

Received October 29, 2018, accepted November 14, 2018, date of publication November 28, 2018, date of current version December 27, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2883610

# UMLPACE for Modeling and Verification of Complex Business Requirements in Event-Driven Process Chain (EPC)

ANAM AMJAD<sup>1</sup>, FAROOQUE AZAM<sup>1</sup>, MUHAMMAD WASEEM ANWAR<sup>1</sup>, WASI HAIDER BUTT<sup>1</sup>, MUHAMMAD RASHID<sup>2</sup>, AND AAMIR NAEEM<sup>1</sup>

<sup>1</sup>Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Islamabad 44000, Pakistan

<sup>2</sup>Computer Engineering Department, Umm Al-Qura University, Makkah 21421, Saudi Arabia

Corresponding author: Muhammad Waseem Anwar (waseemanwar@ceme.nust.edu.pk)

**ABSTRACT** Business processes (BPs) are often modeled to elaborate process-related business requirements (BRs). This leads to verify the complex BRs in early automation stages. Among various BP languages, event-driven process chain (EPC) is a well-known semi-formal modeling language, which is verifiable after transforming it into any other formal language, such as, timed automata or Petri nets. However, full potential of EPC cannot be exploited as yet because existing EPC tools can only model or verify the simple patterns and they lack the modeling/verification of complex patterns. Moreover, only the proprietary tools are available, which limit its applicability toward overwhelming utilization amongst widespread practitioners and research community. This research work is the first attempt to make EPC more expressive in terms of modeling complex patterns for real time systems. Particularly, the UMLPACE (Unified Modeling Language Profile for Atomic and Complex events in EPC) has been developed, which adapts the concepts of UML activity diagram for representing both simple as well as complex patterns in EPC. As a part of research, a complete open source transformation engine is developed to transform UMLPACE source models into timed automata target models for the verification of complex BPs. The implementation of transformation engine is carried out in JAVA language and Acceleo tool through model-to-text transformation approach. Finally, the broader applications of UMLPACE are demonstrated through two benchmark case studies.

**INDEX TERMS** Business processes, BPML, complex events, EPC, UMLPACE, verification.

## I. INTRODUCTION

Business Processes (BPs) consist of pre-defined activities to achieve a certain business objective. In the development of business process, the first step is to gather the customer's requirements which further need to be documented. In the beginning, this documentation is done using natural language which often leads to the misinterpretation. In order to overcome this issue, different process modeling languages are developed. The purpose of these languages is to document the business requirement in an efficient manner without any ambiguity or misinterpretation. The process modeling languages are in graphical or textual formats to facilitate business users. Different process modeling languages (e.g. Business Process Modeling Language - BPMN [1], Business Process Execution Language - BPEL [2], Event-driven Process Chain -EPC [3] and EDOC [4] etc.) represent different

set of notations for modeling but the essence remains same i.e. flow and activities in all of them. These languages are commonly used by business user, business analyst and software developers [5]. The ultimate objective is to verify the correctness of initially collected business requirements in early automation stages.

Event-driven Process Chain (EPC) is a well-known BPML supporting functional, dynamic and organization view. It is a semi-formal language, which supports the modeling and represents the business requirements in graphical view. It has many advantages over other BPML, such as, it has a dynamic view which supports to model the behavior of the system. The primary task of EPC is modeling but it can also perform different activities such analysis, simulation and verification. These activities are often performed with the help of other formal models like Petri net. Events are first class-citizens

meaning EPC revolves around events to model the occurrence of anything happening in the real time world. It is a descriptive language understandable by both human and machine. EPC is widely used in industry and commercial areas [2].

In modern business processes, business requirements can be of two types; simple and complex. Simple requirement can be related to simple patterns involving synchronization, merge or termination patterns. In EPC, events which are satisfying these simple patterns are called atomic events. The issue with these atomic events is the inefficiency to model the complex requirements of the stakeholders. For example, “The order should be delivered to the customer in 24 hours” cannot be catered using atomic events of EPC. To facilitate this requirement, there is a need of complex event pattern (e.g. temporal pattern).

The complex events satisfy the complex requirement of the stakeholders, which can be related to the logical, temporal, spatial, data or trend pattern categories. The simple patterns belong to the workflow patterns whereas complex patterns are introduced from the innovative field of Complex Event Processing (CEP). These complex events are being used in different languages like Kim and Oussena [6] has used Archimate language to facilitate the complex events in order to obtain the full potential of the Archimate. Furthermore, real time applications such as ATM cannot be modeled using atomic events, so complex events are involved as presented in [7]. The advantages of using these complex events in EPC will make it more expressive. Moreover, EPC can be used to model the complex and real time business requirements through these complex event patterns.

Modeling and verification are two main activities for any process modeling language. Modeling of business process is performed for the analysis, improvement, standardization, design and documentation of business process. Modeling is a core element to represent all the activities to be performed in a way that is understandable by stakeholders. Verification of the models being generated is a common trend using process modeling languages. Early verification is performed after modeling phase. Although few studies [7], [8] provide partial support for complex events in EPC, there is no study available yet to the best of our knowledge that provides the full modeling and verification support of complex event patterns in EPC. Consequently, it is not possible to model and verify complex business requirements in EPC.

In this article, a novel framework is introduced to model and verify the complex business requirements by utilizing the major Model Based System Engineering (MBSE) activities i.e. modeling, transformation and verification [9]. The major contributions of this research, as shown in **Figure 1**, are summarized as follows:

- Firstly, the UMLPACE (Unified Modeling Language Profile for Atomic and Complex events in EPC) has been introduced, which adapts the concepts of UML activity diagram for representing both simple as well as complex patterns in EPC. Particularly, UMLPACE profile incorporates three sub-profiles i.e. Events and

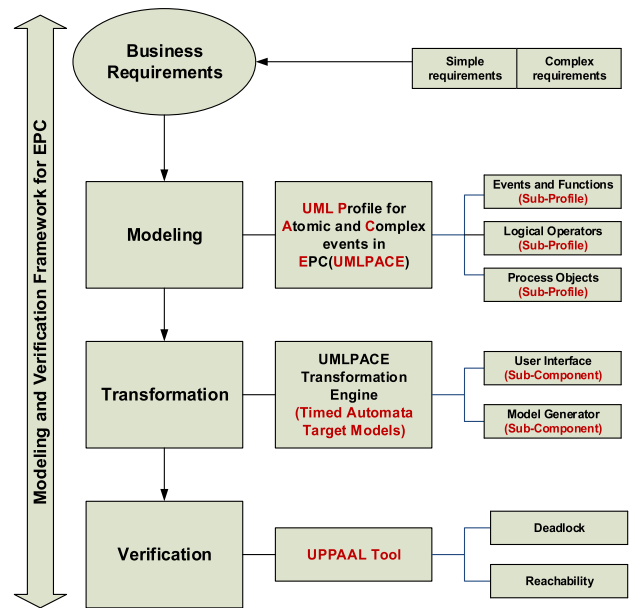


FIGURE 1. Overview of Research.

Function, Logical Operators and Additional Process Objects

- Secondly, a complete open source transformation engine is developed to transform UMLPACE source models into timed automata target models for the verification of complex BPs. Particularly, the implementation of transformation engine is carried out in JAVA language and Acceleo tool through Model-to-Text (M2T) transformation approach.
- Finally, the validation is performed through two benchmark case studies i.e. Automatic Teller Machine (ATM) and Patient Flow System (PFS). Particularly, the complex requirements of both case studies have been modeled through UMLPACE. Subsequently, the UMLPACE source models are automatically transformed to timed automata models through transformation engine. Finally, deadlock and reachability properties are verified through UPPAAL tool.

Rest of the paper is organized as follow; Section II highlights the preliminary concept of this research which involves EPC, literature review, research gaps and proposed solution. Section III provides complete description of the proposed UMLPACE. The implementation details of UMLPACE transformation engine are provided in Section IV. The validation of proposal is performed in Section V through two benchmark case studies. The comparative analysis of UMLPACE with other state of the art approaches and discussion is performed in Section VI. Finally, the conclusion and future work are given in Section VII.

## II. PRELIMINARIES

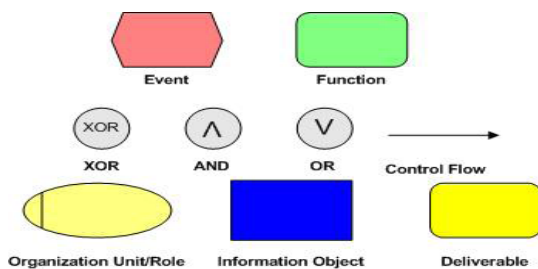
In this section, the syntax and semantics of EPC, related work, research gaps and proposed solution are discussed.

### A. EVENT-DRIVEN PROCESS CHAIN (EPC)

EPC was developed through combined efforts of University of Saarland, Germany and global software system, SAP AG in 1992 [5]. Firstly, it was integrated into the ARIS (Architecture of Integrated Information System) tool. SAP R/3 models were modeled by using EPC in 1992 indicating the usability of EPC as soon as it developed in software global software company. After that, a lot of tools offered business process modeling through EPC such as MS Visio and ADONIS. In the beginning, EPC was developed for modeling where requirements of any business process were gathered and modeled graphically. One of the prominent reasons of EPC popularity is its easy understandability. After gaining much attention in both industrial and commercial area, EPC was used for the simulation or verification. Simulation or verification was performed to make the output model error free.

### B. SYNTAX AND SEMANTICS OF EPC

In **Figure 2**, The EPC notations are provided for depicting the syntax. EPC is a directed connected graph [11], [12]. The notations or elements used for the EPC modeling are as follows:



**FIGURE 2.** Elements of EPC Modeling Notations.

Events are passive elements defining pre-condition and post-condition of the function of EPC. Event is represented as hexagon. Functions are active elements performing activity/action. Function is represented as rounded rectangle. Control Flow is used to connect event, function and logical operators. It is represented as uni-directional arrow. Logical Operators are of three types (AND, OR, XOR). Logical operators are represented as circle in EPC.

- AND operator is used for parallel execution of paths.
- OR operator is used to represent the execution of at least one path.
- XOR: Exactly one path is executed.

Additional process objects are linked with function providing additional information. Deliverable, information object and organization unit belongs to this category [6]

- Deliverable is the products or services produced by the function.
- Information Object represents the input/ output of the function in terms of information.

- Organizational Unit/Role represents the person or role responsible for performing the certain function which is described through organizational role/ unit.

There are few important semantics of EPC those should be taken into consideration for modeling the error-free EPC diagram:

1. EPC always starts with an event called starting event. There can be one or more than one starting event in EPC.
2. EPC ends on the event called ending event. There can be one or more than one ending event in EPC [13], [14].
3. Events and functions should be alternated.
4. Events cannot make decisions like (OR/XOR), event can only be associated with AND operator. Function can use all these three logical operators for decision making.

### C. RELATED WORK

In this section, related work is presented to get insight on the usability and applicability of EPC in different dimensions. Krumeich *et al.* [7] propose the selection of eight complex event patterns from the Complex Event Processing (CEP) and the modeling notations of these complex event patterns through EPC are demonstrated. Annotations are used for the modeling of complex events in EPC. These annotations for modeling are described theoretically; no tool is used or developed. Amjad *et al.* [8] present the verification of two complex event patterns in EPC. For this purpose, verification through two formal methods; timed automata and time Petri net is performed. It is concluded through case study that timed automata supports both complex event patterns. Stefanov and List [11] also extended the EPC language to make it suitable for the performance measurement. Meta-model of EPC is extended and proposed notations are discussed. Gross and Doerr [15] have contributed in the area of Requirement Engineering (RE) and EPC. The comparison on activity diagram and EPC is conducted in the context of business process requirements.

