INSE 6250 – QUALITY METHODOLOGIES FOR SOFTWARE – SUMMER 2024

Modeling and Verification of an InsurancePurchase Website Using UPPAAL

Submitted By

Supriya Avala	**– 40258501**
Bhanu Teja Kakarla	**– 40256174**
Kunati Bala Krishna Yadav	**– 40292128**
Naveen Rayapudi	**– 40291526**

Under the Guidance of

DR. JAMAL BENTAHAR

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. ABSTRACT:

An insurance company is a financial institution that delivers risk management services by issuing policies to both individuals and businesses. These policies are intended to safeguard policyholders from financial losses caused by various risks, including accidents, natural disasters, illness, and other unexpected events. This company collects premium from policyholders and in return agrees to compensate them in case of covered losses. Some of its core features includes users being able to purchase these policies, renew or cancel them and raise claims if any. For performing any of the above mentioned operations, traditionally the users have to visit the Insurance Office or speak with the Insurance Agents. But with the increasing technology and internet accessibility, these kind of in-person operations are becoming outdated, due to some of the major disadvantages like long queues, waiting times and travelling. The goal of this project is to eliminate such problems faced by commuters by modeling an Insurance purchasing website using Uppaal.

Our project investigates the use of Uppaal, an integrated tool environment for modelling and validating real-time systems by modelling an insurance purchasing website. Such websites are critical digital platforms where users may traverse various insurance packages, receive information, and interact with providers. The platform's design prioritizes user-friendly interfaces, straightforward navigation, and rich information to enable seamless user interactions. With Uppaal's modeling, validation, and verification capabilities, developers can ensure the website's stability and performance. It validates temporal logic properties, ensures safety, liveness, and reachability and provides simulation and visualization tools to observe system activity. Integrating Uppaal into the development process guarantees that the insurance website satisfies high standards of functionality, security, and responsiveness, hence improving user experience and operational efficiency.

2. Introduction

In the modern digital era, the insurance industry is increasingly utilizing technology to improve customer experiences and optimize operations. One critical component of this digital transformation is the development of user friendly online platforms for purchasing insurance policies. By making these processes digital, users can avoid long queues and waiting times, by accessing all these services at their fingertips. When providing these excellent features, it is crucial to verify such applications before they reach the market. They must function as intended without any errors as accuracy is paramount and any failure in doing so can lead to significant user dissatisfaction. Therefore, verifying the systems is essential to ensure the quality of such services. Designing user friendly and appealing web pages for these online applications requires considerable time and effort. In this project, we will be modelling an Insurance purchasing website and verify its correctness using CTL on a tool called UPPAAL.

2.1 Formal Methods and Model Checking

Formal testing involves executing a program with a set of inputs and comparing the actual outputs to the expected outputs. However, relying solely on testing for software error detection has several limitations. Testing can only commence once an implementation is available, and correcting errors discovered during testing can require undoing significant work. Errors in the specification phase are particularly costly to rectify if testing is the sole method of error detection.

To address these drawbacks, industries increasingly use formal methods, which go beyond mere specification languages. Formal methods include proof systems that demonstrate each design step preserves the formal meaning of the previous step. This approach allows for mathematical verification of important properties, such as internal consistency, which can also be incorporated as runtime checks in the final program. Constructing a proof of program correctness is a more robust method than testing alone.

In a model-based approach, an abstract mathematical model of the data is built using abstract mathematical types such as abstract state machines. The behavior of operations is then specified in relation to this model. Formal methods like Model Checking have gained popularity because they verify state transition systems and execute automated verification of a system model against its specifications. Unlike testing, which requires user interaction, model checking is an automatic, model-based method for property verification that helps develop high-assurance systems.

One significant advantage of model checking is the availability of efficient implementation tools like UPPAAL, NuSMV, and SPIN, which have successfully verified large real-world systems.

2.2 UPPAAL

UPPAAL is an extensive toolset created for modeling, validating, and verifying real-time systems. Uppaal checks properties expressed in temporal logic, exploring the state space to verify safety, liveness, and reachability. The tool also provides simulation and visualization features, allowing users to observe system behavior over time

Uppaal consists of 3 core components: the Editor, the Simulator and the Verifier. The Editor is used to create and modify system models or templates, that form our system, which needs to be analyzed. The simulator allows users to manually run the system and select transitions or follow a trace generated by the verifier. The Verifier can be used to check the correctness of

our system. Here, the system requirements are converted into Computation Tree Logic(CTL) formulas and are verified against the system.

Several features make Uppaal a preferred choice over other model checkers. One key advantage is that Uppaal provides counterexamples when a property is not satisfied, aiding developers in detecting and fixing errors in the system. Additionally, properties can be easily specified as CTL formulas, and the system can be graphically modeled as finite-state automata, making it user-friendly and efficient for system verification.

