SSW-345 Labs

by

Benjamin Knobloch

Stevens.edu

October 27, 2023

SSW-345 Labs

Benjamin Knobloch
Stevens.edu

This document provides the requirements and design details of SSW-345 Labs. 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**
09/21/2023	BCK: • Created document. • Created team introduction chapter (1).
09/29/2023	NS: • Added Class Diagram section (Refer 2.2).
10/19/2023	NS: • Added Lab Two Chapter (Refer 3).
10/26/2023	BK: • Added chapters for Lab Three (Refer 4) and Lab Four (Refer 5).

Table of Contents

List of Tables

List of Figures

Chapter 1

Team Members

– Ben Knobloch

1.1 Marcos Morales

Hi, my name is Marcos Morales and I am a 3/4 software engineering major here at Stevens. My favorite programming languages are JavaScript and Python because of the convenient syntax. As for my interests outside of programming, I enjoy anything Star Wars and Nintendo related. I recently moved to Hoboken after commuting to Stevens for 2 years from New York so I'm now able to hangout with friends and be more involved in extracurricular activities.

1.2 Benjamin Knobloch

Ben Knobloch is a student of Software Engineering in his Junior year at Stevens Institute of Technology. He is a student tutor on campus, as well as the News Editor for the campus paper and secretary for the Stevens Honor Board and Engineers Without Borders SIT. In his free time, he enjoys reading and playing board games with his friends.

1.3 Noah Spina

Hello, my name is Noah Spina and I am a highly motivated Software Engineer. Specifically, I am interested in Machine Learning and hope to develop ML models to solve difficult problems. At Stevens, I am a brother of Sigma Phi Epsilon and am currently starting a Pickleball Club on campus. Other hobbies of mine include playing video games, playing pickleball, and cooking. Some fun facts about me: I have 4 dogs and 4 cats and my favorite color is white.

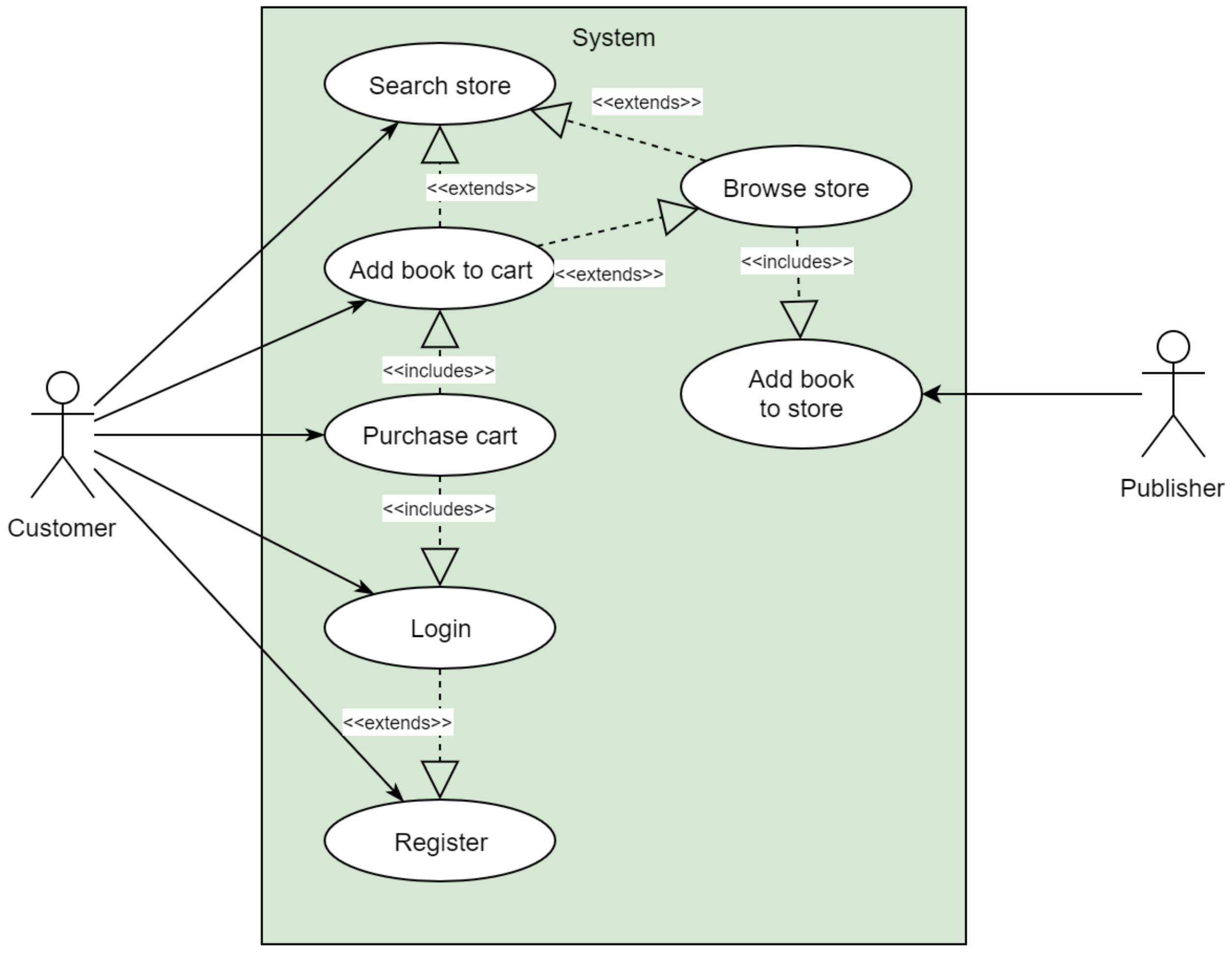
Chapter 2

Lab One: Bookstore UML Modeling

2.1 Use Case Diagram

The use-case diagram includes the standard suite of options available to the customer of the eBook store, as well as the basic interaction a publisher would have with the store. A customer can choose to search the store, add a book to his cart, purchase his cart, login and register. These use cases are also related. Adding a book to a cart is extended by searching or browsing the store, since those are optional actions that can be taken in order to find a book. Purchasing a cart includes adding at least one book to the cart. It also includes logging in, which in turn requires that the user has registered with the store. Lastly, having a store to browse from the user's end includes the assumption that publishers have submitted books to be sold in the store.