Various extensions of EPC are also proposed to make it suitable to the specific domain. For example, Iris Reinhartz-Berger *et al.* [16] present the extension of EPC i.e. Configurable EPC(C-EPC) to support the configuration of EPC. The Application-based Domain Modeling (ADOM) is used for the design of reference models with EPC which increase the flexibility, variability and adaptability of the design. Similarly, Mendling *et al.* [17] extend the EPC to yet-another Event-driven Process Chain (yEPC). The purpose of this extension is to take advantage of YAWL language and consequently, additional support for workflow patterns. Rosemanna and Van der Aalst [18] also present an extension of EPC which is Configurable EPC (C-EPC) for reference modeling. The reason of this extension is to make EPC more expressive towards enterprise systems. Wil M.P. In another study, Van der Aalst *et al.* [19] extend the EPC language to C-EPC to support the configuration for setting the

requirements of the specific organization. The semantics of configurable EPC is presented in the paper. Petri-nets (Work flow nets) are also used for mapping to EPC language. The errors caused at any step are highlighted and suggestion for corrections are made according to the rules provided. Another important extension of EPC is proposed by Xue *et al.* [21] i.e. lightweight event-driven process chain (lightEPC). A complete semantics of extended language is introduced. Also, two error patterns are presented. The rules to eliminate the errors at design time are also discussed.

There exist several studies in the literature that deals with the verification of EPC. Such verification is primarily meant for the elimination of errors which are present in the EPC models. Moreover, EPC is a semi-formal language which is verified through formal methods such as Petri nets. In this context, Mendling *et al.* [22] has presented the prediction and detection of errors in EPC. For this purpose, 600 SAP reference models which are modeled using EPC are selected. The verification is performed by translation it to the YAWL language through WofYawl tool. In another study, Van der Aalst [23] has mapped to the semantics of Petri-nets taking advantage of the high-level language. Finally, the formalization and verification of EPC with Petri-net is performed. In another study, Mendling *et al.* [24] discuss the two main problems of EPC language; OR join and multiple start events. These problems are specified in order to reduce the gap between conceptual models and workflow executions. Finally, the solution of OR join and multiple start events in EPC is proposed. In another important study, Mendling *et al.* [25] generate correct EPC from the configurable EPC. The problems encountered in the generation of correct EPC are also elaborated. Automatic transformation of one language (C-EPC) to another (Correct EPC) is performed which enables the correct EPC by using transformation rules. Gruhn and Laue [26] have also contributed in the area EPC verification. 1000 EPC models are selected for verification. A great number of models are selected from SAP R/3 whereas others are from thesis or textbooks etc. Comparison of the soundness result with three tools (i.e. ProM, EPCTools and YAWL) is performed.

Another interesting area is the translation of EPC graphical models into different textual languages. In this context, Kapuruge *et al.* [12] introduced the textual language for EPC i.e. EPClets. The EPC is a directed graph but this proposed approach is able to generate the text along the EPC graphical description. This study benefits the developers as codes or textual languages are easy to understand for them. The comparison of EPClets and EPC Markup Language (EPML) is also performed. Similarly, Winter and Simon [27] propose a Graphic eXchange Language (GXL) that represents different graphical models in XML. As a proof of concept, EPC and Petri nets are exchanges. Their meta-schema is also discussed for the better description. Mei *et al.* [28] propose Rule merged Event-driven Process Chain (REPC) framework to reduce the gap between process modeling and business process execution. Particularly, four important properties (i.e.

multi-view, semantic maintenance, business rule and execution support) are introduced in REPC. Comparison with other languages such as BPMN, ARIS and SEPC shows that REPC yields better results to support these four properties. In another study, Yongsiriwit *et al.* [29] propose Business Process as a Service (BPaaS) by integrating two business process modeling languages i.e. EPC and BPMN. Another important study is performed by Schunselaar *et al.* [30] where business requirements are modeled through EPC. Subsequently, the EPC models are directly transformed to ERP systems.

#### D. RESEARCH GAPS AND PROPOSED SOLUTION

It is concluded from the above section that there are two types of events in EPC i.e. atomic events and complex events. Atomic events in EPC represent the support of simple patterns such as parallel synchronization. For example, the order is delivered by company and payment is made by the customer represent the parallel synchronization which is represented by the atomic events in EPC. The main activities of EPC are modeling, simulation and verification. In the above related work, these activities are well-supported by atomic events of EPC using different tools. Consequently, simple business requirements are easily modeled and verified through atomic events of EPC.

Complex events in EPC represent the support of complex event patterns such as event time and event cardinality. For example, user cannot enter incorrect PIN in ATM more than three times. This example illustrates the cardinality (three) of the event and it belongs to the complex event category. It is concluded from the related work that the support for complex events in EPC is mandatory to model and verify the complex business requirements. However, the modeling and verification of complex events is not supported yet in EPC. There are limited studies available [7], [8], [31], in literature that try to support complex events in EPC. Therefore, there is a strong need to develop a complete solution that support both atomic and complex events in EPC for the modeling and verification of large and complex business requirements.

This study proposes UMLPACE to support the modeling and verification of both atomic as well as complex events in EPC. Particularly, six complex events are included in EPC [7] as follows:

1. **Event Cardinality:** 'n' represent the number of time an event occurs. So,  $n > 1$  means that any event occurring more than one time defines its cardinality.
2. **Event Exclusion:** A condition where one event should be absent for the occurrence of another event is event exclusion.
3. **Event Sequence:** Event occurring in a specified order defines its sequence.
4. **Event Time:** Event is bound to the time condition. Before or after time conditions can be defined by using event time.

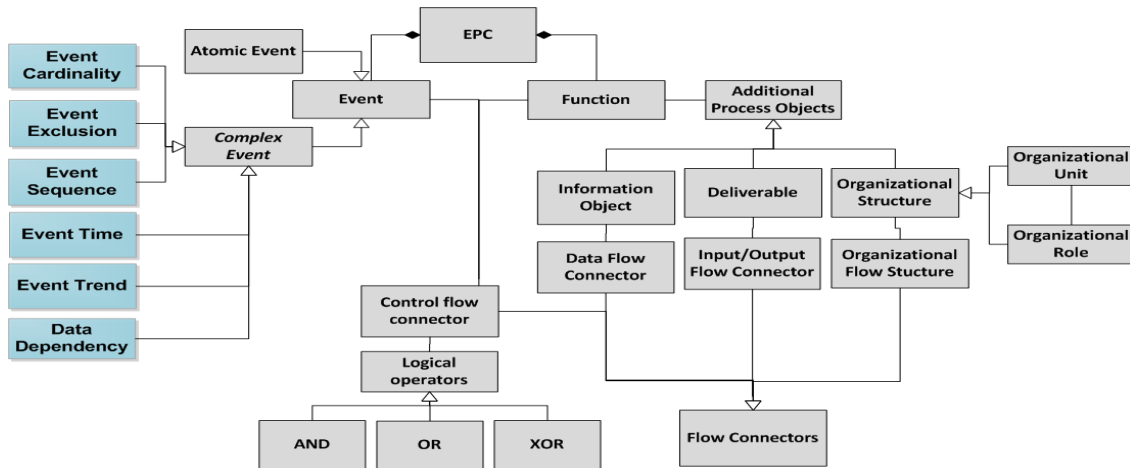


FIGURE 3. Extended EPC Meta-model.

5. **Event Trend:** Two or more than two events can represent the trend of occurring events in increasing or decreasing order.
6. **Data Dependency:** In data dependency, data condition must hold true for the existing data.

### III. UMLPACE

Unified Modeling Language (UML) is a general-purpose language. UML can be extended through profile mechanism to make UML a domain specific modeling language. One example of UML profile is MARTE which was developed for the Modeling and Analysis of Real-Time and Embedded systems [32]. UML profile is a package with the “Profile” guillemet. UML profile mainly consists of stereotypes, tagged values and constraints. We have developed a UML profile that supports the atomic and complex events of EPC. UML 2.0 supports two kind of diagrams i.e. structural diagrams and behavioral diagrams. Examples of structural diagrams are class diagrams and object diagrams. Few examples of behavioral diagrams are activity diagram, state-machine diagram and use-case diagram.

In UMLPACE the concepts of UML activity diagram are extended. The reason to extend activity diagram are as follows: 1) Activity diagram is a behavioral diagram which can be used to represent the behavior of the system. It is the commonly used UML diagram in business process modeling. 2) Activity diagram has semantics similar to the Petri net for the graphical description of the flow oriented processes like EPC [33]. Moreover, EPC and activity diagram share the same semantics. Both languages are used to represent the concurrent flow of the business processes.

#### A. EXTENDED EPC META-MODEL

In this section, meta-model of EPC [10] is extended. Six complex events are introduced in this meta-model (represented as blue color). UMLPACE is developed by using the stereotypes from EPC meta-model [10] and activity diagram

meta-classes. The extended EPC meta-model is shown in **Figure 3**. On the basis of this meta-model, UMLPACE is established. This meta-model of EPC is divided into three categories. 1) Events (atomic and complex) and function 2) Logical operator 3) Additional process objects.

#### B. UMLPACE DESCRIPTION

The UMLPACE is developed in Papyrus modeling editor tool. It is an eclipse plug-in written in Java. **Figure 4** provides the complete view of the UMLPACE. From Figure 3, all the constructs of EPC languages are selected and its UML profile is developed by extending it with semantically equivalent constructs of UML activity diagram [10]. In **Figure 4**, meta-class and stereotypes are used. Stereotypes represents the constructs of EPC language whereas meta-class represent the constructs of UML activity diagram, which is extended form of EPC. It is categorized into three sub-profiles to make it more organized and easy to understand. The division is followed by the meta-model division in **Figure 3**.

- Sub profile for events and function
- Sub profile for logical operators
- Sub profile for additional process objects

##### 1) SUB PROFILE FOR EVENTS AND FUNCTION

The first sub profile contains following EPC elements; start event, end event, atomic event, complex event and function as stereotypes [10]. These can be called basic elements which are minimum requirement to model an EPC. The descriptions of six complex event patterns which are generalized with complex events are also described along with atomic event. This sub profile is shown in Figure 4.

