UNIVERSITÉ Concordia UNIVERSITY

INSE 6250 – QUALITY METHODOLOGIES FOR SOFTWARE

WINTER 2020

A PROJECT REPORT ON "MODELING AND VERIFICATION OF WEB-BASED TICKET VENDING MACHINE FOR MONTREAL METRO"

Submitted By

ADARSH ARAVIND

40082585

Under the Guidance of

DR. JAMAL BENTAHAR

Date of Submission

09-04-2020

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Ticket Vending Machine is a vending machine that produces paper or electronic tickets or recharges a stored-value card or smart card or the user's mobile wallet, typically on a smartphone. For instance, ticket machines dispense train tickets at railway stations, transit tickets at metro stations, tram tickets at some tram stops. The typical transaction consists of a user using the display interface to select the type and quantity of tickets and then choosing a payment method of either cash, credit/debit card or smartcard. The ticket(s) are then printed on a paper and dispensed to the user, or loaded onto the user's smartcard or smartphone. Ticket vending machines are used since the late 1920s for many of its benefits. There are also some disadvantages like long queues, out of service, etc. The goal of this project is to eliminate the problems faced by the commuters and build a web-based ticket vending machine model which will be designed according to Société de Transport de Montreal (STM) machines.

In Montreal, people must top up their OPUS card balance at the beginning of every month via physical Société de transport de Montréal (STM) Ticket Vending Machines (TVM) placed in STM stations. To top up their OPUS card, people often wait in long queues in front of TVM machines and STM office stations. The travelling process could be interrupted because people usually forget to top up their OPUS card on time. To avoid this, the web-based application plays as an online platform allowing travellers to top up their OPUS cards and manage STM transactions themselves. People can connect and top up their OPUS cards or buy tickets online using their desktops or mobile devices at home. The project introduces the concept of an online web application to provide an alternative solution for STM's physical TVM.

However, the verification of such automated models remains the challenge that needs to be solved. As the web applications provide various interesting features and they are used by numerous users, the quality and correctness of these applications are very important. Kripke structure is constructed to model the system, the mathematical model of which is represented by a Finite State Machine (FSM) from my perspective. UPPAAL is used for modeling, verification, and validation of the system. UPPAAL provides a toolbox to verify real-time systems and has been successfully used in case studies of communication protocols and multimedia applications. To perform model checking I'm using CTL formal language to specify properties and verify them using the model checking tool. UPPAAL performs automatic verification and locates errors in application design in the form of counterexamples when my model does not satisfy a certain property.

1. INTRODUCTION

In Montreal, almost everyone uses the metro for their everyday travel and most of the people purchase tickets rather than recharging their OPUS card for a month. Even when the commuters are in a hurry, they have to wait in long queues to buy their travel tickets which takes a lot of time. This problem can be avoided by web-based ticket vending machines where they can recharge their card from anywhere and at any time. This not only avoids long queues, but travellers can keep track of their transaction history, update their profile and can go cashless. When offering all these excellent features, such applications/systems should be verified before going into the market. It should work as per the requirements planned without any errors. The accuracy of such web applications is of utmost importance as the failure often causes user dissatisfaction. Therefore, the verification of such systems is vital to ensure the quality of the service features. Specification of these online applications takes considerable time to be decided to design web pages in a user-friendly and appealing way while navigating through the application. So, in my work, I'll model a system and check its correctness using CTL on a tool called UPPAAL.

1.1 FORMAL METHODS AND MODEL CHECKING

The formal testing method involves running a program with a set of inputs and comparing the actual outputs from the program against the expected outputs. There are several limitations to using testing as the sole approach to software error detection. Testing cannot take place until some implementation is available. Correcting errors uncovered by testing could involve retracing many steps and undoing work previously done. If testing is the only approach to error detection then errors in the specification involve the greatest amount of work to rectify. By considering all the drawbacks of testing, industries use formal methods which are more than just a specification language - it also includes a proof system for demonstrating that each design step preserves the formal meaning captured in the previous step. Important properties (such as internal consistency) of the initial specification can be checked mathematically and incorporated as run-time checks in the final program. Proof of program correctness can be constructed that is a much more robust method of achieving program correctness than is testing alone.

In a model-based approach, an abstract mathematical model is built of the data, using abstract mathematical types such as abstract state machines. The behaviour of the operations is then specified directly concerning this model. A formal method like Model Checking has become popular as it verifies the state transition system and executes an automated verification of a system model concerning its specification. Different from testing which needs user communication, model checking is a model-based, property-verification, automatic method to develop high assurance systems. The other advantage of model checking is that there are several efficient implementation tools such as UPPAAL, NuSMV and SPIN, which had successfully verified large real-world systems.

