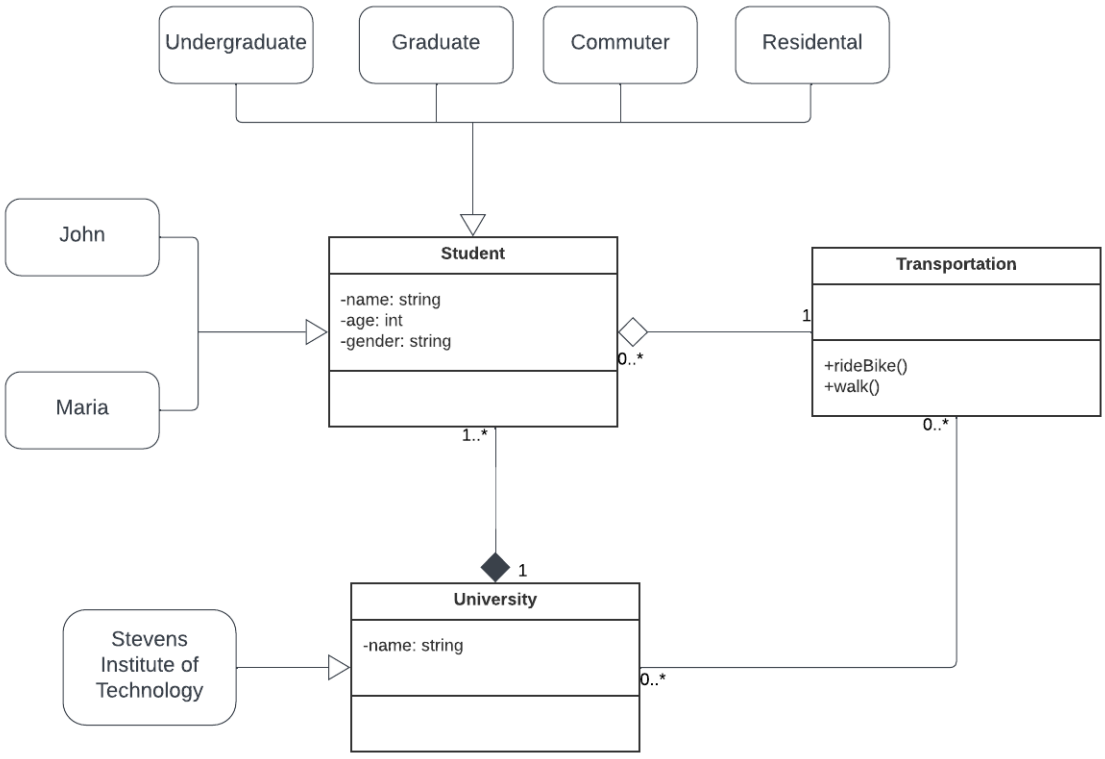


Figure 5.4: Class diagram.

In this class diagram, I successfully used aggregation, generalization, and composition. John and Maria, along with the types of students (Undergraduate, Graduate, Commuter, Residential) are all generalized into Student. Student and Transportation have an aggregation association. If Student gets deleted, Transportation can stand by its own, even if it may have no point of. On the other hand, if University were to be deleted, Student would also be deleted because you cannot have a student of a university if there are no universities that exist, so they have a composition association. In regards to the multiplicities, Student has only one way of transportation and Transportation has zero to many students. Transportation also has zero to many universities, as University has zero to many transportation methods. Student can only have one University and University can have one to many students, as you need students for a school and students can only be enrolled into one university at a time.

