**CS5373 –Software Model and Architecture**

**Spring 2020 – May 05, 2020**

**Term Project - SCOUTS**

**Project Team Members:**

Ashritha Puradamane Balachandra

Manisha Wakase

Rajasree Punneth Radhakrishnan

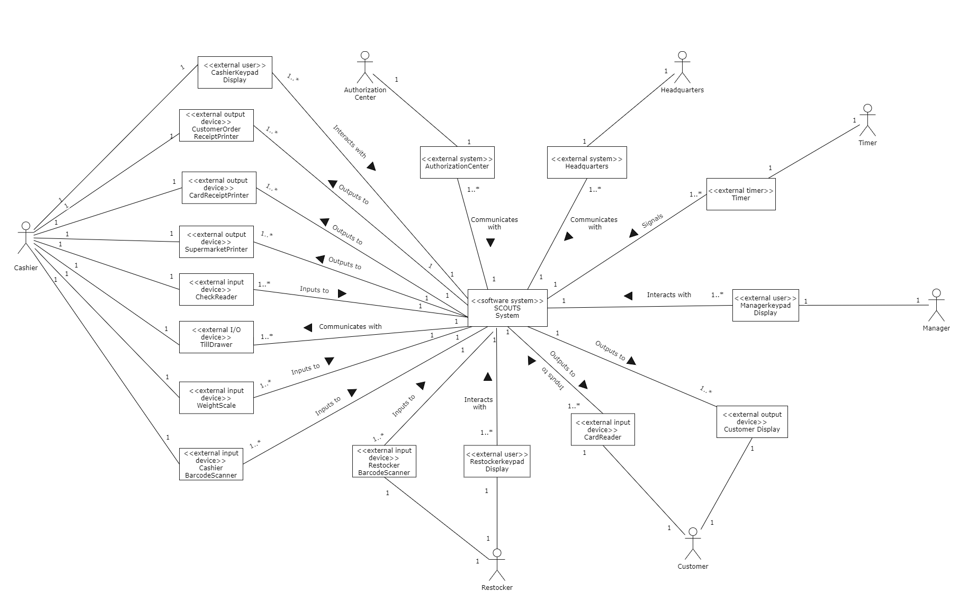
Sai Pranitha Challapuram

Vineela Aradhyula

**Phase 1 - Analysis model of the Supermarket Checkout System (SCOUTS):**

1. **SYSTEM CONTEXT CLASS DIAGRAM:**

**System Software Context Class Diagram:**



**System Context Class Diagram:**

**A close up of a map

Description automatically generated**

1. **STATIC MODEL:**

A close up of a map

Description automatically generated

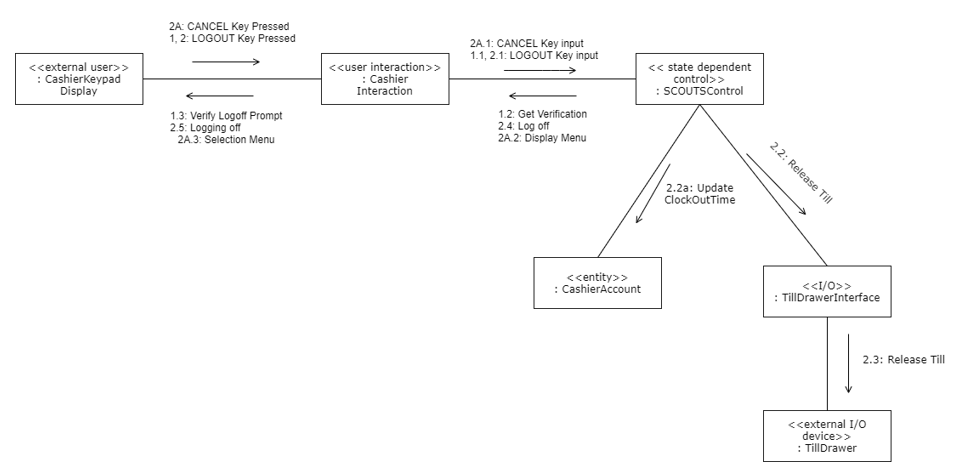
1. **COMMUNICATION DIAGRAMS:**

**Login Communication Diagram:**

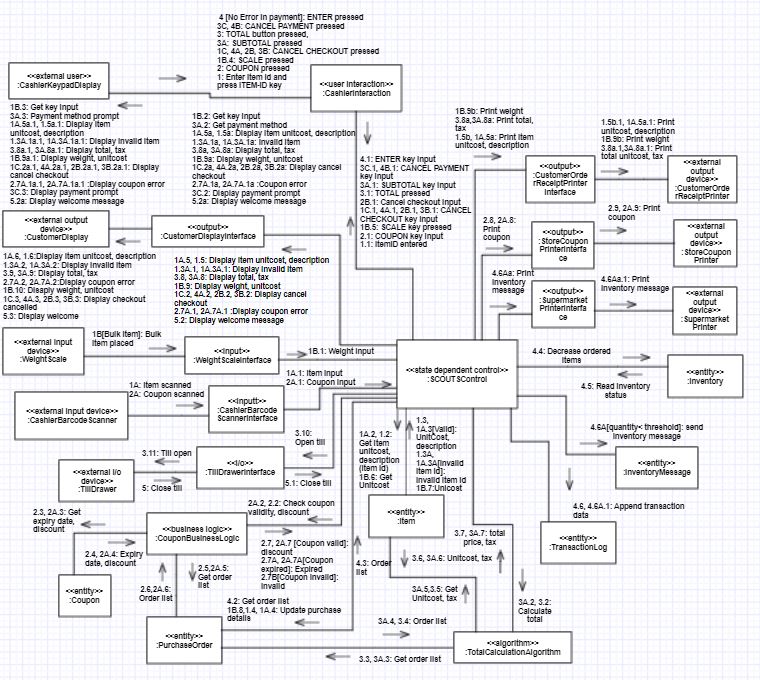
A picture containing screenshot

Description automatically generated

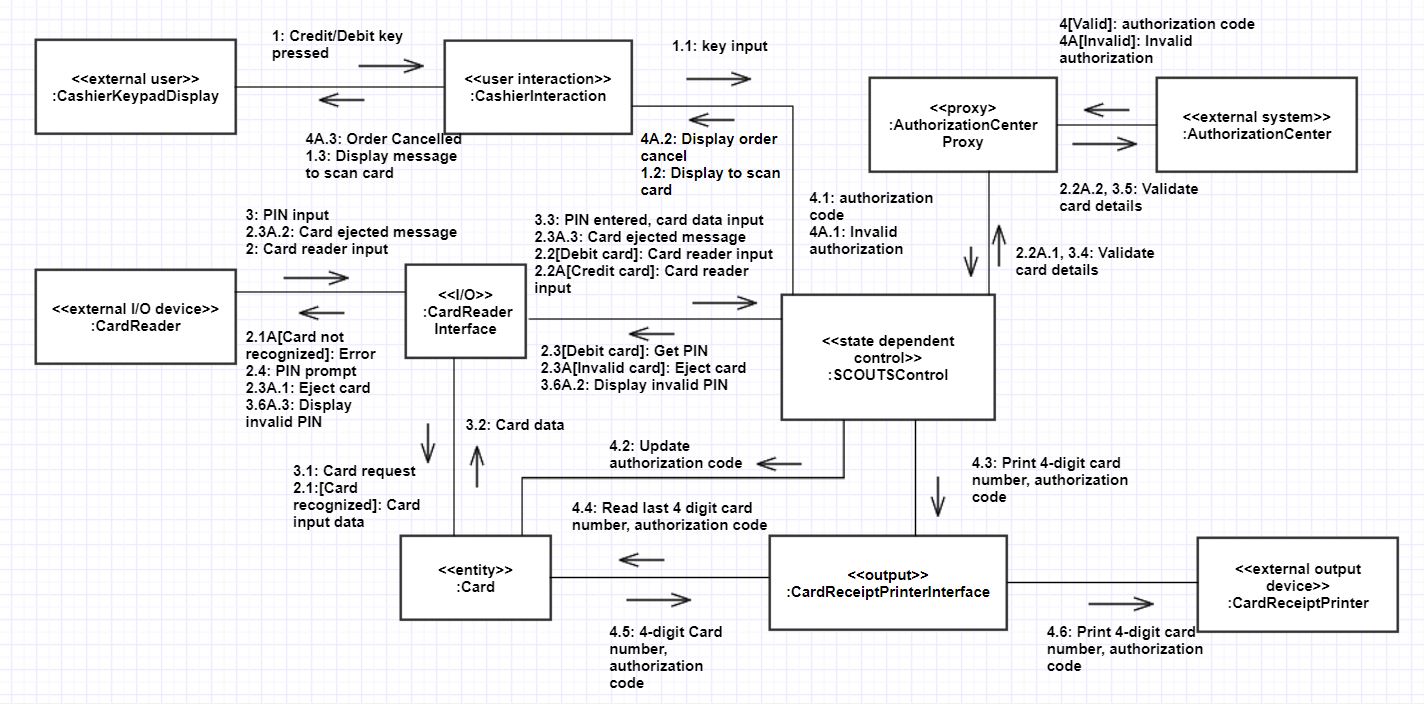
**Logout Communication Diagram:**



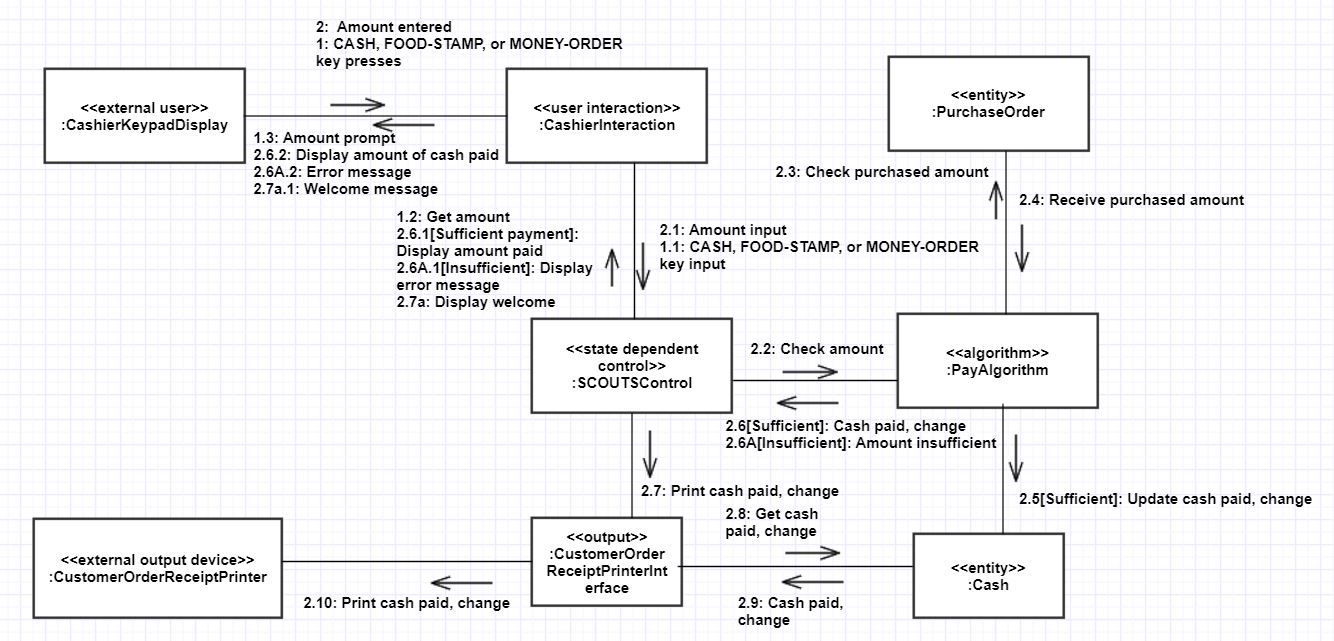
**Checkout Communication Diagram:**



**Pay by Credit/Debit Card Communication Diagram:**



**Pay by Cash Communication Diagram:**



**Pay by Check Communication Diagram:**A close up of a map

Description automatically generated

**Restock Product Inventory Communication Diagram:A close up of text on a white background

Description automatically generated**

**View/Update Product Inventory Communication Diagram:**

A screenshot of a cell phone screen with text

Description automatically generated

A close up of a map

Description automatically generated**Prepare & Send Daily Report Communication Diagram:**

**Send Inventory Message Communication Diagram:**

**A picture containing clock

Description automatically generated**

**Send Product Information Communication Diagram:**

A close up of a map

Description automatically generated

**Object Structuring Criteria Used:**

Entity Objects:

Item, Inventory, TransactionLog, Till, InventoryMessage, Coupon, CashierAccount, Manager, Headquarters, Restocker, Card, Cash, Check, PurchaseOrder

Control Objects:

State dependent control: SCOUTSControl

Timer: ReportTimer

Boundary objects:

User Interaction: CashierInteraction, CustomerInteraction, RestockerInteraction, ManagerInteraction

I/O: TillDrawerInterface, CardReaderInterface

Proxy: HeadquartersProxy

Input: CheckReaderInterface, BarcodeScannerInterface

Output: CustomerOrderReceiptPrinterInterface, StoreCouponPrinterInterface, SupermarketPrinterInterface, CardReceiptPrinterInterface

1. **STATE CHART DIAGRAMS:**

**Checkout StateChart diagram:**

**A close up of text on a white background

Description automatically generated**

**Pay by Credit/Debit Card StateChart diagram:**

A picture containing screenshot

Description automatically generated

**Pay by Cash StateChart diagram:**

A screenshot of a cell phone

Description automatically generated

**Pay by Check StateChart diagram:**

A picture containing screenshot

Description automatically generated

1. **MESSAGE SEQUENCE DESCRIPTIONS:**

**Login:**

1: Cashier presses the “Logon” function key on the cash register keypad.

1.1: Cashier interaction sends the logon key input to system’s SCOUTS Control.

1.2: System’s SCOUTS Control request to get User ID from Cashier Interaction.

1.3: Cashier Interaction prompts user ID on the Cashier keypad display.

2: Cashier enters ID number on the keypad.

2.1: Cashier Interaction sends user id input to system’s SCOUTS Control.

2.2: System’s SCOUTS Control request to get password from Cashier Interaction.

2.3: Cashier Interaction prompts password on the Cashier keypad display.

3: Cashier enters the password on the keypad.

3.1: Cashier Interaction sends password input to system’s SCOUTS Control.

3.2: System’s SCOUTS Control sends User ID and password to Cashier account for verification.

3.3[Valid]: If login ID and password are valid, then Cashier account sends valid user id and password message to system’s SCOUTS Control.

34b: System’s SCOUTS Control sends notification to update check in time to Cashier account.

3.4: system’s SCOUTS Control sends request to Till Drawer Interface to lock the till.

3.5: Till Drawer Interface locks the till.

3.4a: System’s SCOUTS Control request to display welcome message to Cashier.

3.4a.1: Cashier keypad display displays “Welcome” message to Cashier.

**Alternative sequence description:**

3.3A[Invalid]: If login ID and password are invalid, then cashier account sends invalid user id and password message to system’s SCOUTS Control.

3.3A.1: System’s SCOUTS Control request to display invalid login message to cashier.

3.3A.2: Cashier keypad display displays “Invalid Login” message to Cashier.

**Logout:**

1: Cashier presses the “Log Out” function key on the cash register keypad.

1.1: Cashier Interaction sends Logout key input to system’s SCOUTS Control.

1.2: System’s SCOUTS Control request verification code to Cashier Interaction.

1.3: The system prompts the cashier to verify log off on keypad display.

2: Cashier verifies log off by pressing the “Log Out” function key again.