##### «Start\_Event»

It can be viewed in **Figure 3** that EPC is composed of events and function/ functions. Event can be at three positions; start, intermediate or end. In addition, there can be one or more than one start events in EPC. In order to model the start event, we have extended Start\_Event stereotype with InitialNode

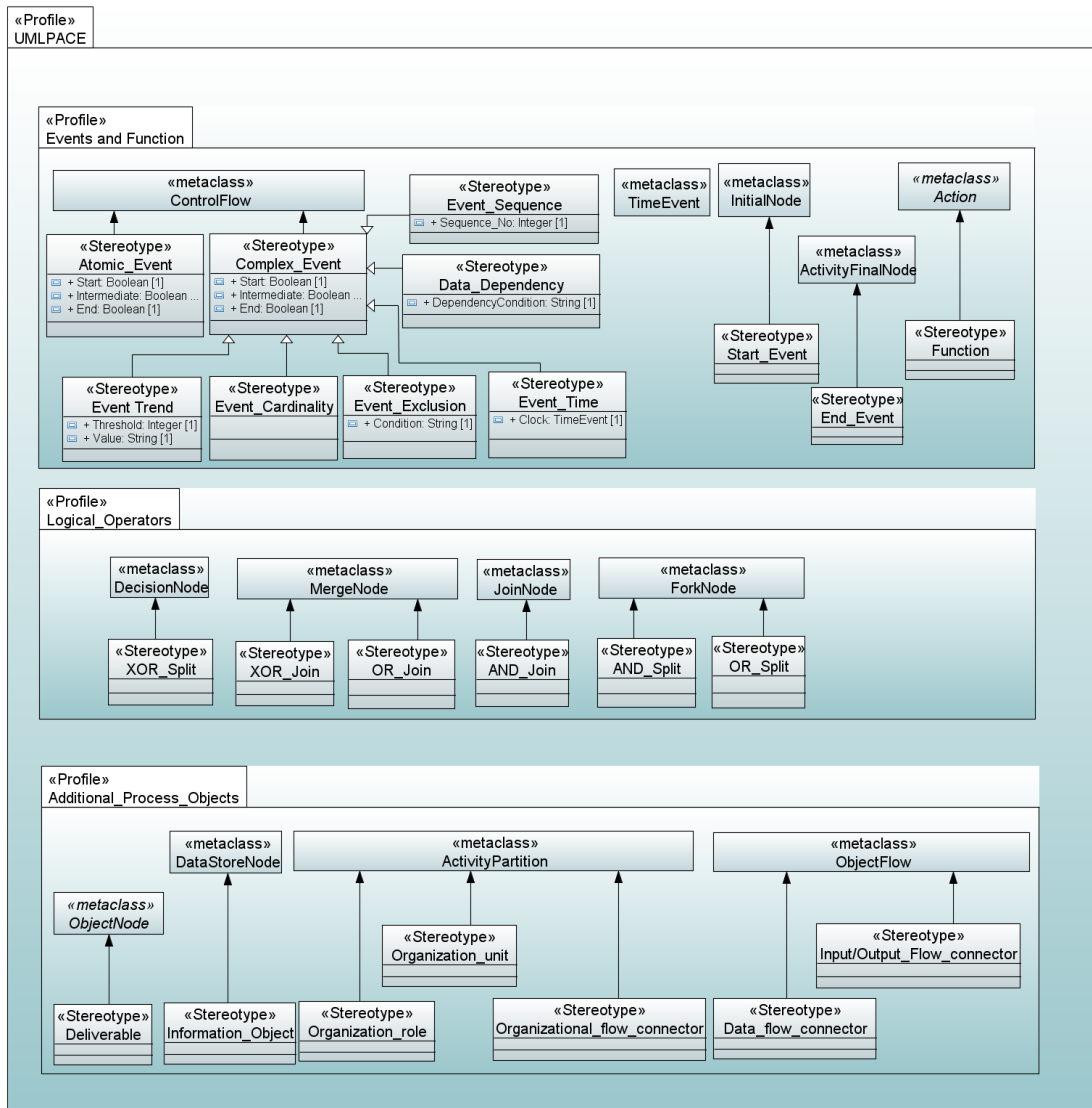


FIGURE 4. UML Profile for Atomic and Complex Events in EPC.

meta-class. By using this stereotype, one or more than one Start\_Event can be mapped to the activity diagram while modeling the business requirements through UMLPACE.

«End\_Event»

According to the meta-model of EPC in Figure 3, there can be one or more than one end events in EPC. End\_Event stereotype is extended with the ActivityFinalNode meta-class. By using the ActivityFinalNode for activity diagram, one or more than one paths as End\_Event stereotype can be represented.

«Function»

The Function can be a complex one and may be an abstract representation. According to EPC meta-model, there should be at least one function to model the EPC. We have extended Function stereotype with Action meta-class. By using this meta-class, activity diagram can facilitate the modeling of

Function of EPC. This stereotype can be applied on Action of activity diagram.

«Atomic\_Event» & «Complex\_Event»

Event represents anything happening in the real world. It can be atomic event or complex event according to the nature of business requirements. There can be minimum two or maximum more than two events in EPC as per EPC meta-model. We have extended Atomic\_Event and Complex\_Events with ControlFlow meta-class. Now, the atomic and complex events can be mapped to the activity diagram using UMLPACE. We have introduced six complex events; event cardinality, event exclusion, event sequence, event time, event trend and data dependency in UMLPACE. These complex events are generalized with the Complex\_Event stereotype. These six complex events are discussed below:

**«Event\_Cardinality»**

Cardinality represents the number of times an event occurs. Event\_Cardinality stereotype is generalized with Complex\_Event stereotype which is further extended with the ControlFlow meta-class. According to the restriction of UML, stereotype cannot be extended with other stereotypes. The stereotypes can only be generalized with other stereotypes. This is the reason event cardinality is generalized with complex event. It is clear from the above statement that by using ControlFlow, Event\_Cardinality stereotype is extended. It does not contain any tagged value. The guard condition on the ControlFlow represents the number of times an event occurs. This stereotype facilitates the full modeling capabilities of Event\_Cardinality in EPC.

**«Event\_Exclusion»**

Event exclusion represents the condition which is excluded for some event to be executed. It is also generalized with complex event which is further extended with ControlFlow meta-class. This stereotype is having a tagged value named Condition. The type of Condition is String. It means that string represent the condition for the exclusion. This stereotype and tagged value enable event exclusion in EPC.

**«Event\_Sequence»**

Event sequence represents the execution of events in sequential order. It was partially achieved by EPC earlier. In UMLPACE, event sequence stereotype make it possible to fully model event sequence. It is extended with ControlFlow meta-class with intermediately Complex\_Event stereotype between these two. This stereotype consists of one tagged value named Sequence\_No where event which is assigned Sequence\_No = 1 is executed first.

**«Event\_Time»**

Time is an important factor while considering many real time applications. This Event time stereotype is generalized with complex event containing one tagged value as clock. TimeEvent is separately placed in this sub-profile as a meta-class. It is used in depicting the type of clock. By using TimeEvent, a clock can represent the before or after conditions related to time.

**«Event\_Trend»**

The trend of two or more than two event needs to be observed in increasing or decreasing order when critical analysis of real time application such as marketing is performed. This stereotype is also extended with ControlFlow as ControlFlow represent the atomic and complex events. This stereotype has two tagged values. One is Threshold which can be any arbitrary number for observing the increasing or decreasing trend pattern. Second tagged value is Function which is of string type. A Function keeps the record of two or more events and increment their values.

**«Data\_Dependency»**

Data dependency is used to check the validity of the accounts by accessing/matching the information from existing database. Here the Data\_Dependency stereotype is extended with the ControlFlow meta-class. Data\_Dependency

stereotype has one tagged value which is DependencyCondition. This should satisfy the condition holding for the dependency. Type of tagged value is String.

**2) SUB PROFILE FOR LOGICAL OPERATORS**

The sub profile for logical operators consists of AND, OR and XOR which are further divided into two categories; split and join. Split performs the division of the outgoing flows and join performs the collection of incoming flows. These logical operators are used to facilitate both atomic and complex events. The Figure 4 illustrates the concepts being used as stereotypes and meta-classes for this sub profile.

**«AND\_Split»**

In EPC, when two parallel executions need to be performed rather than single path to be followed, AND\_Split stereotype is used. This behavior is demonstrated by using AND\_Split in EPC which is divided into parallel paths satisfying one incoming and two or more outgoing arcs. Its equal semantic is ForkNode in activity diagram. Therefore, by using UMLPACE, the AND\_Split stereotype can be applied on the ForkNode meta-class.

**«AND\_Join»**

Opposite to AND\_Split, AND\_Join in EPC has two or more incoming and one outgoing arc representing the parallel execution of the incoming arcs. This stereotype is extended with JoinNode in activity diagram. JoinNode can be modeled using activity diagram and AND\_Join stereotype can be applied using UMLPACE.

**«OR\_Split»**

The need of using OR\_Split arises when user is intended to perform at least one or more than one function. This behavior is captured by OR\_Split in EPC. It has one incoming and two or more outgoing arcs executing minimum one outgoing flow. The OR\_Split stereotype is extended with ForkNode meta-class. In Figure 4, it is illustrated that both AND\_Split and OR\_Split are extended with the ForkNode. The difference among them is the use of guard condition in OR\_Split which controls the outgoing tokens.

**«OR\_Join»**

Unlike OR\_Split, OR\_Join has one or more incoming arcs and one outgoing arc. This stereotype is extended with MergeNode meta-class. The OR\_Join stereotype can be applied on the MergeNode in activity diagram to facilitate OR\_Split.

**«XOR\_Split»**

When exactly one path is selected among multiple paths, the XOR\_Split is used to model this behavior. In EPC, XOR\_Split has one incoming and two or more outgoing arcs where only one outgoing can be executed. XOR\_Split stereotype is extended with DecisionNode meta-class having guard condition on ControlFlow. This guard condition as true/false indicate the decision making process.

**«XOR\_Join»**

In EPC meta-model, XOR\_Join has two or more incoming and one outgoing arc. It is extended with merge node of activity diagram. The purpose to use this stereotype is to

select exactly one path among multiple paths. This ultimately makes a decision using guard condition.

### 3) SUB PROFILE FOR ADDITIONAL PROCESS OBJECTS

Like logical operators, additional process object may or may not part of an EPC. A complete EPC language cannot be presented without additional objects. However, additional process objects can be used to represent the additional information regarding the functions of EPC. In this sub-profile, four additional process objects and three connectors used to link these additional process objects are presented as shown in **Figure 4**. The description of these elements is as follows:

#### « Deliverable »

In presented EPC meta-model (**Figure 3**), the modeling of deliverable is important when a function produce some product or service to the customer. The stereotype Deliverable is extended with the ObjectNode using UMLPACE. Now, the ObjectNode which is present in the activity diagram can illustrate the concept of deliverable in EPC.

#### « Information\_Object »

Information\_Object when associated with the function is used to represent the significant information regarding that function. This stereotype is extended with the DataStoreNode meta-class. Both are semantically equal.

#### « Organizational\_Unit/Organizational\_Role »

In the meta-model of EPC, Organizational\_Unit / Organizational\_Role is used to represent the responsibility of the organization role/unit towards a function. These both stereotypes can be applied to the Activity Partition which represent the swim-lanes used to depict one particular Unit/ Role accountable for a function.

#### « Input/Output\_Flow\_Connector »

In EPC meta-model (**Figure 3**), there is only one input/output flow connector linked with function. This connector is used to link deliverable with function. The same is obtained by the concepts of activity diagram. The stereotype Input\_Output\_Flow\_Connector is extended with the ObjectFlow meta-class. Now, the ObjectFlow is linked with the ObjectNode making the modeling of Input/Output\_Flow\_Connector easy to use.

#### « Data\_Flow\_Connector »

Data\_Flow\_Connector in EPC is used to link Information\_Object with function. ObjectFlow is extended for Input/Output\_Flow\_Connector as well as for Data\_Flow\_Connector.

#### « Organization\_Flow\_Connector »

Organization\_Unit / Organization\_Role is linked with the function with the help of Organization\_Flow\_Connector. This stereotype can be applied to ActivityPartition by using UMLPACE. It is analyzed that all organization related concepts are associated with the activity partition which is used to represent the same concept by swim lanes.