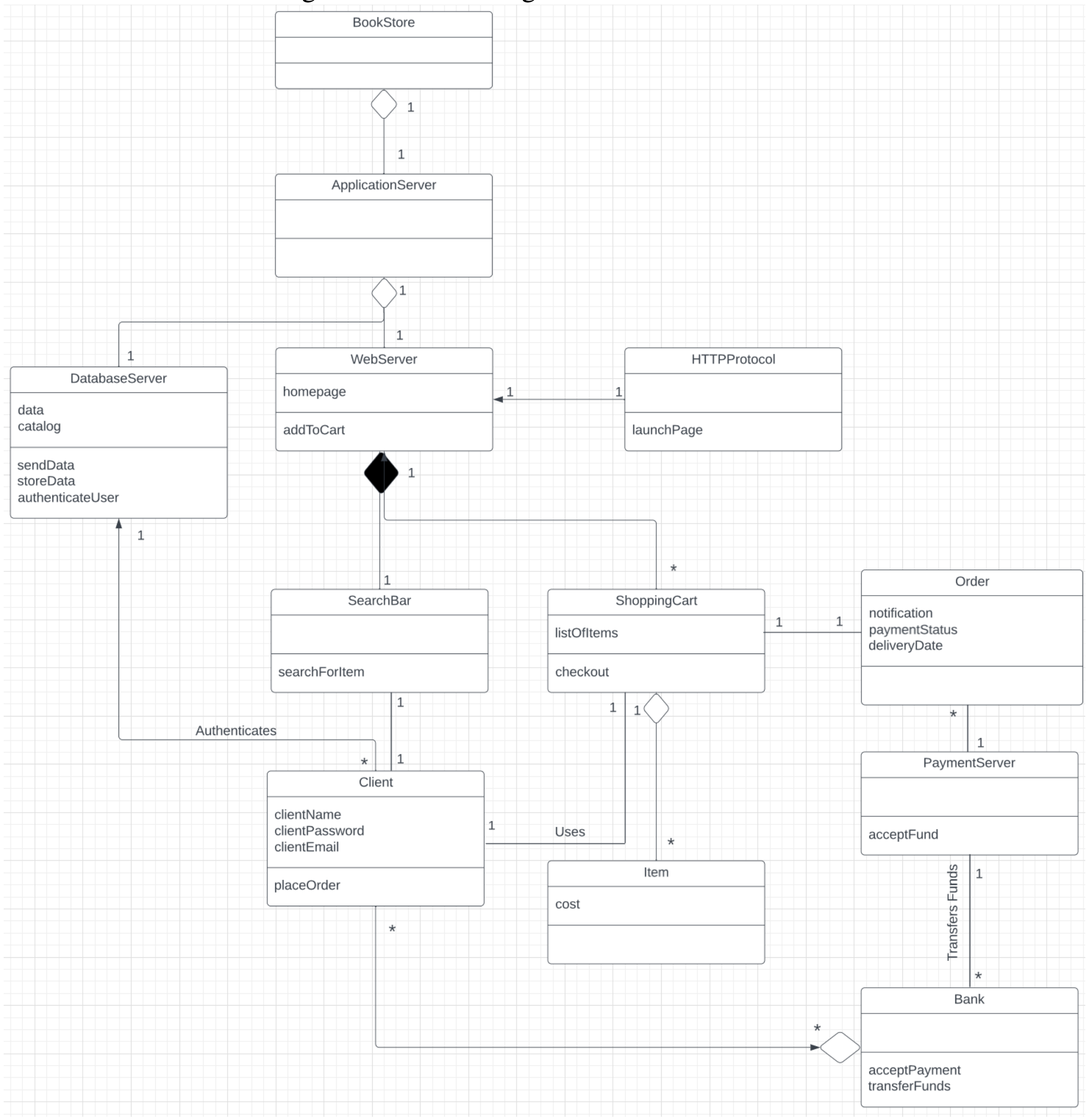
Figure 2.1: Use case diagram for an ebook store



2.2 Class Diagram

The class diagram starts at the top with the main BookStore class. The BookStore class has 1 ApplicationServer class that it runs on. The ApplicationServer class interacts with the DatabaseServer and WebServer class. All of these relationships are aggregations because they all can exist independently, however, each class ensures the smooth operation of the BookStore. We can see that the WebServer class is generalized to the HTTPProtocal. One HTTPProtocol belongs to one WebServer and allows it to run. We can also see how the HTTPProtocol class has the launchPage method. Going back to the WebServer class, we can identify that it is composed of the SearchBar and ShoppingCart classes. The SearchBar allows users to use the searchForItem method in order to add a book to their ShoppingCart. The ShoppingCart class is a container that holds the Client's order until it is ready to be placed. It has many Items in it, however, it is not composed by Items, just aggregated to them. We don't want to forget about the DatabaseServer class. This class authenticates multiple users by validating their clientEmail and clientPassword. The Client class is authenticated by the DatabaseServer and can also interact with the SearchBar to search for Books. Looking at the Order class, it has information about the ShoppingCart's listOfItems and it stores that information in a notification attribute along with the total cost of the ShoppingCart when the order is placed. The Client pays for this Order through the PaymentServer which interacts with a Bank class that transfers the User's funds to the PaymentServer for processing. We can see how the Bank class relates to the Client class as well. This class diagram models a fully functioning e-bookstore and shows the interaction between classes.

