Modeling and Verification of Adaptive eLearning System

Ching Hay Chau

C.H.Chau1@ncl.ac.uk

Abstract. The eLearning system has become a vital component of modern education. To address variations in students' abilities, learning histories, and preferences, adaptive eLearning has been introduced. While numerous models have been developed for adaptive eLearning, their reliability is typically assessed only through student evaluations post-implementation. This dissertation aims to propose a verification tool to assist adaptive eLearning system developers in validating the logic of their designs. Given the lack of standardised models, a higher-level approach to modelling is recommended using Event-Condition-Action (ECA) rules. The dissertation explores the use of the SPIN model checking tool to analyse eLearning models, benefiting both developers and educators. To support this approach, the extension of PROMELA—the programming language used in SPIN—is investigated. A graphical user interface (GUI)-based integrated development environment (IDE) has been developed to manage these extensions and handle Linear Temporal Logic (LTL) formulae for verifying the correctness of eLearning models. Test cases are included to demonstrate the functionality and effectiveness of the PROMELA extensions.

1 Introduction

E-learning, defined as the use of electronic technology in education and sometimes referred to as computer-assisted learning, has become a prevalent teaching method. Numerous Learning Management Systems (LMS) and Massive Open Online Courses (MOOCs) have been launched, particularly in tertiary education. These approaches have proven effective, particularly in subjects that emphasise problem-solving or language learning [1]. With advancements in computer-assisted learning technologies, researchers have increasingly explored personalised approaches to elearning, often supported by artificial intelligence, referred to as adaptive e-learning. The lack of agreed-upon modelling standards [4] has led to diverse approaches in developing adaptive e-learning systems. Researchers have focused on varying aspects of learning events and adaptation processes. For instance, AMASE, developed by Trinity College Dublin, emphasises the learning workflow [5], whereas ALEF, implemented at the Slovak University of Technology in Bratislava, prioritises collaboration among learners to enhance learning efficiency [6]. A 2011 survey concluded that IMS-LD, a modelling language that frames e-learning events as plays in drama for computerisation, is rarely used in proposed adaptive e-learning systems [11]. This diversity in system development focus presents challenges in evaluating system reliability without direct testing on students. Furthermore, the absence of standardised evaluation methods [4] underscores the need for a cross-platform verification tool for adaptive e-learning systems. The absence of model-checking tools for adaptive e-learning systems during the design phase is a critical gap. This project addresses this issue by proposing a verification tool to validate the logical design of adaptive e-learning models before implementation. Although tools like CAVIAr have been developed to validate courseware during the design phase [18], they are limited to assessing teaching materials within a specific model setup. These tools do not verify the underlying system model, leaving the design's potential failures undetected until the evaluation phase, where teachers and students assess its effectiveness. Early-stage verification, ensuring logical correctness of the design prototype, is therefore essential. This work explores the use of the SPIN model checker and the extension of PROMELA, the validation language for SPIN, to verify e-learning systems. The implementation of this tool involves adapting concepts from electronic and smart contracts, as defined in EPROMELA (an extended version of PROMELA) [7,8,9], to educational components. Recent research has demonstrated the applicability of blockchain-based smart contracts—whether centralised, distributed, or hybrid [47,48,49]—across various domains, including education [33,34,35,36], cloud computing, the Internet of Things, and service level agreements (SLAs) [37-46,50,51]. Instead of modelling learning events as plays in drama, as suggested by IMS-LD, this work adopts a contract metaphor using Event-ConditionAction (ECA) rules. A hypothetical example of such a model is provided below:

The model involves student who will perform the following actions repeatedly:

a. *Choose a subject*
b. *Take subject 1*
c. *Take subject 2*
d. *Take an assessment* Operation ‘a’ is performed, followed by ‘b’, ‘c’ or ‘d’ under certain

constraints:

2. Student is *obliged* to choose ‘b’, ‘c’ or ‘d’ after ‘a’.
3. Student is *prohibited* to choose ‘d’ if neither ‘b’ or ‘c’ have been performed.
4. Student is *prohibited* to choose ‘b’ if ‘b’ has been performed.
5. Student is *prohibited* to choose ‘c’ if ‘c’ has been performed.

The execution of each learning activity is based on the right/obligation/prohibition given to a Role player. Right means that student is allowed to execute that event. Obligation means the student is expected to execute a certain event, i.e. compulsory to do it. Prohibition means that student should not do the event unless otherwise notice. A more detailed model may involve more role players such as the learning management system and teachers, with more prohibition and right/obligation assignment. Logical errors are more likely to happen with the increase in complexity. Therefore modelling learning activity is not only a single task but required a continuous check throughout the design period is important.

Modelling learning events is the first step of verification. Transformation of these models and rules into PROMELA is essential for SPIN. In this dissertation method of transformation would be discussed and justified. Moreover, to verify a design is logically correct, only having the model and SPIN verification is not enough. For constraints that are not as obvious as the design or monitoring states of the design, Linear Temporal Logic formulae are required. Discussion of such formulae will be at Section 3.

1.1 Proposed Aim and Objectives

The aim of the project is to *develop a model-checking based verification tool for adaptive e-learning system represented in Event Condition Action (ECA) rules.* This tool is a highlevel tool to be used by teachers for checking their design on the courseware and the learning flow of the course.

The objectives of the project are as follows:

1. To classify the similarity and differences among the adaptive e-learning system

We would like to have an understanding of the current situations about adaptive elearning system. To verify a system, a model of the system is required to set up. To model a system, understanding of the system is required. Therefore classifying similarity and differences of different adaptive eLearning system should be done.

2. To identify usual practices in evaluating the system

It is worth understanding how system developers usually do justification on their product. By knowing the usual methods in evaluating the system, we can identify if there exists any missing part that required a verification to be done and develop the tools.

3. To modify EPROMELA to an education-oriented PROMELA extension

The aim of the project is to develop a model-checking tool. EPROMELA is a verification tools for e-contract which works with SPIN, with extensions of PROMELA and ECA rules. Understanding on how PROMELA can be extended for other purposes is important to achieve the aim of the dissertation.

4. To develop a verification tool for adaptive eLearning system based on SPIN

Developing verification tool for adaptive eLearning system is the main objective of the dissertation. In this project, a verification tools extending PROMELA for adaptive learning systems is the target to develop. A Java based IDE with LTL formulae manager that can execute the SPIN verification is to be built.

5. To evaluate the system using realistic e-learning scenarios

Testing the system with some scenarios to understand how the application can help verifying adaptive learning systems.

1.2 Outline of the dissertation

This dissertation is divided into 6 sections. Section 2 would be the background which introduced related works and concepts like ECA rules, adaptive learning systems, SPIN, PROMELA, Linear Temporal Logics and previous attempts to extend PROMELA. Section 3 would be introducing LPromela, the extension of PROMELA to be developed for adaptive learning systems. Architecture and detail conversion from design draft to ECA rules and from ECA rules to executable PROMELA would be explained. Assertion and LTL formulae usage would be briefly included in the section.

Section 4 introduces the LTL manager developed to manage LTL formulae for LPromela model during verification. In Section 5, some scenarios would be included for discussion to demonstrate how to verify adaptive learning systems through LPromela and LTL. Discussion on Slicing Algorithm would also be mentioned. Then we evaluate the aim and objectives of the dissertation.

Conclusion and future work would be mentioned in Section 6 which rounding up the dissertation by giving a summary of the work done in the project and states the further topics to be research on or implementation can be done about the project.

2 Background

2.1 Event – Condition – Action (ECA) rules

The idea of Event-Condition-Action rules is proposed in 1996 for active database systems, describing the execution flow of the event by definition the event, conditions the event holds and the consequence action [12]. As a popular model used to describe electronic contracts, it has been used to model electronic contracts in EPROMELA, an extension of PROMELA, by specifying the ECA rules in EROP (Event, Right, Obligation and Prohibition) language [7,13,31,32], which used to specific the set of rights, obligations and prohibitions (ROP) of the role player involved, and the conditionaction combinations for each ROP elements.

Consider the example specified in Introduction, we can find the ROP sets specified there. Conditions are set such as the "prohibition" of "taking assessment" would be changed upon execution of any of the two "subjects". To complete the assignment, we also need to specify the actions to take. For example, when the condition of the event "taking assessment" is still prohibited, the action would be "returning to course selection".

Similar to the above example, we can model different learning activities and simulate course adaption by setting up ECA rules. As similar tools on electronic contract has been established [7,8,9], it is reasonable to modify the existing tools into an adaptive eLearning–based tool.

2.2 Adaptive eLearning System

An adaptive e-learning system contains the following characteristics [2,3]:

- Tracking user activities;
- Manage the user activities records with the specified models;
- Manage and select appropriate contents with the analysis on user requirements and their preferences;
- Manage and select specified contents which are based on what have been learnt by the user and other important topics.

A lot of models have been proposed and being used. A survey [11] has been done in December 2009 that 105 articles on popular online bibliographic database has proposed or implemented an adaptive and intelligent system for collaborative learning support (AICLS). Most of the systems have different aims and objectives targeting on. For example Topolar, which developed in University of Warwick [16], put their focus on Social Interaction and based the adaptation mechanism on that; AMASE [5] developed in Trinity College Dublin, on the other hand, does not consider social interactions among students as an important features. Adaptation on the learning workflow is the aim of the project instead.

In [11] focus on modeling are categorized into User/group (modelling the learners), Learning Domain (modelling the knowledge area targeted), and Activity (modelling the way of learning). It also stated three basic targeting directions on the adapted component: Group formation, domain-specific support and peer-interaction support. Upon the discussed systems in [11] most of the systems are targeting the peer interaction technology with the modelling of activity. Moreover, the survey suggested that IMS-LD, a framework used to formally describe teaching-learning process [15] by using a drama play as metaphor, is not popular among the learning systems investigated. Only one case among the investigated models used a standard based modelling to develop the system. This also agreed by [4] which suggested no standards have been concurred in current situation. A verification tools should then be providing a more abstract modelling features that leads to a freedom for the learning system designer to verify a system with a specific aim.

2.3 Validation methods for Adaptive eLearning System

To evaluate the usability of the adaptive learning model, the usual practice is by collecting student-based feedbacks through running the model in a real classroom or a classlike condition [11]. However, software failures always happen because of some very unlikely reason that we usually overlook [14]. Moreover, research suggested that as a control experiment setup is used, which included two models with adaptive and nonadaptive setting respectively, the non-adaptive version would usually be not well designed which at the end gives an unfair comparison between the models [19]. Therefore an independent verification of the model itself before implementation is important by testing the model on every possibility that logically would happen.

Work on validating constructed courseware has been suggested in [18] in order to avoid human errors from involving in the result during evaluation as the system complexity increases. CAVIAr [18] is developed under this motivation, providing a step back before testing an immature model on students. By providing a validation step after implementation of courseware model, logical errors and structure inconsistency are able to be identified. This successfully reduced the chance of human mistakes on overlooking problems during manual evaluation. However this does not apply to the case where the design of the courseware itself has serious consistency problem. As development of courseware is a "complex, time-consuming and expensive task"[18, 20], it is worthless if we find out the critical design problem of a courseware after implementing it. A verification process for the model prototype before implementation is then important to save time and reduce cost on the adaptive learning system development.

2.4 PROMELA and SPIN

SPIN (*S*imple *P*ROMELA *In*terpreter) model checker [14] is a popular distributed system model verification tool developed by the Bell Labs in 1991 by Dr. Holzmann. SPIN verifies system by first modelling the protocol written into PROMELA (*Pro*cess *Me*ta-*La*nguage), a specified language for SPIN by abstracting the concurrent systems to a higher level design such that the system designer can get rid of the details in implementation before verification. Designed for such feature, PROMELA contains components like processes, channels, synchronizing statements, messages, etc., On the other hand, some basic data types like floating point numbers which is not required in design level does not exist.

Apart from verification, SPIN itself is able to simulate the message communication inside the PROMELA model. In default mode the verification is done by Depth-First search, which will go to the end of each possibility branch until the very end and do backtracking for other possibilities. The aim for the verification search is to find out counterexample which violated the assumptions set by the model. Once a counterexample is found, the verification search is terminated and the trail to the counterexample is recorded for simulation to demonstrate the error which serves for debugging purpose.

In this dissertation, as we mentioned in 2.3 that a verification tools allowing an abstraction of model is required for adaptive learning systems, SPIN is then a reasonable tools to use as abstraction is a requirement.

2.5 Linear Temporal Logic

The aim of verification is to identify the possibility of a system, i.e. the things a system can do and the things it cannot do [14]. Assertion is therefore important to identify design requirements that have a chance to be violated no matter it is likely to be happened or not. On the other hand, as limitations assigned in the system, assertion is used to “indicate the validity of these limitations” as quoted in [14].

However, basic assertion offered in SPIN has limitation. It only handles finite automata but not infinite executions. Therefore, temporal logic is important in handling infinite runs. It is used by shaping the logic statement into formula with specified syntax, and SPIN would translate the formula into a never claim statements in PROMELA. As it is only used to define correctness requirements in a PROMELA model, it is used to setup rules that the model should meet. For example, there exists a learning courseware that required students to complete course 1 before working on course 2. The LTL can then be set to express "course 1 is *always* executed before course 2 is executed".

Detailed syntax is out of scope of the discussion and can be found in [14]. The execution would be discussed in Section 3 of the dissertation.

2.6 Extension of SPIN/PROMELA

As mentioned in section 2.3, SPIN is designed originally for distributed systems. However, by extending it, business contracts or other systems can also be verified.