We have proposed eleven stereotypes in the sub-profile for events and function. Six stereotypes are proposed in sub-profile for logical operators. In the last sub-profile for additional process objects, seven stereotypes regarding EPC

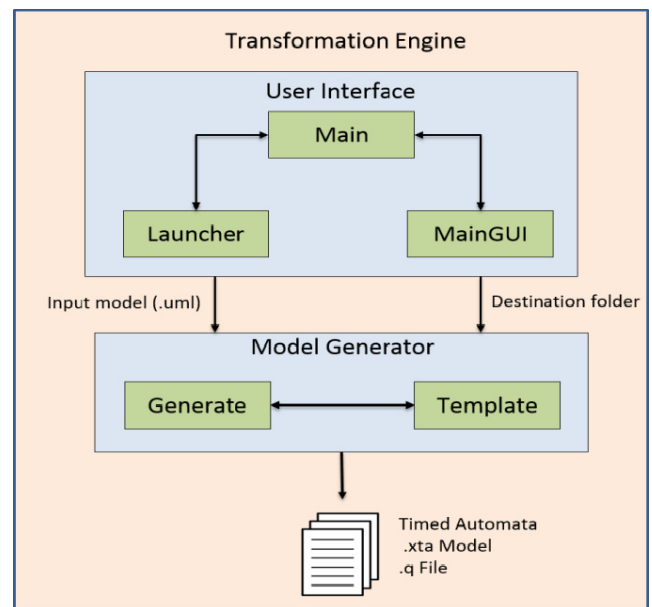
are proposed. The practical application of these stereotypes is further demonstrated in Section V where two case studies (ATM and PSF) are modeled using UMLPACE. The complete UMLPACE profile along with sample case studies can be downloaded at [40].

## IV. TRANSFORMATION ENGINE

As a part of research, a complete transformation engine is developed to automatically transform the source UMLPACE requirements. This section deals with the implementation details of transformation engine. Firstly, the architecture of transformation engine is discussed. Secondly, the transformation rules are presented.

### A. ARCHITECTURE OF TRANSFORMATION ENGINE

The architecture of transformation engine is shown in **Figure 5**. There are two main components of transformation engine i.e. User Interface (UI) and Code generator. The implementation of transformation engine is carried out in JAVA language and Acceleo tool by utilizing model-to-text (M2T) transformation approach.



**FIGURE 5.** Architecture of transformation engine.

*User Interface:* Main, MainGUI and Launcher are the classes used to develop the user interface of transformation engine. Main is used for execution and its functionality resides in MainGUI and Launcher. This interface shown in **Figure 6** consists of various components. Among them, the Input Model and Destination Folder can be obtained using Browse button. The Generate button can generate the timed automata .xta file from UMLPACE models. Status is used to represent whether file is successfully generated or not. By using Reset button, the input model and destination folder can be replaced. At the end, two tools; Papyrus and



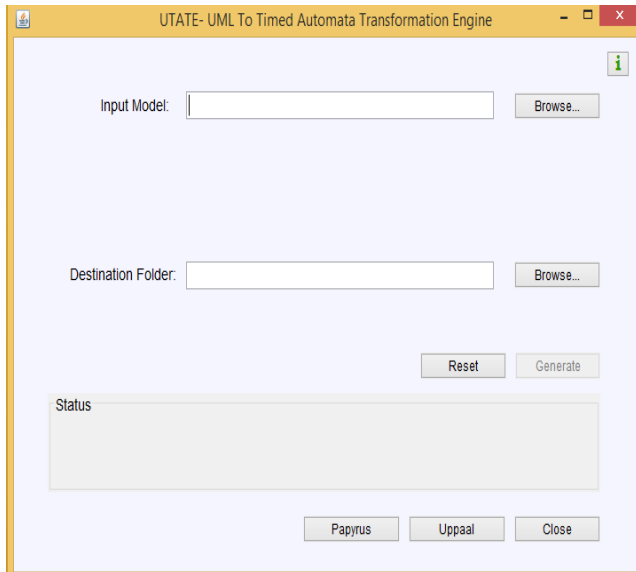


FIGURE 6. User Interface of Transformation Engine.

UPPAAL can be opened directly by using this interface. Close button can be pressed to close the interface view from the screen.

*Model Generator:* The Input Model (.uml) and destination Folder received from the user interface of UTATE (UML to Timed Automata Transformation Engine) are accessed through model generator. It is implemented in java and Acceleo tool to transform the concepts of UMLPACE models to timed automata. Model generator consists of two components; Generate and Template. As a result of execution, two files (i.e. .xta and .q) are generated through generate.mtl file. The file generated with xta extension is basically a timed automata model that is compliant with UPPAAL tool. On the other hand, the file with q extension contains the verification properties.

UTATE is designed to maintain the significant transformation properties as follows:

#### 1) SCALABLE

UTATE is highly scalable as it is based on modular architecture (Figure 5). Therefore, it can easily accommodate the future enhancement as per requirements. For example, it is straightforward to add new transformation rules in order to incorporate more UML activity constructs or even for other formalisms like Petri nets by only modifying Model Generator component of UTATE. Similarly, user interface of UTATE can be upgraded by only modifying User Interface component. In addition, UTATE is built on widely acceptable technologies which are highly supportive for scalability. For example, the core business logic (transformation rules) of UTATE is implemented in Java language which is widely used in several open source model driven projects. To summarize, UTATE is built on a style that is eager to support scalability.

#### 2) DETERMINISTIC

UTATE is deterministic which is intended to produce determined output against provided input. For example, we have written Java code in such a manner that against UML activity diagram, text file of timed automata is generated without any syntactic or semantic error.

#### 3) EASY TO USE

Simple user interface is provided to configure desired options in UTATE as shown in Figure 6. All the important configurations like browsing of input model and destination folder, availability of status bar and generate / close buttons etc. are provided in UTATE with simplicity.

#### 4) CONTROLLED

In UTATE, transformation is performed in a controlled manner e.g. explicit instructions are provided about transformation process etc. Furthermore, transformation mechanism is well documented, guided through user manual and applicability is shown with the help of multiple case studies.

#### 5) OPEN SOURCE

UTATE is publically available [40], so that, both industry and academia can utilize it for further improvements / enhancements as per requirements.

#### 6) CORRECTNESS

All the syntactic and semantic correctness is handled in the UTATE effectively. Syntactic correctness means that the generated target models represent well-formed instances of target language i.e. timed automata.





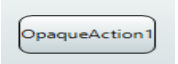



### B. TRANSFORMATION RULES

This section provides the mapping rules which are used to map UMLPACE into timed automata. These rules produce a target model for verification through transformation engine. For the modeling of UMLPACE models, Papyrus modeling editor of Eclipse is used. For the verification, UPPAAL tool is used. UPPAAL is used for the simulation and verification of the models produced using timed automata. Model of UPPAAL is composed of states and transitions. States carry clock and transition carry synchronization, guard and update [35].

#### 1) TRANSFORMATION RULES FOR EVENTS AND FUNCTION

According to the meta-model division in Figure 3, transformation rules are developed for events and function, logical operators and complex events separately. Transformation rule for events and function is provided in Table 1. Three parameters are used in this table: 1) *Activity diagram* depicts the graphical notation used for modeling initial node, final node, control flow and action. 2) *Timed Automata* represents the graphical notation used to model the equivalent notations of activity diagram. 3) *Description* provides the mapping detail of UML activity diagram to timed automata.

**TABLE 1.** Transformation rules for Events and Functions.

Activity Diagram	Timed Automata	Description
InitialNode 	Initial location 	InitialNode in activity diagram is mapped to the initial location in timed automata. Both are semantically equal and used to represent the initial state of the system.
ControlFlow 	Edge 	ControlFlow in activity diagram is mapped to the Edge in timed automata. Both notations represents the transition of one state to another state.
Action 	Location 	Action in activity diagram is transformed to the location in timed automata where it represents some activity to be performed
ActivityFinalNode 	Committedlocation 	ActivityFinalNode is mapped with committed location in Timed automata. Both notations are used to represents the terminating states.

The transformation rules for complex event are associated with the ControlFlow in activity diagram as UMLPACE complex events stereotypes are based on ControlFlow. The transformation rules to transform UMLPACE complex events stereotypes into equivalent timed automata model are provided in the **Table 2**.

## 2) TRANSFORMATION RULES FOR LOGICAL OPERATORS

Transformation rules for logical operators which are used to map the activity diagram into timed automata are provided graphically and semantically in **Table 3**.

Transformation rules for additional process objects are not implemented in the engine because such details are not covered in this study to limit the broader scope. This major focus of this study is the complex / atomic events, functions and logical operators. The complete source code of transformation engine along with user manual can be found at [40].

## V. VALIDATION

The applicability of our approach is presented in this section with the help of ATM and patient flow system case studies in subsequent sections.

### A. ATM CASE STUDY

This case study consists of three sections. Firstly, introduction of requirements are discussed. Secondly, in next section the details of modeling in UMLPACE is provided. Finally, transformation and verification of ATM case study is performed.

**Requirements:** In ATM case study, two scenarios are commonly observed. One is of ATM fraud detection and another of simple transaction process. The biggest challenge involved in banking field comprise of fraud detection and cross selling. The faulty operations regarding bank are often led by the ATM machines or through web services. Our area of focus is ATM fraud detection where the misuse of ATM machines is conducted by the people to gain access to the other people's account for the sake of unauthorized withdraws of the money.

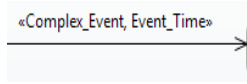
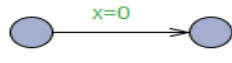
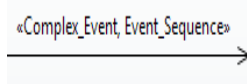
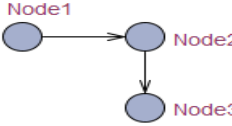
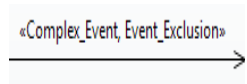

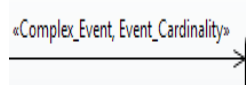
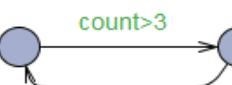
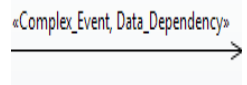
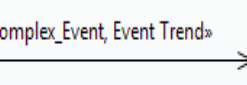
The complex event patterns involved in the modeling of this process enrich the flow and can detect the threat which indicates a fraud. Few examples of such fraud detection are the identification when a customer withdraws a very large amount or changing the password. The requirements we have catered in our process are of two types; one leads to the fraud detection and another leads to the simple transaction process.

1. Card checking should be associated with data dependency where the data in the form of card information (account number) is matched with the existing data to ensure the validity of the card.
2. Event Cardinality is associated with the incorrect PIN. This complex event indicates that if the user enters incorrect PIN more than three times, an event counter measure to fraud detection should be executed.
3. Counter measure to fraud detection is further checked that if incorrect PIN is entered three times and time duration is less than 15 minutes then user's card should be captured.
4. ATM is used for different purpose by different users. Some withdraw cash, some requires balance inquiry and some asks for the statement. Two most frequently used transactions are cash required and balance inquiry required. The trend of these two events is observed by theEvent Trend complex event patterns. Event trend take place with two or more than two events where the trend of the event is observed in increasing or decreasing order. Threshold value is provided to observe the increasing or decreasing trend.
5. Event Exclusion pattern in ATM means that user cannot withdraw more than 25,000 US dollars in one day.

**Modeling:** In **Figure 7**, ATM example is modeled in activity diagram through UMLPACE.

It is mandatory to use UMLPACE stereotypes in order to model the above five requirements. «Start\_Event» stereotype is applied on the initial node User\_Enters\_ATM,

TABLE 2. Transformation rules for complex events.