3. Model

In this project our system consists of three templates namely InsuranceUser (User), Insurance_Server (Server) and PaymentServer. Any user visiting our Insurance Website can browse through all the Insurance plans without any need to SignIn or SignUp. But the user can only purchase one or more plans only if the user has registered in the Insurance Website. So, after visiting the website, the user will be having two options to either SignIn or SignUp. New Users has to SignUp for the first time and then SignIn with the same credentials for every login. The Users can either login through their UserId or their PolicyNumber. The server authenticates these credentials and in case of incorrect credentials, the server restricts the user by redirecting to the login page. And this process has to be repeated by the user until he successfully logs in by entering the correct credentials. Upon successful authentication, the user is provided with five options from the server and the user has to select any one option in that transition. User can either view and update their profile, view plans and add them, purchase plans, view transaction history or raise claims and view their claims history. IWhen a user attempts to purchase a plan, they are given two payment options: Apple Pay and credit or debit card. User upon entering his payment details, the PaymentServer checks if the user's bank has sufficient balance to complete the payment and approves or declines the payment based on the bank balance. The server, recieves this payment status from the PaymentServer and conveys the same to the user as approved or declined accordingly. After the intended operations, the user can logout manually or the loggedin users gets automatically logged out due to session timeout based on a certain duration of inactivity.

Below is the screenshot of InsuranceUser :

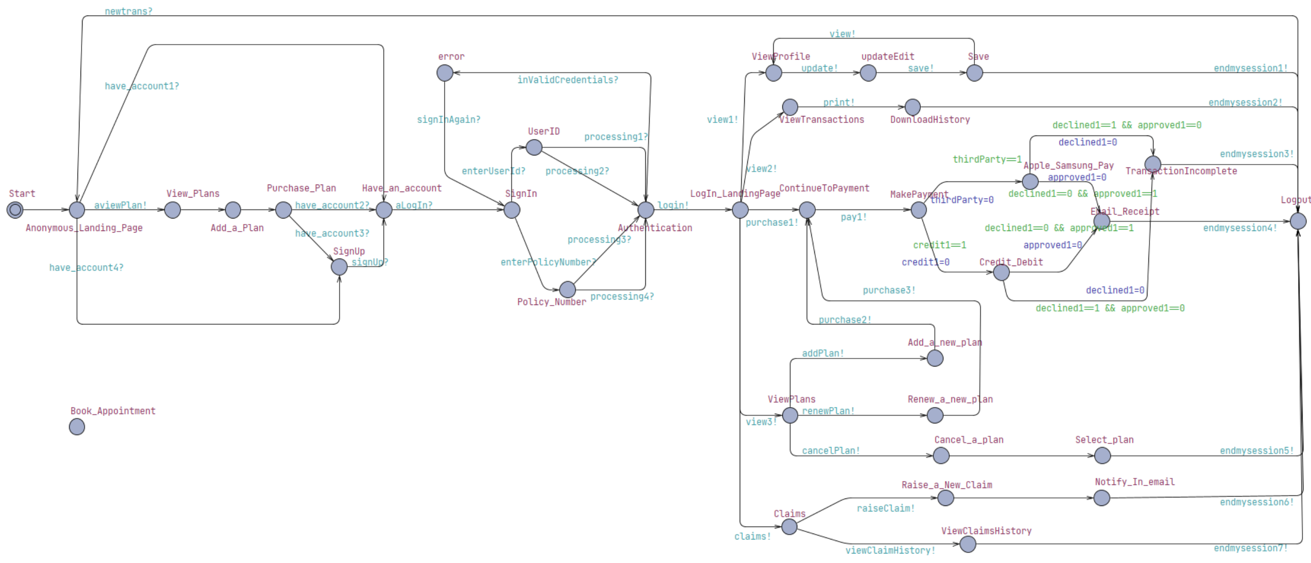


Figure 1 : InsuranceUser Template

Below is the screenshot of Insurance_Server, which acts as a bridge between the InsuranceUser and the PaymentServer using synchronization:

