**Name: Tao Zhang Student No. 7389480**

**The Self-service Presto Card Recharging Machine**

**Part 1**

1. **Description of the system**

This system provide service of helping customers recharge their presto card by themselves instead of waiting in the long queue.

If a customer press a “Recharge Button”, the system will ask him/her to insert the presto card and money. After the customer press the “Confirm” button. The money will deposit into customer’s presto card and a receipt will be printed out and sent to customer.

1. **A brief sketch of the model structure.**



1. **A brief description for each component**

Money Collector: Collect each coin or note and output the amount of money.

Presto Card Reader: Read the information such as the card no, name and the current balance from the Presto Card and forward this information to the Deposit Processor.

Balance Display: Display the amount of money the customer is about to deposit, and display the initial and the final balance of the Presto Card.

Deposit Processor: After receiving the confirm signal, the processor will deposit money which the customer just put in the machine into his/her Presto Card and output the current balance and the corresponding information for the printer.

Receipt Printer: After receiving the information from the Deposit processor, the printer will print out the receipt for the transaction.

**PART 2**

* **Detailed structure**

The Presto Card Recharging Machine consists of six atomic models and one coupled models. Atomic models include MoneyCollector, CardReader, BalanceDisplay, MessageQueue, CentralController,and ReceiptProcessor. The coupled model is RechargeController. The following is the detailed structure of the Presto Card Recharging Machine.



* **Formal specification for each atomic model and coupled model**
  1. *Formal specification for MoneyCollector*

X={money ∈ <N, R>};

Y={money\_amount ∈ <N, R>};

S={phase, sigma, money, preparation\_time};

δext(s,e,x)

{

case phase

passive:

sigma=preparation\_time;

get money\_amount;

phase = busy;

busy:

continue;

}

δint(s,e)

{

case phase

busy: passivate;

passivate: /\*never happens\*/

}

λ(s)

{

Send money\_amount to the port out

}

* 1. *Formal specification for CardReader*

X={card ∈ <N, R>};

Y={card\_info ∈ N};

S={phase, sigma, card\_info<card.id, card.balance>, preparation\_time};

δext(s,e,x)

{

case phase

passive:

sigma=preparation\_time;

get card\_info;

phase = busy;

busy:

continue;

}

δint(s,e)

{

case phase

busy: passivate;

passivate: /\*never happens\*/

}

λ(s)

{

Send card\_info to the port out

}

* 1. *Formal specification for BalanceDisplay*

X={balance ∈ <N, R>};

Y={balance\_display ∈ <N, R>};

S={phase, sigma, balance, preparation\_time};

δext(s,e,x)

{

case phase

passive:

sigma=preparation\_time;

save balance;

phase = busy;

busy:

continue;

}

δint(s,e)

{

case phase

busy: passivate;

passivate: /\*never happens\*/

}

λ(s)

{

Send balance\_display to the port out

}

* 1. *Formal specification for ReceiptProcessor*

X={request\_in, central\_control\_in ∈ <N, R>};

Y={receipt\_display, request\_out ∈ <N, R>};

S={phase, sigma, request, card\_info<card.id, card.balance>, preparation\_time};

δext(s,e,x)

{

case port

request\_in:

send request to the port request\_out;

central\_control\_in:

save card\_info;

sigma=preparation\_time;

phase=busy;

}

δint(s,e)

{

case phase

busy: passivate;

passivate: /\*never happens\*/

}

λ(s)

{

Send card\_info to the port receipt\_display

}

* 1. *Formal specification for MessageQueue*

X = {MoneyCollector\_in∈ < N, R >, CardReader\_in∈ < N, R >, ReceiptRequest\_in ∈ < N, R > , CentralControl\_done∈ < N > };

Y = { message\_out ∈ < N, R >};

S = { Phase, sigma, message\_queue, message, preparationTime }

δext ( s, e, x )

{

case port

message\_in /\* this message includes the message from MoneyCollector,

CardReader\_in and ReceiptRequest \*/

push\_back(message\_queue);

sigma = preparationTime;

phase = busy;

done:

pop\_front ( message\_queue );

if (message\_queue!= 0);

phase = busy;

sigma = preparationTime;

}

δint ( s, e )

{

case phase

busy: passivate;

passive: /\*never happens\*/

}

λ (s)

{

send message to the port message\_out;

}

* 1. *Formal specification for CentralControl*

X = { MessageQueue\_in ∈ < N, R >};