SPIN is based on C. Therefore syntax and library in C can be included in PROMELA for function extension purpose. Researchers have suggested different extensions to get advantage from the popular verification system. A suggested structure in [17] is as follows:

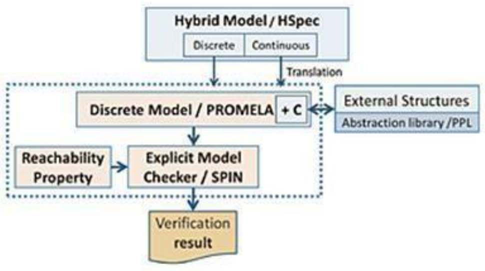


Figure. 1. Architecture of hybrid system verification model checker [17]

Figure 1 demonstrates a design to extend SPIN which can be plugged in and updated easily. External structures including libraries and data structures are imported into original PROMELA code by "#include", "typedef" and "inline". Therefore if any updated is required the SPIN/PROMELA itself does not need to be changed accordingly. A translator is suggested to parse the specified language to PROMELA such that any change of the code does not affect SPIN itself.

Extensions on PROMELA are not a single case. PROBELA [23], a modelling language based on PROMELA for probabilistic model checking with time variables, is an example. There was also trial on extending PROMELA and Spin to real time verification [24]. However these are mostly extending PROMELA based on its original designed domain. To handle other domain-specific concepts and allow PROMELA to serve for more general purposes, an attempt is done to develop language extensions of PROMELA using AbleP [25]. Gaining advantages on the feature of PROMELA which allows embedded C code to be within specification, extension of PROMELA for different purposes outside concurrent system checking is a reasonable task.

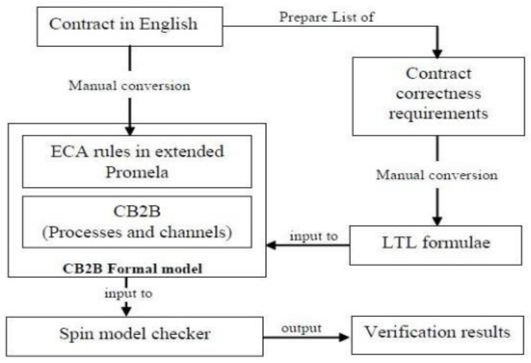


Figure. 2. EPROMELA contract model checking framework [7]

EPROMELA, as proposed in [7], is also one of these extensions. The purpose for EPROMELA is to develop a verification tool for business contract. ECA rules are used in EPROMELA to describe the contract. Figure 2 above shows the framework of the contract checking, where similarities can be found when comparing to Figure 1. Both design work out the modelling in the decided specification language before putting into SPIN for checking. For EPROMELA, it is the EROP which describe the contract in ECA rules. LTL formulae are also set up separately for correctness check. They also show that a "translation" process to SPIN specification is required. In EPROMELA, it is the CB2B shown in Figure 3 which setup a structure on top of PROMELA to simulate a contract as distributed system which SPIN best serves for and to reduce the complexity of writing contract rules in basic PROMELA.

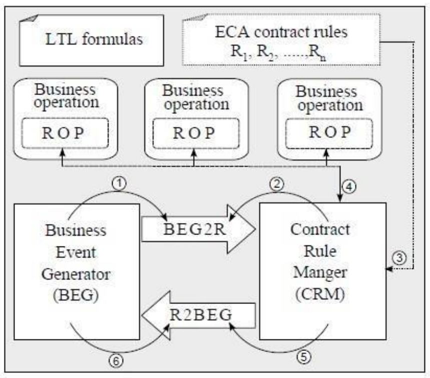


Figure. 3. CB2B model [7]

CB2B model in Figure 3 shows a communication between Business Event Generator and Contract Rule Manager is setup as the base of the whole system. Business Event Generator is in charge of the role players and the possible business operations to be taken, while the Contract Rule Manager is responsible for defining the ECA rules of the business operations and their corresponding actions under different situations like a business failure [7]. This simplifies the PROMELA to be taken and specifies all the related code to be business contract related.

Therefore a similar structure should be built to extend PROMELA for nondistributed system circumstances. In this dissertation, a similar architecture would be set up for adaptive learning systems.

3 LPromela

3.1 ECA rules for learning event

The focus of the verification on adaptive learning systems is to model the learning events and verify the relations between those events. Modelling learning events is thus the first step of developing verification tool. There are many modelling standards proposed of learning events such as IMS-LD, SCORM and IEEE LOM standard [18]. As mentioned in section 2.2, standards used among current adaptive learning systems are not unique. Therefore a modelling method of learning events should be set up in order to map different standards into the tool for verification.

As an example of extension usage of SPIN, EPROMELA [7,21] demonstrates how to model contract in ECA rules and EROP language. ECA rule defines role players, corresponding operations and the Right/Obligation/Prohibition assignment on each condition of a contract, and joining all this into a list of rules. Adaptive learning systems work in similar way. Performance of students and their preference are included such that for a set of learning events defined in the system, right /obligation /prohibition are set on each event.

Consider the following scenario about a small fraction of an adaptive learning system:

Students have applied for a course which consists of the following components:

a. *Choose a learning event*
b. *Take event 1*
c. *Take event 2*
d. *Take an assessment*

Student must choose a learning event before graduated from the course. The assessment cannot be accessed until either event 1 or event 2 is completed. Student can choose to take event 1 or event 2. Once an event is done, the student would not be able to have a retake. If he fails, he would have a chance to retake. Once the assessment is passed, student would be considered graduated from the course.

The above scenario demonstrates that an adaptive learning system is consists of Role player (Student here), Learning Event (event 1, event 2, assessment) and a set of restrictions. Those restrictions can be rewritten in terms of right, obligation and prohibition as follows:

1. Student is *obliged* to choose a learning event.
2. Student is *prohibited* to take ‘d’ until ‘b’ or ‘c’ is executed
3. Student is *prohibited* to take ‘b’ if ‘b’ has been successfully executed (passed).
4. Student is *prohibited* to take ‘c’ if ‘c’ has been successfully executed (passed).
5. Student has the *right* to take ‘b’ if Student failed ‘b’ 6. Student has the *right* to take ‘c’ if Student failed ‘c’
7. System terminated if ‘d’ is successfully completed.

Therefore ECA rules can be used also on adaptive learning systems by rewriting details of the adaptive learning model in contract terms, i.e. including right/obligation/prohibition in the constraints. Hence we can setup the verification tools based on ECA rules following the implementation of EPROMELA.

ECA included Event, Condition and Action. Events and actions are different in different course. However, conditions are similar among most situations. A student would either be passing an examination or fail it, and so do other learning events. As an eLearning platform, there also cases where the technical failure would be affecting the learning progress. Finally there also exists a case where students completed work after the deadline assigned. Four conditions are identified then: Success (Passing the learning event), Learning Failure (Fail the course), Technical Failure and Timeout.

Role players included in adaptive learning systems are teaching materials authors, teachers and students [22]. The learning management system (LMS) itself, which controlled the whole learning experience, is the first interacting character with the students during the learning process. Therefore it is reasonable to count LMS as one of the role players. In this dissertation we will focus on the students activity in adaptive learning system and thus in the discussion below role players will be referring to students and the LMS only.

To write them into formal contract code which can be used in SPIN, a new type LN_EVENT is defined to include the ROP status, execution status and the responsible condition of a learning event.

```
typedef LN_EVENT{
byte name;   byte
role_pl;     bool
right;       bool
oblig;       bool
prohib;      bool
executed;  byte
id;  byte status; }
```

The status here represents the conditions as stated above, i.e. Successful, Technical Failure, Learning Failure and Timeout.

To write the design in ECA rules, list of keywords and definition are required as included below:

Table 1. ECA rules opeartions list

Name	Description
EVENT(Le_i, ROP_i, Status)	Check if *msg1* returns1 and *msg2* returns 1. The *msg1* is responsible to check the Right/Obligation/Prohibition status of the learning event and the role player. The *msg2* is responsible to the execution status (SC/LF/TF/TO) of the event. Block the execution of the specified code if fail. E.g. EVENT(EXAM, IS_O(EXAM, STUDENT) , SC(EXAM))
SC(Le_i), LF(Le_i), TF(Le_i), TO(Le_i), P(Le_i)	Return 1 if the learning event is flagged as stated option, and return 0 if not. SC: Success, LF: Learning process failure, TF: technical failure, TO: timeout, P: Action prohibited
RD(Le_i , RolePlayer, CC, Action)	Return Decision made as a result of the rule execution. Parameter *CC* represents the contract compliance status, i.e. CCR (Contract compliant right) / CCO (Contract compliant obligation) / CCP (Contract compliant prohibition) / NCC (Non-contract compliant). Parameter Action represents the further action after rule execution. It can be CON (Continue) or CND (Contract ended)

SYN(Le_i){
………………….
}
NYS(Le_i)
	Used in RULE to synchronize the current part of execution with another learning event (Le_i). Statements included are executed if Le_i is executed successfully.

SET_R(Le_i , check)	Assign right to perform the learning event Le_i by inputting 1 as check, and re- move the right by inputting 0 E.g. SET_R(CHOOSE, 1)
SET_O(Le_i , check)	Assign obligation to perform the learning event Le_i by inputting 1 as check, and remove the obligation by inputting 0 E.g. SET_O(LECURES, 1)
SET_P(Le_i , check)	Prohibit the learning event Le_i by inputting 1 as check, and remove the prohibition by inputting 0
	E.g. SET_P(EXAM, 1)
SET_X(Le_i , check)	Control the execution status of the learning event Le_i. Initially set to 0. Sets to 1 if event is executed successfully. E.g. SET_X (CW,STUDENT) means CW has been executed by STUDENT.
IS_R(Le_i , RolePlayer)	Return 1 if the RolePlayer has the right to execute learning event Le_i , otherwise return 0 E.g. IS_R(CHOOSE, STUDENT)
IS_O(Le_i , RolePlayer)	Return 1 if the RolePlayer is obliged to execute learning event Le_i , otherwise return 0 E.g. IS_O(LECURES, STUDENT)
IS_P(Le_i , RolePlayer)	Return 1 if the RolePlayer is prohibited to execute learning event Le_i , otherwise return 0 E.g. IS_P(EXAM, STUDENT)
IS_X(Le_i , RolePlayer)	Return 1 if the RolePlayer has executed learning event Le_i , otherwise return 0

Using the above operations list, rules can be written in a specific structure and passed to extended PROMELA model for verification. To make use of the operations, a specific structure is used to include rules in blocks for execution. The structure of the block is as follows:

```
RULE(LN_EVENT){
 WHEN::EVENT(Lei, ROPi, Status)
```

```
  ->{
    ActionBlock;
        RuleDecision;
}
  END(LN_EVENT);  }
```

The action block consists of the actions to take under this certain ROP-condition combination. One of the main actions to be taken is the change of ROP status of learning events such as prohibiting the current event to be taken place after this execution or to assign obligation to the Role Player to complete other learning events.
Take the learning event ‘b’ in above scenario as an example, the eLearning contract rule would be:

```
RULE(b){
 WHEN::EVENT(b, IS_O(b,STUDENT), SC(b))
 ->{
   SET_R(a,1);
   SET_O(b,0);
   SET_P(b,1);
   SET_X(b,STUDENT);
   RD(b, STUDENT, CCO, CO);
    }
 ::EVENT(b, IS_O(b,STUDENT), LF(b))
 ->{    printf("Fail");
SET_O(b,0);
   SET_R(a,1);
   SET_R(b,1);
   RD(b, STUDENT, NCC, CO);
    }
 ::EVENT(b,IS_P(b,STUDENT),SC(b))
 ->{
   SET_R(a,1);
   SET_O(b,0);
   SET_O(c,0);
   SET_O(d,0);
   RD(b, STUDENT, CCP, CO);
 }
  END(b);    }
```

Keyword *printf* of C is also available to be used. Therefore XML can be inserted in serving for systems wanting to collect all the information of the verification. Cases for technical failure and other combinations are not included in the above example as the handling is similar to the above codes.

3.2 Checking of rules

As the metaphor of contract is used on learning events, requirements for contract compliance would be demonstrating the requirements for a valid model. Contract compliance requirements for business operations are as follows [7,21]:

— Business operation bo_i is a subset of the primitive business operations B; — ROP set of the corresponding role players are matched. Role player would have either right, obligation to execute the task or being prohibited to perform it. — bo_i fulfils the constraints specified on the contract.

The compliance rules above are designed in terms of business contract. To provide similar rules for learning system verification, modification is required. Modifying these contract requirements in terms of education would become:

- C1) learning event le_i is a subset of the primitive learning events L;
- C2) The ECA rules set, i.e. the right/obligation/prohibition of an certain operation by the role player is matched;
- C3) Constraints stipulated in the adaptation rules are satisfied.

For any learning event le_i it is only valid if they are included in primitive learning events. Otherwise, it is not well-defined and rules must be setup for the event to give a definition. If C2 is met, the learning event would be considered matching the ROP. If it is not fulfilled, the learning event would be considered as wrongly setup. C3 is referring to the constraints vary from system to system. If C3 is no met, the learning events would be considered as not matching the design of the model and required a review of the events and the model, i.e. the event is invalid. If no obligated learning events that is waiting for executing in the queue and the system is called to be terminated, we can concluded that this trial is a successful one and the model is verified in this particular case. If it terminated normally like this for all possible cases, the model is verified.

The reason why only obligated events in queue are considered as the terminated condition is, only obligated events are some events that must be finished. Therefore, if it is required to be done and is not being executed, this is a violation of the design. Vice versa, if all obligated events have been executed, then there would not have any violation of design found.