1.2 UPPAAL

UPPAAL is an integrated tool environment for modelling, validation, and verification of real-time systems modelled as networks of timed automata, extended with data types. The main purpose of a model checker is to verify the model for a requirement specification. Like the model, the requirement specification must be expressed in a formally well-defined and machine-readable language. Like most of the model checkers, even UPPAAL support three path formulae expressed by temporal logic quantifiers:

- **Reachability:** Is the state formula Ψ satisfied from any reachable state?
- **Safety:** Ψ is invariantly true in all reachable states.
- **Liveness:** Ψ is eventually satisfied.

UPPAAL has 3 core parts: Editor, Simulator, and Verifier. The system editor is used to create and edit the system model to be analyzed. It describes a network of a finite number of non-deterministic finite-state automata. The simulator can be used to run the system manually and choose the transitions to take, or we can go through the trace generated by the verifier. And finally, as mentioned earlier, the verifier is used to check the correctness of the model being built. The requirements are converted into CTL formulas before executing them on the UPPAAL verifier.

There are a few reasons for me to choose UPPAAL over other model checkers. Unlike many other tools, UPPAAL will provide counterexamples when a property isn’t satisfied. This helps a developer to detect and rectify errors in the system, Properties can be easily specified as CTL formulas, System can be modelled graphically as finite-state automata.

2. MODEL

In this project, I have created 3 templates namely TVM_Server, Bank, and the User. It is a timed automaton where a user will have session timeout after a certain duration. Users will have to sign up for the first time and then sign in with the same credentials for every login. The user gets two options to login (through the Phone number or Email ID). The TVM_Server will authenticate the user details and lets the user in if the credentials are valid, else the user gets error notification and has to login again. Once the authentication process has been complete, the user will get four options from the TVM_Server and he will have to choose any one option in that transition. Users can either view his/her profile, Purchase OPUS card, Buy ticket or View transaction history. Users can always update their information and save it into the system. Once the user purchases the OPUS card, he/she will have two options (MonthlyPass and YearlyPass) and if the user wants to buy tickets, the system will provide the user with three sub-options namely OneWayTicket, DailyTicket, WeeklyTicket. Users can also clear or view their transaction history anytime once they sign into the system. After each transition, the user will have to sign in to the system again, to begin with, a new transition.

Below is the screenshot of the TVM_Server model.

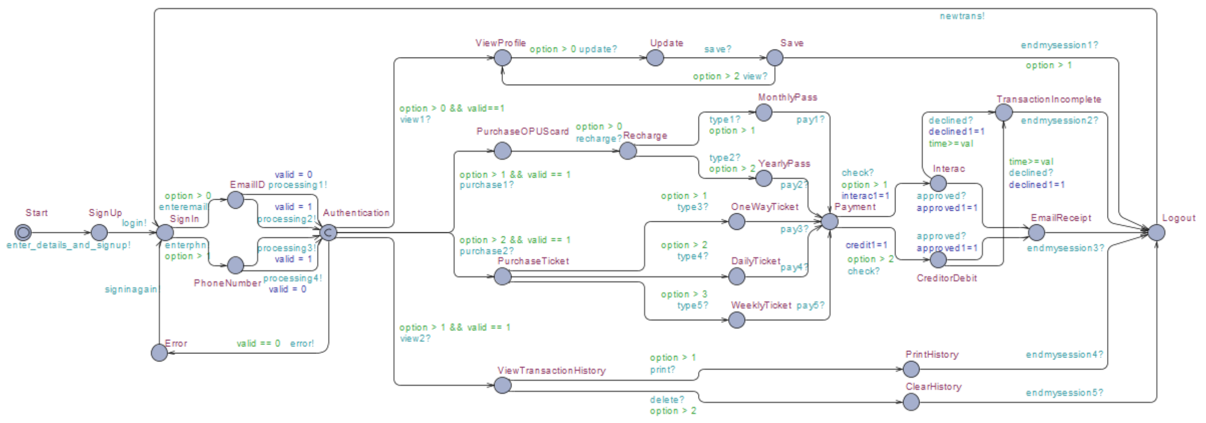


Figure 1: TVM_Server template

The user template is similar to TVM_Server with few changes and synchronization. Synchronization is used mainly to show how the user will interact with the TVM_Server and how the TVM_Server will interact with the Bank template. The message passing can be seen in the simulation of the model. During the payment process, the user will have a session timeout condition where the user can stay on that page for a certain time.

Below is the screenshot of the Bank template which is connected with the TVM_Server to authorize the user's account.