2.1: Cashier Interaction sends Logout key input to system’s SCOUTS Control to log off.

2.2 System’s SCOUTS Control releases till.

2.3: System’s SCOUTS Control sends notification to Cashier account for updating checkout time.

2.4: System’s SCOUTS Control logs off the cashier.

2.5: Log off message is displayed on the keypad display.

**Alternative sequence description:**

2A: Cashier presses the “Cancel” function key on the keypad when the system prompts to verify logout.

2A.1: Cashier Interaction sends Cancel key input to system’s SCOUTS Control.

2A.2: System’s SCOUTS Control request to display menu for Cashier Interaction.

2A.3: Cashier keypad display displays selection menu for Cashier interaction.

**Checkout:**

1: Cashier enters the identification number of each item by keypad, followed by the function key ITEM-ID.

1.1: Cashier interaction sends entered item id to System’s SCOUTS Control.

1.2: System’s SCOUTS Control request to get item price and description using item id from Item.

1.3[Valid]: If item id is valid, item sends item price and description to System’s SCOUTS Control.

1.4: System’s SCOUTS Control update purchase details to Purchase Order.

1.5: System’s SCOUTS Control request to display item price and description to Customer display interface.

1.5.1: Customer display interface sends request to display item price and description on customer display.

1.5a: System’s SCOUTS Control request to display item price and description to Cashier interaction.

1.5.a.1: Cashier interaction sends request to display item price and description on cashier keypad display.

1.5b: System’s SCOUTS Control request to print item price and description to Customer order receipt printer interface.

1.5b.1: Customer order receipt printer prints item price and description.

2: Cashier presses the COUPON function key on keypad.

2.1: Cashier interaction sends COUPON key pressed notification to system’s SCOUTS Control.

2.2: System’s SCOUTS Control request Coupon business logic to check coupon validity and discount.

check expiration date present on COUPON.

2.3: Coupon business logic request to get expiry date and discount from coupon.

2.4: Coupon returns expiry date and discount of the coupon to coupon business logic.

2.5: Coupon business logic request purchase order to send order list.

2.6: Purchase order sends order list to coupon business logic.

2.7[coupon valid]: If coupon is valid, coupon business logic coupon sends applicable discount to system’s SCOUTS Control.

2.8: System’s SCOUTS Control request to print coupon on store coupon printer interface.

2.9: Store coupon printer prints the coupon.

3: Cashier presses TOTAL button on keypad.

3.1: Cashier interaction sends total pressed input to system’s SCOUTS Control.

3.2: System’s SCOUTS Control send request to Total calculation algorithm to calculate total.

3.3: Total calculation algorithm request the Purchase order to send order list to calculate the total.

3.4: Purchase order sends order list to total calculation algorithm.

3.5: Total calculation algorithm request the Item to send unitCost and tax information of all items of order list.

3.6: Item sends unitCost and tax information to Total calculation algorithm.

3.7: Total calculation algorithm calculates total and tax of all ordered items and sends it to system’s SCOUTS Control.

3.8: System’s SCOUTS Control request to display the total price including tax on the Customer display via Customer display interface.

3.8.1: Customer display interface sends request to display the total price including tax on the Customer display.

3.8: System’s SCOUTS Control request to display the total price including tax on the Cashier keypad display via Cashier interaction.

3.9: Cashier interaction sends request to display the total price including tax on the Cashier keypad display.

3.8a: System’s SCOUTS Control request to print total price including tax to Customer order receipt printer interface.

3.8a.1: Customer order receipt printer prints total price including tax.

3.10: System’s SCOUTS Control sends notification to Till drawer interface to open the till.

3.11: Till drawer interface sends request to Till drawer to open.

4[No Error in payment]: If there is no in payment, Cashier presses ENTER key.

4.1: Cashier interaction sends ENTER key pressed notification to System’s SCOUTS Control.

4.2: System’s SCOUTS Control request Purchase order to send order list.

4.3: Purchase order sends order list to System’s SCOUTS Control.

4.4: System’s SCOUTS Control decreases items on the customer order list from Inventory.

4.5: Inventory sends available item id and quantity status to System’s SCOUTS Control.

4.6: System’s SCOUTS Control appends transaction data to transaction log.

5: Cahier closes the till.

5.1: Till drawer interface sends till closed input to System’s SCOUTS Control.

**Alternate sequence description:**

1A: Cashier scans the items using Cashier barcode scanner.

1A.1: Cashier barcode scanner sends scanned item input to

1A.2: System’s SCOUTS Control request to get item price and description using item id from Item.