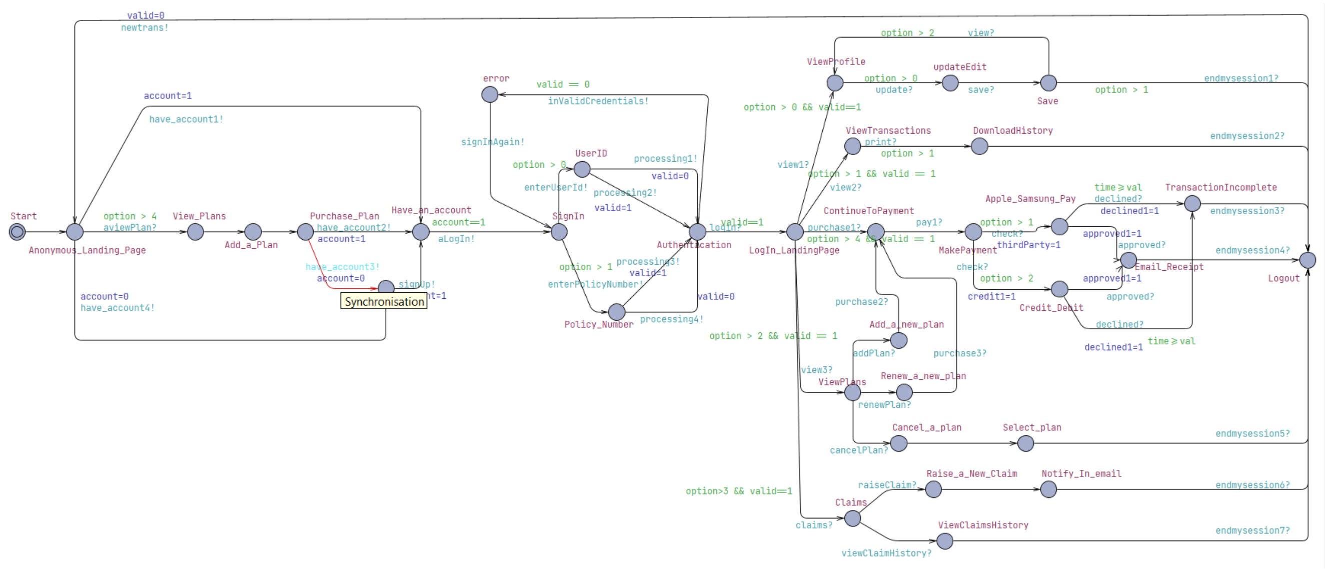


Figure 2 : Insurance_Server Template

Below is the screenshot of PaymentServer :

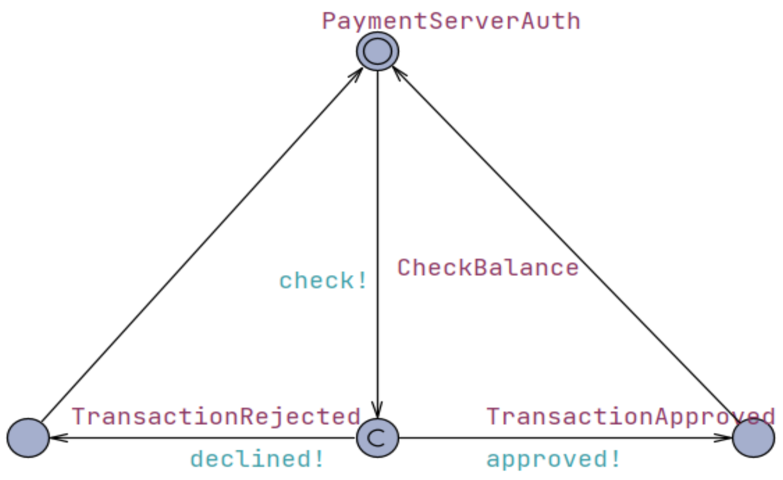
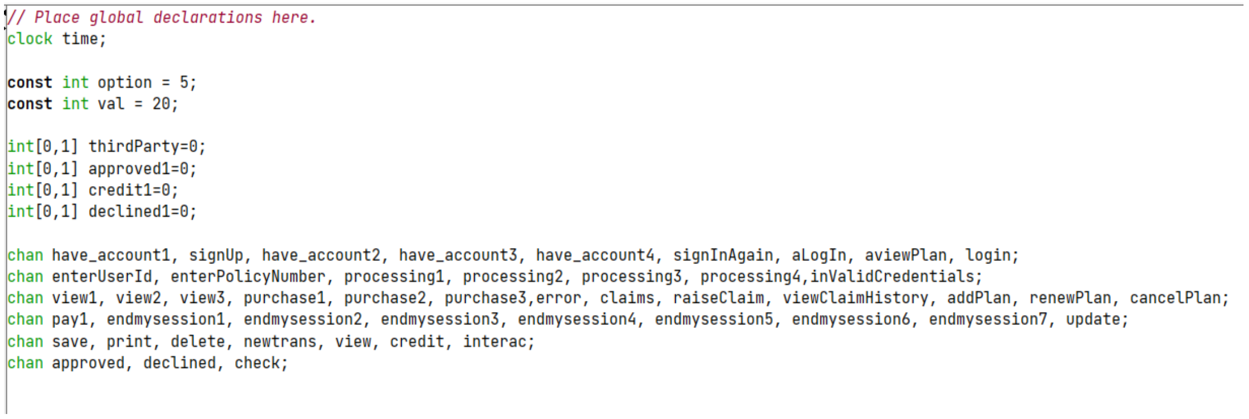


Figure 3 : PaymentServer Template

3.1 Declarations

As shown below I have used various channels for synchronization purposes between our templates.



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

const int option = 5;
const int val = 20;