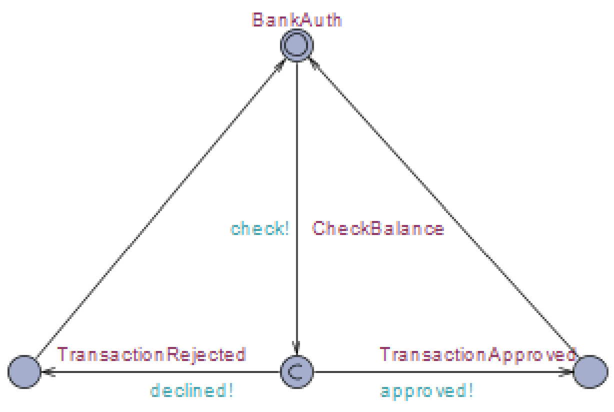
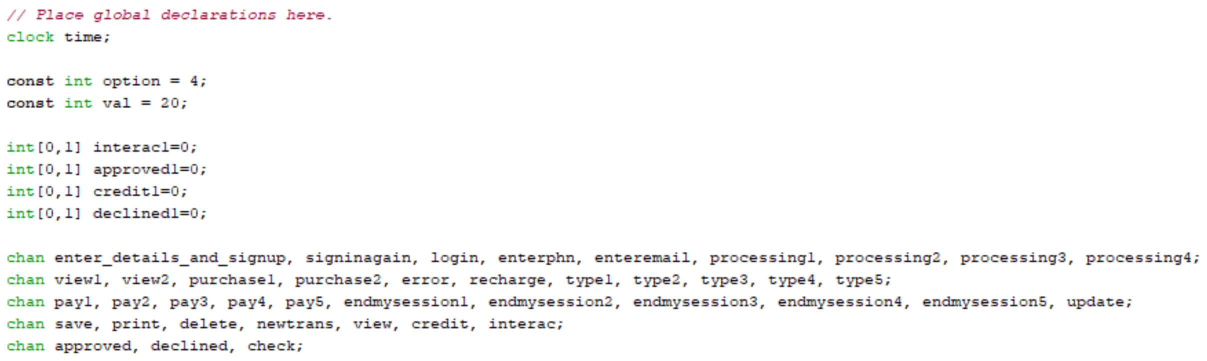


Figure 2: Bank template

Payment can be done either by Interac or using CreditorDebit card options. The payment state is connected to the bank template where the bank will check if the user has sufficient balance in their account to make the payment. If the user has sufficient balance, the transaction will be approved by the bank else it'll be rejected. If the transaction is complete, the User can get the transaction receipt via an email.

2.1 DECLARATIONS

For synchronization purposes, I have user different channels which are shown in the below screenshot.



```
// Place global declarations here.
clock time;

const int option = 4;
const int val = 20;

int[0,1] interac1=0;
int[0,1] approved1=0;
int[0,1] credit1=0;
int[0,1] declined1=0;

chan enter_details_and_signup, signinagain, login, enterphn, enteremail, processing1, processing2, processing3, processing4;
chan view1, view2, purchase1, purchase2, error, recharge, type1, type2, type3, type4, type5;
chan pay1, pay2, pay3, pay4, pay5, endmysession1, endmysession2, endmysession3, endmysession4, endmysession5, update;
chan save, print, delete, newtrans, view, credit, interac;
chan approved, declined, check;
```

Figure 3: Global declarations

Integer variable called val which is constant throughout the process is used for the session timeout. It is compared with the clock time during the bank transaction and if the value is greater than the clock time, the system shuts down and the user will be redirected to the session timeout page where he will be asked to log in and start the transaction from

the beginning. Other integer variables like Interac, approved, credit, declined are used to allow or reject user's access to certain options in the transitions. A constant variable option is used in the TVM_Server to let the user select among the four choices.
Channels approved, declined, and check is used for the synchronization between the Bank and the TVM_Server. And all the other channels are used for the synchronization between the TVM_Server and the User.

3. REQUIREMENTS and PROPERTIES

To begin with the development of any software, we need a set of requirements. Most of the industries rely on user stories and/or use cases to start the development process. Requirements should contain information about how the software will work and interact with the user. Likewise, even in this project, I came up with a set of requirements that the model has to satisfy to be functional.
Below is the list of properties which the model has to satisfy:

⇒ The system should allow the user to login with either Phone number or Email ID.

⇒ The system should authenticate user credentials after login.

⇒ If the user credentials are not valid, the system should send an error notification which allows the user to log in again.

⇒ The system should provide the user with four options after a successful login.

⇒ Users should be able to choose only one option among the four in a transaction.

⇒ If the user updates his/her profile, it should be saved before logout.