1A.3[Valid]: If item id is valid, item sends item price and description to System’s SCOUTS Control.

1A.3[Invalid item id]: If item id is invalid, item sends invalid item id to System’s SCOUTS Control.

1A.4: System’s SCOUTS Control update purchase details to Purchase Order.

1A.5: System’s SCOUTS Control request to display item price and description to Customer display interface.

1A.5.1: Customer display interface sends request to display item price and description on customer display.

1A.5a: System’s SCOUTS Control request to display item price and description to Cashier interaction.

1A.5.a.1: Cashier interaction sends request to display item price and description on cashier keypad display.

1A.5b: System’s SCOUTS Control request to print item price and description to Customer order receipt printer interface.

1A.5b.1: Customer order receipt printer prints item price and description.

1B[Bulk item]: If an item is a bulk item, then item is placed on Weight Scale device.

1B.1: Weight scale interface sends weight input to System’s SCOUTS Control.

1B.2: System’s SCOUTS Control request Cahier interaction for the key input.

1B.3: Cashier interaction request for key input from Cashier Keypad.

1B.4: Cashier presses the SCALE button for bulk item.

1B.5: Cashier interaction sends SCALE key pressed notification to System’s SCOUTS Control.

1B.6: System’s SCOUTS Control request to get item price and description using item id from Item.

1B.7: If item id is valid, item sends item price and description to System’s SCOUTS Control.

1B.8: System’s SCOUTS Control update purchase details to Purchase Order.

1B.9: System’s SCOUTS Control request to display item price and description to Customer display interface.

1B.10: Customer display interface sends request to display item price and description on customer display.

1B.9a: System’s SCOUTS Control request to display item price and description to Cashier interaction.

1B.9a.1: Cashier interaction sends request to display item price and description on cashier keypad display.

1B.9b: System’s SCOUTS Control request to print item price and description to Customer order receipt printer interface.

1B.9b.1: Customer order receipt printer prints item price and description.

1C, 4A, 2B, 3B: Cashier presses the CANCEL CHECKOUT key on keypad.

1C.1, 4A.1, 2B.1, 3B.1: Cashier interaction sends CANCEL CHECKOUT key pressed notification to System’s SCOUTS Control.

1C.2, 4A.2, 2B.2, 3B.2: System’s SCOUTS Control sends cancel checkout to Customer interaction.

1C.3, 4A.3, 2B.3, 3B.3: Customer interaction sends to display cancel checkout on the Customer display.

1C.2, 4A.2, 2B.2, 3B.2: System’s SCOUTS Control sends cancel checkout to cashier interaction.

1C.3, 4A.3, 2B.3, 3B.3: Cashier keypad display displays canceled checkout message to Cashier.

2A: Cashier scans each coupon across the bar-code scanner System’s SCOUTS Control.

2A.1: Barcode scanner interface sends coupon input to System’s SCOUTS Control.

2A.2: System’s SCOUTS Control request Coupon business logic to check coupon validity and discount.

2A.3: Coupon business logic request to get expiry date and discount from coupon.

2A.4: Coupon returns expiry date and discount of the coupon to coupon business logic.

2A.5: Coupon business logic request purchase order to send order list.

2AA.6: Purchase order sends order list to coupon business logic.

2A.7[coupon valid]: If coupon is valid, coupon business logic coupon sends applicable discount to system’s SCOUTS Control.

2.7A, 2A.7A[Coupon expired]: If coupon is expired, coupon business logic coupon sends coupon expired information to system’s SCOUTS Control.

2.7B[Coupon Invalid]: If coupon is invalid, coupon business logic coupon sends coupon invalid to system’s SCOUTS Control.

2.7A.1: System’s SCOUTS Control want to display coupon error on Customer display via Customer display interface.

2.7A.2: Customer display interface sends coupon error information to display on Customer display.

2.7A.1a, 2A.7A.1a: System’s SCOUTS Control want to display coupon error on Cashier keypad display display via Customer interaction.

2.7A.1a.1, 2A.7A.1a.1: Customer interaction sends coupon error information to display on Cashier keypad display.

2A.8: System’s SCOUTS Control request to print coupon on store coupon printer interface.

2A.9: Store coupon printer prints the coupon.

3A: Cashier presses TOTAL button on keypad.

3A.1: Cashier interaction sends total pressed input to system’s SCOUTS Control.

3A.2: System’s SCOUTS Control send request to Total calculation algorithm to calculate total.

3A.3: Total calculation algorithm request the Purchase order to send order list to calculate the total.

3A.4: Purchase order sends order list to total calculation algorithm.

3A.5: Total calculation algorithm request the Item to send unitCost and tax information of all items of order list.

3A.6: Item sends unitCost and tax information to Total calculation algorithm.

3A.7: Total calculation algorithm calculates total and tax of all ordered items and sends it to system’s SCOUTS Control.

3A.8: System’s SCOUTS Control request to display the total price including tax on the Customer display via Customer display interface.

3A.8.1: Customer display interface sends request to display the total price including tax on the Customer display.

3A.8: System’s SCOUTS Control request to display the total price including tax on the Cashier keypad display via Cashier interaction.

3A.8.1: Cashier interaction sends request to display the total price including tax on the Cashier keypad display.

3A.8a: System’s SCOUTS Control request to print total price including tax to Customer order receipt printer interface.

3A.8a.1: Customer order receipt printer prints total price including tax.

3C, 4B: Cashier presses the CANCEL PAYMENT key on keypad.

3C.1, 4B.1: Cashier interaction sends CANCEL PAYMENT key pressed notification to System’s

SCOUTS Control.

3C.2, 4B.2: System’s SCOUTS Control cancel the selected payment method and notify to cashier

interaction.

3C.3, 4B.3: Cashier interaction prompts payment method on the keypad display.

4.6A[quantity < threshold]: If number of items goes below specified threshold, Inventory sends message to System’s SCOUTS Control.

4.6Aa: System’s SCOUTS Control pass the inventory message to supermarket printer interface.

4.6Aa.1: Supermarket printer interface sends inventory message to supermarket printer to print it.

**Pay by Credit/Debit Card:**

1: Cashier presses the CREDIT/DEBIT function key on the Cashier keypad display.

1.1: Cashier interaction sends key input to System’s SCOUTS control.

1.2: System’s SCOUTS control request to display a message to scan the card to Cashier interaction.

1.3: Cashier interaction sends notification to Cashier keypad display a message to scan card.

2: Customer insert the card to card reader

2.1[Card recognized]: if card reader interface recognizes the debit card, then card reader interface sends card input data to Card.

2.2[Debit card]: If debit card, Card reader interface informs the card details to System’s SCOUTS Control.

2.3[Debit card]: If card is debit card, System’s SCOUTS Control request card reader to display need PIN prompt.

2.4: Card reader displays PIN prompt.

3: Customer enters a PIN on keyboard.

3.1: Customer interaction contacts card entity for card details.

3.2: Card provides card date details.

3.3: Customer entered pin and card data is sent to System’s SCOUTS Control.

3.4: System’s SCOUTS Control request validation of customer information by contacting authorization center proxy.

3.5: Authorization center proxy request authorization center to validate card details.

3.6[valid]: if valid customer, authorization center returns authorization code to authorization center proxy.

3.7: Authorization center proxy sends authorization code to System’s SCOUTS Control.

3.8: System’s SCOUTS Control sends update authorization code to Card.

3.9: System’s SCOUTS Control request to print card number and authorization code to customer order receipt Printer interface.

3.10: Customer order receipt Printer interface contacts Card to get last 4-digit card number and authorization code.

3.11: Card sends last 4-digit card number and authorization code to customer order receipt Printer interface.

3.12: Card receipt printer prints card number and authorization code on the customer receipt.

**Alternate sequence description:**

2.1A[Card not recognized]: If card is not recognized, then Card is ejected by card reader.

2.2A[Credit card]: if credit card, Card reader interface informs the card details to System’s SCOUTS Control.

2.2A.1: System’s SCOUTS Control request validation of customer information by contacting authorization center proxy.

2.2A.2: Authorization center proxy request authorization center to validate card details.

2.2A .3[valid]: if valid customer, authorization center returns authorization code to authorization center proxy.

2.2A.4: Authorization center proxy sends authorization code to System’s SCOUTS Control.

2.2A.5: System’s SCOUTS Control sends update authorization code to Card.

2.2A.6: System’s SCOUTS Control request to print card number and authorization code to customer order receipt Printer interface.

2.2A.7: Customer order receipt Printer interface contacts Card to get last 4-digit card number and authorization code.

2.2A.8: Card sends last 4-digit card number and authorization code to customer order receipt Printer interface.

2.2A.9: Card receipt printer prints card number and authorization code on the customer receipt.

2.2A.3A.1, 3.6A.1[Invalid]: If card authorization is invalid, then authorization center returns authorization invalid to System’s SCOUTS Control.

2.2A.3A.2, 3.5A.1: System’s SCOUTS Control request Cashier interaction to display order cancelled to Cashier.

2.2A.3A.3, 3.5A.2: Cashier keypad display shows order cancelled message.

**Pay by Cash:**

1: Cashier presses the CASH, FOOD-STAMP, or MONEY-ORDER function key on the keypad.

1.1: Cashier interaction sends CASH, FOOD-STAMP, or MONEY-ORDER key input to System’s SCOUTS Control.

1.2: System’s SCOUTS Control request for amount being paid.

1.3: System prompts cashier to enter the amount being paid by the customer.

2: Cashier enters the amount of cash received to the system using the keypad.

2.1: Cashier interaction sends amount input to System’s SCOUTS Control.

2.2: System’s SCOUTS Control sends amount to Pay algorithm to determines if sufficient payment is made by cash for purchase order.

2.2a: System’s SCOUTS Control want to display amount paid to Cashier interaction.

2.2a.1: System outputs the amount of cash paid by the customer on the display.

2.3: System’s SCOUTS control send request to Purchase order for the amount.

2.4: Purchase order sends purchased amount to Pay algorithm.

2.5[Sufficient]: If amount is sufficient, Pay algorithm updates to Cash the amount paid and change.

2.6[Sufficient]: If sufficient amount is available, then Pay algorithm sends cash paid amount and change to SCOUTS control.