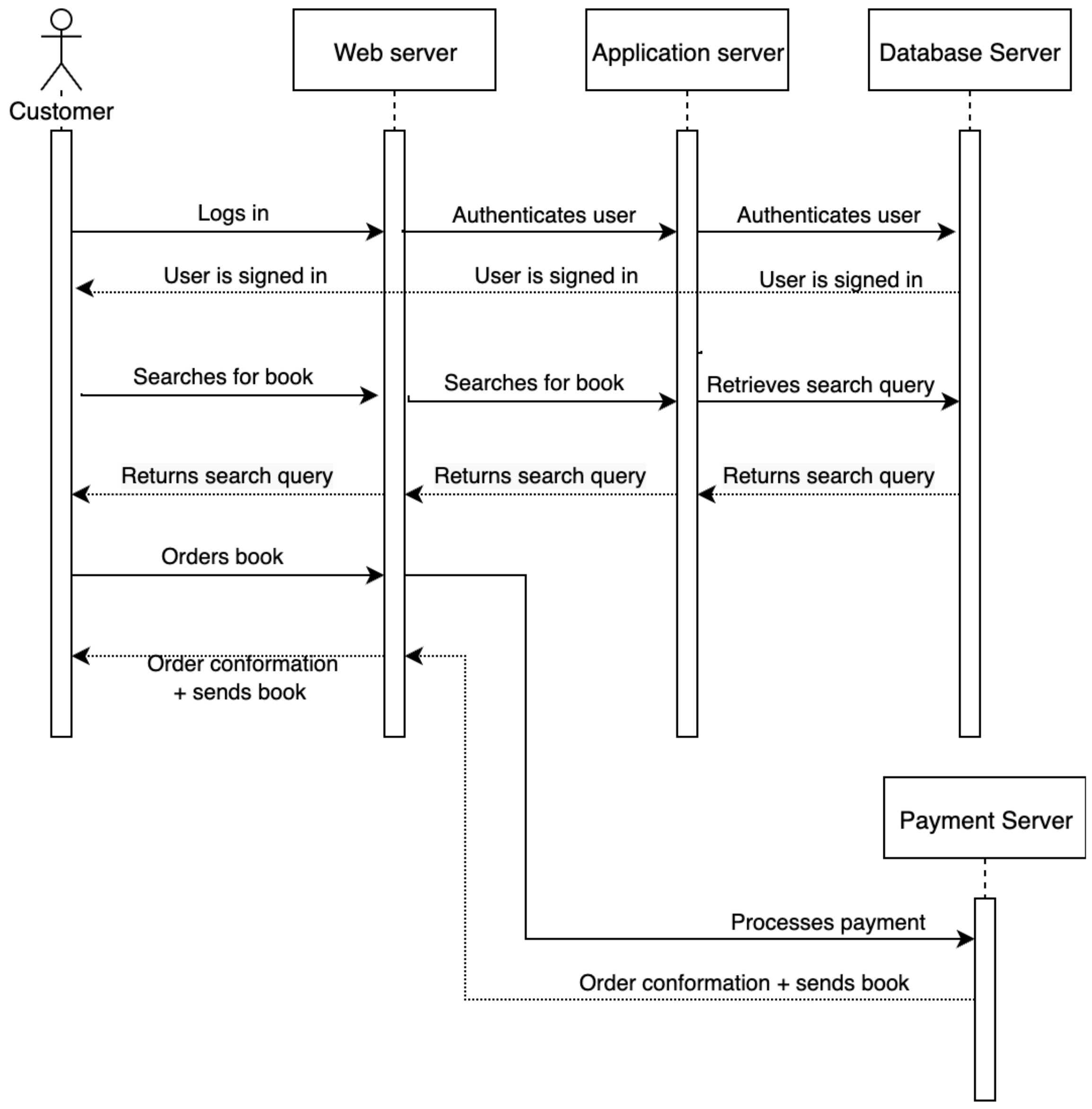
Figure 2.2: Class diagram for an e-book store



2.3 Sequence Diagram

The sequence diagram has customer, web server, application server, database server, and payment server. For every interaction the customer has, something is returned by the web server. The diagram illustrates a typical scenario of a customer ordering a book. First the customer logs in and the web, application, and database servers authenticate the user, returning a message that the user is successfully signed in. Then the customer searches for the book they want and the 3 previously mentioned servers process the search and return the search query. The customer can then order this book which is processed through the payment server which connects with the web server. Lastly, the payment and web server return the order confirmation and sends the customer the book they ordered.

Figure 2.3: Sequence Diagram for an ebook store when a customer searches and orders a book



Chapter 3

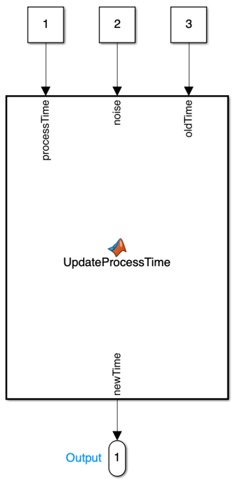
Lab Two: Simulations with Simulink

3.1 Lab Description

In this lab we were tasked with developing two simulations using update process time models. A loop model is then used to simulate repetition of certain processes in the system. The second simulation takes this a step further and adds in a case statement which introduces probabilities of additional processes occurring. To simulate probabilities in Simulink, a probabilistic model is used. The model uses a random number generator and based on the threshold of where the number falls, a process path is chosen While this model requires more code modifications to work with different systems, it reduces the number of connections needed which makes the entire system easier to read.