⇒ Once the user purchases OPUS card, the system should provide two types of recharge options (Monthly pass and Yearly pass)

⇒ If the user purchases tickets, the system should provide the user with three options (One-way ticket, Daily ticket, and Weekly ticket)

⇒ When a user selects view transaction history options, he/she should be able to either print the history or delete it, but not both.

⇒ After purchasing the OPUS card or a ticket, the user should make a payment before login.

⇒ The bank should authenticate the user's account to check if there is a sufficient amount to make a payment and should inform the TVM server.

- ⇒ If the user is out of balance, then the system shouldn't allow the user to purchase anything.
- ⇒ User should be given two options to make the payment (Interac and Credit or Debit card)
- ⇒ If the value of val is greater than the clock time, the server should automatically shut down allowing the user to log in again. This is due to a session timeout.
- ⇒ If the user makes a payment, the receipt should be sent by the server via an email to the user.
- ⇒ If the user is out of balance, then the transaction should not be continued.
- ⇒ For every new transaction, the system should allow the user to log in again.

The model was built according to the above-mentioned specifications in UPPAAL using synchronization between the templates. Any property of software can be broadly categorized into three commonly known properties (Safety property, Liveness property, and Reachable property). Even in the model which I have built, all these properties hold true.

3.1 SAFETY PROPERTIES

Safety properties cover those cases which should not occur in the system. These are the properties that check for the illegal transitions in the model to check the unexpected behaviour of the system. Even in this model, there are a few properties that should not occur during a transaction. For example,

- The user cannot bypass the authentication process.
- Users cannot buy both the OPUS card and the ticket at the same time.
- The system shouldn't allow the users to logout without making a payment for their purchases.
- Users should be able to choose only one among the four given options, but not multiple.

3.2 LIVENESS PROPERTIES

In an application, all the paths should be covered at least once i.e., there must be coverage of all the transitions in the system and it should not be the case that a transition is never covered in the system. For instance,

- If a user login to the system and selects the view profile option, he/she will be able to update and save it before logout.
- If a user logs in to the system and selects purchase OPUS card, then the user can recharge and make a payment if there is a sufficient amount in his/her account which allows the system to generate and email a copy of the transaction receipt before the user logout from the system.
- Likewise, when a user purchases a ticket and if he/she won't have enough balance, the bank should inform the server and the transaction should be incomplete which forces the user to logout from the application.
- If the user selects view transaction history option, he/she can clear or print the history and logout from the system.

3.3 REACHABILITY PROPERTIES

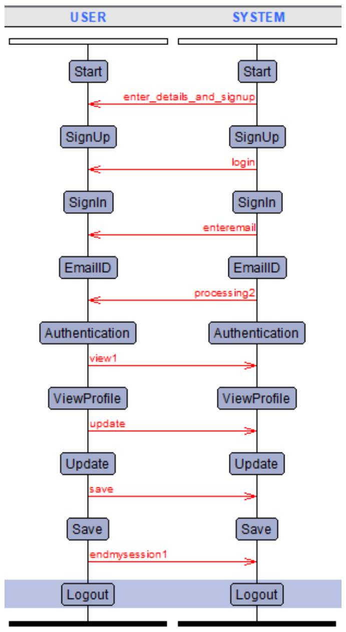
All the states in the system must be reachable by the users at least once at some point in time. Reachability property in a state is defined by E<>, meaning the state will be accessible in at least one path in the future. For instance,

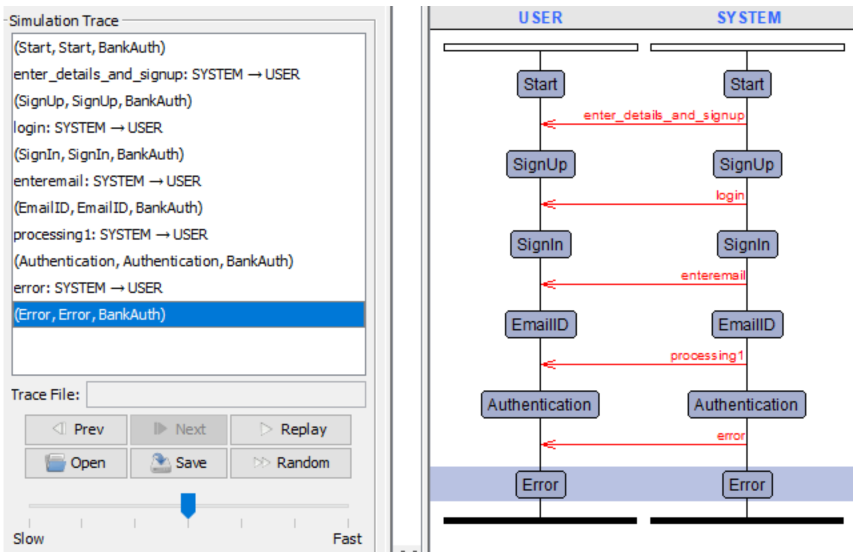
- Once the user logs in to the system, he/she should be able to logout from the system in the future.
- If a user purchases anything, he/she should be able to make a payment in the future.
- After a particular transaction, if the user wishes to perform any other activity, he/she should be allowed to log in again for a new transaction.