2.6.1: If payment is Sufficient, then system’s SCOUTS Control want to display amount of cash paid to Cashier interaction.

2.6.2: Cashier keypad display displays amount of cash paid to Cashier.

2.7: System’s SCOUTS Control requests Customer order receipt Printer interface to print the amount of cash paid by the customer and the change on the customer receipt.

2.8: Customer order receipt Printer interface contacts Cash to get cash paid and change.

2.9: Cash sends cash paid and change to Customer order receipt Printer interface.

2.10: Customer order receipt prints the amount of cash paid by the customer and the change on the customer receipt.

2.7a: System’s SCOUTS Control request Cashier interaction to display welcome menu on keypad display.

2.7a.1: Cashier keypad display displays welcome message.

**Alternate sequence description:**

2.4A[Insufficient payment]: If Insufficient payment, then System’s SCOUTS Control want to display Insufficient payment to Cashier interaction.

2.6A[Insufficient]: If insufficient amount, then Pay algorithm sends insufficient amount message to SCOUTS Control.

2.6A.1: If payment is insufficient, then system’s SCOUTS Control request to display error message to Cashier interaction.

2.6A.2: Cashier keypad display displays error message to Cashier.

**Pay by Check:**

**Message Sequence Description:**

1. Cashier presses the CHECK function key on the keypad.

1.1: Cashier Interaction sends CHECK key entered to System’s SCOUTS Control.

1.2: Display message to scan CHECK.

1.3: Display message to scan CHECK.

2: Check reader scans the input check.

2.1[Check recognized]: If check is recognized, then check reader interface sends check input data to check.

2.2: Card reader interface accepts the check and sends check inserted details to System’s SCOUTS Control.

2.3: System’s SCOUTS Control request for the amount paid.

2.4: system prompts Cashier to enter the amount paid by the customer on cashier keypad display.

3: Cashier enters the amount paid by the customer.

3.1: Cashier interaction sends input amount to System’s SCOUTS Control.

3.2: System’s SCOUTS Control contacts pay algorithm to check due amount.

3.3: Pay algorithm contact Check entity to update amount paid.

3.4: Pay algorithm requests purchase order to check amount.

3.5: Purchase order returns an amount.

3.6[Sufficient payment]: If amount is sufficient, then system’s SCOUTS Control want the cahier to place the check in the customer order receipt printer.

3.7: System’s SCOUTS Control request the customer order receipt printer interface to print date and other required on check.

3.8: Customer order interface request for Customer ID, Store ID, Order No, Date and time to purchase order.

3.9: Purchase order Customer ID, Store ID, Order No, Date and time to printer.

3.10: Printer prints a line containing date, time, store identity, cashier identity, and order number on the check.

**Alternate sequence description:**

2.1A[Check not recognized]: If check is not recognized, then card reader interface sends error message to System’s SCOUTS Control.

2.1A.1: System’s SCOUTS Control request to display error message.

2.1A.2: Printer prints error message.

3.6A[insufficient payment]: If amount is insufficient, then System’s SCOUTS Control requests to display error message.

3.6A.1: Cashier keypad display displays error message.

3.6A.2[Insufficient payment]: Cashier display error message.

**Restock Product Inventory:**

**Message Sequence Description:**

1. Restocker selects restock option from store computer.

1.1: RestockerInteraction sends restock selected input to Restock Business Logic.

1.2: Restock Business Logic asks the RestockerInteraction to prompt for item.

1.3: RestockerInteraction provides item prompt to Restocker.

2: Restocker BarcodeScanner will scan items.

2.1: Scanned item information is sent to RestockerKeyPadDisplayInteraction

2.2: Scanned item information is sent to Restock Business Logic.

2.3: Restock Business Logic requests for item from Item.

2.4: If it is an existing item, then the item details like item id, description, Price are sent to Restock Business Logic.

2.5: Restock Business Logic asks RestockerInteraction for Quantity.

2.6: RestockerInteraction prompts the user to enter item quantity.

3: Restocker enters item quantity.

3.1: RestockerInteraction sends entered item quantity to Restock Business Logic.

3.2: Restock Business Logic updates the item quantity in the Inventory.

**Alternatives**

2.3A: If item is new, Item sends message that item is not present to Restock Business Logic.

2.3A.1: Restock Business Logic asks for item details to RestockerInteraction.

2.3A.2: RestockerInteraction prompts the Restocker to enter item details.

2.3A.3: Restocker enters item details int the prompt.

2.3A.4: RestockerInteraction sends entered item details to Restock Business Logic.

2.3A.5: Restock Business Logic adds new item details to Inventory.

2.3A.6: Restock Business Logic asks RestockerInteraction to display confirmation message.

2.3A.7: RestockerInteraction object displays confirmation to Restocker.

**View/Update Product Inventory:**

**Message Sequence Description:**

1: ManagerKeyPadDisplay has selected view/update option from store computer

1.1: ManagerKeyPadInteraction will display Item prompt to ManagerKeyPadDisplay

2: ManagerKeyPadDisplay will send item input to ManagerKeyPadInteraction.

2.1: ManagerKeyPadInteraction will request item from item.

2.2: Item entity will provide item details.

2.3: Item details will be display to ManagerKeyPadDisplay.

3: ManagerKeyPadDisplay will press OK, SUBMIT, ENTER, GUI button

3.1: ManagerKeyPadInteraction will update the item details

3.1a: ManagerKeyPadInteraction will update the item details in Inventory.

3.1a.1: Inventory will send confirmation when item updated successfully.

3.2: Item will send item id, price, descriptions, discount to ManagerKeyPadInteraction when updated successfully.

3.3: ManagerKeyPadInteraction will display the updated item details to ManagerKeyPadDisplay.

**Prepare & Send Daily Report:**

**Message Sequence Description:**

1.External Timer will send signal to the ReportTimer.

1.1: Report Timer will send signal to DailyReportAlgorithm.

1.2: DailyReportAlgorithm will request details to TransactionLog.

1.3: TransactionLog will send details to DailyReportAlgorithm.

1.4: DailyReportAlgorithm send report to Headquarters proxy.

1.4a: DailyReportAlgorithm will send request to supermarket printer interface to print the report

1.4a.1: SupermarketPrinterInterface will send request to supermarket printer interface to print the report

**Send Inventory Message:**

**Message Sequence Description:**

1.External Timer sends signal to ReportTimer.

1.1: Report Timer sends signal to InventoryMessage.

1.2: InventoryMessage send buffered inventory messages to ReportTimer.

1.3: ReportTimer send buffered inventory messages to Headquarters proxy.

1.4: Headquarters proxy will send inventory message to Headquarters.

**Send Product Information:**

**Message Sequence Description:**

1: Headquarters will send new product information message.

1.1: Headquarters Proxy will update Inventory with new product information.

1.1a: Headquarters Proxy will update Item with new product information.

**f. COMPLETE STATECHART (Based on 1(d)):**

**Complete StateChart Diagram:**

A close up of text on a white background

Description automatically generated

**Phase 2 - Design Model for the Supermarket Checkout System**

1. **Develop a consolidated communication diagram(s) showing all the objects and message interfaces in the system.**