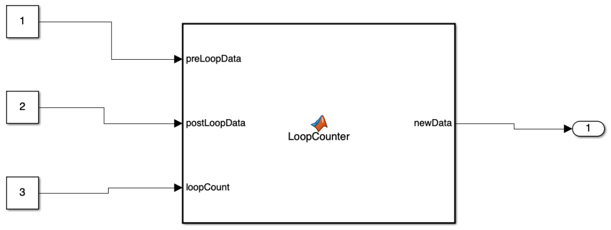
3.2 Update Process Time Model

Figure 3.1: Update Process Time Modeled in Simulink



3.3 Loop Counter Model

Figure 3.2: Loop Counter Modeled in Simulink



3.4 Execution Path Simulation

Figure 3.3: Simple Use Case System Model

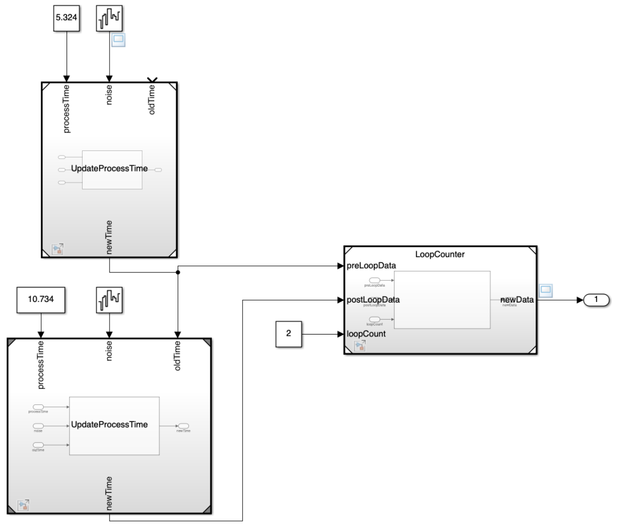
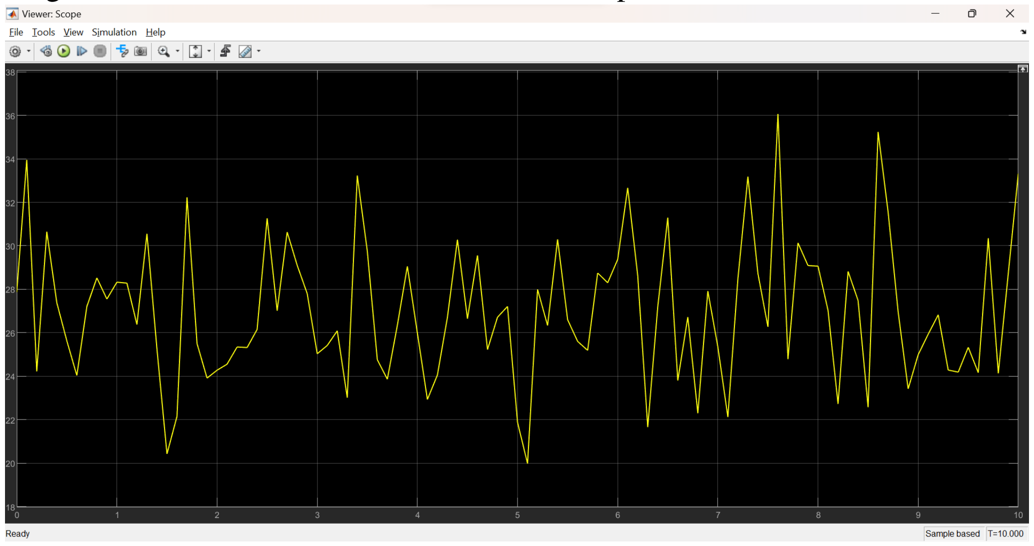


Figure 3.4: The simulated execution of the simple use case model in Simulink



3.5 Probabilistic Looping Model

Figure 3.5: Probabilistic Loop Counter modeled in Simulink

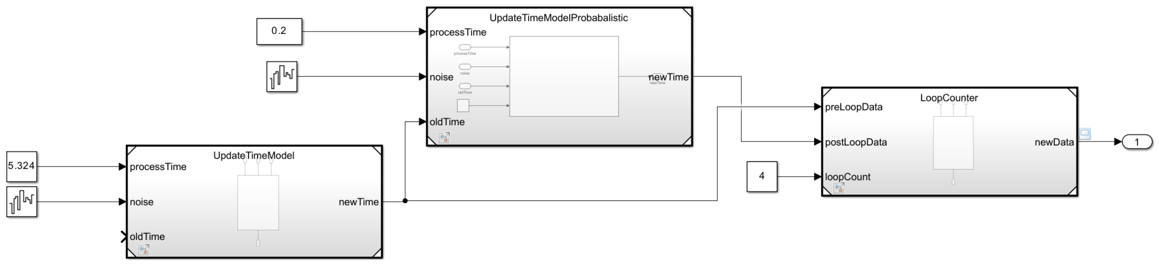


Figure 3.6: The pardo node in the probabilistic model

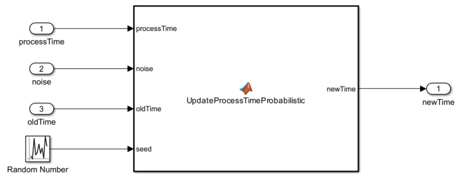
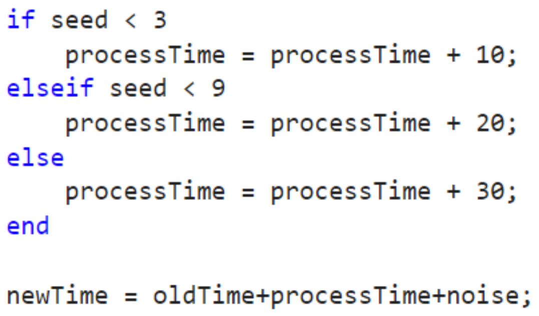


Figure 3.7: The decision statements for the pardo node in the loop counting model



```
if seed < 3
    processTime = processTime + 10;
elseif seed < 9
    processTime = processTime + 20;
else
    processTime = processTime + 30;
end

newTime = oldTime+processTime+noise;
```

Figure 3.8: The scope output for running the probabilistic loop counter model for ten seconds