Activity Diagram (UMLPACE Complex Events Stereotypes)	Timed Automata	Description
	<p>Clock</p> 	In timed automata, value is assigned to the clock representing the time. It is mapped to the Event_Time stereotype.
	<p>Sequential Flow</p> 	Function and events are altered in timed automata. Sequence can be defined which represents which event to come first.
	<p>SELECT on edge</p> 	Event exclusion is mapped to the select condition on the edge of timed automata. It means that event is executed in specified condition. Out of bound condition is not entertained.
	<p>Guard on edge</p> 	In timed automata, guard condition named “count” represents the event cardinality. Count can be any number greater than one.
	<p>Variable declaration</p> <pre>int xyz;</pre>	Variable is declared in timed automata for data dependency which is checked against the value provided in existing data to proceed further.
	<p>Function and threshold</p> <pre>Void UPDATE() { Value++; } int threshold=10;</pre>	In timed automata, the mapping of Event_Trend stereotype holds the function containing increment and threshold to entertain this pattern.

subsequently a Machine\_displays\_message is visible on ATM machine is represented by the «function» stereotype of the UMLPACE. The decision is taken with «XOR\_Split» stereotype which is represented by decision node in activity diagram. Guard condition is applied If ATM\_is\_available == false, then user moves to another machine. Thereafter, activity final node will be executed as «End\_Event». If the ATM\_is\_available == true, then the «function» Use\_inserts\_a\_card is performed by a user. After inserting an ATM card, decision is taken represented by «XOR\_Split» connector where machine decides whether it is a Valid Card which is shown as Valid\_ATM\_card == true or Invalid ATM card is shown as Invalid\_ATM\_card == false. To check the validity of card before decision node, UMLPACE stereotype «Data\_Dependency» is applied. It is used to facilitate the requirement.

If the card is invalid, user can again insert a card else User\_insert\_PIN. Again, when the PIN is inserted, decision is

taken by XOR split connector to see whether user has inserted the PIN correctly or incorrectly. PIN\_correctly\_entered == true represents that pin is correctly entered while PIN\_correctly\_entered == false represents that PIN is incorrectly entered. Here, two stereotypes of «Complex\_Event, Event\_Cardinality» are applied. Stereotype of complex event is used to differentiate from atomic event and event cardinality represents the type of complex event applied. If PIN is incorrectly entered then counter measure to fraud detection as a «function» is executed. Further, the user’s card is captured which shows a fraud case. Exit2 represented by «End\_event» is one of the end events here. It is triggered when two stereotypes «Event\_Time, Event\_Cardinality» are fulfilled where it ensures that incorrect PIN is entered more than three times and within 15 minutes. Along with these two stereotypes, «Complex\_event» is also applied. These three stereotypes together model the requirement 2 and requirement 3.

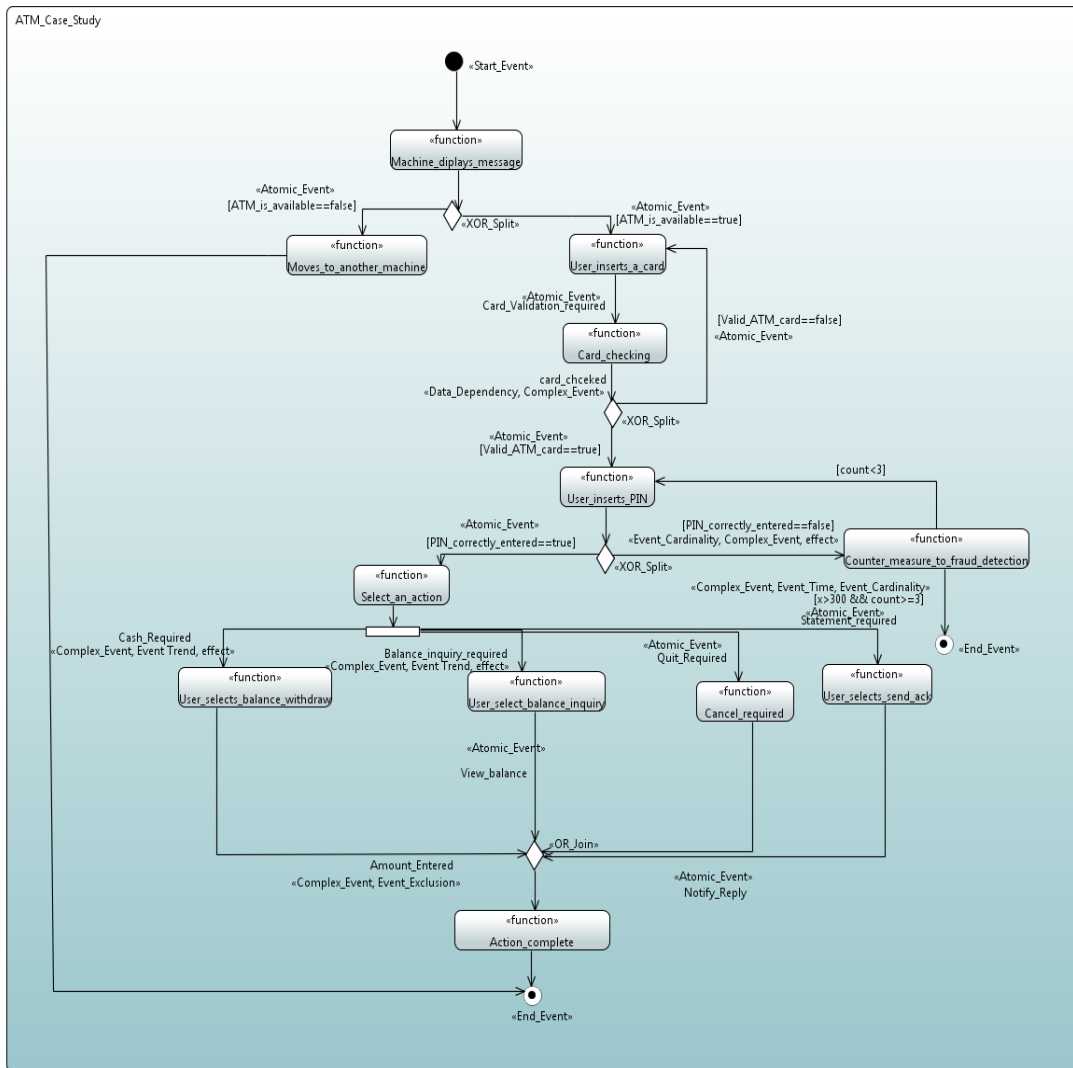


FIGURE 7. Modeling of ATM Requirements through UMLPACE.

If the PIN is entered correctly, then user can see multiple options on the screen and should select one particular action at one time. After Select\_an\_action, «OR\_Split» is placed which represents that user can select minimum one action and can also perform more than one action. Four events; Cash\_required, Balance\_Inquiry Required, Statement\_Required and Exit\_Required are presented. These four events have corresponding functions. When User\_selects\_balance\_Inquiry, current balance will be viewed. If the user\_selects\_statemen\_required, it triggers the send acknowledgement which notifies the reply. User\_select\_cancel\_required automatically pass the flow to the action\_complete. User\_selects\_Balance\_withdarwis associated with UMLPACE stereotype «Event\_Exclusion» where user cannot enter more than 25000 US dollars in one day in order to fulfill the requirement 5. To fulfill the requirement 4, «Event\_Trend» is observed for

Cash\_required and Balance\_inquiry\_required. It represents that among these two events, whether user uses cash or balance inquiry required. It is recorded as a pattern against a user. An arbitrary threshold value, ten (10) is used to observe the increasing or decreasing trend. All these four events reach to the «OR\_Join» which enables the action completed. Then, Exit is another final state.

**Transformation and Verification:** UMLPACE transformation engine is utilized to automatically transform the ATM case study model into timed automata target model in order to perform the verification. UPPAAL tool takes timed automata model in xta format. Furthermore, verification properties (e.g. reachability, deadlock etc.) are expressed in Timed Computational Tree Logic (TCTL) [32]. UMLPACE transformation engine is capable of generating both timed automata model and TCTL properties from the source model as shown in Figure 8. Input Model of Activity Diagram for ATM

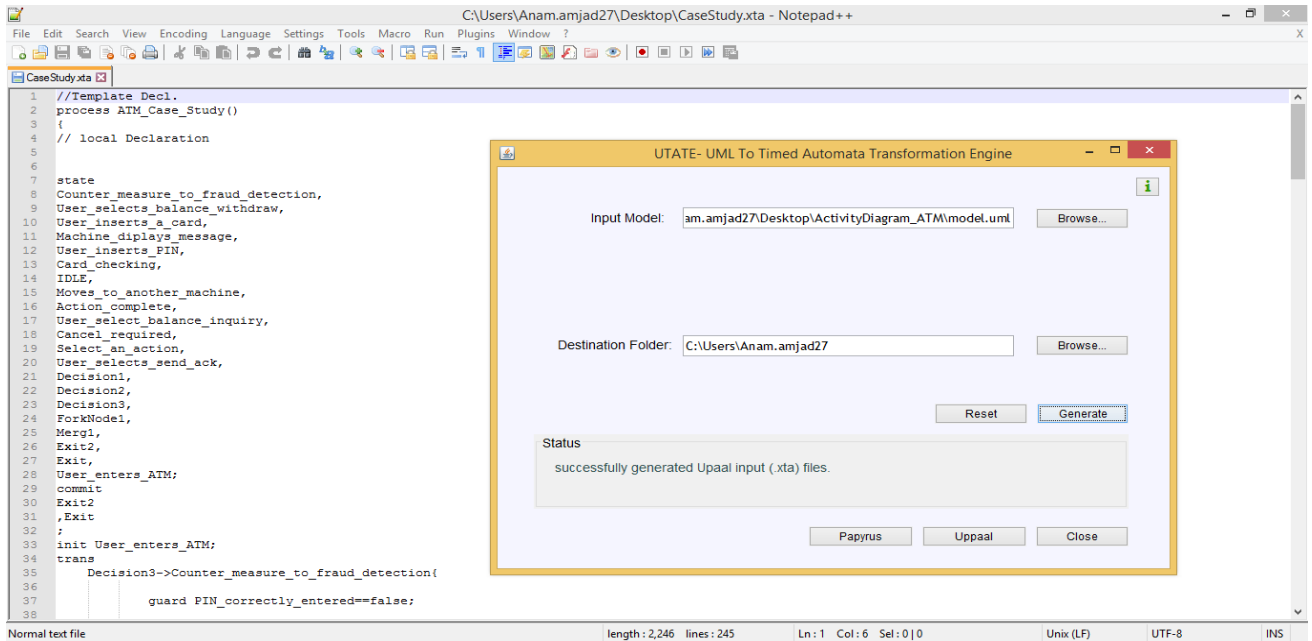


FIGURE 8. Transformation of ATM Model in Timed Automata.

TABLE 3. Transformation rules for logical operator.

Activity Diagram	Timed Automata	Description
		ForkNode in activity diagram and sequential flow in timed automata are semantically equal. An extra place holder needs to be added in timed automata shown as AND_Split.
		JoinNode in activity diagram is mapped to the sequential flow in timed automata. It is analyzed that ForkNode and JoinNode both are mapped to the sequential flow in order to entertain the parallel execution.
		ForkNode with guard condition represent the OR_Split in activity diagram It is mapped to the some locations, edges and an additional edge in timed automata.
		MergeNode with guard condition represents the OR_Join in activity diagram whereas guard condition in timed automata represents the same concept.
		XOR_Split is supported with the help of DecisionNode in activity diagram and guard condition in timed automata.
		XOR_Join is supported with the help of MergeNode in activity diagram and guard condition in timed automata.

is provided. The extension of this model is .uml. It will only select models with .uml extension. Destination Folder is browsed as desktop. After browsing Input Model and Destination Folder, Generate button is enabled. By clicking on the

Generate button, the text file .xta is generated. Status shows that file is successfully generated. This Case Study .xta file is opened in Notepad++ displaying the text obtained from the activity model. Case Study .xta file will automatically produce a model in timed automata.