Take examination as an example. Examination is one of the primitive learning events. So C1 is met. Student is always prohibited to look at the examination questions until the examination time started, but is obliged to take it. If the student start reading the examination questions after examination started, C2 is met. If the student is cheating and viewing the questions before examination begins, C2 is not met. Assume that in the adaptation rules, this student should finish the examination in 2 hours. If student finished the examination at or before 2 hours passed, C3 is met. If the student is not able to complete the work, C3 is violated. If C1, C2 and C3 are all met, the learning event would be called "compliant".

3.3 Architecture of the verification model

The following discussion would be on how the rules are used for verification and how the verification itself is to be done. The following is the architecture of the verification tool:

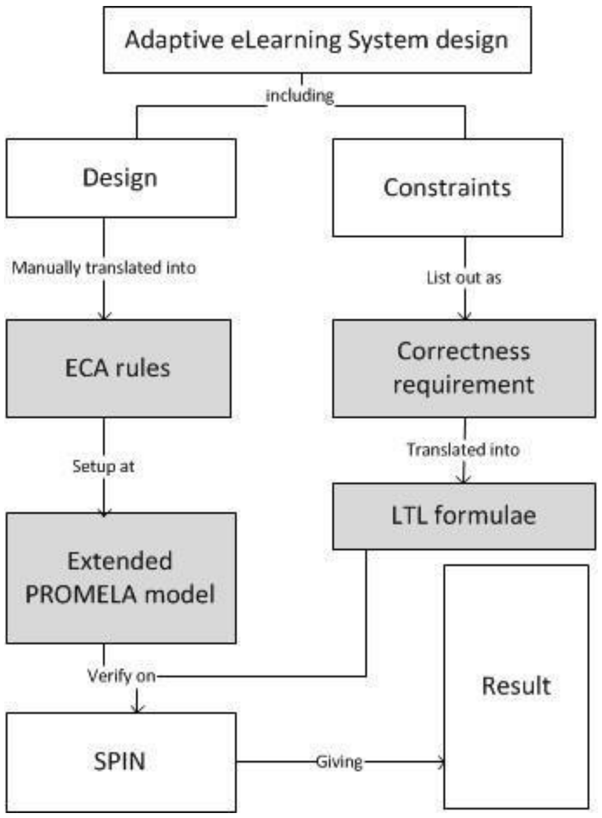


Figure. 4. Architecture of the verification model

A prototype of an adaptive eLearning System can be divided into design and constraints. Design represents the flow of the eLearning system on how the learning events are held and their corresponding actions, i.e. how the adaptive rules are applied. On the other hand, constraints are some statements that the system should be obeying apart from the design. For example, when a learning event is set to be prohibited by student, it cannot be obliged to student at the same time. These limitations, which are generated after fixing the design of the system, are not included in the design but they are part of the system.

Here we will discuss in details how Extended PROMELA model would cooperate with LTL formulae to test the system.

Extended PROMELA. Design of an adaptive eLearning system prototype would lead to a set of ECA rules as presented in previous sections. To execute the verification for all the "contract" requirements stated in section 3.2, there is a need to setup the rules and translate into PROMELA for SPIN to handle.

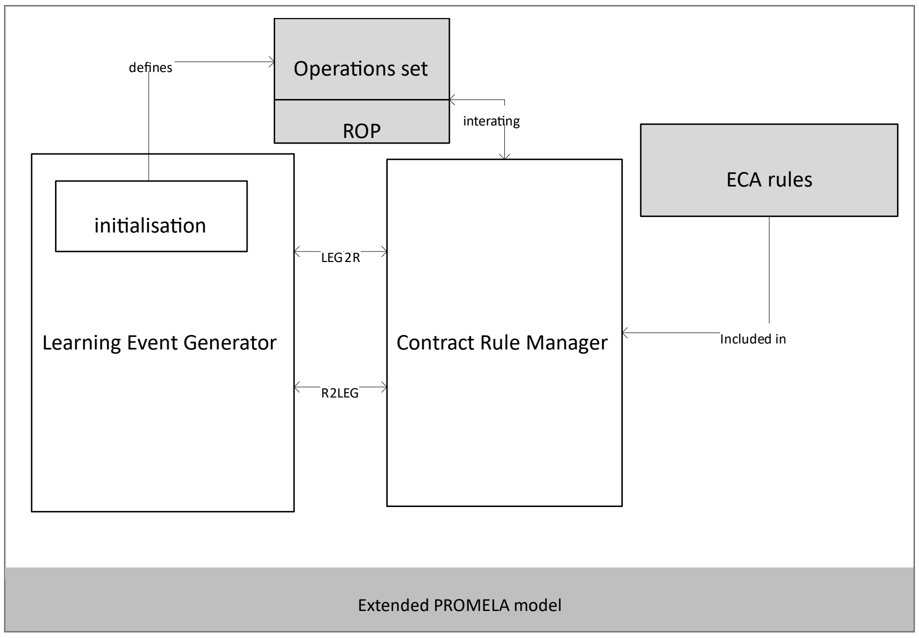


Figure. 5. Extended PROMELA model design

To check the rules and learning events, two proctypes (process behavior in SPIN) are set up for different tasks. Contract Rule Manager (CRM) includes all the ECA rules in the proctype. Learning Event Generator (LEG) defines the queue of learning events and initializes the conditions of these events.

To check the compliance conditions, event in the queue will be called by LEG and the corresponding action under the current condition of the event would be checked in CRM. The action would change the conditions of some events and return them back to the learning event generator for reordering the queue. By repeatedly doing so, all obligated events can be go through until no more events which must be executed are in the queue. This is called a verified situation. However, if it ends in the middle, this can proved that the design of the system is having problem and need modification.

A set of operations are responsible for handling the initialization and verification process. They are defined as follows:

Table 2. Extended PROMELA Operations list

Name	Description
LN_EVENT(Le_i)	Declares *learning events* of PROMELA type *typedef* with the fields: *name, role player, right, obligation, prohibition, execution status, id and status*. The field *Name* is defined by the parameter Le_i of the function E.g. LN_EVENT(LECTURES)
INIT(Le_i , RolePlayer, 1, 0, 0)	Initialise the learning event Le_i with the right, obligation, prohibition status to execute the event by the speci-
	fied RolePlayer E.g. INIT(LECTURES, STUDENT, 0,1,0) means the student is obliged to execute lecture(s).
CONTRACT(Le_i)	Include the specified Rule for a certain event Le_i into contract rule manager (CRM) E.g. CONTRACT(EXAM);
L_E(RolePlayer, Le_i, STATUS)	Send learning event Le_i belongs to the RolePlayer with specified status (S/LF/TF/TO). (S represents SC as in the Rule Operations) E.g. L_E(STUDENT, EXAM, S);
DONE(RolePlayer)	Reset the Execution Trace, i.e. queue E.g. DONE(STUDENT)

These are the operations extending from original PROMELA. Some of them like CONTRACT() and INIT() only require to be executed once in CRM and LEG respectively. However, as discussed above, a recurring check of the obligated events is the key of the verification algorithm. Therefore in LEG, looping would be used to generate learning events as follows: `do`

```
 ::L_E(STUDENT,LECTURES,S);
 ::L_E(STUDENT,LECTURES,TF);
 ::L_E(STUDENT,CW,S);
 ::L_E(STUDENT,CW,LF);
 ::L_E(STUDENT,EXAM,S);
::L_E(STUDENT,EXAM,LF); od
```

The loop will stop until an event called CND (Contract End) as action in CRM and replied to BEG which demonstrates the termination of the checking in queue. If in the queue (the defined event-status combinations inside the loop) there are still obligated events, the contract is having problem. On the other hand, if no obligated events are on the list, the design is said to be verified. Other ways to construct the loop as a method to reduce memory size required, i.e. slicing, will be discussed in Section 5.2.

LTL formulae. Recalling that to ensure a learning event is valid, matching requirement and compliant, there are three conditions to handle, denoted C1, C2 and C3. Moreover, in the verification model architecture discussed above, to fully verify a design correctness requirement check is required. Consider an example as follows: A student would like to choose to take a module "Object-oriented Programming". Here is the course outline:

Lecture 1: "Basic Java Programming" Lecture

2: "Inheritance & Polymorphism"

Lecture 3: "Object-oriented Programming"

Coursework

Exam

Student should take each event one by one, i.e. he cannot take coursework before completing lecture 3, and he cannot take Exam before completing coursework. Consider the teacher set up the verification events queue with a possibility that Learning Failure would be happening in Coursework, and the corresponding action is set as CND. Under ECA rules, coursework under learning failure is uncompleted. Therefore Exam would still in prohibited condition. The only obligated event is coursework and the obligation is removed as CND is called. Therefore under the checking rules, the design is verified.

However, if the teacher is not intending to drop students out from the course under failure of coursework, correctness check is important to ensure some constraints are checked and proven. If the teacher wants Exam to be executed at all case, LTL formulae can be used as follows:

```
ltl p1 { [] <> IS_X(EXAM, STUDENT)};
```

The keyword *ltl* is for inserting LTL formulae in PROMELA. The symbol "<>" means "eventually" and "[]" means "always". Therefore the formula here is means EXAM would be executed at least once by STUDENT. Under the above scenario, error message would be found as this formula is violated under the case of Learning Failure on Coursework and system termination happened. Therefore it ensures some important constraints of the design which are not obviously stated in the ECA rules are being taken into account.

Common and Specified LTL Formulae. There are some LTL formulae that are used quite frequently at all cases, especially considering the use of LPromela here. As ECA rules are used to model LTL formulae, every learning events would have their own "right, obligation, prohibition" condition specified. As a matter of fact, if an event is set with “right”, it cannot be obligated or prohibited. If an event is obligated, there are more constraints than be declared as “right” and cannot be set “prohibited”. If the event is prohibited, there should not be right or obligation declared. Therefore this would be one of the common constraints that every model using LPromela would have come across. Writing this constraint on event EXAM in LTL formula syntax, we will have: `ltl ROPchecker { []( !((IS_R(EXAM, STUDENT) && IS_O(EXAM, STUDENT)) || (IS_R(EXAM, STUDENT) && IS_P(EXAM, STUDENT)) || (IS_O(EXAM, STUDENT) && IS_P(EXAM, STUDENT))) )}`

This would be a common LTL formula as this is a constraint for all LPromela model. However for every model, there would be some constraints which is specified by their own design, a number of specify LTL Formulae would have to be set. Some examples of using specified LTL formulae would be discussed in Section 5.

Assertion. One problem of LTL formulae is that, as only one formula can be checked in one trial, it takes time to handle if there are many constraints to be verified. For example, to check all the events of the above formula *ROPchecker* in the scenario discussed, we need to check the same model for 5 times, one for each event. Therefore for some common and simple constraints, like the ROP constraint mentioned above, can be built in the system using assertions. In LPromela, “right cannot be assigned with prohibition” and “obligation cannot be assigned with prohibition” are worked as inbuilt assertions and would be checked each time when changes of ROP of events happened. The assertion is done as follows:

```
inline SET_R(lo,r){   lo.right=r;
assert(!(lo.right==1 && lo.oblig==1));
assert(!(lo.right==1 && lo.prohib==1)); }
inline SET_O(lo,o){   lo.oblig=o;
assert(!(lo.oblig ==1 && lo.prohib ==1));
assert(!(lo.oblig ==1 && lo.right ==1));
}  inline SET_P(lo,p){   lo.prohib=p;
assert(!(lo.prohib ==1 && lo.right ==1));
assert(!(lo.prohib ==1 && lo.oblig ==1)); }
```

The keyword *assert* is the original keyword in PROMELA. To prevent right assigned with obligation or prohibition at the same time, it is included as part of the parser as above. This method can ensure some simple and general constraints can be checked every time as required without handling the limitation of using LTL formulae one for a time, in exchange by increasing verification performance time. Way to turn these assertion off would be discussed in Appendix E.

4 LPromela LTL manager

The aim of the project is to develop a verification tools on adaptive eLearning system. In previous sections we have discussed a new extension of PROMELA which would allow users of the language to model their design in adaptive learning system to LPromela and verify it in SPIN using command line. However this would only allow users who have the knowledge of setting up LTL formulae and writing PROMELA code to be able to use the tool. As the targeting people of the aim are not only developers but also teachers, a development environment for handling LTL formulae in Graphic User Interface (GUI) should be built.