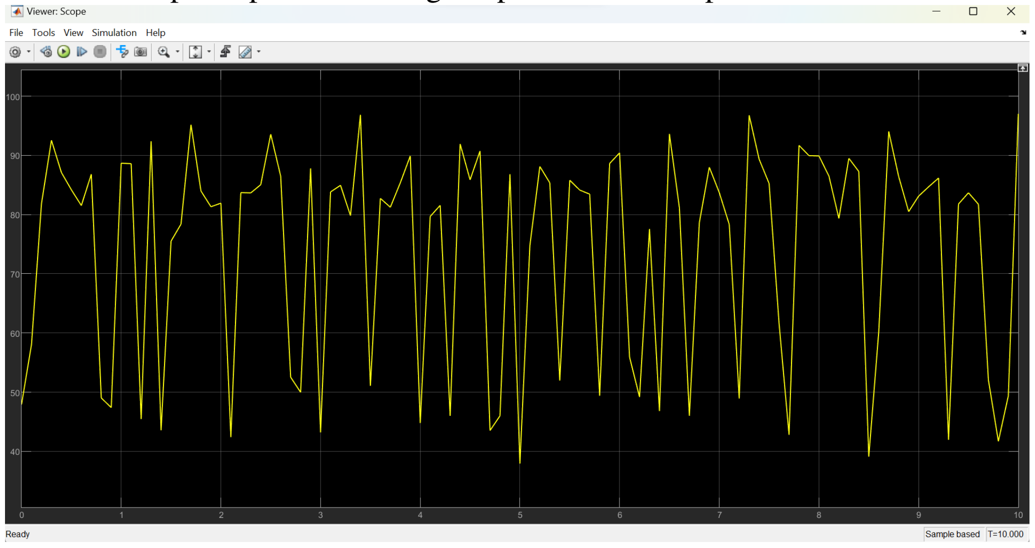
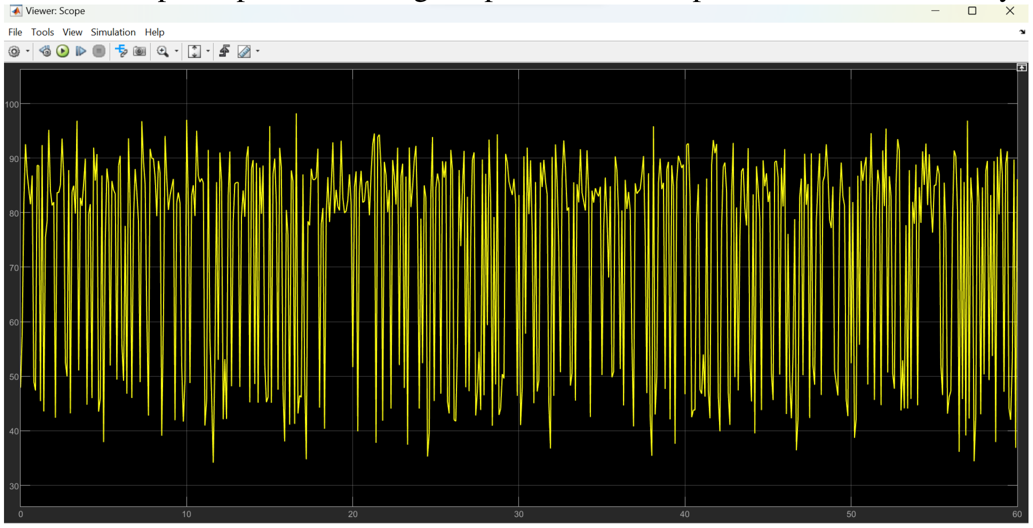


Figure 3.9: The scope output for running the probabilistic loop counter model for sixty seconds



Chapter 4

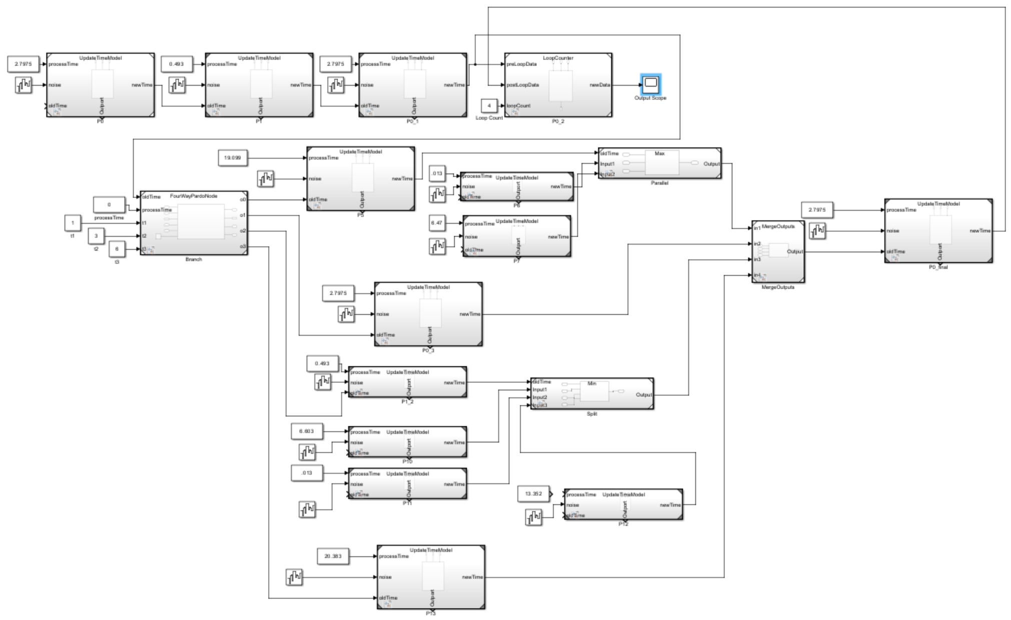
Case Pardo Split Simulation

4.1 Lab Description

This lab is a continuation of the previous lab where we have to create a system model in Simulink, except now we have a more complex system to simulate. It uses a system from a previous homework assignment, which requires us to make models for maximum and minimum nodes as well as use the runtimes that were calculated in that homework assignment.