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

chan have_account1, signUp, have_account2, have_account3, have_account4, signInAgain, aLogIn, aviewPlan, login;
chan enterUserId, enterPolicyNumber, processing1, processing2, processing3, processing4,inValidCredentials;
chan view1, view2, view3, purchase1, purchase2, purchase3,error, claims, raiseClaim, viewClaimHistory, addPlan, renewPlan, cancelPlan;
chan pay1, endmysession1, endmysession2, endmysession3, endmysession4, endmysession5, endmysession6, endmysession7, update;
chan save, print, delete, newtrans, view, credit, interac;
chan approved, declined, check;
```

Figure 4 : Global Variables

The integer variables thirdParty and credit1 are used to represent the type of payments as applePay and credit or debit card respectively, whereas the variables approved1 and declined1 are used to represent the status of payment as approved and declined respectively, if their values are 1. Integer variable called val which is constant through out the process is used for representing the session timeout value. This is compared against the clock time and if this time is greater the val, then the user automatically gets logged out. The other integer variable called option is used to represent the 5 different operations the user can perform, once he/she is successfully authenticated. The channels approved, declined and check are used for synchronization between the PaymentServer and the Insurance_Server, whereas all the other remaining channels are used for synchronization between the InsuranceUser and the Insurance_Server.

4. Requirements and Properties

Any software development process should start with a defined set of criteria. The requirements should detail the program's functionality and user-interface. Similarly, for this project,we have established a set of specifications that the model must satisfy to function correctly.There are several needs for insurance purchasing websites, which may be divided into functional and non-functional criteria. Below is a detailed list of all the things that such a system ought to include:

4.1 Functional Requirements

User Registration and Authentication:

- Users must be able to register with personal information.
- Secure login with email and password.
- Password recovery options (e.g., email verification).

Insurance Plan Management:

- Users should be able to view the Insurance Policies even without signing into the website
- Only Logged In Users should be able to purchase new plans, renew existing ones, and cancel plans.

Policy Management:

- Viewing active and past policies.
- Notifications for policy renewals and updates.

Payment Processing:

- Support for various payment methods, including credit/debit cards and digital wallets.
- Secure payment gateway integration.
- Handling payment failures with appropriate user feedback.

Claims Handling:

- Users can file new claims with detailed descriptions and supporting documents.
- View status of ongoing claims.
- History of past claims and their resolutions.

4.2 Non-Functional Requirements

Security:

- End-to-end encryption for data transmission.

- Secure storage of user data.

Performance:

- Fast loading times and responsive design.
- Scalable infrastructure to handle high traffic volumes.
- Efficient handling of concurrent users and transactions.

Usability:

- Intuitive and user-friendly interface.
- Accessible design .
- Clear navigation and well-organized content.

Maintainability:

- Modular design for easy updates and maintenance.
- Comprehensive documentation for developers and users.

Compliance:

- Adherence to legal and regulatory requirements.
