PLD-SiM: Process Line Diagram Simulator and Modeler

Akshit Rajput[1], Karnika Shivhare[1] and Rushikesh K. Joshi[1]

[1]*Department of Computer Science Engineering, Indian Institute of Technology Bombay, Mumbai, India*

Abstract

This paper introduces a browser-based tool for modeling applications with Process Line Diagrams (PLDs)[1]. The tool aims at providing a modular and logical approach for building process architecture models to bridge the gap between high-level process notations, low-level formal process models and implementation programming languages. While high-level process models primarily rely on manual practices for architectural detailing and trace conformance checking, the paper presents a user-assisted trace simulator to visually inspect the behavior of a process. The interface enables the user to obtain pen-paper snapshots from the process through dry run in the visual simulator. The visual modeler implements the syntactic rules of the PLD primitives, including roles, events, choices, and event synchronization. The same are enforced by the simulator for active prototyping through visual traces.

Keywords

Process Modeler, Workflow Modeling, Process Automation, Process Debugger, Process Line Diagram, PLD-SiM, PLD

1. Introduction

Process Line Diagram (PLD) [1] is a visual modeling approach for modeling of business processes in a modular fashion. It is a logical model of process architecture, which is aimed at bridging the gap between the high-level notational front such as BPMN [2], low-level formal models such as Petri nets [3], and implementation languages. It is known that high-level visual modeling is prone to engineering defects, and high-level visual languages, including the standard specification BPMN, carry notational defects such as non-compactness, bulkiness, complexity [4][5], redundancy [6] [7], and ambiguity [8] [9] [10]. In contrast, low-level formal modeling languages such as Petri nets do not have the richness of high-level notations that are required for easy comprehension at user-level or architectural-level modeling of processes. Trading off these gaps, Process Line Diagram (PLD) provides a minimalistic but a sufficiently high-level toolset of modeling notations. PLDs support modular models of processes, and they also bring the modeling phase closer to the implementation phase by introducing the concept of modular concurrency and structuredness at the modeling stage itself. This early integration of modular concurrency into the process is aimed at reducing error possibilities that may be incorporated during transition into implementation. The PLD approach weighs in the idea of implementation-friendly modeling notation by strongly recommending early modeling in a modular fashion.

PNSE'24, International Workshop on Petri Nets and Software Engineering, 2024

akshitrajput@cse.iitb.ac.in (A. Rajput); karnika@cse.iitb.ac.in (K. Shivhare); rkj@cse.iitb.ac.in (R. K. Joshi)

0009-0005-8654-2437 (A. Rajput); 0000-0001-6490-0380 (K. Shivhare); 0000-0002-2712-1406 (R. K. Joshi)

Its minimalistic toolset consists of 19 elements. Through its role-based modeling features, PLDs drive design thinking in terms of trace conformance with the desired process behavior.

This paper discusses PLD-SiM, a tool which bundles both modeler and simulator for PLD diagrams. The modeler segment of the tool is inclusive of syntactic modeling checks that are performed in accordance with the definition of well-formedness of PLD notations. They include checks for synchronization through naming correspondence in message and event throw-catch pairs. The simulator can be used to execute a modeled process in prototype mode going over its control flows. The simulation process in this tool is equipped with user-interactivity to enable modelers to carry out refinements. By running the process in prototype mode in the simulator, an architect can find the exact location in the model that deviates from the desired process behavior. Once the deviation has been found, iterative refinement to the model can be carried out as a debugging action and the prototype execution can be repeated.

2. Literature Survey

There are tools available for modular process modeling using other notations such as Appian[11], Bizagi [12], and Camunda [10][13] for Bussiness Process Modeling Notation (BPMN)[2] and Decision Modelling & Notation (DMN)[14]. Along with their modeling properties, these tools provide low-code automation that facilitates rapid design, development, and deployment of workflows and applications for organizations. AgroUML [15], MagicDraw tool [16], and Dia [17] are some tools that are used to model Unified Modeling Language (UML) [18]. However, UML does not support the richness of abstractions required for business process modeling. CPN tools (currently CPN IDE) [19] presents a graphical environment for the development, and simulation of colored Petri net models for the design and analysis of complex systems. Tools Appian[11], Bizagi[12] and Camunda [10][13] are used for high-level modeling notations BPMN and DMN.