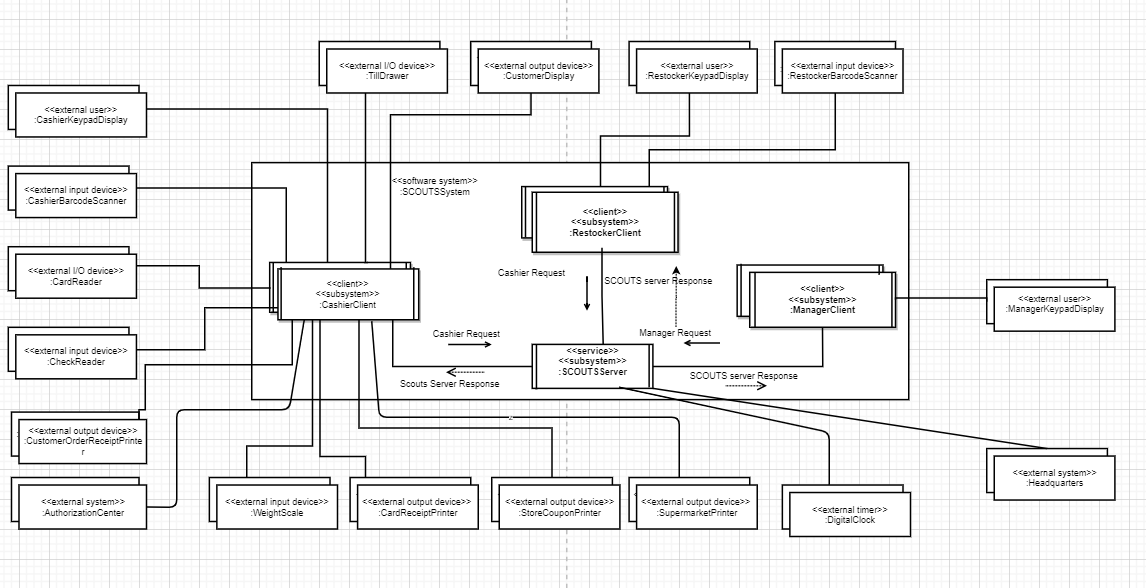
**Integrated Communication Diagram:**

**A close up of text on a black background

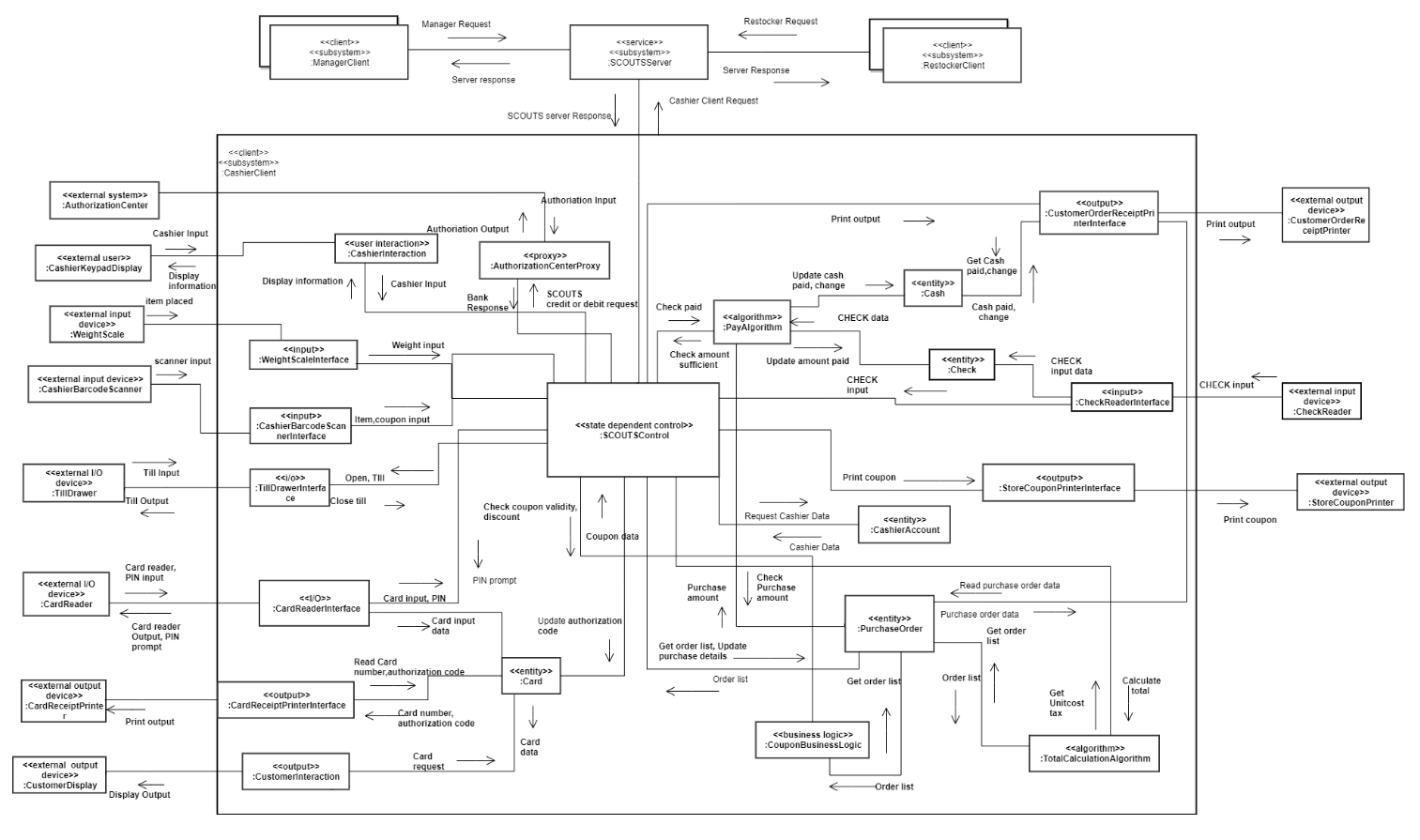
Description automatically generated**

1. **Define the subsystem architecture, (depicted on a concurrent communication diagram), showing the clients and server of the Scouts System. Define the message communication interfaces between the clients and server.**

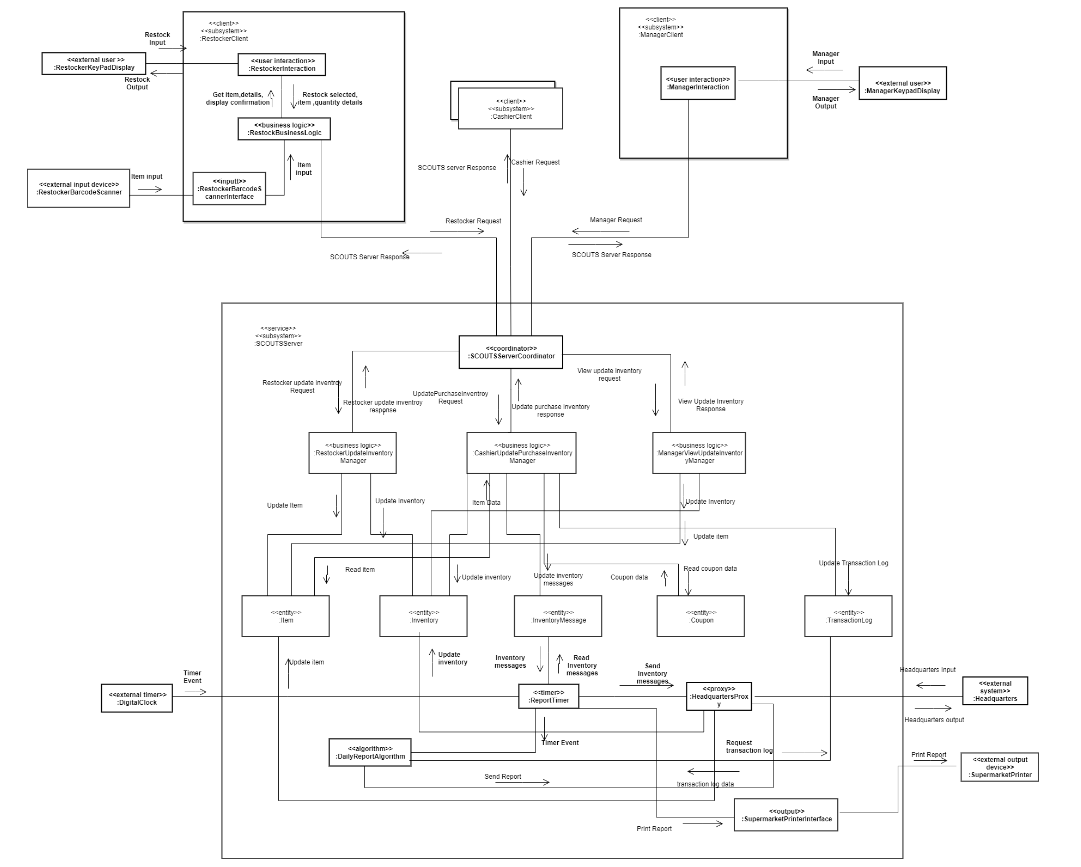
**Subsystem design –high level communication diagram for SCOUTS System:**



**Integrated Communication Diagram for Cashier Client subsystem:**



**Integrated Communication Diagram for SCOUTS Server, Manager and Restocker subsystem:**



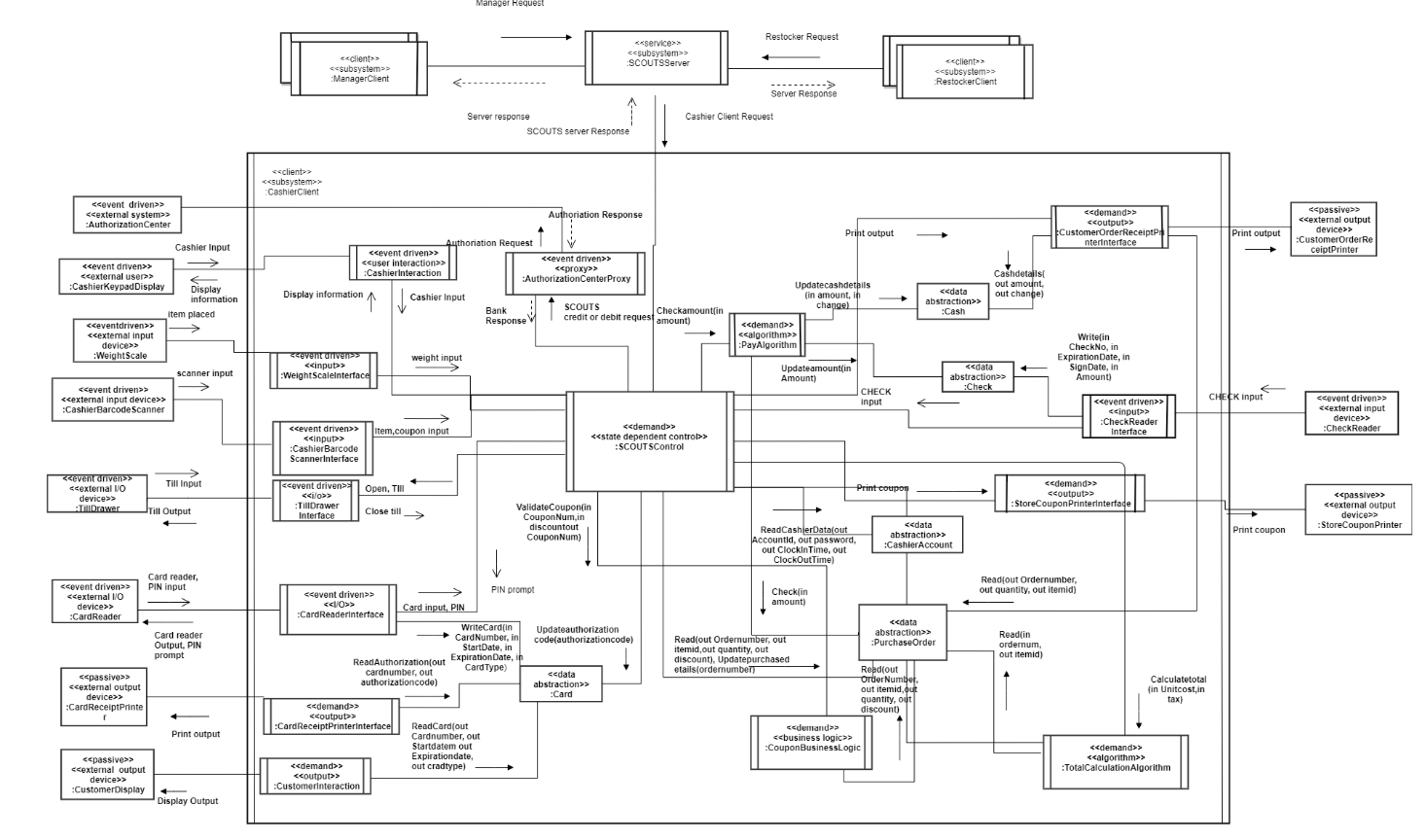
1. **Define the task architecture (depicted on concurrent communication diagrams) showing the concurrent tasks in each subsystem and the interfaces between them. Describe the criteria used for task structuring. (Define the message communication interfaces.)**