- Clear terms and conditions, privacy policy, and user agreements.

4.3 Safety Properties

Safety features guarantee that "something bad" never occurs while the system is in use. These requirements must constantly be met in order to avoid disastrous mishaps or crucial mistakes.

- The system is designed to handle unsuccessful payments by sending the user back to the payment procedures page, preventing incomplete transactions from causing problems.

.

- Guarantees that only permitted users can enter the system, preventing unauthorized access.
- The model includes mechanisms to detect and correct errors, ensuring all specified requirements and constraints are satisfied.

4.4 Liveness Properties

Liveness guarantee that "something good" will ultimately occur. These characteristics ensure that the system will advance and ultimately arrive at a desired state.

- Users may make sure that these crucial tasks are finally finished by purchasing insurance policies and submitting claims.
- Email alerts are sent to users when there are new purchases or claims, keeping them updated on the progress of their actions.

- Customers have the ability to examine their past transactions and claims, guaranteeing that the system keeps track of and eventually grants access to this data.

4.5 Reachability Properties

The ability to reach a certain state from the starting state is guaranteed by reachability characteristics. These characteristics are essential for confirming that the system can transition between all required states while it is operating.

- The concept makes sure that the states entailing viewing and purchasing are attainable by enabling consumers to examine insurance plans on the home page and purchase preferred plans.

- In order to confirm that the authentication status is reachable, users can log in with their policy number or email.

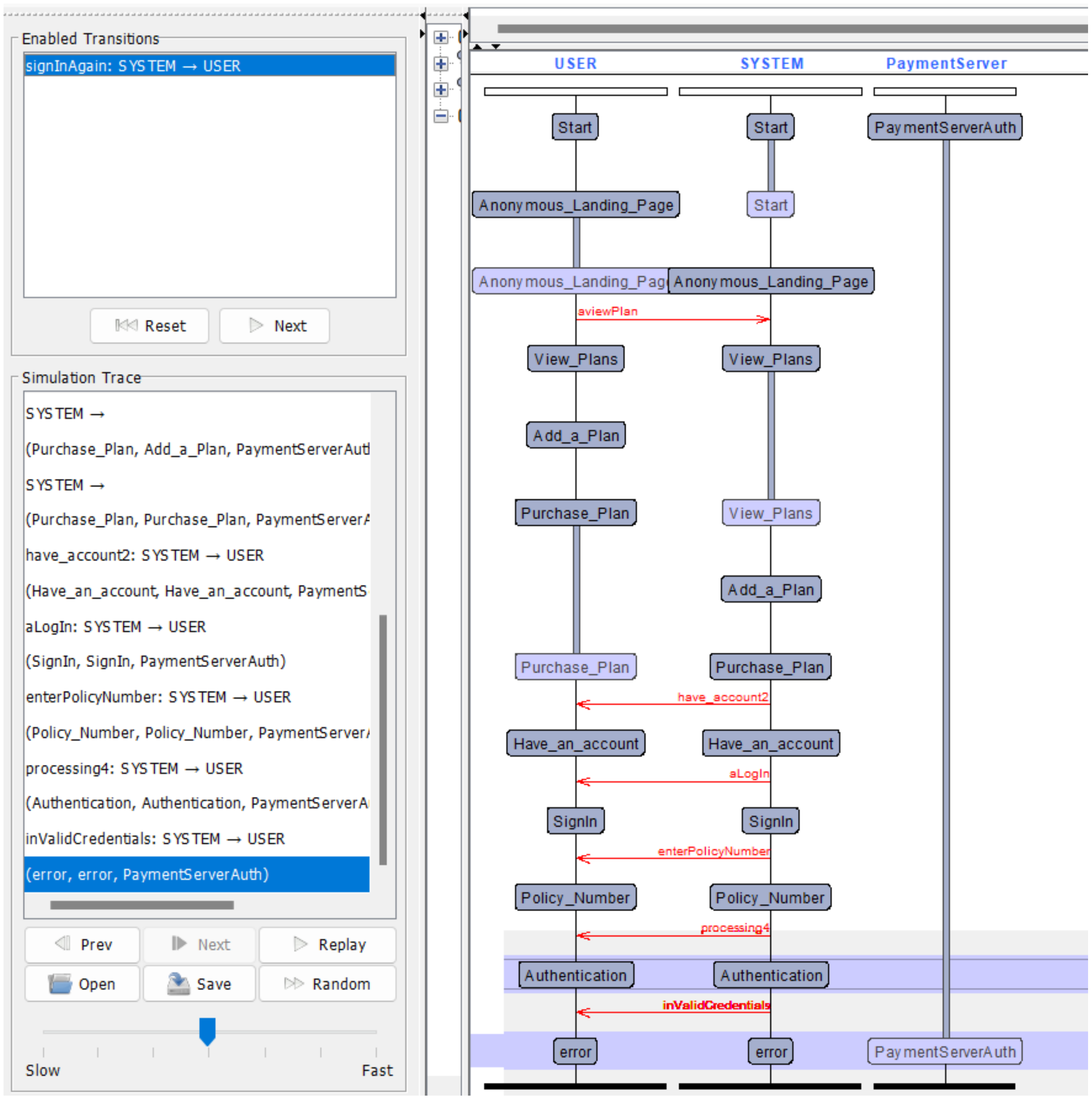
- The system makes sure that the payment state is approachable by supporting a number of payment methods, including Apple/Samsung Pay and Credit/Debit.

- To make sure that the states concerning claim management are attainable, users have the ability to submit new claims and see claim histories.

Achieving all of these qualities in the model is the aim.

5. Simulation

Simulation traces are visible in UPPAAL and other model checker tools, which aids in tracking a specific transition. When something goes wrong with the system, this is useful. Tracing the transition makes it simpler to identify the problem and allows us to describe the system appropriately.The following actions may be made to incorporate this into UPPAAL:1) Emulate the States and Transitions of the System, 2) For shared variables, use global declarations, and for template synchronisation, use channels, 3) Define CTL Properties, 4)Evaluate properties and run simulations.


Enabled Transitions

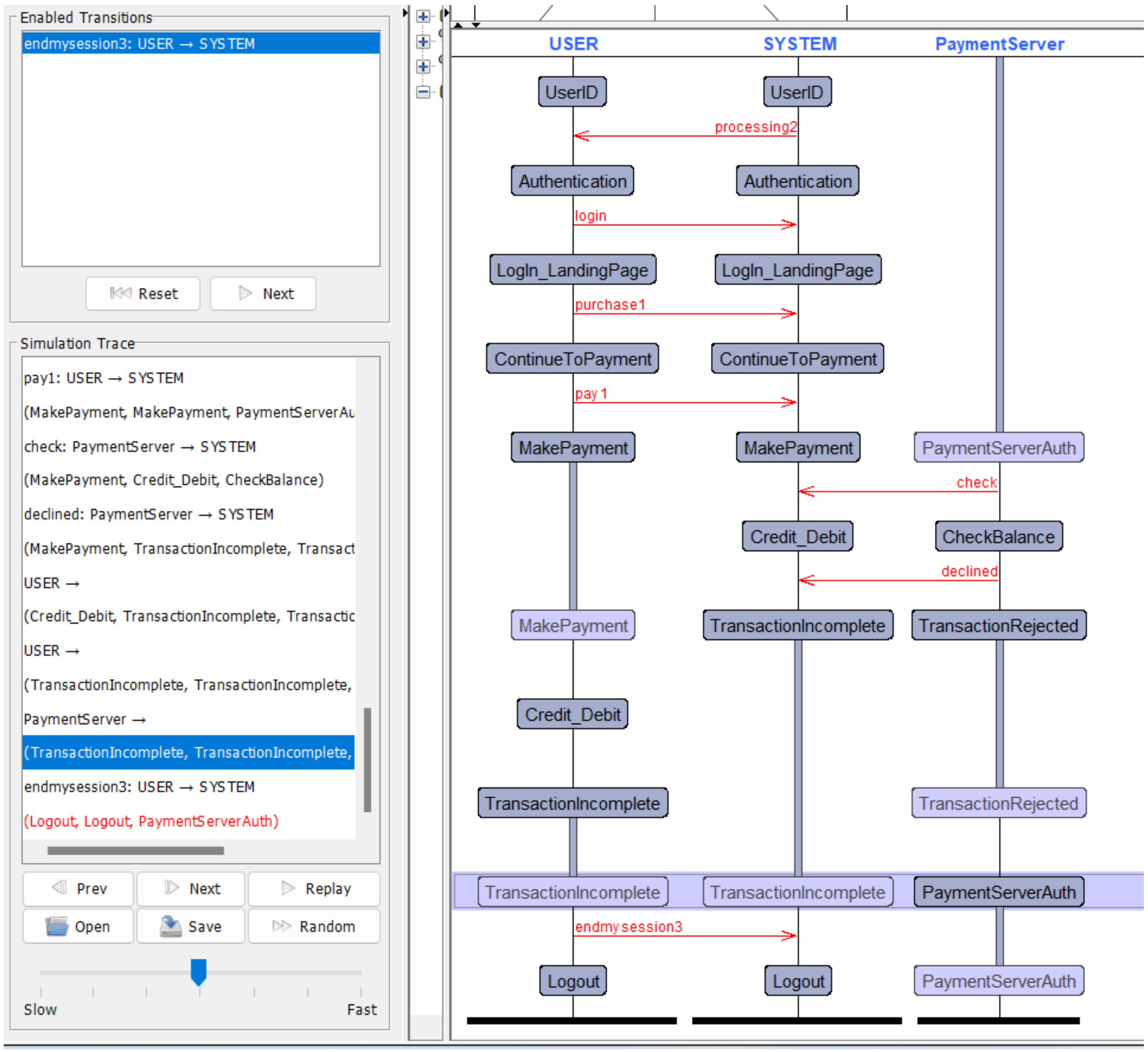
signInAgain: SYSTEM → USER

Simulation Trace

SYSTEM →
(Purchase_Plan, Add_a_Plan, PaymentServerAut
SYSTEM →
(Purchase_Plan, Purchase_Plan, PaymentServerA
have_account2: SYSTEM → USER
(Have_an_account, Have_an_account, PaymentS
aLogIn: SYSTEM → USER
(SignIn, SignIn, PaymentServerAuth)
enterPolicyNumber: SYSTEM → USER
(Policy_Number, Policy_Number, PaymentServer/
processing4: SYSTEM → USER
(Authentication, Authentication, PaymentServerA
inValidCredentials: SYSTEM → USER
(error, error, PaymentServerAuth)

Figure 5 : Authentication Error Simulation

The picture above shows the authentication error on entering the incorrect credentials by the user. According to the model, the user will receive an error message from the server informing them that they need to log in using the right credentials.