When the model is loaded in the UPPAAL, its syntax is checked first. This validation by using simulation ensures the correctness and completeness of the model uploaded. After simulation, verification is performed. We have only considered two verification properties.

- **Deadlock:** Deadlock stops any state from going to the next state.
- **Reachability:** Soundness is when an initial state is reachable to the final state through some firing sequence. Reachability defines that specific condition holds in some state from the initial state. The reachability to final state can be considered as soundness in EPC.

It can be seen from the **Figure 9** that the two properties are successfully verified. First property A [] not deadlock represents that there is no deadlock in the system. Second property E<> Process. Exit2 represents that the final state ‘Exit 2’ is reachable from the initial state by using any path. Exit 2 refers to the fraud detection based end event as shown in **Figure 9**.

**B. Patient Flow System Case Study**

To keep the patient flow system efficient is the forefront duty of every hospital. Patient flow systems are made to ensure the care and quality provided to the customer with best of their services. This system is an example of real-time

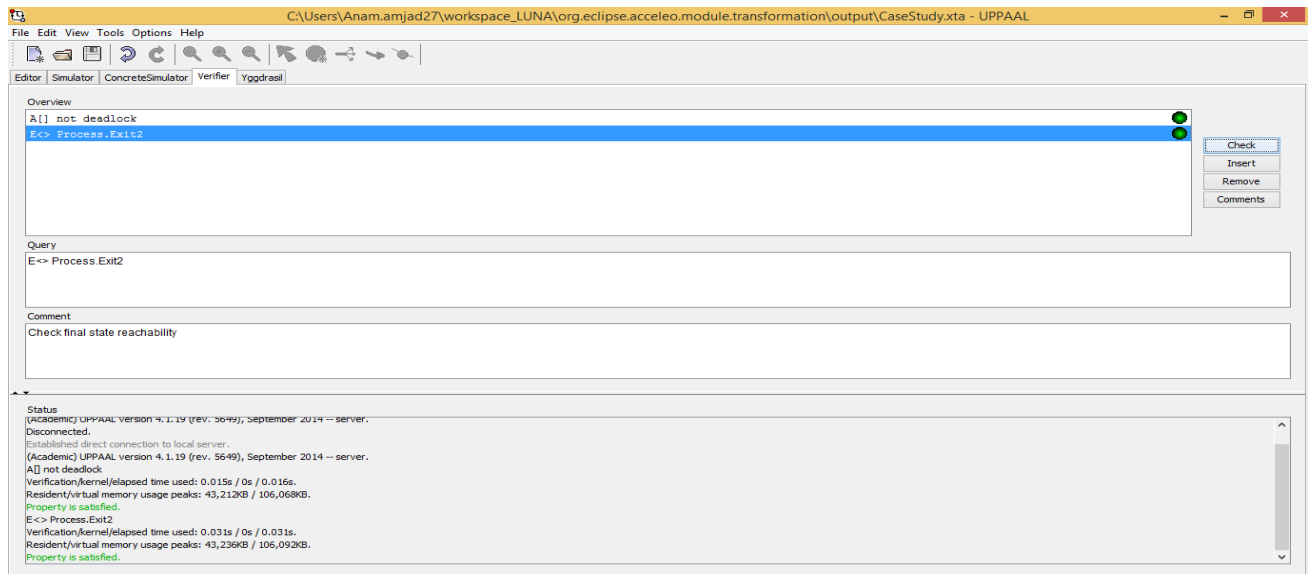


FIGURE 9. Verification of ATM Case Study in UPPAAL.

system. Also, it is a complex process which consists of simple as well as some complex requirements. The requirements need to be modeled and verified. Requirement, modeling, transformation and verification is carried out in this section.

**Requirement:** Following are the key requirements of this case study.

1. Event Time: The laboratory tests should be available to the patient within specified time.
2. Event Sequence: Sequence of following occurring events should be followed. Sequence 1 means that this event should execute prior to sequence 2. And sequence 2 means that it should be executed prior to sequence 3. sequence 3 is executed at the last.
  - Sequence 1: Medical history needed.
  - Sequence 2: Physical examination required.
  - Sequence3: Laboratory tests required.
3. Data Dependency: The patient should visit the doctor according to the visiting number allotted.

**Modeling:** In Figure 10, Patient's visits the starting event represented as stereotype «Start\_Event», then an activity is performed which is representing as Checking doctor's availability. This activity is represented by stereotype «function» in UMLPACE. A decision is taken with the help of stereotype «XOR\_Split» connector showing whether the Doctor is available or Doctor is unavailable. If the doctor is unavailable the guard condition Doctor\_is\_avalible == false executes then some temporary arrangements are made. The temporary arrangement means arranging a duty doctor to examine the patient. In second case, if the doctor is available then Doctor\_is\_available == true is executed and check-up is performed with Doctors\_examination. Patient\_turn is linked

with the «Complex\_Event, Data\_Dependency» to satisfy the **requirement 3**.

This check-up is further split with a AND operator indicating the parallel execution of three events; Medical history needed, physical examination and laboratory tests required. These should be performed in sequence provided and stereotype «AND\_Join» collects the results of the execution. The «Complex\_Event, Event\_Sequence» stereotype on these three events depicts that sequence is followed and **requirement 2** is satisfied. Further, on step three, «Event\_Time» is applied to fulfill the **requirement 1**. At first, control is given to the medical history needed, then physical examination is provided and after these two laboratory tests are made. After it, the disease is diagnosed and prescription\_provided by the doctor. The stereotype «XOR\_Join» can have the output from one path at one time. After it, stereotype «Payment\_made» is applied on the action of the activity diagram. Activity final node named Patient\_exit indicates that the checkup is being completed and it must be followed by the payment by the patient. Turn to the next patient is the stereotype «End\_Event» in this example.

**Transformation and Verification:** UMLPACE transformation engine is utilized to automatically transform PFS model into timed automata target model. We are omitting the details of transformation and verification process for PSF case study because such details are already described in ATM case study. Figure 11 and Figure 12 represents the transformation and verification of PFS case study. However, interested readers can find complete models of both ATM and PFS case studies along with transformation engine here [40]. The transformed model is verified with deadlock and reachability to final state i.e. Turn\_of\_next\_patient as shown in Figure 12.

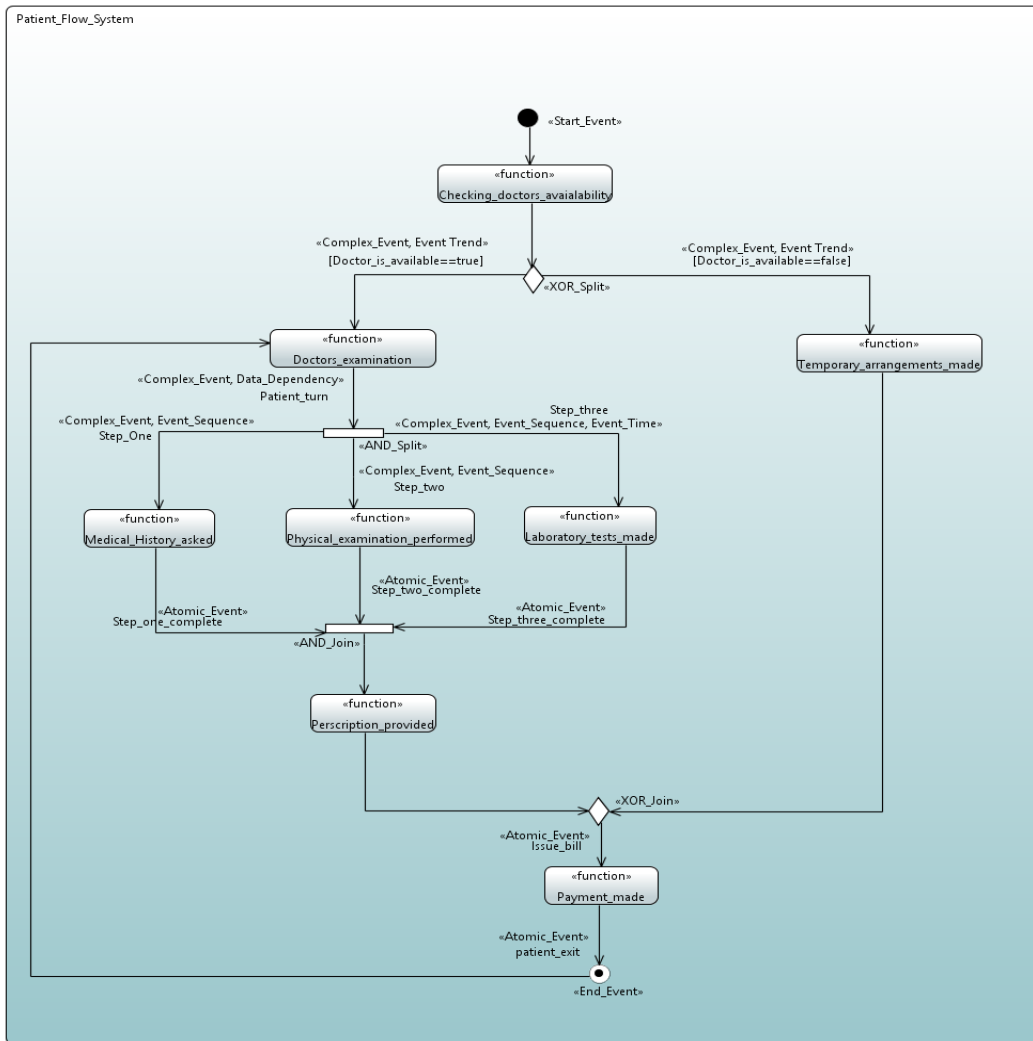


FIGURE 10. Modeling of PFS Requirements through UMLPACE.

VI. COMPARATIVE ANALYSIS AND DISCUSSION

In this section, comparative analysis with the existing literature is performed to highlight the importance of the work done. We have found four studies where complex event patterns are considered in the context of EPC. We have used five evaluation parameters to perform comparative analysis of studies with UMLPACE as follows: 1) Reference No is used to refer the selected research study. 2) No of complex event patterns represents the number of complex events patterns which are targeted in the particular study. The selected research studies are checked against 3) Modeling, 4) Transformation and 5) Verification of the complex events in EPC. Particularly, the objective is to evaluate the modeling, transformation and verification support for complex events in EPC. The parameters 3, 4 and 5 are evaluated as Yes, No or Partial. 6) Tool support is represented in last column.