4.1 Design

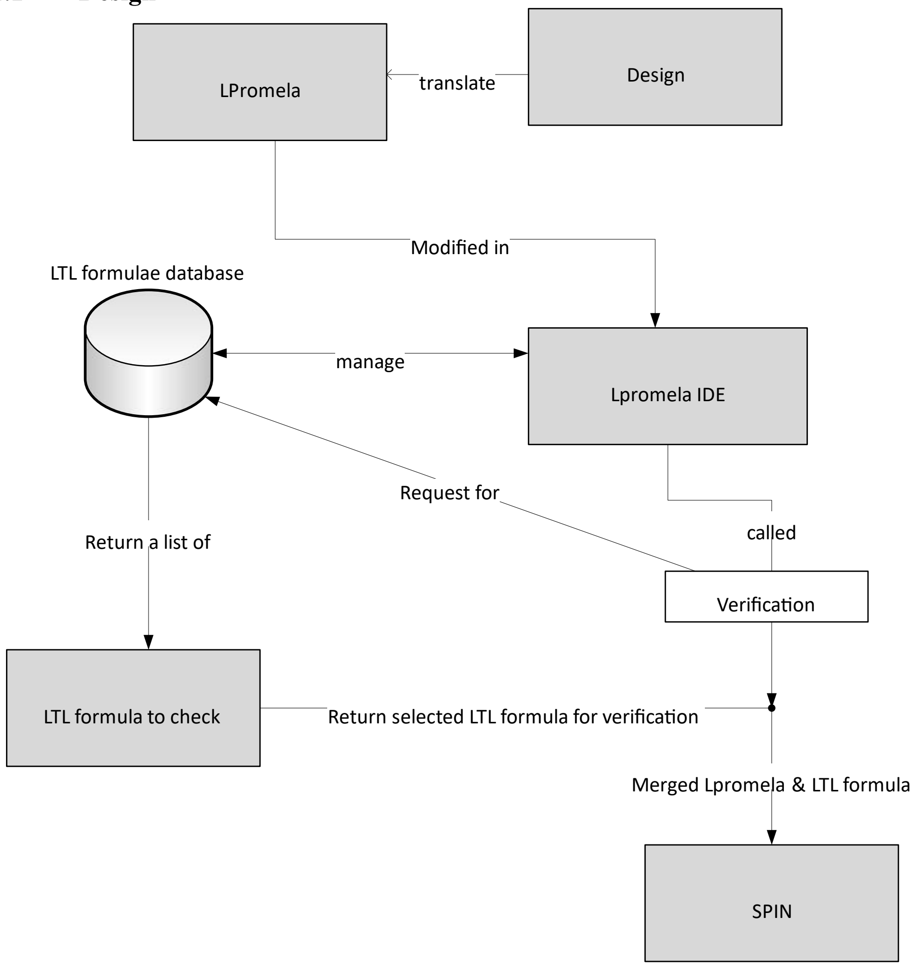


Figure. 6. Architecture of LPromela LTL Manager

Above is the design of the LTL Manager. Viewing of LEG, CRM and ECA rules are done on the manager. It is also responsible for managing a set of LTL formulae which can be chosen to check when verification of design is called. SPIN would be running in the background and would return the result to the LTL manager. If errors found, the found counter example is shown as simulation in the manager.

An independent database is required to setup to store all the LTL formulae. Consider a school with some number of teachers with no knowledge on LTL using LPromela to verify their teaching materials, the LTL formula built by experts can be freely retrieved by these teachers. Therefore, by setting up the database in the school server, teachers can access to the set of formulae and find those best fit for the situation.

4.2 Implementation of LTL Manager

Similar function of managing LTL formulae has been achieved by Jim Sun on a project for translating BPMN to PROMELA [26][30]. However, it is deeply connected with the verification of BPMN graph so it cannot directly be used as the LPromela LTL Manager. Therefore, related design was extracted out and re-implemented with LPromela LTL Manager.

Language and Tools used. The LTL manager is implemented using Java language. The Java editor used is Eclipse Luna. For the GUI implementation, WindowBuilder has been used. WindowBuilder is an open source project from Eclipse to work as a plugin in Eclipse which provide a drag-and-drop interface to deploy Java Swing components and SWT components into the program to facilitate the setup out GUI.

The Database used is MySQL. To connect MySQL with Java, an external JAR file mysql-connector-java is included in the library. The choice of MySQL is based on a few factors. MySQL is a very popular open-source project on relational database system [27]. Support of JAVA connection to MySQL is also well established [28]. Therefore MySQL is chosen to be the database managing LTL formulae.

Structure of the software. Classes are divided into groups: GUI, LTL Managing and Database tools. The classes responsible for GUI are responsible for creating the interface that the user can access. Frames for LPromela and ECA rules code, windows for verification, simulations, control panel for LTL formulae management and viewing LTL formulae are controlled under the GUI classes. Classes under LTL Managing group are responsible for modelling LTL formulae into a specified Java object. Classes for Database tools are responsible for the connection to the database and handle all the SQL queries to retrieve those formulae from database or insert formulae into it.

Except for the GUI, most of the classes have been implemented in BPMN2PROMELA and thus only modification is needed to use it as LTL manager.

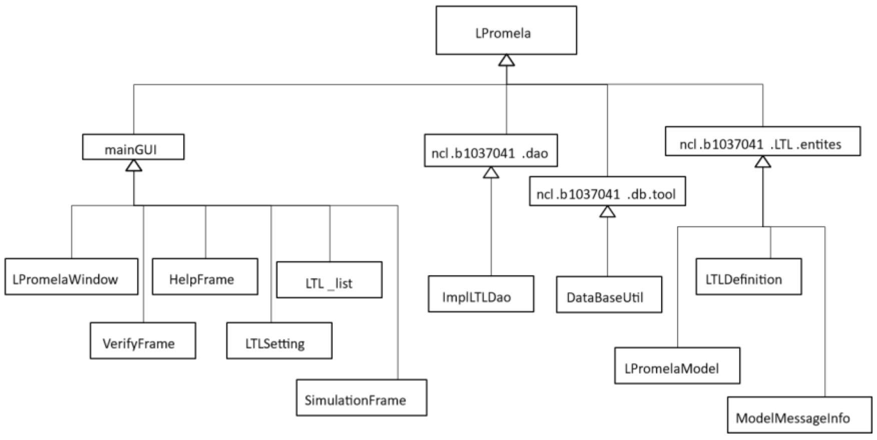
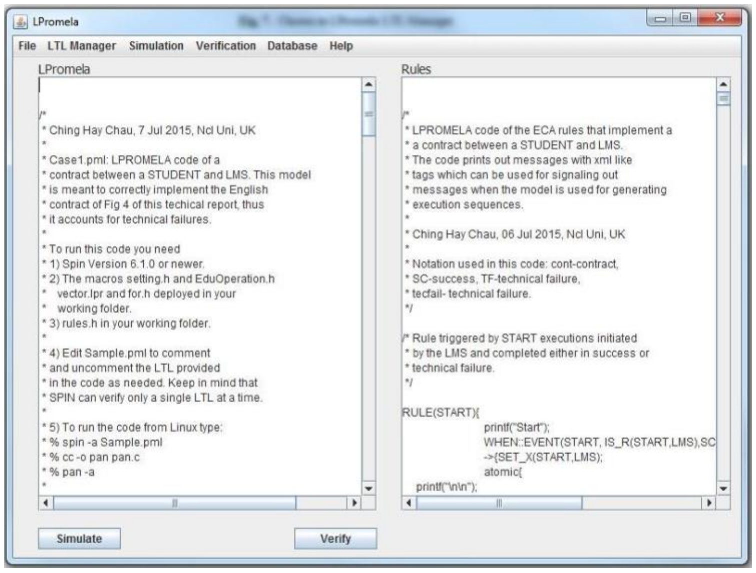


Figure. 7. Classes in LPromela LTL Manager

Figure 7 concluded the description above. All the classes required are listed in. Descriptions of them would be introduced one by one below:

mainGUI. Under mainGUI package there are 6 classes: LPromelaWindow, VerifyFrame, SimulationFrame, LTLSetting, LTL_list and HelpFrame.



```
/*
 * Ching Hay Chau, 7 Jul 2015, Ncl Uni, UK
 *
 * Case1.pml: LPROMELA code of a
 * contract between a STUDENT and LMS. This model
 * is meant to correctly implement the English
 * contract of Fig 4 of this techical report, thus
 * it accounts for technical failures.
 *
 * To run this code you need
 * 1) Spin Version 6.1.0 or newer.
 * 2) The macros setting.h and EduOperation.h
 *    vector.lpr and for.h deployed in your
 *    working folder.
 * 3) rules.h in your working folder.
 *
 * 4) Edit Sample.pml to comment
 * and uncomment the LTL provided
 * in the code as needed. Keep in mind that
 * SPIN can verify only a single LTL at a time.
 *
 * 5) To run the code from Linux type:
 * % spin -a Sample.pml
 * % cc -o pan pan.c
 * % pan -a
 *
```

```
/*
 * LPROMELA code of the ECA rules that implement a
 * a contract between a STUDENT and LMS.
 * The code prints out messages with xml like
 * tags which can be used for signaling out
 * messages when the model is used for generating
 * execution sequences.
 *
 * Ching Hay Chau, 06 Jul 2015, Ncl Uni, UK
 *
 * Notation used in this code: cont-contract,
 * SC-success, TF-technical failure,
 * tecfail- technical failure.
 */

/* Rule triggered by START executions initiated
 * by the LMS and completed either in success or
 * technical failure.
 */

RULE(START){
                printf("Start");
                WHEN::EVENT(START, IS_R(START,LMS),SC
                ->{SET_X(START,LMS);
                atomic{
   printf("\n\n");
```

Figure. 8. LPromelaWindow

LPromelaWindow is the main class, controlling the initialization of the software and closing events. I/O of the PROMELA model files would be controlled within the class. It contains a User Interface which linked to all the functions of the LTL manager.

VerifyFrame is the frame responsible for the verification event. When verification is called in LPromelaWindow, this frame would be taken charge asking the LTL formula to verify and returning the verification result to user. If deadlock or violation of assertion/LTL formulae is found, SimulationFrame would be created demonstrating a counterexample would be pop out.

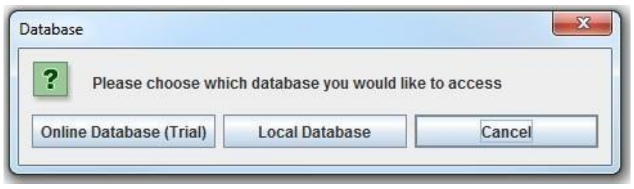
SimulationFrame is responsible for showing the simulation trial of the PROMELA model built. SPIN has a function to assign which trail to show and in this class the same function is implemented to show different simulation tracks.

LTLSetting is the management frame for all the LTL formulae in the database. It is responsible for adding new LTL formula, listing all the LTL formulae built and deleting the unwanted LTL formulae. LTL_list shows all the details of the LTL formulae. HelpFrame is just a pop message containing information about the software.

LTL entities. LTL entites define all the objects that modelling the LTL formulae and objects to collect in MySQL database. LPromelaModel is responsible for storing the login information of each PROMELA model loaded and ModelMessageInfo is responsible for returning informations of the model.

LTLdefintion defines LTL object with formula, description and nickname. Nickname is the name used to let users who do not have knowledge on LTL formula know the meaning of the formula. Description converts directly the symbol of formula to English, and formula stores the formula. ID is assigned for each entry as a primary key in the table.

Database. Two classes are responsible for database connection. DataBaseUtil is responsible for setting up the connection and close it. A function to switch among database has been added for users to connect to a different database if they would like to.


Database

Please choose which database you would like to access

Figure. 9. Switching Database

ImplLTLDao serves as a bridge between LTLSetting, LTL entites classes and the database. Retrieving the connection set in DataBaseUtil, SQL queries are sent from ImplLTLDao for all the actions required a database connection and return the corresponding result. For example, when a user added a new LTL formula to the software in LTLSetting, it is passed to ImplLTLDao to translate as an SQL query to pass the insertion to the database. ImplLTLDao will pass the query to the database and receive feedback from it.

Detailed operations using LTL Manager would be described in Appendix.

5 Testing and Evaluation

In this section, some cases would be discussed to evaluate the usability of the language, followed by a performance test for LPromela and a unit test for the LTL manager.

5.1 Case Studies

Scenario 1 – Learning Progress Adaptation. The first example is an adaptive eLearning system that focuses on adjusting the learning pace of the student during the course. There are three components in the course: *LECTURES* (Denoted as *L1, L2, L3)*, *CW* (Coursework) and *EXAM*.

1. There are 3 lectures, 1 coursework and 1 exam
2. After 3 lectures, a coursework will be given. Student must finish the coursework before progressing.
3. After all lectures and coursework are finished, an exam will be given. Student must finish the exam.
4. If the student does not finish the coursework/exam within the specified time, then the course will be marked as fail

Step 1 – Declaration Part.

```
/*Declaration of some variables for checking*/
bool fail=FALSE; int lectures=3; int LCount=0; int
cw=1; int CwCount=0; int exam=1; int
ExamCount=0;

/*Declaration of Role Player*/
RolePlayer(STUDENT, LMS);

/*Declaration of Learning Event*/
LN_EVENT(START);
LN_EVENT(L1); /*A list of lectures */
LN_EVENT(L2);
LN_EVENT(L3);
LN_EVENT(CW); /* All Coursework */ LN_EVENT(EXAM);
/* All Exam */
```

Step 2 – Contract initialisation.

```
DONE(STUDENT);
DONE(LMS);
INIT(START, LMS, 1,0,0);
INIT(L1,  STUDENT, 0,0,1);
INIT(L2,  STUDENT, 0,0,1);
INIT(L3,  STUDENT, 0,0,1);
INIT(CW1, STUDENT, 0,0,1);
INIT(EXAM, STUDENT, 0,0,1);
```

Step 3 – Deriving contract rules. The natural language of the learning workflow is manually converted into the following ECA rules:

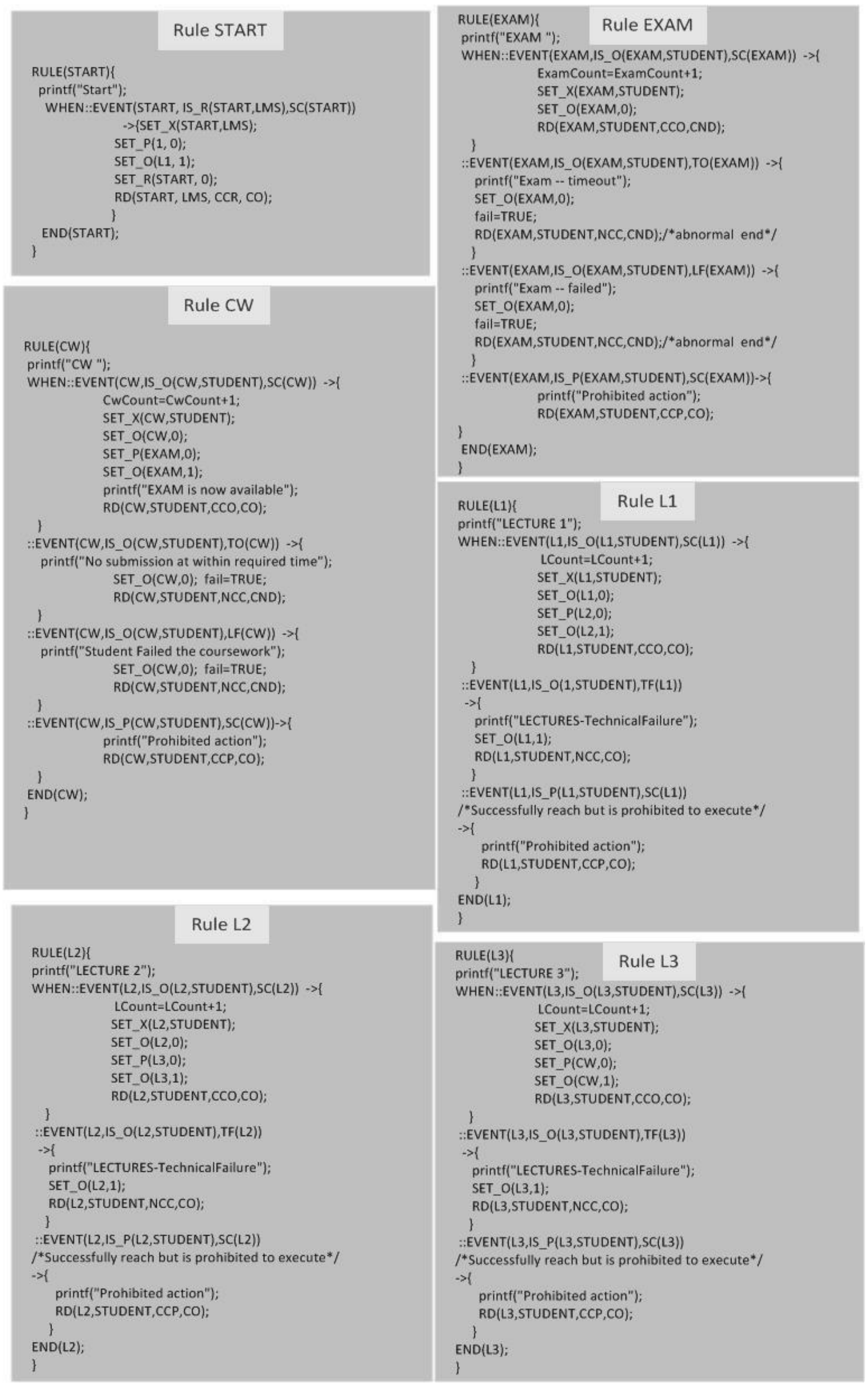
1. Rule(START) – LMS starts offering course materials.
2. Rule(L1), Rule(L2), Rule(L3) – Student takes lectures
3. Rule(CW) – Student works on coursework 4. Rule(EXAM) – Student takes examination.

Implementations of the rules are as below. Timeout of Rule(CW) and Rule(EXAM) are handled as well as technical failure of lectures events.

Figure 10 demonstrates how the designed learning rules are setup in terms of ECA. RULE(L1), RULE(L2), RULE(L3) allows it to handle two status: successfully executed (SC) and technical failure (TF). From the rules we can find that when a technical error take place, i.e. failures happening to the learning management system preventing students from accessing the lecture contents, recovery of the system would take place and let the student return to their original progress. Once a lecture is finished successfully, the next stage is obligated to be executed. For L1 the next stage would be L2, and L2 would be followed by L3, while L3 is followed by CW.

Successfully executed has an counterpart to control the condition where the event is prohibited to be executed. It is classified as successfully executed because the students does successfully access the event. The only difference is that the student does not have the right to finish the event. Therefore it is controlled under the status SC.

Similar implementations are found in CW and EXAM rules. However, technical failure does not take into account on both events in this case as the events are timed event in reality and when technical failure takes place it does not do immediate effects on the student learning progress and would be recovered later by extension of deadline or other measures. Therefore there are learning failure LF and timeout TO taken into account when considering the time where student has finished the event. Learning Failure (LF) is responsible for handling the case of a student failed the coursework or examination which means that some further actions out of the scope of the course should be taken place. Timeout is set in corresponding to the rule 4 of the scenarios stated above. If a student is unable to finish the coursework by deadline or fail to complete the exam within the stated duration, it is also treated as failing the course, and a termination (CND) of the learning progress is executed. As a termination of learning is found, it is considered as a non-compliant case (NCC).



Rule START

```
RULE(START){
  printf("Start");
   WHEN::EVENT(START, IS_R(START,LMS),SC(START))
               ->{SET_X(START,LMS);
             SET_P(1, 0);
             SET_O(L1, 1);
             SET_R(START, 0);
             RD(START, LMS, CCR, CO);
             }
   END(START);
}
```

Rule CW

```
RULE(CW){
 printf("CW ");
 WHEN::EVENT(CW,IS_O(CW,STUDENT),SC(CW)) ->{
             CwCount=CwCount+1;
             SET_X(CW,STUDENT);
             SET_O(CW,0);
             SET_P(EXAM,0);
             SET_O(EXAM,1);
             printf("EXAM is now available");
             RD(CW,STUDENT,CCO,CO);
   }
 ::EVENT(CW,IS_O(CW,STUDENT),TO(CW)) ->{
    printf("No submission at within required time");
               SET_O(CW,0); fail=TRUE;
               RD(CW,STUDENT,NCC,CND);
   }
 ::EVENT(CW,IS_O(CW,STUDENT),LF(CW)) ->{
    printf("Student Failed the coursework");
               SET_O(CW,0); fail=TRUE;
               RD(CW,STUDENT,NCC,CND);
   }
 ::EVENT(CW,IS_P(CW,STUDENT),SC(CW))->{
             printf("Prohibited action");
             RD(CW,STUDENT,CCP,CO);
   }
 END(CW);
}
```

Rule L2

```
RULE(L2){
printf("LECTURE 2");
WHEN::EVENT(L2,IS_O(L2,STUDENT),SC(L2)) ->{
             LCount=LCount+1;
             SET_X(L2,STUDENT);
             SET_O(L2,0);
             SET_P(L3,0);
             SET_O(L3,1);
             RD(L2,STUDENT,CCO,CO);
   }
 ::EVENT(L2,IS_O(L2,STUDENT),TF(L2))
  ->{
    printf("LECTURES-TechnicalFailure");
    SET_O(L2,1);
    RD(L2,STUDENT,NCC,CO);
   }
 ::EVENT(L2,IS_P(L2,STUDENT),SC(L2))
/*Successfully reach but is prohibited to execute*/
->{
     printf("Prohibited action");
     RD(L2,STUDENT,CCP,CO);
    }
END(L2);
}
```

Rule EXAM

```
RULE(EXAM){
 printf("EXAM ");
 WHEN::EVENT(EXAM,IS_O(EXAM,STUDENT),SC(EXAM)) ->{
             ExamCount=ExamCount+1;
             SET_X(EXAM,STUDENT);
             SET_O(EXAM,0);
             RD(EXAM,STUDENT,CCO,CND);
   }
 ::EVENT(EXAM,IS_O(EXAM,STUDENT),TO(EXAM)) ->{
   printf("Exam -- timeout");
   SET_O(EXAM,0);
   fail=TRUE;
   RD(EXAM,STUDENT,NCC,CND);/*abnormal end*/
  }
 ::EVENT(EXAM,IS_O(EXAM,STUDENT),LF(EXAM)) ->{
   printf("Exam -- failed");
   SET_O(EXAM,0);
   fail=TRUE;
   RD(EXAM,STUDENT,NCC,CND);/*abnormal end*/
  }
 ::EVENT(EXAM,IS_P(EXAM,STUDENT),SC(EXAM))->{
             printf("Prohibited action");
             RD(EXAM,STUDENT,CCP,CO);
}
 END(EXAM);
}
```

Rule L1

```
RULE(L1){
printf("LECTURE 1");
WHEN::EVENT(L1,IS_O(L1,STUDENT),SC(L1)) ->{
              LCount=LCount+1;
             SET_X(L1,STUDENT);
             SET_O(L1,0);
             SET_P(L2,0);
             SET_O(L2,1);
             RD(L1,STUDENT,CCO,CO);
   }
 ::EVENT(L1,IS_O(1,STUDENT),TF(L1))
  ->{
   printf("LECTURES-TechnicalFailure");
   SET_O(L1,1);
   RD(L1,STUDENT,NCC,CO);
   }
 ::EVENT(L1,IS_P(L1,STUDENT),SC(L1))
/*Successfully reach but is prohibited to execute*/
->{
     printf("Prohibited action");
     RD(L1,STUDENT,CCP,CO);
   }
END(L1);
}
```

Rule L3

```
RULE(L3){
printf("LECTURE 3");
WHEN::EVENT(L3,IS_O(L3,STUDENT),SC(L3)) ->{
             LCount=LCount+1;
             SET_X(L3,STUDENT);
             SET_O(L3,0);
             SET_P(CW,0);
             SET_O(CW,1);
             RD(L3,STUDENT,CCO,CO);
   }
 ::EVENT(L3,IS_O(L3,STUDENT),TF(L3))
  ->{
   printf("LECTURES-TechnicalFailure");
   SET_O(L3,1);
   RD(L3,STUDENT,NCC,CO);
  }
 ::EVENT(L3,IS_P(L3,STUDENT),SC(L3))
/*Successfully reach but is prohibited to execute*/
->{
     printf("Prohibited action");
     RD(L3,STUDENT,CCP,CO);
   }
END(L3);
}
```

Figure. 10. Learning Progress Adaptation rule set

Step 4 – Prepare LTL formulae. As an example, three LTL formulae are listed below P1, P2 and P3 to verify different adaptation properties.

```
ltl p1 {[]((IS_O(L1, STUDENT))->!(IS_O(CW, STUDENT) &&
IS_O(EXAM, STUDENT)))} ltl p2 {[]((IS_O(L3, STUDENT))-
>(IS_P(EXAM, STUDENT)))} ltl p3 {[] (<>
(IS_O(EXAM,STUDENT)))}
```

The formula p1 is to verify that if only one learning event would be taken place at one time. The formula p2 is to verify that if there are still lectures going on, no exam would be taken place. The formula p3 is to check if EXAM is always eventually performed at all time, no matter the students fail it or not.

Step 5 - Verification. After step 1 to step 4, the contract-like model is verifiable by using Spin model checker, with p1, p2 and p3. The verification without any LTL formulas shows that things run as planned. Formula p1 and p2 are demonstrating that the designed rules are following these constraints as expected. However, formula p3 shows clauses that give violation to the model.

In checking of p3, SPIN returned the following result:

```
State-vector 104 byte, depth reached 260, errors: 1
```

Error is found and a counterexample is given, showing the case that when CW is in status TO, student is declared as failed the course and would not be able to continue the study. Therefore EXAM is never reached. If the teacher wants all students to take place in the examination, rule of CW for TO and LF can be modified as follow:

```
::EVENT(CW,IS_O(CW,STUDENT),TO(CW))  ->{     printf("No
submission at within required time");
              SET_O(CW,0);  fail=TRUE;
              SET_P(EXAM,0);
              SET_O(EXAM,1);
              RD(CW,STUDENT,NCC,CO);
 }
 ::EVENT(CW,IS_O(CW,STUDENT),LF(CW))  ->{
printf("Student Failed the coursework");
              SET_O(CW,0);  fail=TRUE;
              SET_P(EXAM,0);
              SET_O(EXAM,1);
              RD(CW,STUDENT,NCC,CO);

 }
```

Under this modification, learning progress continues no matter the student fail the coursework or not. By checking it again, SPIN returned the following result:

```
State-vector 104 byte, depth reached 264, errors: 1
```

The counterexample is reviewed and demonstrates a case that after L3, student keep trying access EXAM but not CW until the end of the search as SPIN viewed it as cycle. It therefore demonstrates a design problem of the rules that if a student would never follow the standard procedures of learning the current system does not support a measure to handle it.

Evaluation. This test case provides a scenario which there is an in-depth design problem of the situation as discussed above. It also demonstrates that counterexample of LTL formula offered by SPIN only provide situation for one of the reasons that lead to the violation of the constraints. Therefore a repeated LTL formulae check is essential.

Scenario 2 – Course registration adaptation. The second case to consider is the adaptation process of course registration. The learning management system (LMS) is responsible for suggesting a number of courses that are suitable for students according to their individual preferences and previous knowledge, as well as preventing inappropriate courses being taken by the students.

1. The Student has the right to request for suggested course.
2. The LMS is obliged to provide a suitable course
3. The Student is obliged to choose in the list
4. The LMS is obliged to respond accept or decline to the choice
5. The Student is obliged to choose again if the choice is declined, until all the choices are rejected.