4.2 Simulating a Process with a Pardo Node Split

Figure 4.1: Simulation Model



This figure shows our full Simulink model for the Pardo Split system. The branching node is created using a multiple output function that uses a probabilistic block to choose one of its four branches based on a random number. The pardo node is created by comparing two input processes and choosing the longer process time, simulating a parallel process. The split node, similarly, is created by comparing three input processes and choosing the shortest process time. Finally, each branch is merged by funnelling each signal into a single block that selects the only non-negative signal to propagate further in the simulation.

Figure 4.2: Four-way branch node

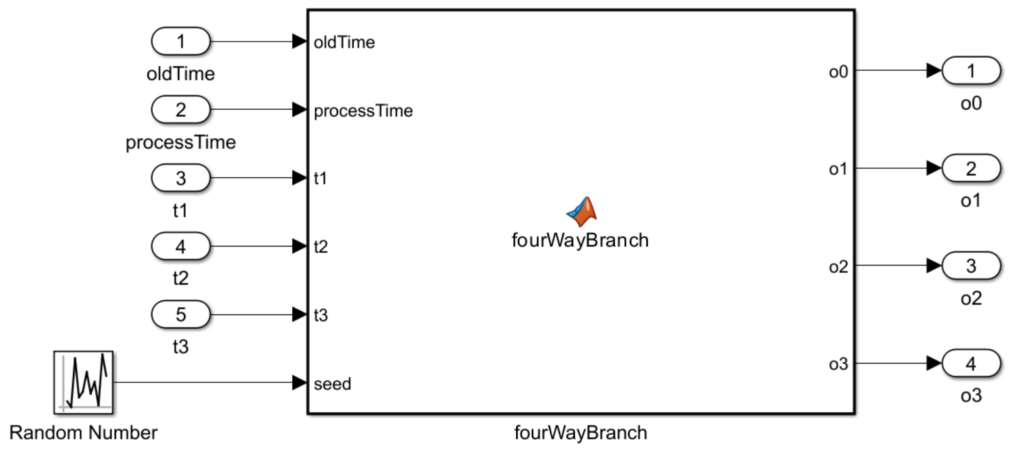
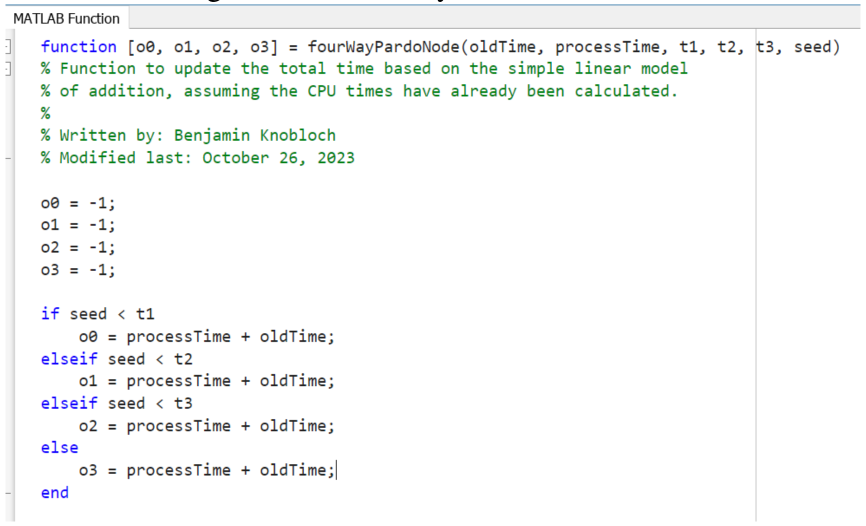


Figure 4.3: Four-way branch node code



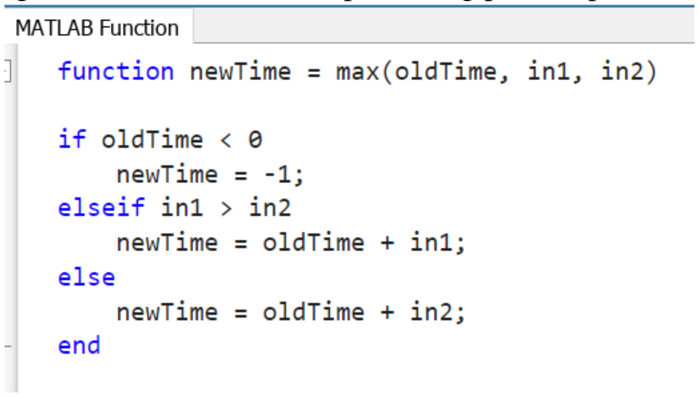
```
function [o0, o1, o2, o3] = fourWayPardoNode(oldTime, processTime, t1, t2, t3, seed)
% Function to update the total time based on the simple linear model
% of addition, assuming the CPU times have already been calculated.
%
% Written by: Benjamin Knobloch
% Modified last: October 26, 2023

o0 = -1;
o1 = -1;
o2 = -1;
o3 = -1;

if seed < t1
    o0 = processTime + oldTime;
elseif seed < t2
    o1 = processTime + oldTime;
elseif seed < t3
    o2 = processTime + oldTime;
else
    o3 = processTime + oldTime;
end
```

Our output produced a result consistent with our hand-completed calculations, but with additionally variability due to the addition of a noise factor in the simulation. The total process time of the system ranged from about 0 to about 150 seconds, and was mostly stratified into two main process lengths that are clearly visible in the final scope output.