In Table 4, total four studies in the given domain are selected. Different research papers have used different number of complex event patterns. For example, study [7] has selected eight complex event patterns, [8] and [20] both have selected two complex event patterns and [31] has selected one complex event pattern. In the column targeting number of complex event pattern, our proposed approach has used 6 complex event patterns. Although [7] has used more number of patterns than our proposed approach, this study has neither performed transformation nor verification. Modeling is also performed with the help of annotations. The business process modeling language is same in all the papers i.e. EPC. It can be analyzed from the Table 4 that two studies ([7] and [20]) have performed modeling of complex events in EPC. On the other hand, studies [8], [31] do not purely deal with the modeling of complex events and only conceptual mapping of complex events is provided by utilizing the existing modeling

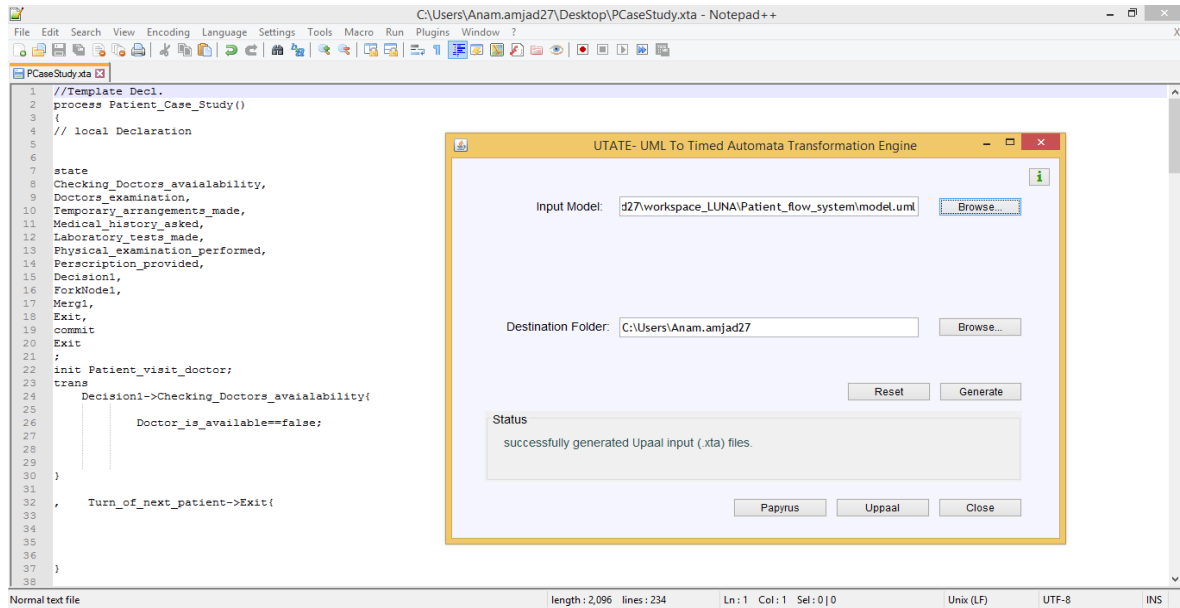


FIGURE 11. Transformation of PFS case study.

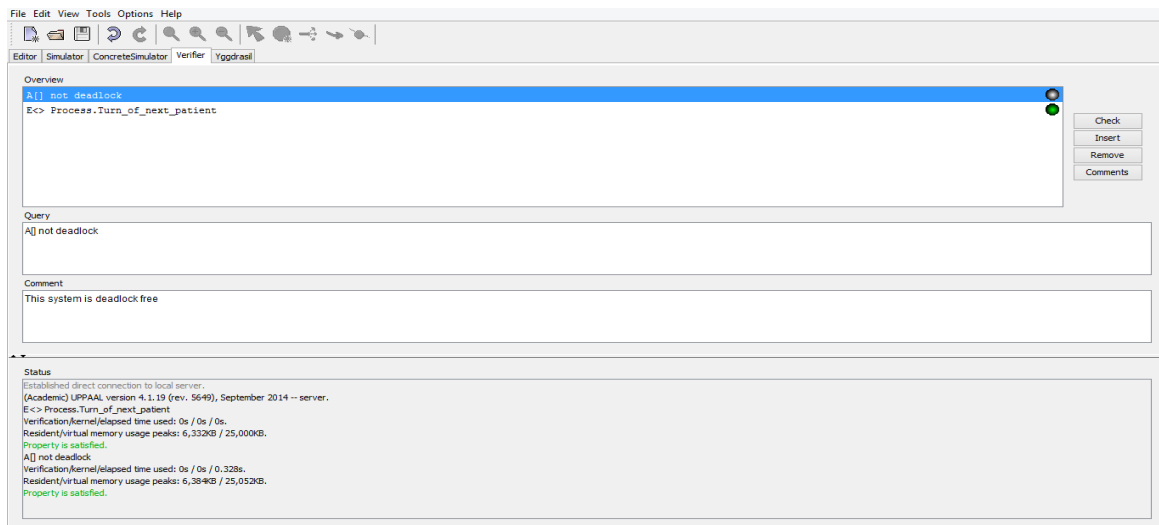


FIGURE 12. Verification of PFS case study.

concept of EPC. It can also be analyzed from the **Table 4** that there does not exist any study that provides complete tool support in the form of transformation engine. In study [8], verification of complex event is performed, however, transformation support is partial because only mapping of EPC to verification language is performed without any proper tool support. Finally, it can be concluded from the **Table 4** that UMLPACE is the only approach that fully support modeling, transformation and verification of complex events patterns in EPC. Furthermore, it also provides complete tool support as a UMLPACE transformation engine.

## A. DISCUSSION

In this article, UMLPACE is introduced to include the modeling, transformation and verification support for complex events in EPC. There are few attempts in this area,

for example, Birgit Korherr and Beate List presented UML profile for EPC but it only covers atomic events [41]. Therefore, this study is the first attempt to propose the modeling of six complex events (Event cardinality, event exclusion, event sequence, event time, event trend and data dependency) in EPC by using UMLPACE. Besides complex events, UMLPACE includes all other elements (atomic events, logical operators and additional process objects) of EPC in this profile to give a complete view of the EPC. Therefore, by using UMLPACE, equal benefits to both a business user and software developer are gained. Some advantages of UMLPACE are as follows.

- **Real time:** UMLPACE enables the modeling and verification of complex real time business processes. It is **not** possible to manage real times business requirements in EPC before UMLPACE.



TABLE 4. Comparative analysis with state-of-the-Art.

Reference #	No. of complex event patterns	Modeling	Verification	Transformation	Tool Support
[7]	8	Yes	No	No	No
[8]	2	No	Yes	Partial	No
[20]	2	Yes	No	No	No
[31]	1	No	Yes	Yes	No
UMLPACE	6	Yes	Yes	Yes	Yes

- **Behavioral modeling:** For model to be useful, it should be correct and meaningful. UMLPACE supports the behavioral modeling as it is based on the semantics of activity diagram and EPC language. Consequently, it is verified to obtain correct and meaningful models [42].
- **Formal Verification:** UMLPACE is fully supported by transformation engine to automatically generate timed automata target models from the source models. Consequently, complex and real time business requirements can be formally verified in early automation stages.

Although UMLPACE incorporates six important complex events in EPC for modeling and verification of complex business requirements, it is still required to include more complex events in UMLPACE to manage modern business requirements particularly for real time systems. Furthermore, UMLPACE does not provide the complete support for additional process objects. Despite such minor limitations, UMLPACE provides strong basis to model and verify complex and real time business processes. If we talk about the modeling capability of the UMLPACE with other approaches, so it is two step modeling. In first step, we have to develop a UML activity diagram. This UML activity diagram is created with the help of mapping given in the UMLPACE. In the second step, UMLPACE need to be applied on the activity diagram. This complexity of the UMLPACE is associated with two-step process. The modeling is considered incomplete without performing these two steps. By using UTATE, the model we have developed using UMLPACE can be verified at design phase. Early verification at design phase is used to improve the software development life cycle by identifying errors or bugs.

It can be argued that the parallel execution of fork node in activity diagram can be modeled using the timed automata CCS parallel composition operator which allows interleaving of actions as well as hand-shake synchronization. In this regard, it is important to mention here that the fork is a control node in UML activity diagram which has one incoming edge and multiple outgoing edges. It is used to split incoming flow into multiple concurrent flows within a single process. On the other hand, there are two major elements (i.e. sequential flow and CCS operator) in timed automata to achieve the concurrent flows. Essentially, CCS operator is used to achieve

concurrency where more than one processes combined into single process [43]. In Contrast, sequential flow is used to achieve concurrency in timed automata where modeling of a single process is performed. As we have used a single activity in UML activity diagram which is transformed to a single process in timed automata, the mapping of fork node to sequential flow is carried out in UTATE. Furthermore, the major functionality can be achieved through sequential flow only because it can produce same resultant tokens. Therefore, we have ignored the transformation of fork to CCS operator in UTATE to avoid complexity.

It can also be argued that the business artifacts representation methods have evolved little over the last decades and other modeling languages such as BPMN naturally replaced EPC in different areas. BPMN is considered as one of the most popular and standard language by Object Management Group (OMG). However, the key advantages of EPC over BPMN are as follows:

- 1) **Increased usability:** To model complex event patterns (e.g. event cardinality or event trend), EPC represent a promising means for depicting these patterns in relation to business processes. As these complex processes are generated from business scenarios for which non-technical experts are responsible, therefore, EPC being a simple and easy to understand language is used by both non-technical and domain experts. On the other hand, in order to understand BPMN, domain experts are required.
- 2) **Wide tool support:** There is a wide variety of EPC tools available such as ARIS, Microsoft Visio 41] etc. at industrial and academic level. A complete tool support for the atomic and complex events of EPC is also provided in this article which helps the user to model and verify real-time systems. In addition, we have developed Platform Independent Model (PIM) approach to model EPC for complex processes. By using PIM approach, modeling can be performed by utilizing the concepts of any other business process modeling language as well. For example, UMLPACE can easily be extended to incorporate the concepts of Business Process Modeling Notation (BPMN) or Business Process Execution Language (BPEL) depending

upon the domain and user's requirements. Furthermore, UTATE can be upgraded to generate the other required target models like Petri Nets etc. In contrast, BPMN tools are not flexible and usually deal with the platform specific models.

It is important to understand that EPC is still being widely used and cannot be replaced completely with BPMN [41]. In addition, EPC has made an irreplaceable impact in different domains since its development. Such as in industry, EPC plays an important role as SAP models are developed in EPC since its development. By adding six complex patterns (i.e. event cardinality, event sequence, event time, event exclusion, event trend and data dependency) in EPC, it is now becomes highly suitable for real time systems.

## VII. CONCLUSION AND FUTURE WORK

This article reduces the design and verification gap of the complex and real time business processes by utilizing the concepts of Event-driven Process Chain (EPC). For this purpose, UMLPACE (Unified Modeling Language Profile for Atomic and Complex events in EPC) is introduced for the modeling of complex and real time business requirements at Platform Independent Model (PIM) level. Particularly, the concepts of UML activity diagram are utilized in UMLPACE for representing both simple as well as complex patterns in EPC. As a part of research, a complete open source UML to Timed Automata Transformation Engine (UTATE) is developed to automatically transform UMLPACE source models into timed automata target models for the early verification of complex and real time business processes. Finally, the applicability of UMLPACE is demonstrated through two benchmark case studies.

UMLPACE is the first attempt to model and verify complex and real time business requirements in the context of EPC. Moreover, it is based on UML concepts that provide higher level of abstraction. Furthermore, it fully supports formal verification through sophisticated transformation engine. Consequently, it is equally beneficial for business analyst, business user and software developers that lead to simplify the design, verification and implementation of modern business processes. Business users and business analyst can easily model the complex requirements using the proposed framework whereas this research works aims to facilitate the software developers with familiar notations of UML activity diagram for modeling without any additional efforts and time to understand the EPC language. Although UMLPACE incorporates six important complex events in EPC for modeling and verification of complex business requirements, it is still required to include more complex events in UMLPACE to manage modern business requirements particularly for real time systems. Therefore, in future, we intend to enrich the UMLPACE by introducing more behavioral patterns e.g. subscription and consumption patterns. Moreover, we also intend to include spatial patterns for complex scenarios where event location is an important factor.