The goal is to satisfy all these properties in the model.

4. SIMULATION

In UPPAAL or any other model checker tool, simulation traces can be seen which helps to trace a particular transition. This is helpful when the system doesn’t work as expected. If we trace the transition, it’s easier to find the error and can model the system accordingly. Below is a screen capture (Figure 4) of the simulation where the user selects view profile option and he updates the information. Figure 5 shows the wrong credentials entered by the user. As modeled, the server will throw an error message to the user so that the user will be informed to login with the correct credentials. Simulation trace can be seen in figure 5. Message passing between the templates can also be seen in the diagrams.



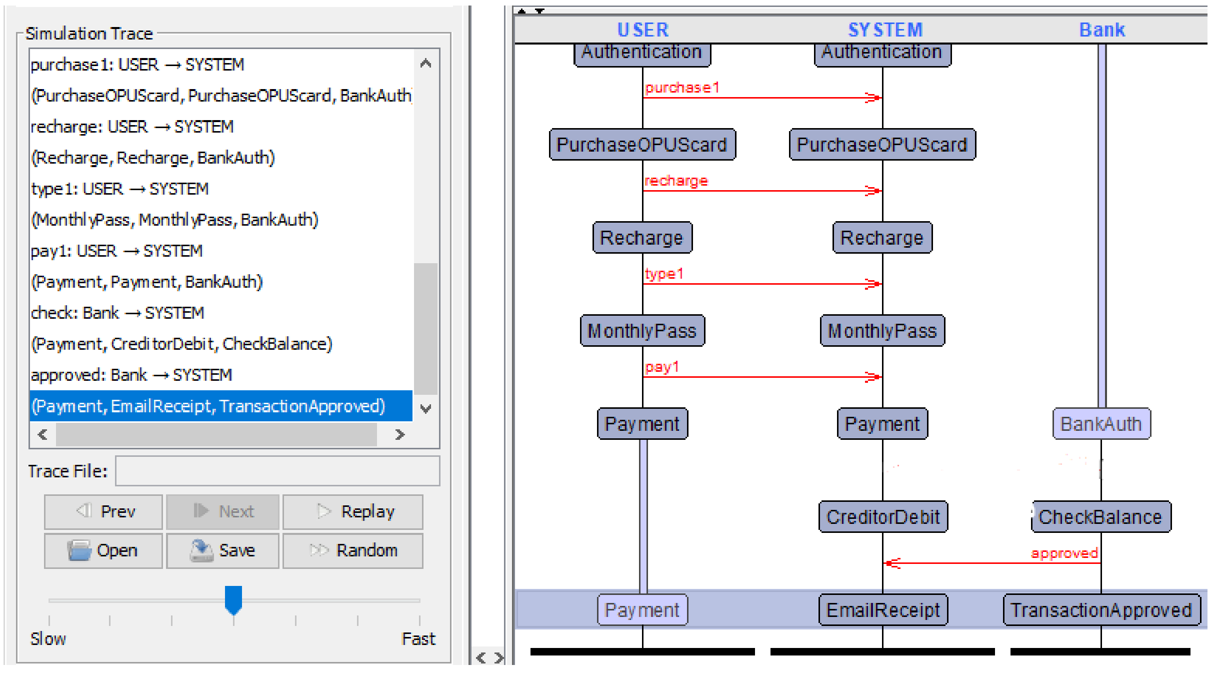

Simulation Trace

```
(Start, Start, BankAuth)
enter_details_and_signup: SYSTEM → USER
(SignUp, SignUp, BankAuth)
login: SYSTEM → USER
(SignIn, SignIn, BankAuth)
enteremail: SYSTEM → USER
(EmailID, EmailID, BankAuth)
processing1: SYSTEM → USER
(Authentication, Authentication, BankAuth)
error: SYSTEM → USER
(Error, Error, BankAuth)
```

Trace File:

Figure 4: View profile **Figure 5: Authentication error**

Below is a screenshot of the simulation when the transaction has been approved by the bank when the user selects an option to recharge the OPUS card.