Y = { ReceiptProcessor\_out ∈ < N, R>, BalanceDisplay\_out∈ < N, R >, CentralControl\_done ∈ < N > };

S = { Phase, sigma, balance, card\_info, preparationTime, serviceTime }

δext ( s, e, x )

{

message\_in:

if (message.value <10 ) /\* this message is from MoneyCollector\*/

save balance;

sigma = preparationTime;

phase = busy;

elseif (message.value >=10) /\* this message is from CardReader\*/

save the card\_info;

elseif ( message.value = -1) /\*this message is from ReceiptProcessor\*/

card\_info.balance =card\_info.balance+MoneyCollector.balance;

sigma = preparationTime;

phase = busy;

}

δint ( s, e )

{

case phase

busy: passivate

passivate: /\* never happens\*/

}

λ (s)

{

send card\_info to the port receipt\_out;

send balance to the port balance\_out;

send done to the port done;

}

* 1. *Formal specification for RechargeController*

RechargeController = < X, Y, { MessageQueue, CentralControl }, EIC, EOC, IC, SELECT >

X = { money\_in, card\_in, receipt\_request\_in}

Y = { balance\_out, receipt\_out }

EIC = { ( MoneyCollector.out, money\_in), (CardReader.out, card\_in), (ReceiptProcessor.out, receipt\_request\_in)}

EOC = { (balance\_out, BalanceDisplay.in), (receipt\_out, ReceiptProcessor.in) }

IC = { (money\_in, MessageQueue.in), (card\_in, MessageQueue.in), (receipt\_request\_in, MessageQueue.in), (CentralControl.out, balance\_out), (CentralControl.out, receipt\_out), (CentralControl.out , MessageQueue.done), (MessageQueue.out, CentralControl.in)}

SELECT : ({MessageQueue, CentralControl}) = MessageQueue

**PART 3**

**Simulation Results.**

Prior to explaining the simulation input and output, a list of values that passed into the simulation are predefined

**money\_in**: money value inserted into the Money Collector. Range from 5 to 100

**card\_in**: the value is always 1 which represents a card has been inserted into the Card Reader. Inside the Card Reader, a three-digit card number is randomly generated, ranging from 100 to 999.

**request\_in:** receipt print request, value is always –1

The following is the couple.ev and its corresponding output file after running the simulation.

**Couple.ev**

00:00:01:00 card\_in 1 // the card was insert into the Card Reader

00:00:10:00 money\_in 10 // insert 10 dollars into the Money Collector

00:00:15:00 money\_in 20 // insert 20 dollars into the Money Collector

00:00:20:00 money\_in 20 // insert 20 dollars into the Money Collector

00:00:35:00 request\_in -1 // request for the receipt

00:02:30:00 money\_in 50 // insert 50 dollars into the Money Collector

00:02:33:00 money\_in 100 // insert 100 dollars into the Money Collector

00:02:40:00 request\_in -1 // request for the receipt

**Coupleout**

00:00:03:000 balance\_out 0 // the initial balance of Presto Card is 0

00:00:12:000 balance\_out 10 // the current balance is 10

00:00:17:000 balance\_out 30 // the current balance is 30

00:00:22:000 balance\_out 50 // the current balance is 50

00:00:36:000 balance\_out 50 // the current balance is 50

00:00:37:000 request\_out 706 // return the card number 706

00:02:32:000 balance\_out 100 // the current balance is 100

00:02:35:000 balance\_out 200 // the current balance is 200

00:02:41:000 balance\_out 200 // the current balance is 200

00:02:42:000 request\_out 706 // return the card number 706

The results show that simulation is correct. And a number of different testing cases are performed to ensure the correctness of the simulation. Please refer to the zip file attached for more testing cases performed and their results.

Individual testing cases.

Before assembling all atomic models together, individual testing cases are being performed to verify the correct behavior of each model. The following are a few sample testing cases performed on selected model.

E.g. the EV file for atomic model MoneyCollector.

00:00:01:05 money\_in 5

00:00:07:10 money\_in 5

00:00:13:25 money\_in 20

00:00:20:40 money\_in 10

00:00:30:60 money\_in 10

00:00:40:60 money\_in 20

“money\_in” represents the input port of MoneyCollector, the value “5” represents the money value.

Each model is tested based on its functionality, inputs and outputs.

e.g. in the testing of the Message Queue, different input ports accept user requests and queued up the messages. A message is released when a done message is received from the Central Controller. Testing cases are performed to make sure there is no message lost during the queueing time and there is no message released until requested by the Central Controller via the ‘done’ message.