**Integrated Communication Diagram (after task structuring and defining task interfaces):**

A close up of text on a white background

Description automatically generated

**Task architecture – revised concurrent communication diagram for Cashier Client subsystem:**



**Task architecture – revised concurrent communication diagram for SCOUTS Server subsystem, Restocker subsystem and Manager Subsystem:**

A close up of text on a white background

Description automatically generated

1. **Define the information hiding classes in the system. Define the operations of each class.**

**Cashier Client Subsystem Information Hiding Classes**

A screenshot of a social media post

Description automatically generated

**SCOUTS Server Subsystem Information Hiding Classes**

A screenshot of a cell phone

Description automatically generated

**Restocker Client Subsystem Information Hiding Classes**

A screenshot of a cell phone

Description automatically generated

**Manager Client Subsystem Information Hiding Classes**

A screenshot of a cell phone

Description automatically generated

**f) Develop a task behavior specification for each concurrent task in the system, showing how each task responds to the inputs it receives.**

**Event Sequencing Logic for Card Reader Interface Task:**

Initialize card reader;

**loop**

-- Wait for external interrupt from card reader

**wait** (cardReaderEvent);

Read card data held on card’s magnetic strip;

**if** card is recognized

**then** -- Write card data to Card object;

Card.writeCard (**in** CardNumber, **in** StartDate, **in** ExpirationDate, **in** CardType);

-- send card Inserted message to SCOUTS Control;

send (SCOUTSControlMessageQ, cardInserted);

-- Wait for message from SCOUTS Control;

receive (cardReaderMessageBuffer, message);

**if** message = eject

**then**

--Eject card;

CardReaderInterface.eject();

-- Send card Ejected message to SCOUTS Control;

send (SCOUTSControlMessageQ, cardEjected);

**elseif** message = Get PIN

**then**

-- Display PIN prompt;

CardReaderInterface.displayPrompt();

**elseif** message = Invalid PIN

**then**

-- Display invalid PIN message;

CardReaderInterface.display(invalidPIN);

**else** error condition;

**end if;**

**else** -- card was not recognized so eject;

Eject card;

**end if;**

-- Wait for PIN eneted by Cashier;

**wait** (PINEnteredEvent);

Read entered PIN;

-- Read card details from Card object;

Card.read(**out** CardNumber, **out** StartDate, **out** ExpirationDate);

-- Send PIN and card details to SCOUTS Control;

send (SCOUTSControlMessageQ, pinAndCardDetailsInput);

**end loop;**

**Event Sequencing Logic for Check Reader Interface Task:**

Initialize check reader;

**loop**

-- Wait for external interrupt from check reader

**wait** (CheckReaderEvent);

Read check data

**if** Check is recognized

**then** -- Write card data to Check object;

Check.write (**in** CheckNo, **in** ExpirationDate, **in** SignDate, **in** Amount);

-- send check inserted message to SCOUTS Control;

send (SCOUTSControlMessageQ, checkInserted);

-- Wait for message from SCOUTS Control;

receive (checkReaderMessageBuffer, message);

**if** message = eject

**then**

--Eject check;

CheckReaderInterface.eject();

-- Send check Ejected message to SCOUTS Control;

send (SCOUTSControlMessageQ, checkEjected);

**else** error condition;

**end if;**

**else** – Check not recognized, so send message to SCOUTS control object;

send (SCOUTSControlMessageQ, checkInvalid);

**end if;**

**end loop;**

**Event Sequencing Logic for Cashier Interaction Task:**

Initialize cashier interaction;

**loop**

-- Wait for external interrupt from cashier

**wait** (keyPressedEvent);

Read KeyPressed input;

**if** input = ITEM-ID

**then**

-- Send ITEM-ID key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, GetItem);

**elseif** input = COUPON

**then**

-- Send COUPON key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, couponPressed);

**elseif** input = SCALE

**then**

-- Send SCALE key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, scalePressed);

**elseif** input = CANCEL CHECKOUT

**then**

-- Send CANCEL CHECKOUT key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, cancelCheckoutPressed);

**elseif** input = SUBTOTAL

**then**

-- Send SUBTOTAL key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, subtotalPressed);

**elseif** input = TOTAL

**then**

-- Send TOTAL key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, totalPressed);

**elseif** input = CANCEL PAYMENT

**then**

-- Send CANCEL PAYMENT key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, cancelPaymentPressed);

**elseif** input = ENTER

**then**

-- Send ENTER key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, enterPressed);

**elseif** input = Debit

**then**

-- Send Debit key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, debitPressed);

**elseif** input = Credit

**then**

-- Send Credit key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, creditPressed);

**elseif** input = Credit

**then**

-- Send Credit key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, creditPressed);

**elseif** input = Amount

**then**

-- Send Amount input message to SCOUTS Control;

send (SCOUTSControlMessageQ, amountEntered);

**elseif** input = CASH

**then**

-- Send CASH key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, cash Pressed);

**elseif** input = FOOD-STAMP

**then**

-- Send FOOD-STAMP key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, foodStampPressed);

**elseif** input = MONEY-ORDER

**then**

-- Send MONEY-ORDER key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, moneyorderPressed);

**elseif** input = CHECK

**then**

-- Send CHECK key input message to SCOUTS Control;

send (SCOUTSControlMessageQ, checkPressed);

**else** error condition

**end if;**

--Messages from SCOUTS control are received in prompt message queue

receive(promptMessageQueue, message);

Extract message name and message parameters from message;

**If** message = GetKeyInput

**then**

--Prompt for key input;

CashierInteraction.displayPrompt ();

**elseif** message = GetPaymentMethod

**then**

--Prompt for payment method;

CashierInteraction.displayPrompt();

**elseif** message = unitcostDescription

**then**

--Display item details;

CashierInteraction.display(unitcost, description);

**elseif** message = invalidItem

**then**

--Display Invalid item;

CashierInteraction.display(invalidItem);

**elseif** message = totalTax

**then**

--Display total and tax;

CashierInteraction.display (total, tax);

**elseif** message = weightUnitCost

**then**

--Display weight and unitcost;

CashierInteraction.display (weight, unitCost);

**elseif** message = cancelCheckout

**then**

--Display checkout cancelled;

CashierInteraction.display (cancelledCheckout);

**elseif** message = couponInvalid

**then**

--Display coupon invalid;

CashierInteraction.display (couponInvalid);

**elseif** message = welcome

**then**

--Display welcome message;

CashierInteraction.display (welcome);

**else** error condition;

**end if;**

**end loop;**

**Event Sequencing Logic for Customer Interaction Task:**

Initialize Customer interaction;

**loop**

--Messages from SCOUTS control are received in customerDisplayMessageQ

receive(customerDisplayMessageQ, message);

Extract message name and message parameters from message;

**if** message = unitcostDescription

**then**

--Display item details;

CustomerDisplayInterface.display(unitcost, description);

**elseif** message = invalidItem

**then**

--Display Invalid item;

CustomerDisplayInterface.display(invalidItem);

**elseif** message = totalTax

**then**

--Display total and tax;

CustomerDisplayInterface.display (total, tax);

**elseif** message = weightUnitCost

**then**

--Display weight and unitcost;

CustomerDisplayInterface.display (weight, unitCost);

**elseif** message = cancelCheckout

**then**

--Display checkout cancelled;

CustomerDisplayInterface.display (cancelledCheckout);

**elseif** message = couponInvalid

**then**

--Display coupon invalid;

CustomerDisplayInterface.display (couponInvalid);

**elseif** message = welcome

**then**

--Display welcome message;

CustomerDisplayInterface.display (welcome);

**else** error condition;

**end if;**

**end loop;**

**Event Sequencing Logic for Manager Interaction Task:**

Initialize Manager interaction;

**loop**

-- Wait for external interrupt from Manager

**wait** (keyPressedEvent);

Read KeyPressed input;

**if** input = view

**then**

-- display item id prompt

MangerInteraction.displayPrompt ();

--wait to Manager to enter ItemId;

**wait**(ItemIdEvent)

--read item details

Item.readItemDetails(**in** itemId **out** unitCost, **out** description, **out** discount);

-- Display Item details to Manager;

MangerInteraction.display(unitCost, description, discount);

**else if**  input = update

**then**

-- display item id prompt

MangerInteraction.displayPrompt(ItemId);

--wait to Manager to enter ItemId;

**wait**(ItemIdEvent)

– Update item details to Item object

Item.UpdateItem(**in** UnitCost, **in** description, **in** discount);

--wait to Manager to enter OK or SUBMIT;

**wait**(okEvent)

--Update item details in Inventory

Inventory.UpdateItem(**in** itemId, **in** itemDescription, **in** discount);

--Display updated information to Manager

MangerInteraction.display(Item id, item description, item discount);

**end else if;**

**end loop;**

**Event Sequencing Logic for Restocker Interaction Task:**

Initialize Restocker interaction;

**loop**

-- Wait for external interrupt from Restocker

**wait** (RestockSelectedEvent);

--send restock selected input to Restock business logic

send(RestockbusinessQ, restockSelected);

**wait**(RequestItemIdEvent);

--Display prompt for Item Id

RestockerInteraction.displayPrompt();

--wait for itemId entered or scanned by restocker

**wait**(getItemIdEvent);

--send itemId input to Restock business logic

send(RestockbusinessQ, itemId);

--Send itemId to Restocker Business logic;

Send(RestockerBusinessLogic, itemId);

**wait**(RequestQuantityEvent);

--Display prompt for Item quantity

RestockerInteraction.displayPrompt ();

**wait**(getItemQantityEvent);

--send item qiantity input to Restock business logic

send(RestockbusinessQ, itemquantity);

**end loop;**

**Event Sequencing Logic for Cashier Barcode scanner Interface Task:**

Initialize Cashier barcode reader;

**loop**

-- Wait for external interrupt from barcode reader

**wait** (cashierbarcodeInputEvent);

barcodeInput = scan();

Read the barcode input;

**if** barcodeInput

**then**

-- Send input message to SCOUTS Control;

send (SCOUTSControlMessageQ, CashiecBarcodeInput);