## REFERENCES

- [1] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Inf. Softw. Technol.*, vol. 50, no. 12, pp. 1281–1294, Nov. 2008.
- [2] H. Mili, G. Tremblay, G. B. Jaoude, É. Lefebvre, L. Elabed, and G. El Boussaidi, "Business process modeling languages: Sorting through the alphabet soup," *J. Comput. Surv.*, vol. 43, no. 1, pp. 1–56, 2010.
- [3] E. Söderström, B. Andersson, P. Johannesson, E. Perjons, and B. Wangler, "Towards a framework for comparing process modelling languages," in *Proc. 14th Int. Conf. Adv. Inf. Syst. Eng. (CAISE)*, Toronto, ON, Canada, 2002, pp. 600–611.
- [4] O. Kath, "Towards executable models: Transforming EDOC behavior models to CORBA and BPEL," in *Proc. 8th IEEE Int. Enterprise Distrib. Object Comput. Conf. (EDOC)*, Monterey, CA, USA, Sep. 2004, pp. 267–274.
- [5] G. Jošt, J. Huber, M. Heričko, and G. Polančič, "An empirical investigation of intuitive understandability of process diagrams," *Comput. Standards Interfaces* vol. 48, pp. 90–111, Nov. 2016.
- [6] H. Kim and S. Oussena, "A case study on modeling of complex event processing in enterprise architecture," in *Proc. ICEIS*, vol. 3, 2012, pp. 173–180.
- [7] J. Krumeich, N. Mehdiyev, D. Werth, and P. Loos, "Towards an extended metamodel of event-driven process chains to model complex event patterns," in *Proc. Int. Conf. Conceptual Modeling*, Stockholm, Sweden, 2015, pp. 119–130.
- [8] A. Amjad, F. Azam, M. W. Anwar, and W. H. Butt, "Verification of event-driven process chain with timed automata and time Petri nets," in *Proc. 9th IEEE GCC Conf. Exhib.*, Manama, Bahrain, May 2017, pp. 1–6.
- [9] M. W. Anwar, M. Rashid, F. Azam, and M. Kashif, "Model-based design verification for embedded systems through SVOCL: An OCL extension for SystemVerilog," *Design Automat. Embedded Syst.*, vol. 21, no. 1, pp. 1–36, 2017.
- [10] B. Korherr and B. List, "A UML 2 profile for event driven process chains," in *Proc. Int. Conf. Res. Practical Issues Enterprise Inf. Syst. (CONFENIS)*, Vienna, Austria, 2006, pp. 161–172.
- [11] V. Stefanov and B. List, "A performance measurement perspective for event-driven process chains," in *Proc. 16th Int. Workshop Database Expert Syst. Appl. (DEXA)*, Aug. 2005, pp. 967–971.
- [12] M. Kapurige, J. Han, A. Colman, and U. Rüegg, "EPClets—A lightweight and flexible textual language to augment EPC process modelling," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun./Jul. 2014, pp. 693–700.
- [13] E. Kindler, "On the semantics of EPCs: Resolving the vicious circle," *J. Data Knowl. Eng.*, vol. 56, no. 1, pp. 23–40, 2005.
- [14] J. Mendling and M. Nüttgens, "EPC markup language (EPML): An XML-based interchange format for event-driven process chains (EPC)," *Inf. Syst. e-Bus. Manage.*, vol. 4, no. 3, pp. 245–263, 2005.
- [15] A. Gross and J. Doerr, "EPC vs. UML activity diagram—two experiments examining their usefulness for requirements engineering," in *Proc. IEEE 17th Int. Conf. Requirements Eng.*, Aug./Sep. 2009, pp. 47–56.
- [16] I. Reinhartz-Berger, P. Soffer, and A. Sturm, "Extending the adaptability of reference models," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 40, no. 5, pp. 1045–1056, Sep. 2010.
- [17] J. Mendling, G. Neumann, and M. Nüttgens, "Yet another event-driven process chain," in *Proc. 3rd Int. Conf. Bus. Process Manage. (BPM)*, Nancy, France, 2005, pp. 428–433.
- [18] M. Rosemann and W. M. P. van der Aalst, "A configurable reference modelling language," *Inf. Syst.*, vol. 32, no. 1, pp. 1–23, 2005.
- [19] W. M. P. van der Aalst, M. Dumas, F. Gottschalk, A. H. M. ter Hofstede, M. La Rosa, and J. Mendling, "Preserving correctness during business process model configuration," *Formal Aspects Comput.*, vol. 22, no. 3, pp. 459–482, 2009.
- [20] R. Rieke and Z. Stoyanova, "Predictive security analysis for event-driven processes," in *Computer Network Security (Lecture Notes in Computer Science)*, vol. 6258. Berlin, Germany: Springer, 2010, pp. 321–328.
- [21] S. Xue, B. Wu, and J. Chen, "lightEPC: A formal approach for modeling personalized lightweight event-driven business process," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun./Jul. 2013, pp. 1–8.
- [22] J. Mendling, H. M. W. Verbeek, B. F. van Dongen, W. M. P. van der Aalst, and G. Neumann, "Detection and prediction of errors in EPCs of the SAP reference model," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 312–329, 2008.
- [23] W. M. P. van der Aalst, "Formalization and verification of event-driven process chains," *Inf. Softw. Technol.*, vol. 41, no. 10, pp. 639–650, 1999.

- [24] J. Mendling, B. F. van Dongen, and W. M. P. van der Aalst, "Getting rid of OR-joins and multiple start events in business process models," *J. Enterprise Inf. Syst.*, vol. 2, no. 4, pp. 403–419, 2008.
- [25] J. Mendling, J. Recker, M. Rosemann, and W. van der Aalst, "Generating correct EPCs from configured C-EPCs," in *Proc. ACM Symp. Appl. Comput. (SAC)*, Dijon, France, 2006, pp. 1505–1510.
- [26] V. Gruhn and R. Laue, "A comparison of soundness results obtained by different approaches," in *Proc. Int. Conf. Bus. Process Manage. (BPM)*, Berlin, Germany, 2010, pp. 501–512.
- [27] A. Winter and C. Simon, "Using GXL for exchanging business process models," *Inf. Syst. e-Bus. Manage.*, vol. 4, no. 3, pp. 285–307, 2005.
- [28] S. Mei, H. Cai, and F. Bu, "Multi-view service-oriented rule merged business process modeling framework," in *Proc. IEEE 6th Int. Symp. Service Oriented Syst. (SOSE)*, Dec. 2011, pp. 175–180.
- [29] K. Yongsiriwit, N. Assy, and W. Gaaloul, "A semantic framework for configurable business process as a service in the cloud," *J. Netw. Comput. Appl.*, vol. 59, pp. 168–184, Jan. 2016.
- [30] D. M. M. Schunselaar, J. Gulden, H. van der Schuur, and H. A. Reijers, "A systematic evaluation of enterprise modelling approaches on their applicability to automatically generate ERP software," in *Proc. 18th Conf. Bus. Inform. (CBI)*, Paris, France, Aug./Sep. 2016, pp. 290–299.
- [31] S. Denne, "Verifying properties of (timed) event driven process chains by transformation to hybrid automata," in *Proc. EPK*, 2006, pp. 157–176.
- [32] M. Faugère, T. Bourbeau, R. de Simone, and S. Gérard, "MARTE: Also an UML profile for modeling AADL applications," in *Proc. 12th IEEE Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, Jul. 2007, pp. 359–364.
- [33] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, "Model-driven development using UML 2.0: Promises and pitfalls," *Computer*, vol. 39, no. 2, pp. 59–66, Feb. 2006.
- [34] M. Tom and P. V. Gorp, "A taxonomy of model transformation," *Electron. Notes Theor. Comput. Sci.*, vol. 152, pp. 125–142, Mar. 2006.
- [35] S. Mallek, N. Daclin, V. Chapurlat, and B. Vallespir, "Enabling model checking for collaborative process analysis: from BPMN to 'network of timed automata,'" *J. Enterprise Inf. Syst.*, vol. 9, no. 3, pp. 279–299, 2015.
- [36] W. M. P. van der Aalst, "Formalization and verification of event-driven process chains," *J. Inf. Softw. Technol.*, vol. 41, no. 10, pp. 639–650, 1999.
- [37] W. M. P. van der Aalst et al., "ProM 4.0: Comprehensive support for real process analysis," in *Proc. 28th Int. Conf. Appl. Theory Petri Nets Other Models Concurrency (ICATPN)*, Siedlce, Poland, 2007, pp. 484–494.
- [38] F. Cassez and O. H. Roux, "Structural translation from time Petri nets to timed automata," *J. Syst. Softw.*, vol. 79, no. 10, pp. 1456–1468, 2006.
- [39] P. Bouyer, P.-A. Reynier, and S. Haddad, "Extended timed automata and time Petri nets," in *Proc. 6th Int. Conf. Appl. Concurrency Syst. Design (ACSD)*, Jun. 2006, pp. 91–100.
- [40] *UMLPACE Transformation Engine*. Accessed: Apr. 2018. [Online]. Available: <http://ceme.nust.edu.pk/ISEGROUP/UMLPACE/umlpace.html>
- [41] A. Amjad, F. Azam, M. W. Anwar, W. H. Butt, and M. Rashid, "Event-driven process chain for modeling and verification of business requirements—A systematic literature review," *IEEE Access*, vol. 6, pp. 9027–9048, 2018.
- [42] E. A. Lee, "Constructive models of discrete and continuous physical phenomena," *IEEE Access*, vol. 2, pp. 797–821, Aug. 2014.
- [43] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Advanced Course Petri Nets*. Berlin, Germany: Springer, 2003, pp. 87–124.



**FAROOQUE AZAM** is currently an Adjunct Faculty with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. He has been teaching various software engineering courses since 2007. His areas of interests are: model driven software engineering, business modeling for Web applications, and business process reengineering.



**MUHAMMAD WASEEM ANWAR** is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, CEME, National University of Sciences and Technology, Pakistan. He is a Senior Researcher and an Industry Practitioner in the field of embedded and control systems. His major research area is: model-based system engineering for complex and large systems.



**WASAI HAIDER BUTT** is an Assistant Professor with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. His areas of interests are: model driven software engineering, Web development, and requirement engineering.



**MUHAMMAD RASHID** received the bachelor's degree in electrical engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2000, the master's degree in embedded systems design from the University of Nice, Sophia-Antipolis, France, in 2006, and the Ph.D. degree in embedded systems design from the University of Bretagne Occidentale, Brest, France, in 2009. He is an Assistant Professor with the Computer Engineering Department, Umm

Al-Qura University, Mecca, Saudi Arabia.



**ANAM AMJAD** received the B.S. degree in computer sciences from International Islamic University and the M.S. degree in software engineering from the National University of Sciences and Technology, Pakistan, where she is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, CEME. Her area of research is business process automation through model driven software engineering.



**AAMIR NAEEM** is currently pursuing the master's degree with the Department of Computer and Software Engineering, CEME, National University of Sciences and Technology, Pakistan. His area of research is business process automation through model driven software engineering.

...