Figure 4.4: Max function representing parallel processes

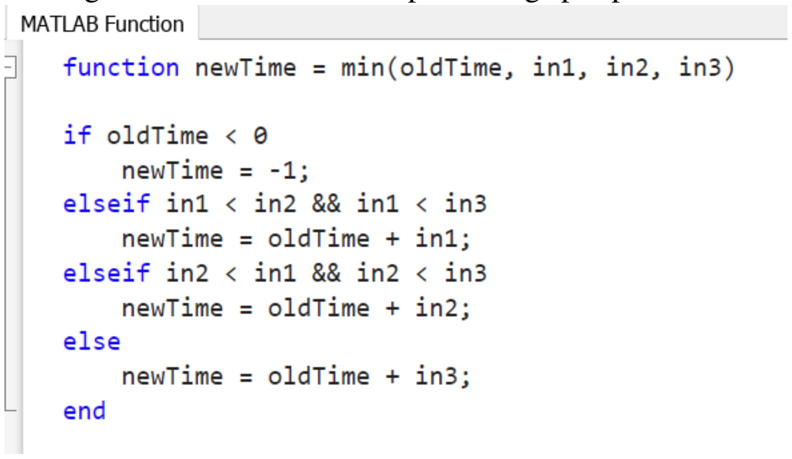


MATLAB Function

```
function newTime = max(oldTime, in1, in2)

if oldTime < 0
    newTime = -1;
elseif in1 > in2
    newTime = oldTime + in1;
else
    newTime = oldTime + in2;
end
```

Figure 4.5: Min function representing split processes

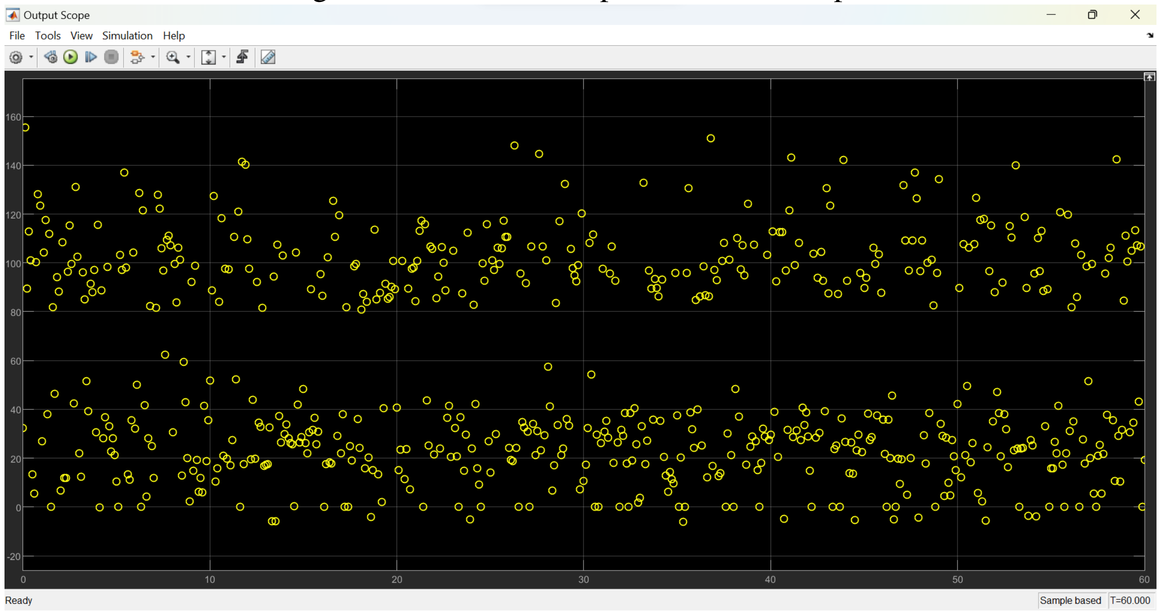


MATLAB Function

```
function newTime = min(oldTime, in1, in2, in3)

if oldTime < 0
    newTime = -1;
elseif in1 < in2 && in1 < in3
    newTime = oldTime + in1;
elseif in2 < in1 && in2 < in3
    newTime = oldTime + in2;
else
    newTime = oldTime + in3;
end
```

Figure 4.6: Pardo node split simulation output



Chapter 5

Lab Four: System Execution Modeling

5.1 Lab Description

In this lab we were tasked with analyzing the performance of a payment system. The lab entailed comparing the throughputs of a 5 year period and observing the utilization throughout this periods.

5.2 Quantitative Analysis of Payment Services Exercises

1. The bottleneck of the online bookstore is the merchant's servers CPU because it has the longest service demand out of all the resources. The maximum throughput would then be $\frac{1}{0.2459s} = 4.067$ jobs per second.

2.

Year 1:

Merchant's Server CPU: ((245.9 ms * 1 s/1000 ms)/1.00 speedup) * 14 tps = **3.44 s**

Payment Gateway's CPU: ((102.5 ms * 1 s/1000 ms)/1.00 speedup) * 14 tps = **1.44 s**

Year 2:

Merchant's Server CPU: ((245.9 ms * 1 s/1000 ms)/1.60 speedup) * 19 tps = **2.92 s**

Payment Gateway's CPU: ((102.5 ms * 1 s/1000 ms)/1.60 speedup) * 19 tps = **1.22 s**

Year 3:

Merchant's Server CPU: ((245.9 ms * 1 s/1000 ms)/2.56 speedup) * 23 tps = **2.209 s**

Payment Gateway's CPU: ((102.5 ms * 1 s/1000 ms)/2.56 speedup) * 23 tps = **0.921 s**

Year 4:

Merchant's Server CPU: ((245.9 ms * 1 s/1000 ms)/4.01 speedup) * 26 tps = **1.594 s**

Payment Gateway's CPU: ((102.5 ms * 1 s/1000 ms)/4.01 speedup) * 26 tps = **0.665 s**

Year 5:

Merchant's Server CPU: ((245.9 ms * 1 s/1000 ms)/6.41 speedup) * 29 tps = **1.112 s**

Payment Gateway's CPU: ((102.5 ms * 1s/1000 ms)/6.41 speedup) * 29 tps = **0.464 s**

The CPU demands for the merchant server and the payment server are vastly improved from year 1 to year 5. The Merchant Server CPU time was 3.44 seconds and the Payment Gateway CPU time was 1.44 seconds in year one. By year five, these numbers decreased to 1.112 seconds and .464 seconds.