Step 1 – Declaration Part.

```
/*Declaration of some variables*/
bool choose1=FALSE; bool
choose2=FALSE; bool choose3=FALSE;
bool all=FALSE;  /*Declaration of
Role Player*/
RolePlayer(STUDENT, LMS);

/*Declaration of Learning Event*/
LN_EVENT(RegReq); /*Registration request*/
LN_EVENT(RegReply); /*Return the list of course*/
LN_EVENT(C1); /*Choosing Course 1*/
LN_EVENT(C2); /*Choosing Course 2*/
LN_EVENT(C3); /*Choosing Course 3*/
LN_EVENT(ChooseAccept); /*Accept the choice*/
LN_EVENT(ChooseReject); /*Reject the choice*/
```

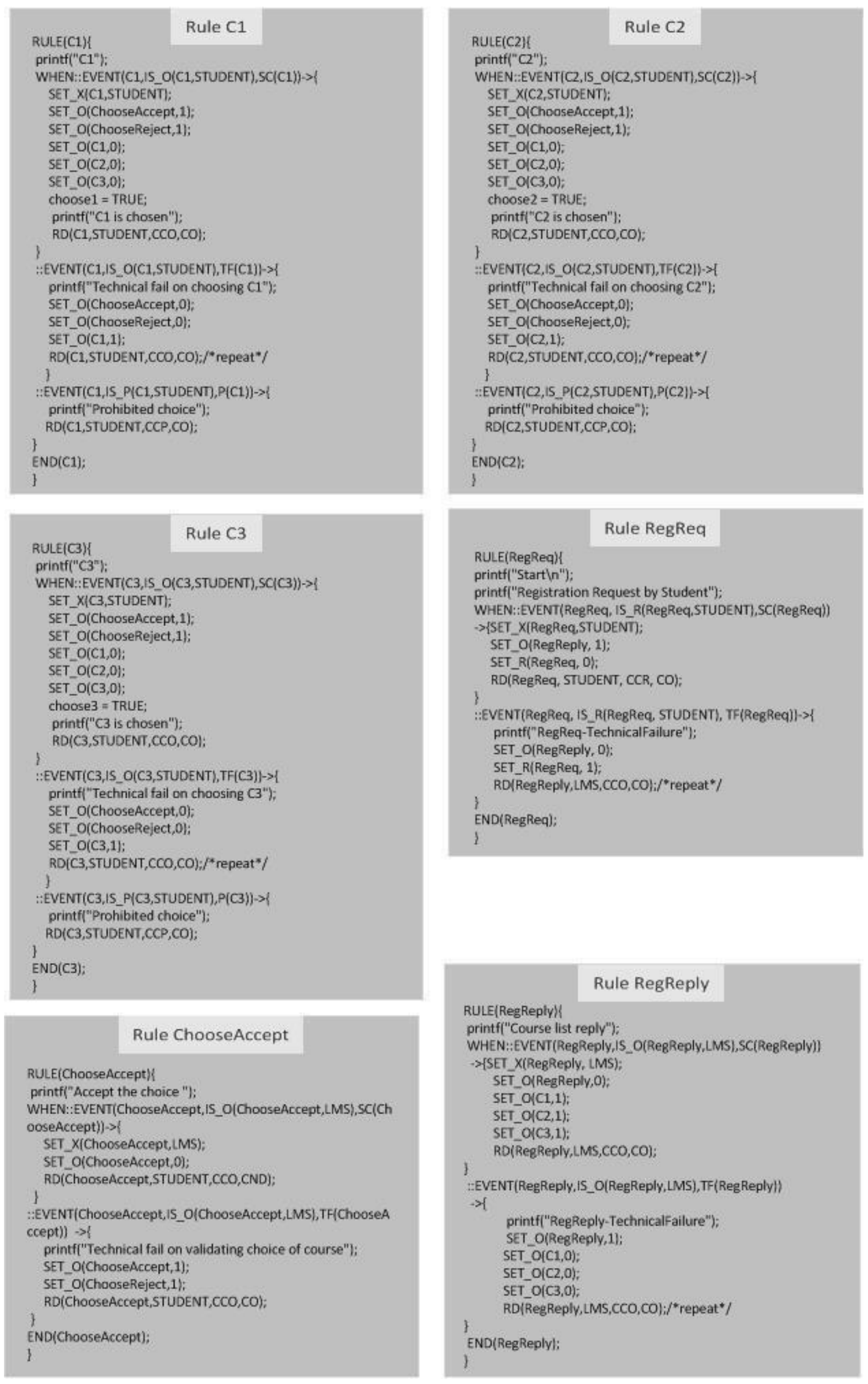
Step 2 – Contract initialisation.

```
  DONE(STUDENT);
  DONE(LMS);
  INIT(RegReq, STUDENT, 1,0,0);
  INIT(RegReply,  LMS, 0,0,0);
  INIT(C1, STUDENT, 0,0,0);
  INIT(C2, STUDENT, 0,0,0);
  INIT(C3, STUDENT, 0,0,0);
INIT(ChooseAccept, LMS, 0,0,0);
  INIT(ChooseReject, LMS, 0,0,0);
```

Step 3 – Deriving contract rules. The model of the registration system is manually converted into the following ECA rules:

1. Rule(RegReq) – Student requesting registration of course
2. Rule(RegReply) – LMS replies with a set of courses
3. Rule(C1) – Student choose Course 1
4. Rule(C2) - Student choose Course 2
5. Rule(C3) - Student choose Course 3
6. Rule(ChooseAccept) - LMS accepts the student to be in the chosen course
7. Rule(ChooseReject) – LMS rejects the student to be in the chose course

Figure 11 and 12 included the implementation of the ECA rules for the adaptation events. As the scenario focuses on the course selection adaptation, learning failure and timeout are not major considerations. Only successful (SC) and technical failure (TF) are considered. Taking Rule(ChooseReject) as an example, a set of if-condition clauses are considered for meeting different situations. For number of courses have been chosen and being rejected, different course would be suggested to students to see if they would like to take it, until there are no course to be chosen from.
One thing to notice in the design is that, for every technical failure taken place, the recovery point would be taken place at the point before the LMS choice is made. For example in RULE(C3), the technical failure would allow the LMS to be choosing between ChooseAccept and ChooseReject, while technical failure happned in ChooseAccept and ChooseReject also recovered to the point where the choice between accept and reject has not been taken place yet.



Rule C1

```
RULE(C1){
 printf("C1");
 WHEN::EVENT(C1,IS_O(C1,STUDENT),SC(C1))->{
    SET_X(C1,STUDENT);
    SET_O(ChooseAccept,1);
    SET_O(ChooseReject,1);
    SET_O(C1,0);
    SET_O(C2,0);
    SET_O(C3,0);
    choose1 = TRUE;
     printf("C1 is chosen");
     RD(C1,STUDENT,CCO,CO);
 }
 ::EVENT(C1,IS_O(C1,STUDENT),TF(C1))->{
    printf("Technical fail on choosing C1");
    SET_O(ChooseAccept,0);
    SET_O(ChooseReject,0);
    SET_O(C1,1);
    RD(C1,STUDENT,CCO,CO);/*repeat*/
   }
 ::EVENT(C1,IS_P(C1,STUDENT),P(C1))->{
    printf("Prohibited choice");
   RD(C1,STUDENT,CCP,CO);
}
END(C1);
}
```

Rule C2

```
RULE(C2){
 printf("C2");
 WHEN::EVENT(C2,IS_O(C2,STUDENT),SC(C2))->{
    SET_X(C2,STUDENT);
    SET_O(ChooseAccept,1);
    SET_O(ChooseReject,1);
    SET_O(C1,0);
    SET_O(C2,0);
    SET_O(C3,0);
    choose2 = TRUE;
     printf("C2 is chosen");
     RD(C2,STUDENT,CCO,CO);
 }
 ::EVENT(C2,IS_O(C2,STUDENT),TF(C2))->{
    printf("Technical fail on choosing C2");
    SET_O(ChooseAccept,0);
    SET_O(ChooseReject,0);
    SET_O(C2,1);
    RD(C2,STUDENT,CCO,CO);/*repeat*/
   }
 ::EVENT(C2,IS_P(C2,STUDENT),P(C2))->{
    printf("Prohibited choice");
   RD(C2,STUDENT,CCP,CO);
}
END(C2);
}
```

Rule C3

```
RULE(C3){
 printf("C3");
 WHEN::EVENT(C3,IS_O(C3,STUDENT),SC(C3))->{
    SET_X(C3,STUDENT);
    SET_O(ChooseAccept,1);
    SET_O(ChooseReject,1);
    SET_O(C1,0);
    SET_O(C2,0);
    SET_O(C3,0);
    choose3 = TRUE;
     printf("C3 is chosen");
     RD(C3,STUDENT,CCO,CO);
 }
 ::EVENT(C3,IS_O(C3,STUDENT),TF(C3))->{
    printf("Technical fail on choosing C3");
    SET_O(ChooseAccept,0);
    SET_O(ChooseReject,0);
    SET_O(C3,1);
    RD(C3,STUDENT,CCO,CO);/*repeat*/
   }
 ::EVENT(C3,IS_P(C3,STUDENT),P(C3))->{
    printf("Prohibited choice");
   RD(C3,STUDENT,CCP,CO);
}
END(C3);
}
```

Rule RegReq

```
RULE(RegReq){
printf("Start\n");
printf("Registration Request by Student");
WHEN::EVENT(RegReq, IS_R(RegReq,STUDENT),SC(RegReq))
->{SET_X(RegReq,STUDENT);
   SET_O(RegReply, 1);
   SET_R(RegReq, 0);
   RD(RegReq, STUDENT, CCR, CO);
}
::EVENT(RegReq, IS_R(RegReq, STUDENT), TF(RegReq))->{
    printf("RegReq-TechnicalFailure");
    SET_O(RegReply, 0);
    SET_R(RegReq, 1);
    RD(RegReply,LMS,CCO,CO);/*repeat*/
}
END(RegReq);
}
```

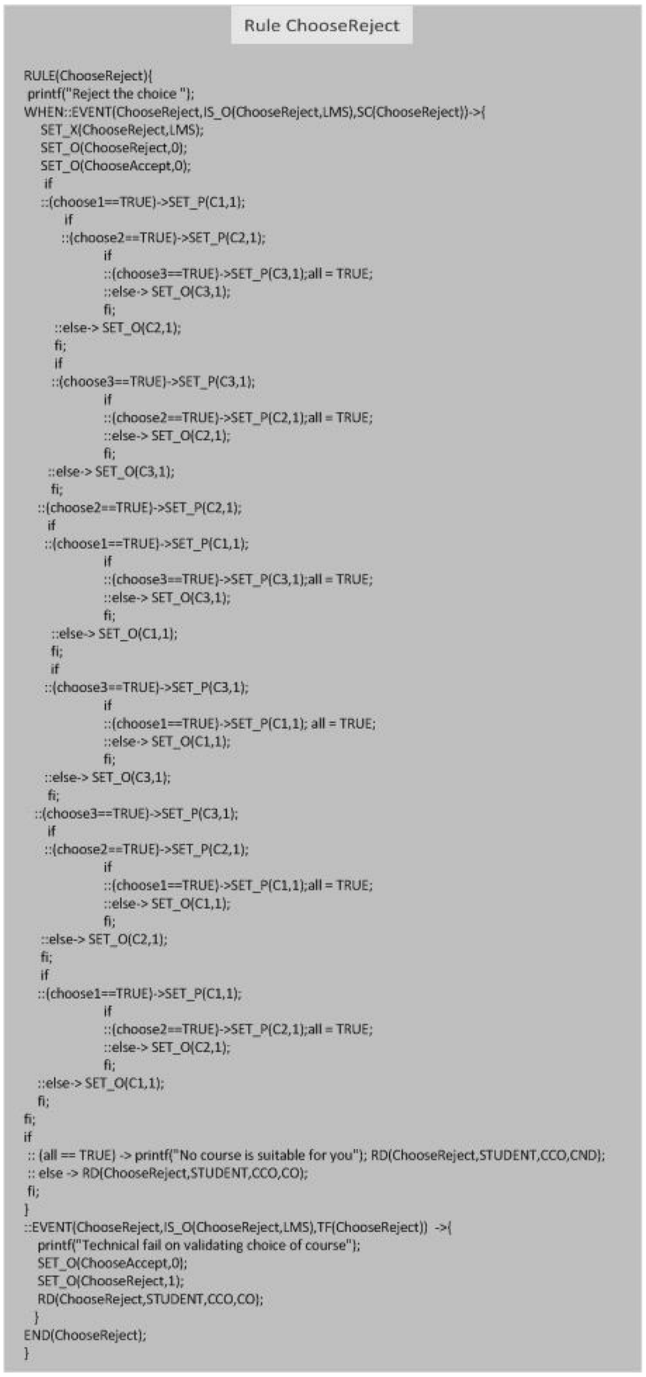
Rule ChooseAccept

```
RULE(ChooseAccept){
 printf("Accept the choice ");
WHEN::EVENT(ChooseAccept,IS_O(ChooseAccept,LMS),SC(Ch
ooseAccept))->{
   SET_X(ChooseAccept,LMS);
   SET_O(ChooseAccept,0);
   RD(ChooseAccept,STUDENT,CCO,CND);
 }
::EVENT(ChooseAccept,IS_O(ChooseAccept,LMS),TF(ChooseA
ccept))  ->{
   printf("Technical fail on validating choice of course");
   SET_O(ChooseAccept,1);
   SET_O(ChooseReject,1);
   RD(ChooseAccept,STUDENT,CCO,CO);
}
END(ChooseAccept);
}
```

Rule RegReply

```
RULE(RegReply){
 printf("Course list reply");
 WHEN::EVENT(RegReply,IS_O(RegReply,LMS),SC(RegReply))
  ->{SET_X(RegReply, LMS);
      SET_O(RegReply,0);
      SET_O(C1,1);
      SET_O(C2,1);
      SET_O(C3,1);
      RD(RegReply,LMS,CCO,CO);
}
 ::EVENT(RegReply,IS_O(RegReply,LMS),TF(RegReply))
  ->{
         printf("RegReply-TechnicalFailure");
         SET_O(RegReply,1);
        SET_O(C1,0);
        SET_O(C2,0);
        SET_O(C3,0);
        RD(RegReply,LMS,CCO,CO);/*repeat*/
}
 END(RegReply);
}
```