The PLD approach prescribes a role-centric process-behavior-driven modeling with a focus on structuredness in the initial stages. In order to aid automation of architectural detailing and trace conformance checking for PLD models, the PLD-SiM modeler implements a visual simulator.

3. PLD-SiM (PLD Simulator and Modeler)

The PLD-SiM tool follows a two-tier architecture with three components, namely, Interface, Modeler, and Simulator. Now we discuss the working and use of each of the three components.

3.1. PLD-SiM Interface

The user interface comprises a toolkit panel, a canvas, and a functionality group. It is responsible for maneuvering the entire functioning of PLD-SiM processes, inclusive of model designing in PLD, trace simulation, process debugging, load-store of processes from and to JSON format, generating operational XML for processes etc. Now, we describe the components of the interface to further elaborate Figure 1.

On the left-hand side of the canvas in Figure 1, modeling elements of PLD-SiM are enlisted. They include a total of 19 elements. In addition, it also includes an annotation text-box, and a join primitive for conditional and selection guard. These 21 elements in the toolset can be used in combination to model a process. The entire toolset is summarised in Table 1.

Role: It is the first used element in the toolset, and it represents the role of an actor, which may be a logical active process entity or a person or a resource in real life. It is a unit of concurrency. Every other model element is part of some or the other role in the model. Without a role, no element except textboxes can be placed. A role is created as a vertical flow line, which can be extended as and when more elements are added to it. When a new role or any other element such as a message element is created, the tool prompts for its name. In Figure 1, the modeled process has *Order Fulfillment* and *Procurement* as its two roles.

Task: It denotes an activity performed by a role. In design terms, it usually represents the execution of a function performing some specific functionality to generate a desired result in the process. A task is depicted as a small filled circle, and it is always associated with some role. Examples of tasks in Figure 1 are *Check Availability*, *inform customer*, *order from supplier*, *ship article*, *check availability with supplier*, and *Financial settlement*. They are from two roles in the model. In a Petri net model, a task is typically depicted as a transition, whereas, BPMN has multiple special notations for different types of tasks.

Gateway: There are two types of gateways, namely, conditional gateway and selection gateway. The former represents a branching with conditions mentioned on their respective paths. On the other hand, a selection gateway also depicts an exclusive choice, but it models a non-deterministic branching such as user selection. In both gateways, only one path continues. In Figure 1, the gateway after task *check availability* in role *Order Fulfillment* is a conditional gateway.

Guards: They provide event and message based branching options in the PLD-SiM, permitting

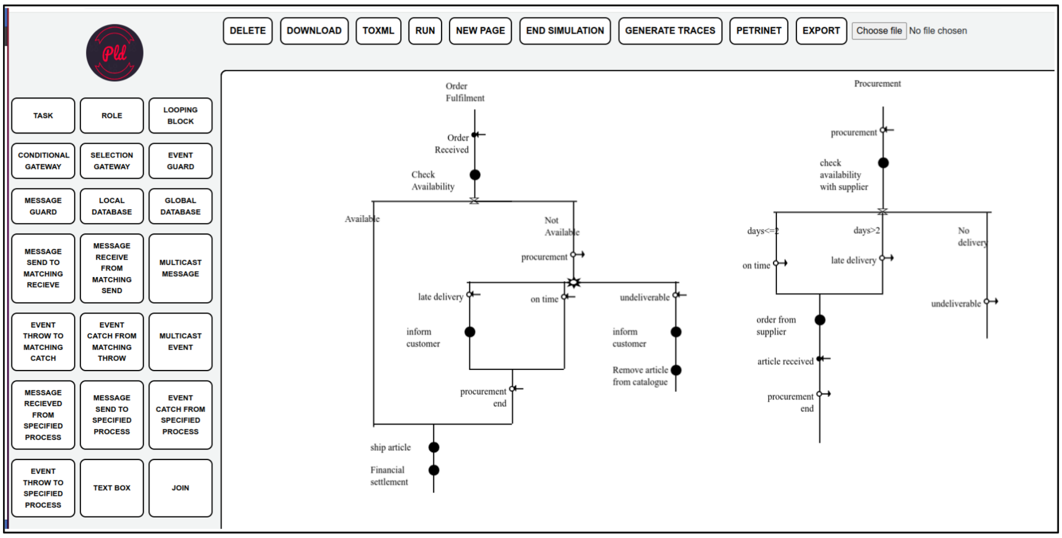

Figure 1: PLD-SiM user interface with an Order Fulfillment and Procurement process modeled

a role to wait for some specific message or an event notification arriving from another role. If one of the guard messages or event notifications is received, the respective branch is continued. In Figure 1, the guard after event *procurement* is an event notification guard. It branches based on the type of event notification received in the immediate previous event receive action. In Figure 1, there are three types of procurement notifications: *late delivery*, *undeliverable*, and *on time delivery*.