Simulation Trace

```
purchase1: USER → SYSTEM
(PurchaseOPUScard, PurchaseOPUScard, BankAuth
recharge: USER → SYSTEM
(Recharge, Recharge, BankAuth)
type1: USER → SYSTEM
(MonthlyPass, MonthlyPass, BankAuth)
pay1: USER → SYSTEM
(Payment, Payment, BankAuth)
check: Bank → SYSTEM
(Payment, CreditorDebit, CheckBalance)
approved: Bank → SYSTEM
(Payment, EmailReceipt, TransactionApproved)
```

Trace File:

Figure 6: Transaction approved

These simulations are helpful in situations where the system doesn't satisfy certain property and gives a counterexample. In such times, we can easily find where the model isn't working as expected.

5. VERIFICATION

In any development activity, it is important to check the correctness of the software. In my project, to check the correctness of the model, I have used CTL to convert and verify the properties of the model. And in this section, I'll show the properties and equivalent CTL queries that can be verified in UPPAAL.

In UPPAAl, it's possible to verify only two kinds of properties (Reachable properties and Liveness properties)

Therefore, I have used E<> logical expression in the form of CTL to define the reachability properties and all the properties that are listed in table 1 are satisfied.

The notations that can be used to represent A[], E[], A<> and E<> operators of CTL are shown in table 2 and all the mentioned properties are verified.

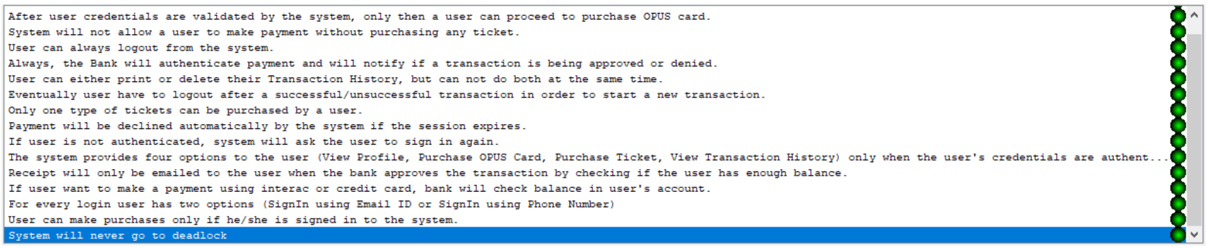
Property No.	Property Name	Property Definition
1	E<>(SYSTEM.SignUp)	It covers the SignUp state at least once in the future.
2	E<>(SYSTEM.SignIn)	It covers the SignIn state at least once in the future.
3	E<>(SYSTEM.Authentication)	It covers the Authentication state at least once in the future.
4	E<>(SYSTEM.ViewProfile)	It covers the ViewProfile state at least once in the future.
5	E<>(SYSTEM.PurchaseOPUScard)	It covers the PurchaseOPUScard state at least once in the future.
6	E<>(SYSTEM.PurchaseTicket)	It covers PurchaseTicket state at least once in the future.
7	E<>(SYSTEM.ViewTransactionHistory)	It covers ViewTransactionHistory state at least once in the future.
8	E<>(SYSTEM.Logout)	It covers the Logout state at least once in the future.

Table 1: CTL queries representing Reachability properties

Property No.	Property Name	Property Definition
1	A[] not deadlock	This property checks if there's a deadlock in the system.
2	A<> SYSTEM.option>1 && USER.valid == 1 imply SYSTEM.PurchaseOPUScard	After user credentials are validated by the system, only then a user can proceed to purchase OPUS card.
3	E<> (USER.SignIn and option == 1)	This property checks if the user has logged.
4	E<> SYSTEM.Logout	This property checks if the user can always logout from the system.
5	A<> (time>=val imply SYSTEM.TransactionIncomplete)	This property checks if the transaction was completed.
6	A<> SYSTEM.PurchaseOPUScard || SYSTEM.PurchaseTicket imply USER.SignIn	This property checks if a user can make purchases only if he/she is signed in to the system.
7	E<> ((USER.TransactionIncomplete || USER.EmailReceipt) imply USER.Logout) imply USER.SignIn	User should logout after successful transaction in order to start a new transaction.

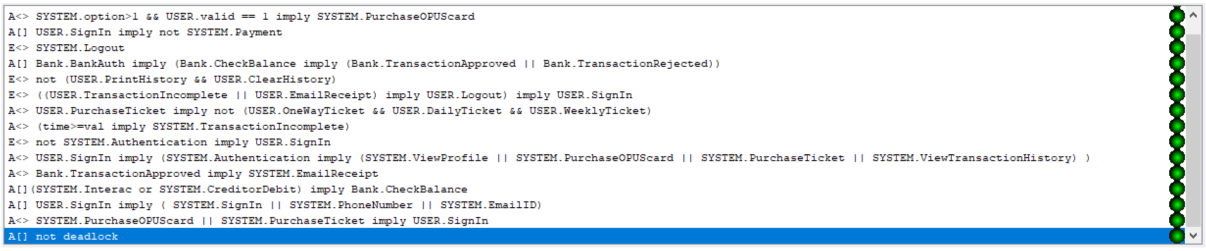
Table 2: CTL queries representing Liveness properties