Figure. 11. Registration rules



Rule ChooseReject

```
RULE(ChooseReject){
 printf("Reject the choice ");
WHEN::EVENT(ChooseReject,IS_O(ChooseReject,LMS),SC(ChooseReject))->{
   SET_X(ChooseReject,LMS);
   SET_O(ChooseReject,0);
   SET_O(ChooseAccept,0);
    if
   ::(choose1==TRUE)->SET_P(C1,1);
       if
      ::(choose2==TRUE)->SET_P(C2,1);
               if
               ::(choose3==TRUE)->SET_P(C3,1);all = TRUE;
               ::else-> SET_O(C3,1);
               fi;
     ::else-> SET_O(C2,1);
     fi;
     if
    ::(choose3==TRUE)->SET_P(C3,1);
               if
               ::(choose2==TRUE)->SET_P(C2,1);all = TRUE;
               ::else-> SET_O(C2,1);
               fi;
    ::else-> SET_O(C3,1);
    fi;
  ::(choose2==TRUE)->SET_P(C2,1);
    if
   ::(choose1==TRUE)->SET_P(C1,1);
               if
               ::(choose3==TRUE)->SET_P(C3,1);all = TRUE;
               ::else-> SET_O(C3,1);
               fi;
    ::else-> SET_O(C1,1);
    fi;
    if
   ::(choose3==TRUE)->SET_P(C3,1);
               if
               ::(choose1==TRUE)->SET_P(C1,1); all = TRUE;
               ::else-> SET_O(C1,1);
               fi;
   ::else-> SET_O(C3,1);
    fi;
  ::(choose3==TRUE)->SET_P(C3,1);
    if
   ::(choose2==TRUE)->SET_P(C2,1);
               if
               ::(choose1==TRUE)->SET_P(C1,1);all = TRUE;
               ::else-> SET_O(C1,1);
               fi;
   ::else-> SET_O(C2,1);
   fi;
   if
  ::(choose1==TRUE)->SET_P(C1,1);
               if
               ::(choose2==TRUE)->SET_P(C2,1);all = TRUE;
               ::else-> SET_O(C2,1);
               fi;
  ::else-> SET_O(C1,1);
  fi;
fi;
if
 :: (all == TRUE) -> printf("No course is suitable for you"); RD(ChooseReject,STUDENT,CCO,CND);
 :: else -> RD(ChooseReject,STUDENT,CCO,CO);
 fi;
}
::EVENT(ChooseReject,IS_O(ChooseReject,LMS),TF(ChooseReject))  ->{
  printf("Technical fail on validating choice of course");
  SET_O(ChooseAccept,0);
  SET_O(ChooseReject,1);
  RD(ChooseReject,STUDENT,CCO,CO);
 }
END(ChooseReject);
}
```

Figure. 12. Registration rules (cont.)

Step 4 – Prepare LTL formulae. There are some properties of interest are specified as LTL formulae as follows:

```
ltl p1 {! <> (IS_X(ChooseAccept,STUDENT) )} ltl p2 { <>
((IS_X(C1,STUDENT)) ||(IS_X(C2,STUDENT)) ||
(IS_X(C3,STUDENT) ))};
```

Formula p1 is responsible to check if there exists a case that ChooseAccept would not eventually be executed. Formula p2 is to check if there exist other possibilities that the student would choose something out of C1, C2 or C3.

Step5 - Verification. As in scenario 1, step 1 to step 4 gives a general model of the design. By testing it through SPIN, no errors have been found. Formula p1 was verified successfully. In p1 cases for a trail not accessing the ChooseAccept route is verified. By tracing the routes manually a similar conclusion can be made. It is because by tracing the rules at course selection part there exists a chance that all three courses are being rejected and it is terminated as no course can be selected. So p1 is true. However, error is reported for p2. By simulating the counterexample, similar to scenario 1 there exists a case that an infinite loop of technical failure situation would stop the student from going forward to the next stage, and modification of rules are needed correspondingly.

Evaluation. It is found that in cases where course selection is the focus of the system, LPromela is still able to provide verification and design mistakes can be found through testing different constraints using LTL formulae.

5.2 Software Design Methodology

Generally speaking, the LPromela and LPromela LTL Manager are developed upon an agile development model with Test-driven development (TDD). TDD is a framework suggesting that before any code implemented, a test case should be given first, and use the code to remove fail test cases and reduce complexity of the code [29]. For the development of LPromela and the LTL Manager, a scenario (as discussed in previous sections) is set first and codes are written to fit it. Testing the codes with the scenario will come up with a pass/fail result and the processes repeat until all requirements are implemented.

5.3 Testing Strategy

Testing is the key element in TDD and tests are done. Unit Testing is the main method used to see if a requirement is successfully implemented. Therefore System test is done on each requirement on LTL manager.

LPromela itself is a language based on EPROMELA. Therefore tests are done regarding on specific samples and performance. Testing for specific scenarios is discussed above and in Appendix B. Other tests would be discussed below.

5.4 Performance Test and Slicing.

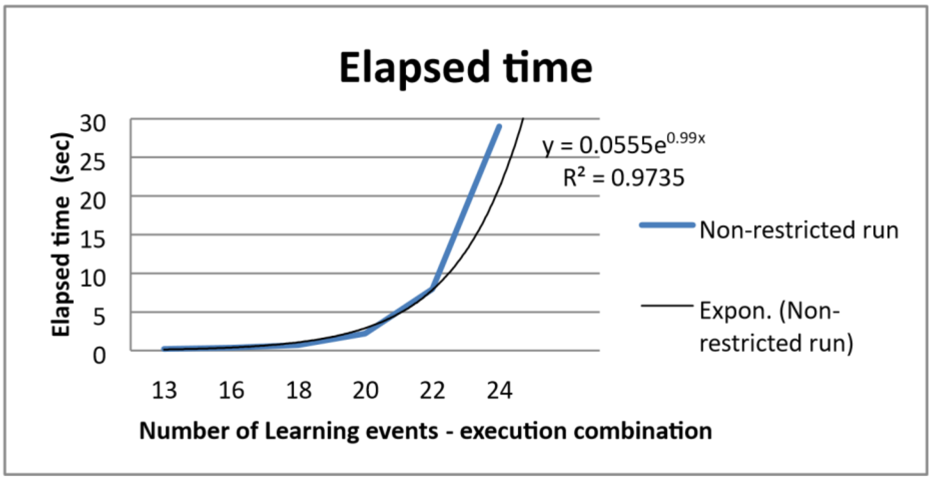


Figure. 13. Verification time versus number of events to verify

Under testing by adjusting the size of model as shown in figure 13, it is shown that the performance drops rapidly for a small increase of learning event-status combination.

Spin uses Depth-first search to deal with a PROMELA models. It means that when the model is very complex, i.e. having a lot of states, the size of the state space would be very large and the searching time would be very long, with a large amount of memory is occupied for the search.

The following is an example when a PROMELA model with some basic learning events.

```
:: L_E(LMS, START, S);
:: L_E(STUDENT, L1,  S);
:: L_E(STUDENT, L1,  TF);
:: L_E(STUDENT, L2,  S);
:: L_E(STUDENT, L2,  TF);
:: L_E(STUDENT, L3,  S);
:: L_E(STUDENT, L3,  TF);
```

When no restriction is added, Spin will go through all the possible states combination to run through, especially when recurring features are introduced. For example in this case, all the technical failure (TF) of Lectures (L_) would cause a recursion to the event itself, as shown in the Figure 13 below. Therefore for Spin there exists a situation that a lecture event keeps having TF for a number of times before it reach the other events, and repeatedly happened until the end, and a large memory is occupied to handle the case.

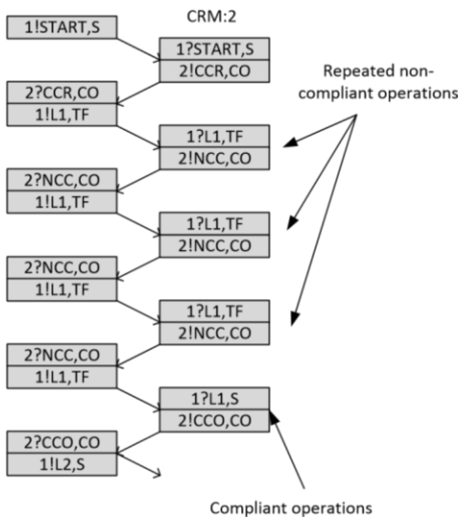


Figure. 14. Random run of non-restricted PROMELA model

To reduce the size of state space and the complexity, a restriction technique called *slicing* is used [10]. Through implementing guards and removing unnecessary states, a smaller model would be obtained. The strength of the guard would be depends on the objective of the verification being made to give a reasonably smaller model. Recall that there are three conditions to check as mentioned in previous section:

- C1) learning event le_i is a subset of the primitive learning events L;
- C2) The ECA rules set, i.e. the right/obligation/prohibition of an certain operation by the role player is matched;
- C3) Constraints stipulated in the adaptation rules are satisfied.

Guards are added on purpose. It will be used to verify specify restriction, or be used simply to reduce search base. For example if we only want to verify C3, i.e. to ensure that constraints are added logically, we could have a very strict guard which provide a search base focusing on the constraints itself. If we would like to check C1 and C2, we would rather make it non-deterministic, i.e. with no guards or fewer guards to loosen the restrictions. Take the learning events mentioned above as an example, if we only want to verify the expected workflow, we can modify the part as follows:

```
::L_E(LMS, START, S) -> L_E(STUDENT, L1,  S) ->
```

```
L_E(STUDENT, L2,  S) -> L_E(STUDENT, L3,  S) -> ...;
```

This modification would strictly limit the flow of the events to check learning operations which would have checked the possibility of logical error for the constraints inside. If we would like to test it with C1 and C2 but reduce the size of the state space, a less tightened guard would be used like:

```
 :: L_E(STUDENT, L1,  S);
 :: L_E(STUDENT, L1,  TF) ->  L_E(STUDENT, L1,  S);
 :: L_E(STUDENT, L2,  S);
 :: L_E(STUDENT, L2, TF) -> L_E(STUDENT, L2, S);
 :: L_E(STUDENT, L3,  S);
 :: L_E(STUDENT, L3,  TF) -> L_E(STUDENT, L3,  S);
........
```

This modification would then be give more freedom to the verifier by reducing all the repeated non-compliant events to be executed only once compared to the case shown in Figure 13. In such case all the conditions would have been checked by removing unnecessary reduplication.

The setting of the guard varies as the purpose varies so as the effect of the slicing. Moreover, the number of learning events – execution combination would also affect the effectiveness of slicing. Here we take the second modification descried above as an example to compare the results of both restricted and non-restricted.

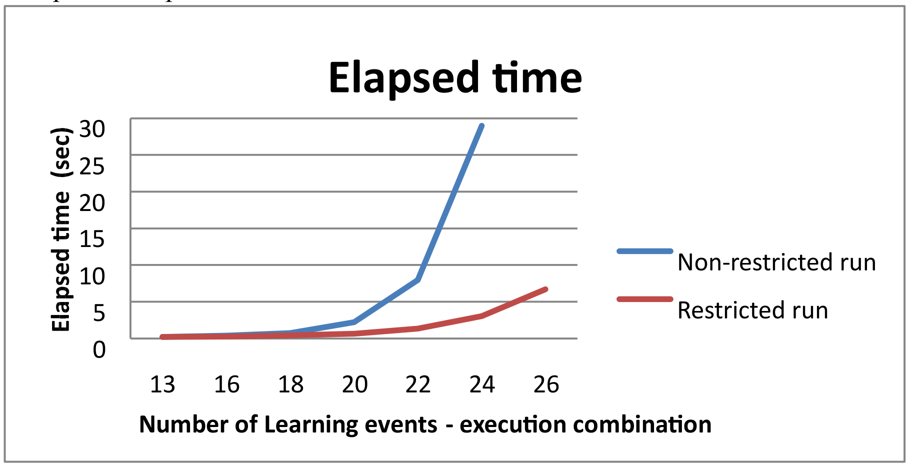


Figure. 15. Comparison of elapsed time in restricted and non-restricted run

From Figure 15 we can find that both elapsed time and memory usage increased with number of events exponentially when no restriction is added. For more learning events are included, the difference of time and memory usage between non-restricted run and restricted run enlarged. Further result of the performance test could be find in Appendix C.

5.5 Testing of LPromela LTL Manager

There are two forms of test performed: Integrated Testing and System Testing. For Integration testing, it is to ensure different components are integrated correctly. In this project, integrated testing is to check if SPIN successfully called from LTL Manager and return the same result as in command line, and MySQL server connection is correct. System testing is to check if all requirements are met. In this project, the aim of the test is to check if LPromela model is successfully checked in SPIN, LTL formulae are added correctly.
Test chosen and results are shown in Appendix D.

6 Conclusion and Future Work

6.1 Evaluation against Aim and Objectives

The aim of the project is to *develop a model-checking based verification tool for adaptive e-learning system represented in Event Condition Action (ECA) rules.* The aim is completed by constructing an extension of PROMELA for eLearning purpose, LPromela, and an LTL manager facilitate the verification is implemented.

The objectives of the project are as follows and evaluation would be done correspondingly.

To classify the similarity and differences among the adaptive e-learning system. In Section 2.2, we have found there is a large spectrum of adaptive eLearning system. As from [11] and other models discussed, many of them are greatly diverse in terms of design domain and concepts. Similarities are all concluded in the definition stated in [2,3]. Therefore this objective is fulfilled.

To identify usual practices in evaluating the system. In Section 2.3, a discussion on current methods to validate and evaluate adaptive learning systems is included. CAVIAr is discussed as a tool proposed for system validation and student evaluation is the main method used in most system for evaluation. This provides evidence on the need of verification tools on the design before implementation.