Communications: Messages and events are used for modeling inter-role interactions. Event-based and message-based interactions along with their variants are possible in PLD-SiM. A *message send* and a matching *message receive* primitives are used for message communication. The name of the Message received and the message sent need to match. In Figure 1, *Order* in the *Customer* role is a message send action, and *Order* in the *Order Fulfillment* role is a message received. In *message send to a specified process* messaging mechanism, a message is sent to a particular role. *Message Receive from a specified Process* receives a message from a specified role. The name of the target role is specified.

For event-based communication, PLD-SiM has a separate toolset. The primitive *Event Throw to Matching Catch* represents an event throw, which is supposed to be caught by a matching *Event Catch from Matching Throw* in another role. In figure 1, the *procurement* in the *Order Fulfillment* role is an example of Event Throw to Matching Catch, and *procurement* in the Procurement role is an example of *Event Catch from Matching Throw*. This event communication is performed over matching event names, irrespective of the role in which they appear. PLD-SiM also supports role-specific event communication via primitives *Event Throw to Specified Process* and *Event Catch from Specified Process*.

Multicast Message and *Multicast Event* are communication channels for multicast mode. These bulk operations can be chosen to enhance parallelism in communication.

Textbox aids modelers in adding additional captions or annotations in a process. It can exist independently as a reference note. They do not have any behavioral implications.

Join button enables modelers to merge branches inside a single role.

3.2. PLD-SiM Modeler

The PLD-SiM modeler is responsible for providing meaning to the elements, connecting them into a process, defining and storing required states for simulation, and ensuring modeling validations of a PLD process. The interface is responsible for handling the location of each element of the drawing on the canvas, whereas, the PLD-SiM modeler defines the relative position of each element in the model. Moreover, the PLD-SiM modeler defines JSON properties and definitions for each element of the model. It then connects all the elements to form them into an in-memory process data structure in accordance with PLD rules.

PLD-SiM Modeler is equipped with a set of process validations that are required to be met during the process. For each element, while performing addition, modification, or removal on the canvas, the Interface interacts with the Modeler component to validate the change a user wishes to draw. It then creates a validated code-based representation for the modeled PLD process in JavaScript with data in JSON notation. This data includes coordinates on canvas, IDs, names and properties of elements, and the connections between elements.

Element Name	Element Design
Role	
Task	
Condition Gateway	
Selection Gateway	
Event Guard	
Message Guard	
Message Send to Message Receive	
Message Receive from Matching Send	
Message Send to Specified Process	<P> <M>
Message Receive from Specified Process	<P> <M>
Multicast Message	
Event Throw to Matching Catch	
Event Catch from Matching Throw	
Event Throw to Specified Process	<P> <M>
Event Catch from Specified Process	<P> <M>
Multicast Event	
Local Database	
Global Database	
Loopback	
Join	