**else** error condition;

**end if;**

**end loop;**

**Event Sequencing Logic for Restocker Barcode scanner Interface Task:**

Initialize Restocker barcode reader;

**loop**

-- Wait for external interrupt from barcode reader

**wait** (restockerBarcodeInputEvent);

barcodeInput = scan();

Read the barcode input;

**if** barcodeInput

**then**

-- Send input message to SCOUTS Control;

send (SCOUTSControlMessageQ, restockerBarcodeInput);

**else** error condition;

**end if;**

**end loop;**

**Event Sequencing Logic for Weight scale Interface Task:**

Initialize wight scale;

**loop**

-- Wait for external interrupt from weight scale

**wait** (ItemPlacedEvent);

Read the barcode input;

wightInput = getWeight(item);

**if** wightInput

**then**

-- Send input message to SCOUTS Control;

send (SCOUTSControlMessageQ, weightInput);

**else** error condition;

**end if;**

**end loop;**

**Event Sequencing Logic for Till Drawer interface Task**

Initialize till drawer;

**loop**

-- Wait message from SCOUTS control

receive(SCOUTSControlMessageQ, message);

**if message =** lockTill

**then**

-- lock the till

tillDrawerInterface.lock();

**elseif message =**  tillOpen

**then**

-- Open the till

tillDrawerInterface.Open();

**elseif message =**  tillRelease

**then**

-- release the till

tillDrawerInterface.release();

**else** error condition;

**end if;**

-- Wait for external interrupt from cashier

**wait** (tillInputEvent);

Read the till input;

**if** tillInput

**then**

-- Close the till;

tillDrawerInterface.Close();

-- Send till input message to SCOUTS Control;

send (SCOUTSControlMessageQ, tillClose);

**end if;**

**end loop;**

**Event Sequencing Logic for Coupon Business logic Task:**

Initialize for Coupon Business logic;

**loop**

--Messages from other senders are received on CouponBusinessLogicQ

Receive(CouponBusinessLogicQ, message);

--Extract the event name and any message parameters;

Read the Coupon number and discount;

ValidateCoupon(**in** CouponNum, **in** discount, **out** CouponNum)

**wait** (validateCouponEvent);

-- Get coupon expiry date and discount from Coupon object.

Coupon.read(**out** expiryDate , **out** discount);

-- Get order details from PurchaseOrder object

PurchaseOrder.read(**out** OrderNumber, **out** ItemId, **out** quantity, **out** discount);

isExpired(**in** expiryDate **out** isValid);

**if** isValid

**then**

discountValid = isDiscountApplicable(**in** Coupon.discount, **in** PurcaseOrder.discount);

**if** discountValid

**then**

-- send coupon valid to SCOUTS control

send (SCOUTSControlMessageQ, couponValid);

**else** – Purchased item is not valid for coupon discount

send (SCOUTSControlMessageQ, couponInvalid);

**else** – Coupon expired

send (SCOUTSControlMessageQ, couponValid);

**end if;**

**end loop;**

**Event Sequencing Logic for Restocker Business logic Task:**

Initialize for Restocker Business logic;

**loop**

-- Wait for interrupt from Restocker interaction

**wait** (RestockSelectedEvent);

Read the input from Restocker interaction;

Item.readItem(**in** ItemId, **out** ItemDscription);

**if** ! ItemDscription

**then**

--Add item to the inventory

Inventory.add(**in** ItemID, **in** ItemDscription);

**else** – --Update item quantity

Inventory.UpdateInventory(**in** itemId, **in** quantity);

**end if;**

**end loop;**

**Event Sequencing Logic for Report timer Task:**

Initialize Digital clock;

**loop**

-- Wait for interrupt from digital clock

**wait** (digitalClockEvent);

Read the timer input from digital clock;

timerInput = getTimerInput();

**if** timerInput

**then**

--Send timer input to Daily report algorithm;

send(reportAlgorithmQueue, **in** timerInput);

**--**Read inventory message from Inventory object;

inventoryMessage = InventoryMessage.read(**out** inventoryMessage);

--Send inventory message to Headquarters proxy

Send(headquartersProxy, **in** inventoryMessage);

**end if;**

**end loop;**

**Event Sequencing Logic for Total Calculation Algorithm Task:**

Initialize for Total calculation algorithm;

**loop**

--Messages from SCOUTS control are received on totalAlgorithmQ

Receive(totalAlgorithmQ, message);

--Extract the event name and any message parameters;

Read the Coupon number and discount;

**wait** (totalCalculateEvent);

-- Get order details from PurchaseOrder object

PurchaseOrder.read(**in** orderNumber **out** ItemId);

--loop through all itemId in purchased order

**while** ItemId

-- Get unitCost, tax for each ordered item from Item object.

Item.read(**in** ItemId, **out** unitCost, **out** tax);

--add unitCost to UnitCost array and dd tax to tax array

**end while;**

--Calculate total unitCost and total tax;

calculateTotal(**in** unitCost, **in** tax, **out** total, **out** totalTax)

--Send total, tax to SCOUTS control

send (SCOUTSControlMessageQ, **out** total, **out** totalTax);

**end loop;**

**Event Sequencing Logic for Pay Algorithm Task:**

Initialize for Pay algorithm;

**loop**

--Messages from SCOUTS control are received on payAlgorithmQ

Receive(payAlgorithmQ, message);

--Extract the event name and any message parameters;

Read the amount to check;

CheckAmount(amount);

**wait** (paymentResponseEvent);

-- Get purchase amount from PurchaseOrder object

PurchaseOrder.CheckinAmount(**out** amount);

**if** Cash.amount

**then**

--Check if amount is sufficient

isAmountSufficient(**in** PurchaseOrder .amount, **in** Cash.amount **out** sufficient);

**if** sufficient

**then**

--Update cash objct with amount and change;

Cash.updateCashDetails(**in** amount, **in** change);

--Send amount sufficient message to SCOUTS Control;

send (SCOUTSControlMessageQ, amountSufficient);

**else** – amount Insufficient

send (SCOUTSControlMessageQ, amountInsufficient);

**end if;**

**elseif** Check.amount

**then**

isAmountSufficient(**in** PurchaseOrder .amount, **in** Check.amount **out** sufficient);

**if** sufficient

**then**

--Send amount sufficient message to SCOUTS Control;

send (SCOUTSControlMessageQ, amountSufficient);

**else** – amount Insufficient

send (SCOUTSControlMessageQ, amountInsufficient);

**end if;**

**end else if;**

**end if;**

**end loop;**

**Event Sequencing Logic for Daily Report Algorithm Task:**

Initialize for Daily report algorithm;

**loop**

-- Wait for interrupt from ReportTimer

**wait** (ReportTimerEvent);

Read Report timer message;

--Request ItemId, quantity and total revenue from TransactionLog object

TransactionLog.readTransaction(**out** Transaction Id, **out** itemId, **out** quantity, **out** totalItemSold)**;**

calculateRevenue(**in** Transaction Id, **in** itemId, **in** quantity, **in** total ItemSold **out** revenue);

generateReport( item sold, revenue);

--Send report to HeadquartersProxy;

send(HeadquarterProxyQueue, report);

**end loop;**

**Event Sequencing Logic for Card receipt Printer Interface Task:**

Initialize Card receipt Printer;

**loop**

-- Wait for external interrupt from SCOUTS control

**wait** (cardReceiptPrintEvent);

--Read received printOutput message;

**if** printOutput

**then**

-- Read last 4 digit card number and authorization code from Card object;

printOutputMessage = Card.readAuthorization(**out** cardNumber, **out** authorizationCode);

--Print the output message;

cardRecieptPrinter.print (cardNumber, authorizationCode);

**end if;**

**end loop;**

**Event Sequencing Logic for Customer order receipt Printer Interface Task:**

Initialize Customer order receipt Printer;

**loop**

-- Wait for external interrupt from SCOUTS control

**wait** (receiptPrintEvent);

--Read received printOutputMessage;

-- extract the parameters from the printOutputMessage

**if** parameters = unicost, description

**then**

– Print the output message

customerOrderReceiptPrinterInterface.print(unicost, description);

**elseif** parameters = total,tax

– Print the output message

customerOrderReceiptPrinterInterface.print(total,tax);

**elseif** parameters = weight,unitCost

– Print the output message

customerOrderReceiptPrinterInterface.print(weight,unitCost);

**Else** error condition;

**end if;**

**end loop;**

**Event Sequencing Logic for Supermarket Printer Interface Task:**

Initialize Supermarket Printer;

**loop**

-- Wait for external interrupt from SCOUTS control

**wait** (InventoryMessagePrintEvent);

--Read received printInventoryMessage;

**if** printInventoryMessage

**then** – Print the output message

superMarketPrinterInterface.print(printInventoryMessage);

**end if;**

**end loop;**

**Event Sequencing Logic for Store coupon Printer Interface Task:**

Initialize Supermarket Printer;

**loop**

-- Wait for external interrupt from SCOUTS control

**wait** (couponPrintEvent);

--Read received printCouponMessage;

-- Extract the printCouponMessage as output

**if** output

**then** – Print the output message

storeCouponInterface.print(storeCoupon);

**end if;**

**end loop;**

**Event Sequencing Logic for Headquarter proxy Task:**

Initialize Headquater;

**loop**

-- Wait for external interrupt from Headquaters or ReportTimer or DailyReportAlgorithm

**wait** (timerEvent or headquartersEvent or algorithmDemand);

--Read incoming message;

**if** timerEvent

**then**

– send inventory message to headquarters;

Send(headquartersQueue, message);

**elseif** algorithmDemand

**then**

– send inventory message to headquarters;

Send(headquartersQueue, message);

**elseif** headquartersEvent

**then**

– Update product information in Item object;

Item.updateItem(**in** itemid ,**in** unitCost**, in** description, **in** quantity, **in** discount, **in** tax);

– Update inventory information in Inventory object;

Inventory.updateItemdescrition(**in** itemid ,**in** unitCost**, in** description, **in** quantity, **in** discount, **in** tax);

**end if;**

**end loop;**

**Event Sequencing Logic for Authorization center proxy Task:**

Initialize Authorization center;