3. Year 3 will vary from year 5 because the processing speed will be faster. The CPU times for year 3 is .817 s and .341 s for the Merchant Server and the Payment Gateway, respectively. However, the time in year 5 are .412 s and.172 s. This is much faster than year 3 due to the speedups.

Year 3:
Merchant's Server CPU: (((245.9 ms * 0.37) * 1 s/1000 ms)/2.56 speedup) * 23 tps = **0.817 s**
Payment Gateway's CPU: (((102.5 ms * 0.37) * 1 s/1000 ms)/2.56 speedup) * 23 tps = **0.341 s**

Year 4:
Merchant's Server CPU: (((245.9 ms * 0.37) * 1 s/1000 ms)/4.01 speedup) * 26 tps = **0.590 s**
Payment Gateway's CPU: (((102.5 ms * 0.37) * 1 s/1000 ms)/4.01 speedup) * 26 tps = **0.246 s**

Year 5:
Merchant's Server CPU: (((245.9 ms * 0.37) * 1 s/1000 ms)/6.41 speedup) * 29 tps = **0.412 s**
Payment Gateway's CPU: (((102.5 ms * 0.37) * 1s/1000 ms)/6.41 speedup) * 29 tps = **0.172 s**

4. Utilization = Volume of Transactions per Second * Service Demand

Old Algorithm

Year 1:
Merchant's Server CPU:
Utilization = ((245.9 ms * 1 s / 1000 ms) * 14 tps) * 100 percent = **344.26%**
CPUs need for 60 percent utilization = 344.26 percent/60 percent = **5.74 → 6 CPUs**
Payment Gateway's CPU:
Utilization = ((102.5 ms * 1 s / 1000 ms) * 14 tps) * 100 percent = **143.5%**
CPUs need for 60 percent utilization = 143.5 percent/60 percent = **2.39 → 3 CPUs**

Year 2:
Merchant's Server CPU:
Utilization = ((245.9 ms * 1 s / 1000 ms) * 19 tps) * 100 percent = **467.21%**
CPUs need for 60 percent utilization = 467.21 percent/60 percent = **7.79 → 8 CPUs**
Payment Gateway's CPU: Utilization = ((102.5 ms * 1 s / 1000 ms) * 19 tps) * 100 percent = **194.75%**
CPUs need for 60 percent utilization = 194.75 percent/60 percent = **3.25 → 4 CPUs**

Year 3:
Merchant's Server CPU:
Utilization = ((245.9 ms * 1 s / 1000 ms) * 23 tps) * 100 percent = **565.57%**
CPUs need for 60 percent utilization = 565.57 percent/60 percent = **9.43 → 10 CPUs**
Payment Gateway's CPU:
Utilization = ((102.5 ms * 1 s / 1000 ms) * 23 tps) * 100 percent = **235.75%**
CPUs need for 60 percent utilization = 235.75 percent/60 percent = **3.93 → 4 CPUs**

Year 4:
Merchant's Server CPU:
Utilization = ((245.9 ms * 1 s / 1000 ms) * 26 tps) * 100 percent = **639.34%**
CPUs need for 60 percent utilization = 639.34 percent/60 percent = **10.66 → 11 CPUs**
Payment Gateway's CPU:
Utilization = ((102.5 ms * 1 s / 1000 ms) * 26 tps) * 100 percent = **266.5%**
CPUs need for 60 percent utilization = 266.5 percent/60 percent = **4.44 → 5 CPUs**

Year 5:
Merchant's Server CPU:
Utilization = ((245.9 ms * 1 s / 1000 ms) * 29 tps) * 100 percent = **713.11%**
CPUs need for 60 percent utilization = 245.9 percent/60 percent = **11.89 → 12 CPUs**
Payment Gateway's CPU:
Utilization = ((102.5 ms * 1 s / 1000 ms) * 29 tps) * 100 percent = **297.25%**
CPUs need for 60 percent utilization = 297.25 percent/60 percent = **4.95 → 5 CPUs**
CPUs need for 60 percent utilization = 143.5 percent/60 percent = 2.39 CPUs
New Algorithm (Years 3-5):

Year 3:
Merchant's Server CPU:
Utilization = ((245.9 ms * .37 * 1 s / 1000 ms) * 23 tps) * 100 percent = **209.26%**
CPUs need for 60 percent utilization = 209.26 percent/60 percent = **3.49 → 4 CPUs**
Payment Gateway's CPU:
Utilization = ((102.5 ms * .37 * 1 s / 1000 ms) * 23 tps) * 100 percent = **87.23%**
CPUs need for 60 percent utilization = 87.23 percent/60 percent = **1.45 → 2 CPUs**

Year 4:
Merchant's Server CPU:
Utilization = ((245.9 ms * .37 * 1 s / 1000 ms) * 26 tps) * 100 percent = **236.56%**
CPUs need for 60 percent utilization = 236.56 percent/60 percent = **3.94 → 4 CPUs**
Payment Gateway's CPU:
Utilization = ((102.5 ms * .37 * 1 s / 1000 ms) * 26 tps) * 100 percent = **98.60%**

CPUs need for 60 percent utilization = 266.5 percent/60 percent = **1.64 → 2 CPUs**

Year 5:
Merchant's Server CPU:
Utilization = ((245.9 ms * .37 * 1 s / 1000 ms) * 29 tps) * 100 percent = **263.85%**
CPUs need for 60 percent utilization = 263.85 percent/60 percent = **4.40 → 5 CPUs**
Payment Gateway's CPU:
Utilization = ((102.5 ms * .37 * 1 s / 1000 ms) * 29 tps) * 100 percent = **109.98%**
CPUs need for 60 percent utilization = 297.25 percent/60 percent = **1.83 → 2 CPUs**

Bibliography

Index