Table 1: **PLD-SiM Toolset**

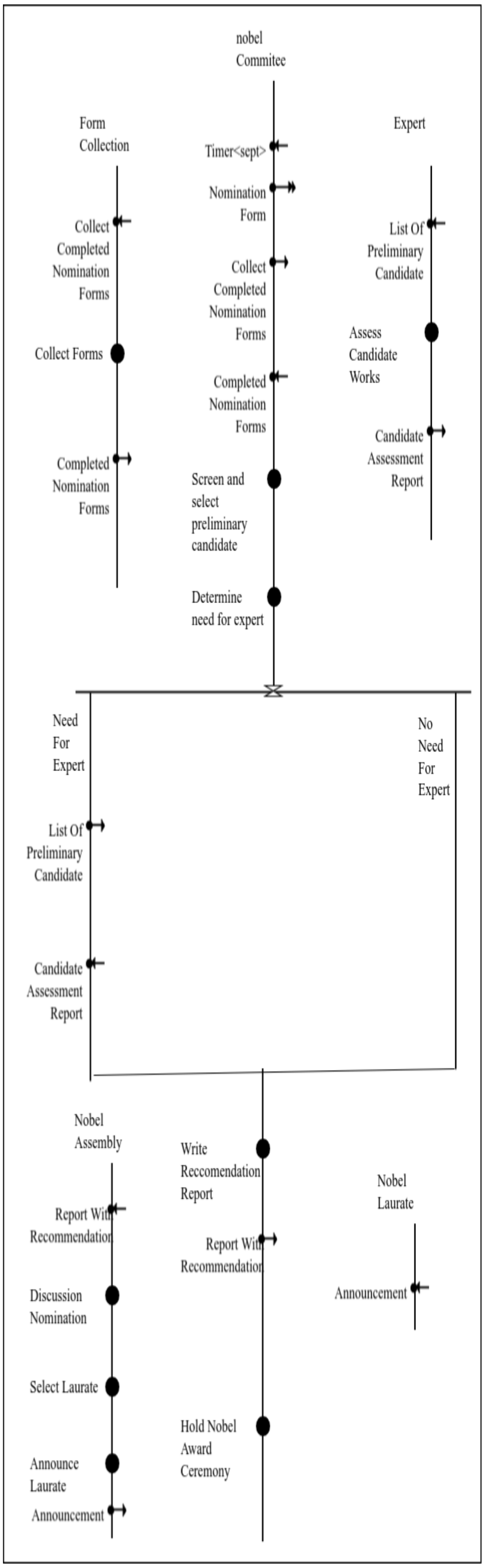

Figure 2: **Nobel prize process model**

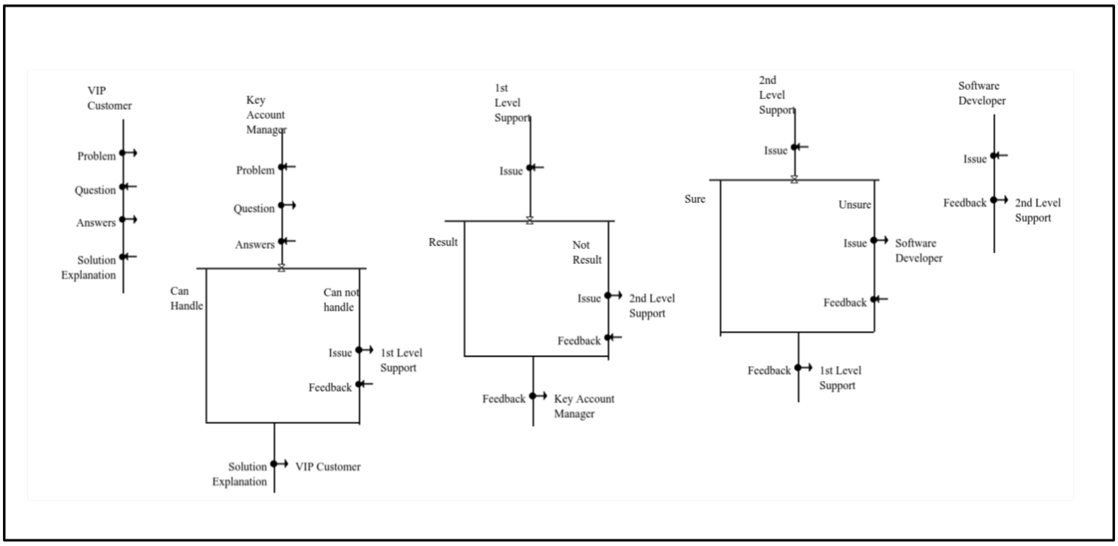

Figure 3: **Incident management process**

3.3. Modeling Validations

The tool does not allow any validation criteria to be violated at any point in time during modeling. The following are a few modeling validations imposed by the tool.

Inclusion Properties ensure syntactically correct inclusion of an element inside another element in a model. A task can only appear over a role in the process drawing, i.e. a task definition can only be included inside a role definition. A role is a part of the process definition. Also, a guard can only be defined within a role. Each branch in an Event-based guard begins with an event catch, and each branch in a Message-based guard begins with a message received.