Below is the screen capture of the properties verified in UPPAAL



```
After user credentials are validated by the system, only then a user can proceed to purchase OPUS card.
System will not allow a user to make payment without purchasing any ticket.
User can always logout from the system.
Always, the Bank will authenticate payment and will notify if a transaction is being approved or denied.
User can either print or delete their Transaction History, but can not do both at the same time.
Eventually user have to logout after a successful/unsuccessful transaction in order to start a new transaction.
Only one type of tickets can be purchased by a user.
Payment will be declined automatically by the system if the session expires.
If user is not authenticated, system will ask the user to sign in again.
The system provides four options to the user (View Profile, Purchase OPUS Card, Purchase Ticket, View Transaction History) only when the user's credentials are authent...
Receipt will only be emailed to the user when the bank approves the transaction by checking if the user has enough balance.
If user want to make a payment using interac or credit card, bank will check balance in user's account.
For every login user has two options (SignIn using Email ID or SignIn using Phone Number)
User can make purchases only if he/she is signed in to the system.
System will never go to deadlock
```

Figure 7: Properties in UPPAAL



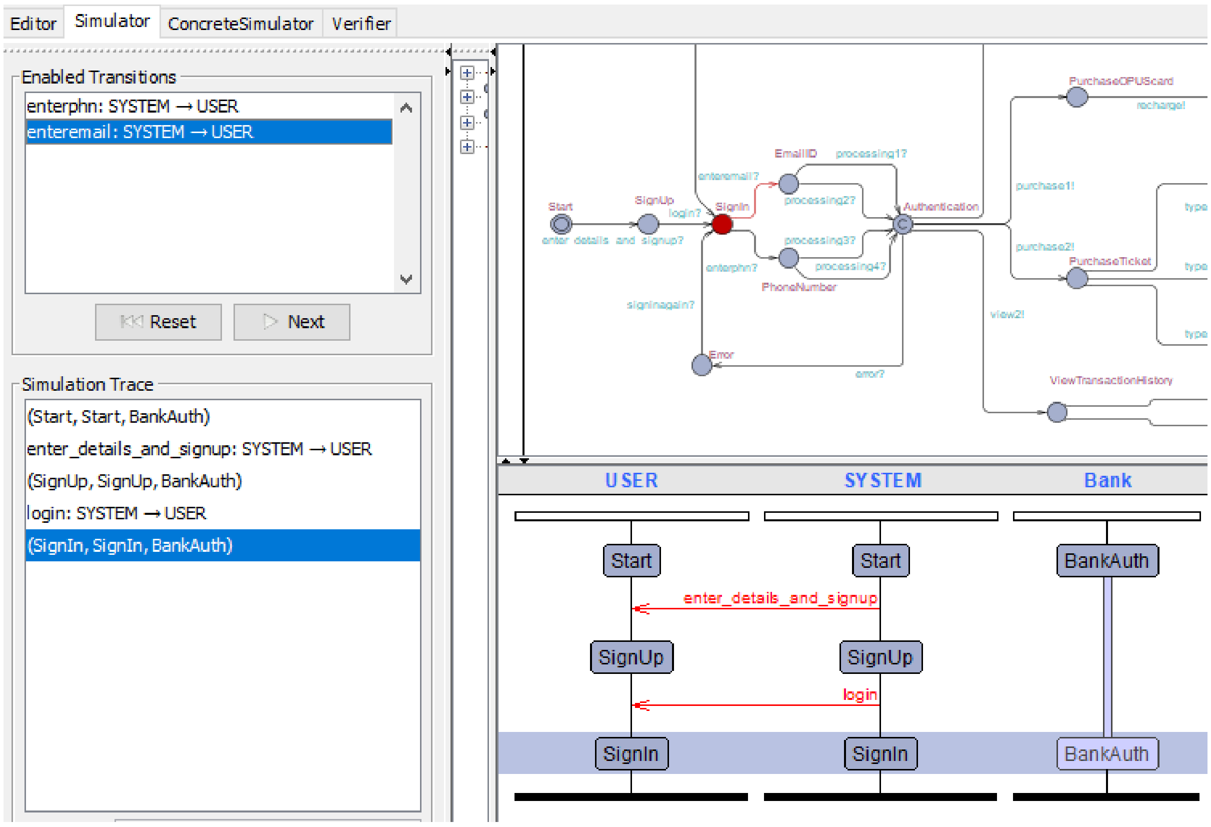
```
A<> SYSTEM.option>1 && USER.valid == 1 imply SYSTEM.PurchaseOPUScard
A[] USER.SignIn imply not SYSTEM.Payment
E<> SYSTEM.Logout
A[] Bank.BankAuth imply (Bank.CheckBalance imply (Bank.TransactionApproved || Bank.TransactionRejected))
E<> not (USER.PrintHistory && USER.ClearHistory)
E<> ((USER.TransactionIncomplete || USER.EmailReceipt) imply USER.Logout) imply USER.SignIn
A<> USER.PurchaseTicket imply not (USER.OneWayTicket && USER.DailyTicket && USER.WeeklyTicket)
A<> (time>=val imply SYSTEM.TransactionIncomplete)
E<> not SYSTEM.Authentication imply USER.SignIn
A<> USER.SignIn imply (SYSTEM.Authentication imply (SYSTEM.ViewProfile || SYSTEM.PurchaseOPUScard || SYSTEM.PurchaseTicket || SYSTEM.ViewTransactionHistory) )
A<> Bank.TransactionApproved imply SYSTEM.EmailReceipt
A[](SYSTEM.Interac or SYSTEM.CreditorDebit) imply Bank.CheckBalance
A[] USER.SignIn imply ( SYSTEM.SignIn || SYSTEM.PhoneNumber || SYSTEM.EmailID)
A<> SYSTEM.PurchaseOPUScard || SYSTEM.PurchaseTicket imply USER.SignIn
A[] not deadlock
```