```
Enabled Transitions
endmysession3: USER → SYSTEM

Simulation Trace
pay1: USER → SYSTEM
(MakePayment, MakePayment, PaymentServerAu
check: PaymentServer → SYSTEM
(MakePayment, Credit_Debit, CheckBalance)
declined: PaymentServer → SYSTEM
(MakePayment, TransactionIncomplete, Transact
USER →
(Credit_Debit, TransactionIncomplete, Transactic
USER →
(TransactionIncomplete, TransactionIncomplete,
PaymentServer →
(TransactionIncomplete, TransactionIncomplete,
endmysession3: USER → SYSTEM
(Logout, Logout, PaymentServerAuth)
```

Figure 6 : Payment Simulation

The picture above illustrates the PaymentServer interacting with the Insurance_Server after entering some invalid credit or debit card details by the user while trying to purchase an insurance plan from the website.

6. Verification

Ensuring software correctness is crucial in any development process. In our project, we have used CTL to verify the properties of the model. This section details the properties and their corresponding CTL queries that are verified using UPPAAL.

UPPAAL allows for the verification of two main types of properties: reachability properties and liveness properties. The CTL operators "A[]", "A<>", "E[]", and "E<>" are used to represent various logical expressions. Among these the operator "E<>" is used to define the reachability properties, whereas the rest other operators are helpful in defining the liveness properties of the System.These notations and their applications are detailed in Table 1, and all specified properties have been successfully verified.

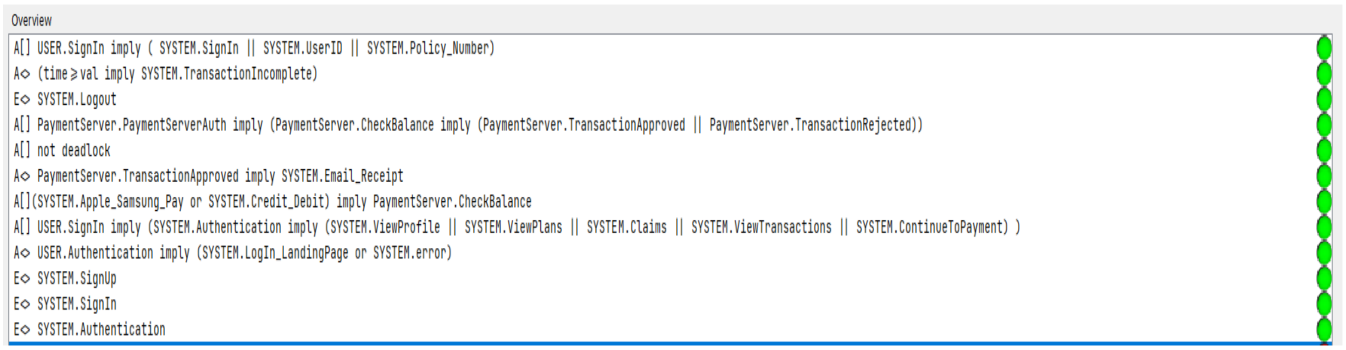
Sl No.	Property Name	Property Definition
1	E<> SYSTEM.SignUp	Eventually there exists at least one path, where the System goes to the SignUp state
2	E<> SYSTEM.SignIn	Eventually there exists at least one path, where the System goes to the SignIn state
3	E<> SYSTEM.Authentication	Eventually there exists at least one path, where the System goes to the Authentication state
4	E<> SYSTEM.Logout	Eventually there exists at least one path, where the System goes to the Logout state

Table 1: CTL queries for verifying the Reachability properties

Property No.	Property Name	Property Definition
1	A[] not deadlock	Sytem will never encounter a deadlock
2	A[] USER.SignIn imply (SYSTEM.SignIn || SYSTEM.UserID || SYSTEM.Policy_Number)	For every login user has two options (SignIn using User ID or SignIn using Policy Number)
3	A[] PaymentServer.PaymentServerAuth imply (PaymentServer.CheckBalance imply (PaymentServer.TransactionApproved || PaymentServer.TransactionRejected))	Always, Bank will authenticate payment and will notify if a transaction is being approved or denied
4	A<> PaymentServer.TransactionApproved imply SYSTEM.Email_Receipt	Receipt will only be emailed to the user when the bank approves the transaction by checking if the user has enough balance.
5	A[](SYSTEM.Apple_Samsung_Pay or SYSTEM.Credit_Debit) imply PaymentServer.CheckBalance	user can make a payment using Apple/Samsung Pay or credit/Debit card, bank will check balance in user's account.
6	A[] USER.SignIn imply (SYSTEM.Authentication imply (SYSTEM.ViewProfile || SYSTEM.ViewPlans || SYSTEM.Claims || SYSTEM.ViewTransactions ||	The system provides various options to the user such as : View Profile, Purchase plans, continue to payment(if logging in to buy a plan), View Transaction History, Raise claims(iff the user's credentials are authenticated).

	SYSTEM.ContinueToPayment))	
7	A<> USER.Authentication imply (SYSTEM.LogIn_LandingPage or SYSTEM.error)	After user enters his credentials, the system should either show the Landing page after successful login or it should go to an error state, incase of incorrect credentials

Table 2: CTL queries for verifying the Liveness properties