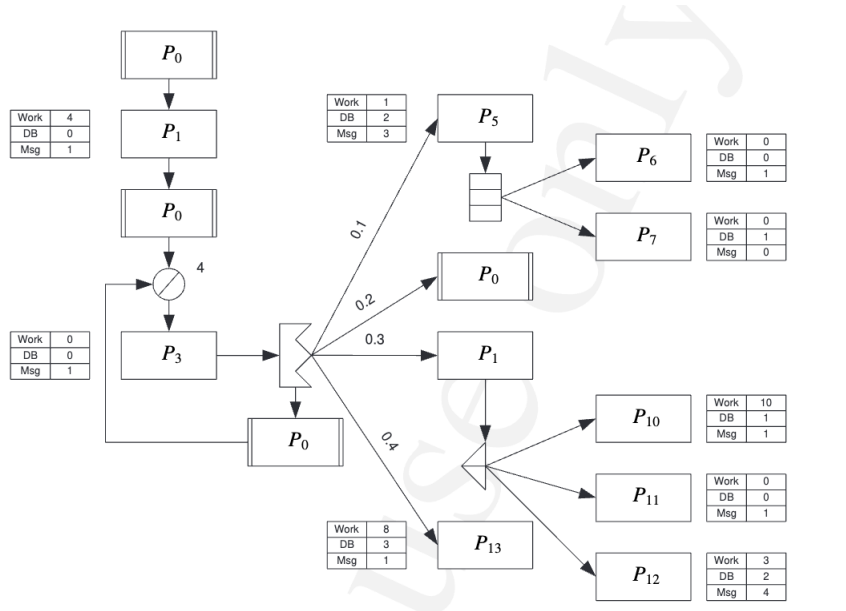
Chapter 6

Software Execution Model

– Anthony Rizzuto

6.1 Introduction

I was provided the software execution model (Figure 6.1) below and was prompted to calculate the best, worst, and average demand of the execution path.



	Device	CPU	Disk	Network
Hardware	Quantity	3	2	1
	Service Unit	KInstructions	I/O Operations	Messages
Software	Work	400	0	0
	DB	1500	150	2
	Msg	10	0	1
Time	Service Time (s)	0.0001	0.02	0.01

P_0

Processing Step	1x CPU (KInst)	1x Disk (I/O)	1x Net (Msg)
openPage()	100	2	100
getCriteria()	200	0	2
search()	200	10	2
closePage()	20	2	100
displayItem()	35	0	30
getItem()	220	0	10

Figure 6.1: Provided software execution model.

6.2 Analysis

To calculate the total demand time of each package, I did the following process:

First, I divided the process into the three groups CPU, Disk, and Network. I then took the demand of the Work, DB, and Msg, provided in the ”Software” chart and multiplied these values by their respective demands for each package. After calculating these demands, I took the sum of these demands, multiplied them by the device’s service time, and multiplied the product by the quantity of the device. This process was repeated for each device.

For example, P1 provided the following demands: Work = 4, DB = 0, and Msg = 1. I started with the hardware device ”CPU”. For the software, Work = 400*4; DB = 1500*0; Msg = 10*1. I added these products to arrive at a demand of 1610, which was multiplied by the provided service time of 0.0001 seconds, and I arrived at .161 seconds. There is a quantity of 3 CPUs, so I had to multiply the demand time of one CPU with P1 by 3 and got 0.483s. After repeating this process for ”Disk” and ”Network” and using their respective values, I arrived at a final answer of P1 = 0.493s.

This process was then repeated for each package, and instead of manually doing this over and over again, I plugged this model in Microsoft Excel and used equations to calculate the remaining packages. The results are shown below in Figure 6.2.

P0	2.7975
P1	0.493
P3	0.013
P5	13.099
P6	0.013
P7	6.47
P10	7.683
P11	0.013
P12	13.352
P13	20.383

Figure 6.2: Package demand times.

P0 was actually calculated using a different approach, due to the provided information. For each piece of hardware, all the processing steps were added and then multiplied by their respective service times. The sum of these three demand times was the final value of 2.7975.

6.3 Best Demand

The best demand was calculated by finding the fastest execution path. To determine the fastest path, I worked backwards starting with the end of the path and ending with the beginning, P0. The different nodes were also kept in mind, such as the repetition node, which made subsequent nodes

repeat 4 times.

The best demand path ended up being the following formula: P0 + P1 + P0 + 4(P3 + P1 + P11 + P0), which resulted in 19.354 seconds, or the path to P1 after the case node.

6.4 Worst Demand

The worst demand path was the slowest execution path, which happened to be the following path: P0 + P1 + P0 +4(P3 + P13 + P0), and this resulted in 98.862 seconds, or the path to P13.

6.5 Average Demand

The average demand path was calculated by taking the case node and applying the weighed chances of subsequent path occurring and adding this to the path taken to reach the case node. This can be displayed by using the following formula: P0 + P1 + P0 +4[P3 + 0.1(P5+P7) + 0.2(P0) + 0.3(P1 + P11) + 0.4(P13) + P0], which results in an average execution path demand of 60.6156 seconds.

Bibliography

Index