Naming mandates all roles and tasks to have names. Primitives *Message Send to Matching Receive*, *Message Catch from Matching Send*, *Multicast Message* have a mandatory message name attached to them. For primitives *Message Send To Specified Process* and *Message Receive from Specified Process*, besides the message name, a role name also needs to be identified, in order to name the role to which the message is to be sent or from which the incoming message is to be caught. Similarly, *Event Throw to Matching Catch*, *Event Catch from Matching Throw*, *Event Throw to Specified Process* and *Event Catch from Specified Process* have mandatory event descriptions attached to them. These communication primitive pairs are thus identified with their matching names, using which a communication gets established.

Branching validations All branches in conditional gateways have conditions specified in text form. Each flow branch inside a role following an event guard must have *Event Catch from Matching Throw* as its first element. Similarly, after a *Message Guard* in every branch, the role flow must have a *Message Receive from Matching Send* primitive as its first element. These well-formedness rules of PLD are ensured by the PLD-SiM modeler. The tool prompts the user whenever naming information, and mandatory primitives are required from the user.

3.3.1. PLD Simulator

The PLD-SiM Simulator is responsible for providing simulation of the modeled processes. It utilizes the model data generated by the modeler to create a simulation environment for the process, which is presented on the interface for the users to view and interact to carry out a simulation run. The tool uses a token-based and interactive method with a three-colored scheme for simulation. All elements and the process are shown in black color during the modeling mode. In simulation mode, black color represents all deactivated places of the process. Green color token showcases all available choices for the process at a given point in the simulation run. Multiple elements might have a green token in a state at a single point of time. Red color tokens are used for guard and gateway choices. A user can pick any of the green or red color tokens to move into the next step in the simulation. The simulator then generates the tokens for the next possible token places according to the current choice of the token interactively indicated by the user. Thus, the PLD-SiM modeler tool provides a user interactive simulation process for the users to verify and debug the modeled process. Another simulation run of the same process can provide a different trace as per the choices made by the user during the simulation.

The simulator enables *selective tracing and debugging* of a process, as any trace can be simulated by the user as far as it is within the model's state space. Interactive tracing can be used in debugging of a process to carry out conformance validation as per the trace requirements.

4. Example Display

In this section, we discuss PLD models of two examples that have been modeled in the tool. These examples are modeled based on the examples given in the OMG's (Object Management Group) BPMN by Example [20][2].

4.1. Nobel Prize process

Figure 2 shows the PLD model of the noble prize process [20] in the PLDMS tool. [Note: Since the tool at present does not support timed processes, the timer in the original process has been modeled as a start trigger message.] The process starts in September, when a *timer trigger* is activated at the Nobel committee role. The committee multicasts a *Nomination Form* and sends a message *Collect Completed Nomination Forms* to role *Form Collection*. Role *Form Collection* completes task *Collect Completed Nomination Forms* and sends message *Completed Nomination Forms* to role *nobel Committee*, which completes the task *Screen and select preliminary candidate*.

After this, a condition guard is placed, which models the condition by the committee regarding the need for an expert. The two conditions *Need For Expert* and *No Need For Expert* provide the two alternative flows. If needed, role *Expert* is contacted with a message *List Of Preliminary Candidate*. Role *Expert* carries out task *Assess Candidate Work*. In this task, the expert accesses the work of candidates for analysis. The expert sends back a message *Candidate Assessment Report* to role *noble comittee*. In the case of selection of another condition not needing an expert, the committee carries out task *Write Recommendation Report*. At this time, both the branches of the condition guard are merged. The role *nobel Commitee* carries out task *Write Recommendation Report* and sends message *Report With Recommendation* to role *Nobel Assembly*. Role *noble* *committee* sends a recommendation report to role *Nobel Assembly*, via message *Report With Recommendation*. Role *Nobel Assembly* carries out task *Discussion of Nomination*, and it completes task *Select Laurate*. After selecting the laurate, they make an announcement. A message is sent to both *Nobel Laurate* and *nobel Commitee*, via a multicast message *Announcenment*. The committee holds the ceremony, with task *Hold Nobel Prize Ceremony* to complete the process.

4.2. Incident Management

Figure 4 shows the PLD simulation trace of the incident management process shown in Figure 3. Each state shows the role that is active in that step. In this process, a role *VIP Customer* sends a message *Problem* to role *Key Account Manager*. Role *Key Account Manager* asks questions to role *VIP Customer* via message *Question*. Role *VIP Customer* replies to those questions through message *Answers*. After analyzing the answers from role *VIP Customers*, role *Key Account Manager* executes a condition regarding handling of the issue. The two conditions *Can Handle* and *Can not Handle* provide the two alternative flows. If role *Key Account Manager* can not handle the issue, then the role sends a message Issue to another specific role *1st level support*. This communication is done using a message *Message Sent to Specified Process* element. This message variant is used to send a message to a particular role. If the branch *Can Handle* is chosen by role *1st Level Support*, a message *Solution Explanation* is sent to role *VIP Customer*.