```
A[] USER.SignIn imply ( SYSTEM.SignIn || SYSTEM.UserID || SYSTEM.Policy_Number)
A<> (time≥val imply SYSTEM.TransactionIncomplete)
E<> SYSTEM.Logout
A[] PaymentServer.PaymentServerAuth imply (PaymentServer.CheckBalance imply (PaymentServer.TransactionApproved || PaymentServer.TransactionRejected))
A[] not deadlock
A<> PaymentServer.TransactionApproved imply SYSTEM.Email_Receipt
A[](SYSTEM.Apple_Samsung_Pay or SYSTEM.Credit_Debit) imply PaymentServer.CheckBalance
A[] USER.SignIn imply (SYSTEM.Authentication imply (SYSTEM.ViewProfile || SYSTEM.ViewPlans || SYSTEM.Claims || SYSTEM.ViewTransactions || SYSTEM.ContinueToPayment) )
A<> USER.Authentication imply (SYSTEM.LogIn_LandingPage or SYSTEM.error)
E<> SYSTEM.SignUp
E<> SYSTEM.SignIn
E<> SYSTEM.Authentication
```

Figure 7 : CTL Queries

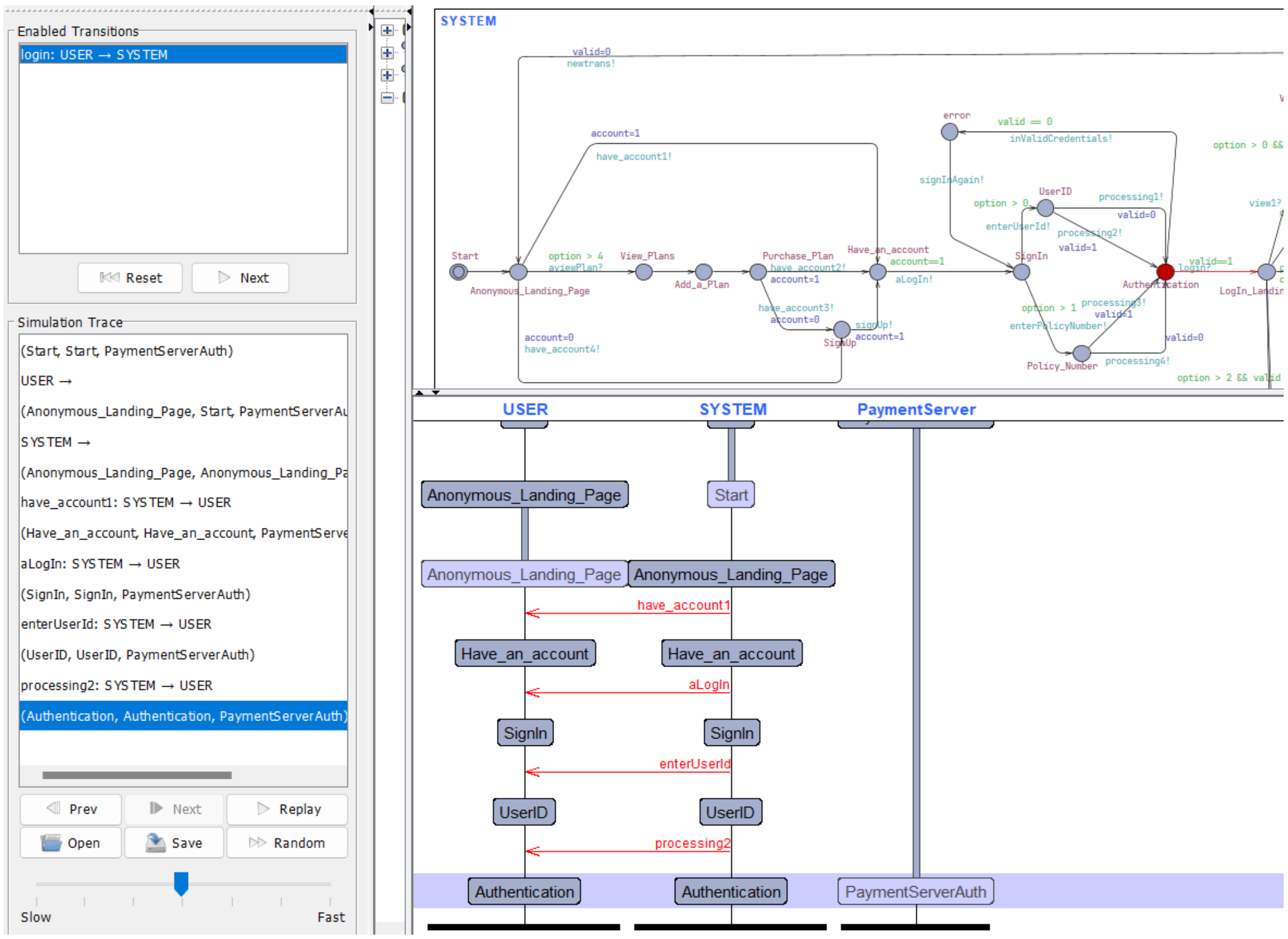
6.1 Counter Example

While the system has to satisfy certain properties, there are instances where the system doesn't satisfy some properties as well. Its common in all Model Checking tools to state, if the properties are satisfied or not by the system. UPPAAL stands one step ahead from the rest of the tools by showing a counter example or trace where the property doesn't satisfy. This error trace helps in visualizing the exact scenarios where the system disobeys our requirements and is helpful in rectifying the errors accordingly.

For example, after authentication, we check if the system can be in both error state and LoginIn_LandingPage at the same time using the below CTL Query :

| A[] USER.Authentication imply (SYSTEM.LogIn_LandingPage and SYSTEM.error)

But, this is not practically possible and the same is shown in the Simulation with a Counter Trace.


Enabled Transitions

login: USER → SYSTEM

Simulation Trace