**loop**

-- Wait for external interrupt from Authorization center

**wait** (authorizationCenterEvent);

--Read incoming message;

**if** message = validAuth

**then**

-- send authorization code to SCOUTS Control;

send (SCOUTSControlMessageQ, authorizationCode);

**elseif** message = InvalidAuth

**then**

-- send invalid authorization to SCOUTS Control;

send (SCOUTSControlMessageQ, invalidAuthorization);

**else** error condition;

**end if;**

**loop**

--Receive message from SCOUTS Control

Receive (SCOUTSControlMessageQ, message);

--Send request to authorization center

Send(AuthorizationCenter, message);

**end loop;**

**end if;**

**end loop;**

**Event Sequencing Logic for SCOUTS control Task:**

**loop**

-- Messages from all senders are received on Message Queue

Receive (SCOUTSControlMessageQ, message);

-- Extract the event name and any message parameters

-- Given the incoming event, lookup state transition table;

-- change state if required; return action to be performed;

newEvent = message.event outstandingEvent = true;

**while** outstandingEvent **do**

SCOUTSStateMachine.processEvent (**in** newEvent, **out** action);

outstandingEvent = false;

-- Execute action(s)

**case** action **of**

Display key prompt: -- Display key prompt to Cashier

Send(PromptMessageQueue, getKeyInput);

Payment method prompt: -- Display payment method prompt to Cashier

Send(PromptMessageQueue, getPaymentMethod);

Display Item details: -- Display item details to Cashier and customer

Send(PromptMessageQueue, unitCostDescription);

Send(customerDisplayMessageQ, unitCostDescription);

Display Invalid item: -- Display invalid item to Cashier and customer

Send(PromptMessageQueue, invalidItem);

Send(customerDisplayMessageQ, invalidItem);

Display total and tax: -- Display total and tax to Cashier and customer

Send(PromptMessageQueue, totalTax);

Send(customerDisplayMessageQ, totalTax);

Display weight and unitcost: -- Display weight and unitcost to Cashier and customer

Send(PromptMessageQueue, weightUnitcost);

Send(customerDisplayMessageQ, weightUnitcost);

Display cancel checkout: -- Display cancel checkout to Cashier and customer

Send(PromptMessageQueue, cancelCheckout);

Send(customerDisplayMessageQ, cancelCheckout);

Display Coupon invalid: -- Display Coupon invalid to Cashier and customer

Send(PromptMessageQueue, couponInvalid);

Send(customerDisplayMessageQ, couponInvalid);

Get PIN: -- prompt for PIN

Send(CardReaderMessageBuffer, getPIN);

Display Invalid PIN: -- Display item details to Cashier and customer

Send(CardReaderMessageBuffer, invalidPIN);

Open till: -- send request to Till drawer interface to open till

Send(TillDrawerInterface, openTill);

Close till: -- send request to Till drawer interface to close till

Send(TillDrawerInterface, closeTill);

Check amount: -- send request to pay algorithm to check the paid amount

Send(payAlgorithmQ, amount);

Send(customerDisplayMessageQ, unitCostDescription);

Get Item: --Get item unitcost, description from Item

send (Item, Item, **in** itemId, **out** itemResponse);

newEvent = itemIdEvent; outstandingEvent = true;

Print order details: -- Print receipt and display details;

send (customerOrderReceiptPrinterInterface, **in** receiptInfo);

send (promptMessageQueue, displayItemCostandDescription);

newEvent = receiptPrint; outstandingEvent = true;

Check coupon: -- Check coupon validity and discount details

send (CouponBusinessLogic, **in** coupon, **out** discount);

newEvent = checkcouponResponse; outstandingEvent = true;

Print coupon: -- Print coupon;

send (storeCouponPrinterInterface, **in** couponInfo);

newEvent = couponPrint; outstandingEvent = true;

Calculate Total: -- Send request to TotalCalculationAlgorithm

send (TotalCalculationAlgorithm, **out** total, **out** tax);

newEvent = totalResponse; outstandingEvent = true;

Update inventory: -- Send request to decrease ordered items to Inventory

send (Inventory, **in** items, **out** InventoryStatus);

newEvent = inventoryResponse; outstandingEvent = true;

Update ordered list: -- Update Purchase order

send (**in** updateRequest);

newEvent = updateResponse; outstandingEvent = true;

Send Inventory message: --Send the inventory message

send (InventoryMessage, **in** iInventoryMessage);

Open till: -- Send request to open the till

send (tillDrawerInterface, **in** openTill)

Payment request: -- Send request to Payment subsystem

send (Payment, **in** paymentdetails, **out** paymentStatus);

newEvent = paymentResponse; outstandingEvent = true;

Get Order details: -- Request for purchase order list from Purchase order

send (**in** queryRequest, **out** queryResponse);

newEvent = purchaseOrderResponse; outstandingEvent = true;

Print inventory message: -- Print inventory message on SupermarketPrinter;

send (supermarketPrinterInterface, **in** inventoryMessage);

newEvent = InventoryMessagePrint; outstandingEvent = true;

Append Transaction log: -- Append the transaction log

send (transactionLog, **in** transactionData);

Check authorization: -- Request card authorization from Authorization center

send (authorizationCenter, **in** cardDetails, **out** authorizationStatus);

newEvent = authorizationResponse; outstandingEvent = true;

Request cashier data: -- Request cashier details from cashier account

send (cashierAccount, **in** cashierId, **out** Cashier details);

newEvent = authorizationResponse; outstandingEvent = true;

Get PIN: -- Prompt for PIN;

send (promptMessageQueue, displayPINPrompt);

Invalid PIN Action: -- Display Invalid PIN prompt;

send (promptMessageQueue, displayInvalidPINPrompt);

paymentTransaction.updatePINStatus (invalid);

Invalid authorization code: -- Display authorization code;

send (promptMessageQueue, displayInvalidAuthorizationCode);

Invalid item id: -- Display invalid item id;

send (promptMessageQueue, displayInvalidItem);

Invalid coupon: -- Display coupon error;

send (promptMessageQueue, displayInvalidCoupon);

Amount insufficient: -- Display amount insufficient message

send (promptMessageQueue, displayAmountInsufficient);

Payment success: -- send inventory message after update Inventory

Inventory.updateInventory(quantity, inventoryDetails);

send(inventoryMessageQ, message);

TransactionLog.append(TransactionLogQ, message);

Cancel checkout: -- Display Cancel checkout message

send (promptMessageQueue, displayCancelCheckout);

Update authorization code: -- Update authorization code linked to the card

send (card, in authorizationCode);

Print Card receipt: -- Print inventory message on SupermarketPrinter;

send (cardReceiptPrinterInterface, **in** lastFourDigit, **in** authorizationCode);

newEvent = cardReceiptPrint; outstandingEvent = true;

Eject: -- Eject debit/credit card;

Send(cardReaderInterface, eject);

Display Ejected: -- Display card ejected prompt;

Send(promptMessageQueue, displayEjected);

Send(customerDisplayMessageQ, displayEjected);

Display message: -- Display message on cashier keypad display to the Cashier

send (promptMessageQueue, welcome);

Send(customerDisplayMessageQ, welcome);

**end case;**

**end while;**

**end loop;**

**Event Sequencing Logic for SCOUTS Server:**

**loop**

receive (Client, Message) from Banking Service Message Queue; Extract message name and message parameters from message;

**case** Message of

Restocker Update Inventory:

--Restocker updates the item quantity in the inventory if item already exists

RestockerUpdateInventroyManager.UpdateInventory

(in itemID, in quantity, out restockerResponse);

‐‐If successful, restockerResponse is inventory updated

‐‐otherwise restockerResponse is update failed

reply (Client, restockerResponse);

Restocker add new item:

--If item does not exist, restocker enters the item details and updates inventory

RestockerUpdateInventroyManager.AddNewItem

(in itemID, in description, in price, in discount, in quantity, out restockerResponse);

‐‐If successful, restockerResponse is new item added and inventory updated

‐‐otherwise restockerResponse is error adding new item

reply (Client, restockerResponse);

Cashier Update Purchase Inventory:

--Cashier updates the inventory when each purchase is done

CashierUpdatePurchaseInventoryManager.UpdatePurchaseInventory

(in itemID, in quantity, out cashierResponse);

‐‐If successful, cashierResponse is updated inventory

‐‐otherwise cashierResponse is update failed

reply (Client, cashierResponse);

Cashier read item:

--Cashier reads item details

CashierUpdatePurchaseInventoryManager.ReadItem

(in itemID, out itemDetails);

‐‐If successful, cashierResponse is item details

‐‐otherwise cashierResponse is error

reply (Client, cashierResponse);

Cashier read coupon:

--Cashier reads coupon data

CashierUpdatePurchaseInventoryManager.ReadCoupon

(in itemID, out couponData);

‐‐If successful, cashierResponse is coupon data

‐‐otherwise cashierResponse is error

reply (Client, cashierResponse);

Cashier update transaction log:

--Cashier updates transaction log

CashierUpdatePurchaseInventoryManager.UpdateTransactionLog

(in transactionDetails, out cashierResponse);

‐‐If successful, cashierResponse is transaction log updated

‐‐otherwise cashierResponse is update failed

reply (Client, cashierResponse);

Cashier update inventory message:

--Cashier updates inventory message

CashierUpdatePurchaseInventoryManager.UpdateInventoryMessage

(in inventoryMessages, out cashierResponse);

‐‐If successful, cashierResponse is inventory message updated

‐‐otherwise cashierResponse is update failed

reply (Client, cashierResponse);

Manager View Inventory:

--Manager views the product inventory

ManagerViewUpdateInventoryManager.ViewProductInventory

(in itemID, out itemDetails);

‐‐If successful, managerResponse is item details

‐‐otherwise managerResponse is error

reply (Client, managerResponse);

Manager Update Inventory:

--Manager updates the product inventory

ManagerViewUpdateInventoryManager.UpdateProductInventory

(in itemDetails, out managerResponse);

‐‐If successful, managerResponse is product inventory updated

‐‐otherwise managerResponse is update failed

reply (Client, managerResponse);

**end case;**

**end loop;**