On receiving message *Issue*, if the first level support can handle the issue and execute condition *Result*, it sends a message *Feedback* to role *Key Account Manager*. If the branch with condition *Not Result* is chosen, a message *Issue* is sent to role 2^{nd} *Level Support*. The role 2^{nd} *Level Support* makes a conditional decision about handling the issue. The conditions specified are *sure* and *unsure*. If role 2^{nd} *Level Support* is sure of the result, it sends a message *Feedback* to role 1^{st} *Level Support*. If it is unsure about the solution, then it forwards message *Issue* to role *Software Developer*. Role *Software Developer* returns message *Feedback* to role 2^{nd} *level support* as its solution. In this way, the communication proceeds to sort out the issue raised. These steps are shown in Figure 4 depicting the simulation trace.

4.3. A brief comparison to BPMN

Figure 3 shows a model of the incident management process in PLD-Sim. Owing to its richness, a BPMN diagram typically reflects a complex structure and a high notational knowledge even while making a relatively simpler process. On the other hand, the PLD model shows a simpler, and an ambiguity-free architecture. The simulation run of the same is shown in Figure 4. In the PLD-SiM tool, communication mechanisms like message and event send and receive, with a compact toolset comprising 19 elements reduces learning time to start with modeling a process, avoiding possible ambiguities and variations, and model complexity. Model complexity is usually observed in high-level modeling notations as in BPMN[21], which has a large set of symbols and multiple alternatives. The incident management process modeled in PLD-SiM requires only five elements, namely, the *Message Send*, *Message Receive*, *Task*, *Role*, and *Conditional gateway* to complete the whole process. A smaller set of elements is intended to make it easier to model and learn for a new user. Whereas, modeling in BPMN[21] requires an understanding of a wide variety of symbols and abstractions.