```
(Start, Start, PaymentServerAuth)
USER →
(Anonymous_Landing_Page, Start, PaymentServerAu
SYSTEM →
(Anonymous_Landing_Page, Anonymous_Landing_Pa
have_account1: SYSTEM → USER
(Have_an_account, Have_an_account, PaymentServe
aLogIn: SYSTEM → USER
(SignIn, SignIn, PaymentServerAuth)
enterUserId: SYSTEM → USER
(UserID, UserID, PaymentServerAuth)
processing2: SYSTEM → USER
(Authentication, Authentication, PaymentServerAuth)
```

Figure 8 :Counter-Example Trace

7. Conclusion

Through this project, we have understood the importance of framing the correct specifications and verifying them by modelling these system specifications in a visualization tool like UPPAAL and test it much before the start of the development phase. By modeling and verifying the specifications in advance, we can avoid the expensive and time-consuming task of altering specifications after development. This proactive approach ensures that the system meets all requirements from the beginning, preventing the need to undo and redo significant amount of work.

In conclusion, in this project, we have modeled and verified our system against its specifications, such that the system can be developed without any further changes or discrepancies in the specifications. However, if needed, Insurance companies can implement this into a fully functional application, with additional features as required by adding them into the specifications and then by modelling and verifying them, before moving on to the development stage. Such digital systems in the Insurance Industry, can help in reducing the long queues and waiting times for the users.

8. References

[1] Ahmed, Z., Shaikh, M. K., & Mahar, I. (2018). Model Checking Knowledge And Commitments In Multi-Agent Systems Using Actors And UPPAAL. https://www.researchgate.net/publication/324202437

[2] Andersson, P., David, A., Larsen, K. G., Legay, A., Mikucionis, M., & Poulsen, D. (2021). UPPAAL Tutorial. https://uppaal.org/texts/21-tutorial.pdf

[3] Larsen, K. G., & Pettersson, P. (1997). Uppaal in a Nutshell. https://www.semanticscholar.org/paper/Uppaal-in-a-nutshell-Larsen-Pettersson/ec48771b3e35ed451c721d99805541c2ad431a8a?utm_source=direct

[4] Rajeev Alur, Costas Courcoubetis, and David Dill. 1993. Model-checking indense real-time. Information and Computation 104, 1 (1993), 2–34.

[5] Uppaal. (n.d.). Verifier. https://docs.uppaal.org/gui-reference/verifier/

[6] Amato, F., Casola, V., Cozzolino, G., De Benedictis, A., Mazzocca, N., & Moscato, F. (year). A Security and Privacy Validation Methodology for e-Health Systems.