To modify EPROMELA to an education-oriented PROMELA extension. In Section 3, a discussion starting from ECA rules to implement LPromela model is discussed. A number of features are added in:

- Specified ECA rules for modeling eLearning events
- An operation parser to translate operations to PROMELA

Current investigated cases are mostly abstract and are part of a realistic case. Therefore the development of LPromela can be more completed by considering real case like verification of MOOC design.

To develop a verification tool for adaptive eLearning system based on SPIN. In Section 4, a verification tool in GUI mainly for managing LTL formulae is built.

Features implemented are as follows:

- A GUI for LTL formulae management
- Automated verification with LTL formulae
- Automated guided simulation when errors are found in verification All functions required to check the model and model with LTL formulae are implemented. However, as the target of the project is not only for professional SPIN users but also teachers with no knowledge about SPIN and LTL formulae, some functions should be implemented in the future:
- A setup wizard for LPromela model in GUI for teachers
- Graphical representation of eLearning model for user without knowledge of PROMELA

To evaluate the system using realistic e-learning scenarios. Some test cases are provided above to explain how the tool can be used to help verifying the design of an adaptive eLearning System. Test case with a real course structure is modelled as test in Appendix B. However, cases of larger scale system like MOOC should also be investigated in the future.

6.2 Conclusions and Future Works

Adaptive eLearning is still an on-going research field. The missing of standards, large spectrum of adaptive eLearning systems and lack of testing method before implementation as mentioned before are things that can be researched on. In this dissertation we have discussed the need and implementation of a verification tool for the design stage of an adaptive eLearning system. Design and implementation of LPromela, the extension of PROMELA on the issue, and a GUI for managing LTL formulae and handling LPromela code is introduced. Finally some cases are investigated with the testing results included. We have also evaluated the works that have been done.

The dissertation is a first attempt of such tools and there are some potential further works to be completed. Scenarios on course learning workflow and course selection have been discussed. However at [11] we also identified that except activities and domain models, peer-interaction is one of the most popular features to be included in adaptive eLearning system and verification is needed. Test case on that can be more complex. Another extension direction would be investigating the use of the tools on verifying design of Massive Open Online Course (MOOC), which has a largest variety of students and adaptation is probably required.

The tool itself is also not totally mature. As the main target group of the project is teachers, who may not have the knowledge about PROMELA or even programming, a graphical representation and a GUI for modeling the eLearning activities would be a

future work that should be taken. Moreover, the current verification result and simulation tracks are both textual. Using of SPIN self-generated graphical representation of simulation tracks and standard XML tags to be generated during simulation, extension of using graphical representation to demonstrate the simulation would be giving the teacher a help and a more user-friendly verification experience. Therefore, a set of standardized XML tags should be setup, and an extension of current LTL manager can be implemented to collect these XML and translate them into graphical representation.

Current LTL formulae management system used MySQL database, and everyone who has the access right to a particular server would be able to see every formula recorded in that database. Correlating LTL formulae with the specific model would be a better implementation as LTL formulae are usually model-specific instead of generally used. This would be one of the extension works that can be done.

7 References

1. Griff, E. R., & Matter, S. F. (2013). Evaluation of an adaptive online learning system. British Journal of Educational Technology, 44(1), 170-176.
2. Paramythis, A., & Loidl-Reisinger, S. (2003). Adaptive learning environments and elearning standards. In Second European Conference on e-Learning (Vol. 1, pp. 369-379).
3. Laroussi, M. (2012, January). Ontology in adaptive Learning environment. In Workshop on Learning Technology for Education in Cloud (LTEC'12) (pp. 167177). Springer Berlin Heidelberg.
4. Zafar, A. and Albidewi, I. (2015) "Evaluation Study of eLGuide: A Framework for Adaptive e-Learning" *Computer Applied Engineering Education* 23 (4). 542-555.
5. Staikopoulos, A., O'Keeffe, I., Rafter, R., Walsh, E., Yousuf, B., Conlan, O., & Wade, V. (2012). AMASE: A framework for composing adaptive and Personalised Learning Activities on the Web. In Advances in Web-Based Learning-ICWL 2012 (pp. 190-199). Springer Berlin Heidelberg.
6. Bieliková, M., Šimko, M., Barla, M., Tvarožek, J., Labaj, M., Móro, R., ... & Ševcech, J. (2014). ALEF: from application to platform for adaptive collaborative learning. In Recommender Systems for Technology Enhanced Learning (pp. 195225). Springer New York.

7. Solaiman, E. Sfyrakis, I. Molina-Jimenez, C., (2016). A state aware model and architecture for the monitoring and enforcement of electronic contracts. 18th Conference on Business Informatics (CBI), IEEE.
8. Solaiman, E. Sfyrakis, I. Molina-Jimenez, C., (2016). High Level Model Checker Based Testing of Electronic Contracts. In Cloud Computing and Services Science, Springer.
9. Solaiman, E., Sfyrakis, I., & Molina-Jimenez, C. (2015). Dynamic Testing and Deployment of a Contract Monitoring Service., 5th International Conference on Cloud Computing and Services Science (CLOSER 2015).
10. Millett, L. I., & Teitelbaum, T. (2000). Issues in slicing PROMELA and its applications to model checking, protocol understanding, and simulation. International Journal on Software Tools for Technology Transfer, 2(4), 343-349.
11. Magnisalis, I., Demetriadis, S., & Karakostas, A. (2011). Adaptive and intelligent systems for collaborative learning support: a review of the field. Learning Technologies, IEEE Transactions on, 4(1), 5-20.
12. Dittrich, K. R., Gatziu, S., & Geppert, A. (1995). The active database management system manifesto: A rulebase of ADBMS features. In Rules in Database Systems (pp. 1-17). Springer Berlin Heidelberg.
13. Molina-Jimenez, C. Shrivastava, S. Solaiman, E. Warne, J. (2004). Run-time monitoring and enforcement of electronic contracts. Journal of Electronic Commerce Research and Applications, Elsevier.
14. Holzmann, G. J. (2004). The SPIN model checker: Primer and reference manual (Vol. 1003). Reading: Addison-Wesley.
15. Caeiro, M., Anido, L., & Llamas, M. (2003). A critical analysis of IMS Learning Design. In Designing for Change in Networked Learning Environments (pp. 363367). Springer Netherlands.
16. Shi, L., Al Qudah, D., & Cristea, A. I. (2013, July). Designing social personalized adaptive e-learning. In Proceedings of the 18th ACM conference on Innovation and technology in computer science education (pp. 341-341). ACM.
17. Gallardo, M. D. M., & Panizo, L. (2014). Extending model checkers for hybrid system verification: the case study of SPIN. Software Testing, Verification and Reliability, 24(6), 438-471.
18. Melia, M., & Pahl, C. (2009). Constraint-based validation of adaptive e-learning courseware. Learning Technologies, IEEE Transactions on, 2(1), 37-49.
19. Mustafa, Y. E. A., & Sharif, S. M. (2011). An approach to adaptive e-learning hypermedia system based on learning styles (AEHS-LS): Implementation and evaluation. International Journal of Library and Information Science, 3(1), 15-28.
20. Dagger D.(2006) Personalised eLearning Development Environments. PhD dissertation, Univ. of Dublin, 2006
21. Abdelsadiq, A. (2013). A toolkit for model checking of electronic contracts. PhD dissertation, Newcastle University, 2013.

22. Kostolányová, K., Šarmanová, J., Takács, O. (2012) Adaptive e-learning and Its Evaluation. Journal on Efficiency and Responsibility in Education and Science, 5(4), 212-225
23. Baier, C., Ciesinski, F., & Größer, M. (2004, June). PROBMELA: a modeling language for communicating probabilistic processes. In Formal Methods and Models for Co-Design, 2004. MEMOCODE'04. Proceedings. Second ACM and IEEE International Conference on (pp. 57-66). IEEE.
24. Tripakis, S., & Courcoubetis, C. (1996). Extending promela and spin for real time. In Tools and Algorithms for the Construction and Analysis of Systems (pp. 329348). Springer Berlin Heidelberg.
25. Mali, Y., & Van Wyk, E. (201s). Building extensible specifications and implementations of Promela with AbleP. In Model Checking Software (pp. 108125). Springer Berlin Heidelberg.
26. Jim, W. S. (2012). Design and Implementation of a BPMN to PROMELA Translator. MSc Dissertation Project. Newcastle University, 2012.
27. DB-Engines Ranking of Relational DBMS. (2015). Retrieved August 24, 2015, from http://db-engines.com/en/ranking/relational+dbms
28. Oracle (2015) MySQL Connector/J Developer Guide. Retrieved August 24, 2015, from http://dev.mysql.com/doc/connector-j/en/index.html
29. Beck, K. (2003). Test-driven development: by example. Addison-Wesley Professional.
30. Solaiman, E. Sun, W. Molina-Jimenez, C. (2015). A Tool for the Automatic Verification of BPMN Choreographies. 12th IEEE International Conference on Services Computing (SCC).
31. Solaiman, E. Molina-Jimenez, C. Shrivastava, S. (2003). Model checking correctness properties of electronic contracts. 2003: First International Conference Service-Oriented Computing-ICSOC, Springer.
32. C. Molina-Jimenez, S. Shrivastava, E. Solaiman, and J. Warne, "Contract representation for run-time monitoring and enforcement," in IEEE International Conference on E-Commerce (CEC 2003), 2003.
33. B. Awaji, E. Solaiman, and A. Albshri, "Blockchain-based applications in higher education: A systematic mapping study," in 5th ACM International Conference on Information and Education Innovations, 2020.
34. B. Awaji, E. Solaiman, and L. Marshall, "Blockchain-based trusted achievement record system design," in 5th ACM International Conference on Information and Education Innovations, 2020.
35. B. Awaji, E. Solaiman, and L. Marshall, "Investigating the requirements for building a blockchain-based achievement record system," in ACM 5th International Conference on Information and Education Innovations, 2020.
36. B. Awaji and E. Solaiman, "Design implementation and evaluation of blockchain-based trusted achievement record system for students in higher

education," in 14th International Conference on Computer Supported Education (CSEDU), 2022.

37. A. Noor, K. Mitra, E. Solaiman, A. Souza, D. Jha, U. Demirbaga, P. Jayaraman, N. Cacho, and R. Ranjan, "Cyber-physical application monitoring across multiple clouds," Computers and Electrical Engineering, 2019.
38. E. Solaiman, R. Ranjan, P. P. Jayaraman, and K. Mitra, "Monitoring internet of things application ecosystems for failure," IT Professional IEEE, 2016.
39. A. Alqahtani, Y. Li, P. Patel, E. Solaiman, and R. Ranjan, "End-to-end service level agreement specification for iot applications," in 2018 IEEE International Conference on High Performance Computing & Simulation (HPCS), 2018.
40. A. Alqahtani, E. Solaiman, P. Patel, S. Dustdar, and R. Ranjan, "Service level agreement specification for end-to-end iot application ecosystems," Software: Practice and Experience, 2019.
41. A. Alqahtani, D. N. Jha, P. Patel, E. Solaiman, and R. Ranjan, "SLA-aware approach for iot workflow activities placement based on collaboration between cloud and edge," in 1st Workshop on Cyber-Physical Social Systems (CPSS), 2019.
42. A. Alzubaidi, E. Solaiman, P. Patel, and K. Mitra, "Blockchain-based sla management in the context of iot," IT Professional IEEE, 2019.
43. A. Alzubaidi, K. Mitra, P. Patel, and E. Solaiman, “A blockchain-based approach for assessing compliance with sla-guaranteed iot services”2020 ieee international conference on smart internet of things (smartiot), IEEE, 2020.
44. A. Aldweesh, M. Alharby, E. Solaiman, and A. van Moorsel, "Performance benchmarking of smart contracts to assess miner incentives in ethereum," in IEEE 14th European Dependable Computing Conference (EDCC), 2018.
45. A. Albshri, B. Awaji, and E. Solaiman, "Investigating the requirement of building blockchain simulator for iot applications," in 2022 IEEE International Conference on Smart Internet of Things (SmartIoT), 2022.
46. A. Albshri, A. Alzubaidi, B. Awaji, and E. Solaiman, "Blockchain simulators: a systematic mapping study," in 2022 IEEE International Conference on Services Computing (SCC), 2022.
47. C. Molina-Jimenez, I. Sfyrakis, E. Solaiman, I. Ng, M. W. Wong, A. Chun, and J. Crowcroft, "Implementation of smart contracts using hybrid architectures with on-and off-blockchain components,". 8th International Symposium on Cloud and Service Computing (SC2), IEEE, 2018.

48. C. Molina-Jimenez, E. Solaiman, I. Sfyrakis, I. Ng, and J. Crowcroft, "On and off-blockchain enforcement of smart contracts," Euro-Par 2018: European Conference on Parallel Processing, Springer, 2018.
49. E. Solaiman, T. Wike, and I. Sfyrakis, "Implementation and evaluation of smart contracts using a hybrid on-and off-blockchain architecture," Concurrency and Computation: Practice and Experience, 2021.
50. A. Alzubaidi, K. Mitra, and E. Solaiman, “A blockchain-based SLA monitoring and compliance assessment for IoT ecosystems”, Journal of Cloud Computing: Advances, Systems and Applications, Springer, 2023.
51. A. Albshri, A. Alzubaidi, M. Alharby, B. Awaji, K. Mitra, E. Solaiman, “A conceptual architecture for simulating blockchain-based IoT ecosystems”, Journal of Cloud Computing: Advances, Systems and Applications, Springer, 2023.