Figure 8: CTL queries

All the properties which I had planned initially for web-based Ticket Vending Machine were converted to CTL queries and were satisfied on UPPAAL.

5.1 COUNTEREXAMPLE

It is also important to note that the system will not satisfy certain conditions. And these can be avoided when testing activity is performed. Likewise, in my project, UPPAAL gives a counterexample for the properties that are not satisfied. Consider an example where the user will be allowed to log in to the system by entering both Email ID and phone number which is not possible. For this property, below is the attached screen capture.



```
Enabled Transitions
enterphn: SYSTEM → USER
enteremail: SYSTEM → USER

Simulation Trace
(Start, Start, BankAuth)
enter_details_and_signup: SYSTEM → USER
(SignUp, SignUp, BankAuth)
login: SYSTEM → USER
(SignIn, SignIn, BankAuth)
```

Figure 9: Counterexample for Sign In

As shown in the trace above, the user can only choose to sign in using the email ID or phone number. i.e., there exists only one path from the sign-in state. So, when a CTL query A[] USER.SignIn imply (SYSTEM.PhoneNumber && SYSTEM.EmailID) will not satisfy and the tool gives a counterexample.

Counterexamples are not generated by all the model checker tools. This is one of the main reasons for me to choose UPPAAL as a model checker for my project. With counterexamples, we can also see why the system doesn't satisfy certain properties and can correct them accordingly.

6. CONCLUSION

Takeaways from this project:

→ How to model a real-time system.
→ How to check the correctness of the model using a model checker tool.
→ Converting the properties of a system into a formal specification language.
→ Understanding and implementing CTL queries.

The system which I have built in this project can be implemented into a fully functional application with additional features as required and can be used by commuters in Montreal. A device with an internet connection is the only requirement to use this application. This eliminates the disadvantages of physical TVM machines like long queues and saves a lot of time for commuters.

7. REFERENCES

[1] Lecture notes of INSE 6250, Dr. JAMAL BENTAHAR.

[2] Official website of UPPAAL, http://www.uppaal.org/

[3] A tutorial on UPPAAL, Gerd Behrmann, Alexandre David, and Kim G. Larsen, Department of Computer Science, Aalborg University, Denmark

[4] Comparison of Model Checking Tools for Information Systems, Marc Frappier, Benoît Fraikin, Romain Chossart, Raphaël Chane-Yack-Fa, Mohammed Ouenzar. https://doi.org/10.1007/978-3-642-16901-4_38

[5] MODEL CHECKERS –TOOLS AND LANGUAGES FOR SYSTEM DESIGN- A SURVEY, Shubha Raj K B and Suryaprasad J Department of Computer Science and Engineering, PESIT-Bangalore South Campus, Bengaluru, Karnataka, India.

[6] Computational Tree Logic (CTL) Lecture 16, http://www.inf.ed.ac.uk/teaching/courses/propm/papers/CTL.pdf

You can find my work at the below-mentioned links:

GitHub: https://github.com/AdarshArvind/Model-checking-with-UPPAAL

Google Drive: https://drive.google.com/drive/u/0/folders/1sRIcaRil5wyo8NcFYDJcyu6gWkjkEZwH