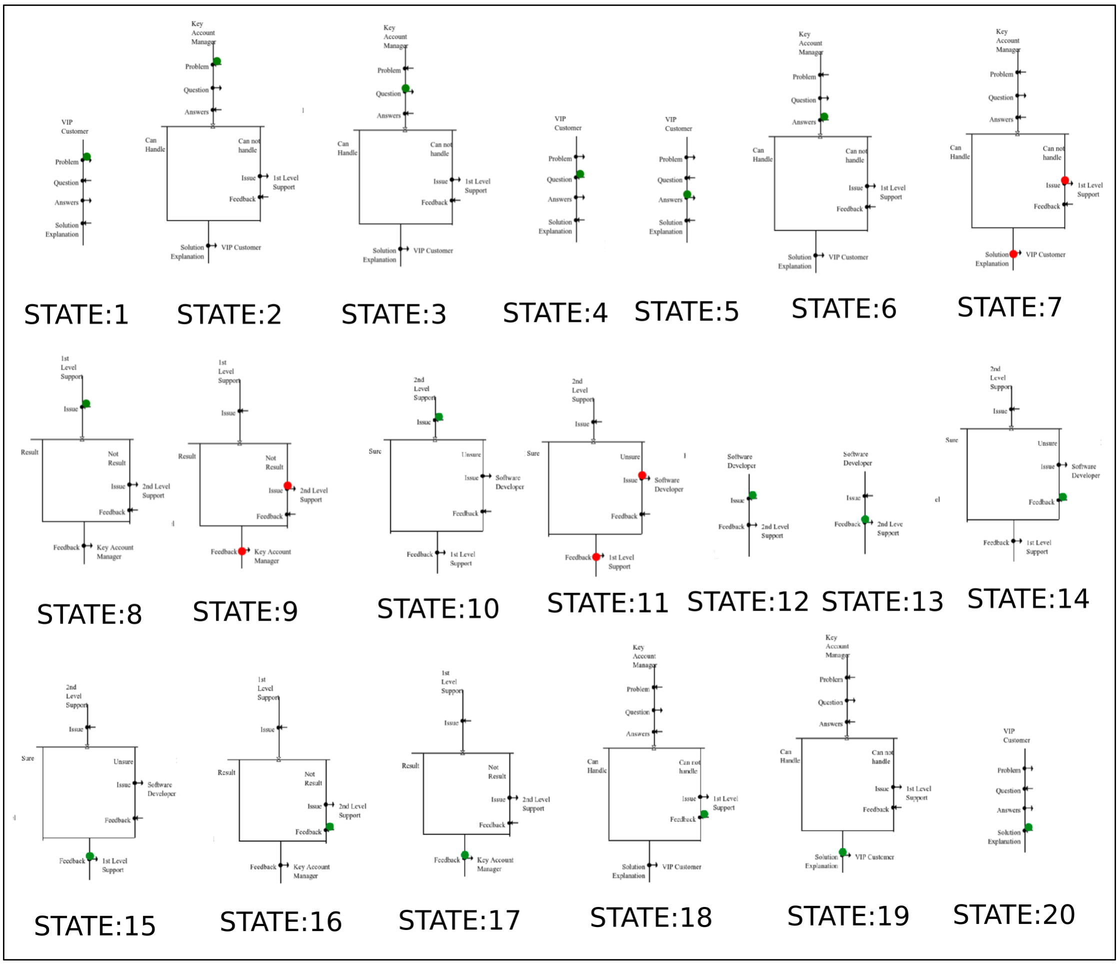

Figure 4: Simulation states of a trace in Incident Management process in the PLD-SiM

5. Conclusions and future work

We presented PLD-SiM, a tool that provides a platform to model and simulate PLD processes. PLD is an alternative to BPMN with a compact notation set, and a tracing-friendly notation around concurrent roles, intra-role branching, and inter-role communication. The user can simulate and debug the process model using selective inputs from the user, as the tool provides a seamless approach to debugging the processes. Further, we look forward to convert the processes into Petri Nets for formal analysis, and to integrate code generation for automation.

References

[1] K. Shivhare, R. K. Joshi, Process line diagrams (plds): An approach for modular process modeling, in: Proceedings of the 16th Innovations in Software Engineering Conf., 2023.

[2] Object Management Group (OMG), OMG Business Process Model and Notation (BPMN) Version 2.0 Specification, 2011. URL: http://www.omg.org/spec/BPMN/2.0.

[3] T. Murata, Petri nets: Properties, analysis and applications, Proc. of the IEEE 77 (1989).

[4] A. Suchenia, Towards a taxonomy of business process and its anomalies, International Journal of Computer Science & Network Security 21 (2021) 230–240.

[5] M. z. Muehlen, J. Recker, How much language is enough? theoretical and practical use of the business process modeling notation, Seminal Contributions to Information Systems Engineering: 25 Years of CAiSE (2013) 429–443.

[6] H. Leopold, J. Mendling, O. Günther, Learning from quality issues of bpmn models from industry, IEEE software 33 (2015) 26–33.

[7] T. Rozman, G. Polancic, R. V. Horvat, Analysis of most common process modeling mistakes in bpmn process models, Eur SPI'2007 (2008).

[8] E. Börger, O. Sörensen, B. Thalheim, On defining the behavior of or-joins in business process models., J. Univers. Comput. Sci. 15 (2009) 3–32.

[9] F. Corradini, C. Muzi, B. Re, L. Rossi, F. Tiezzi, Global vs. local semantics of bpmn 2.0 or-join, in: 44th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2018, Springer, 2018, pp. 321–336.

[10] F. Corradini, C. Muzi, B. Re, L. Rossi, F. Tiezzi, Bpmn 2.0 or-join semantics: Global and local characterisation, Information Systems 105 (2022) 101934.

[11] A. R. Kunduru, Cloud bpm application (appian) robotic process automation capabilities, Asian Journal of Research in Computer Science 16 (2023) 267–280.

[12] O. Gjoni, Comparison of two model driven architecture approaches for automating business processes, moskitt framework and bizagi process management suite, Mediterranean Journal of Social Sciences 6 (2015) 615.

[13] E. Schäffer, V. Stiehl, P. K. Schwab, A. Mayr, J. Lierhammer, J. Franke, Process-driven approach within the engineering domain by combining business process model and notation (bpmn) with process engines, Procedia CIRP 96 (2021) 207–212.

[14] J. Taylor, J. Purchase, Real-world decision modeling with DMN, Meghan-Kiffer Press Tampa, 2016.

[15] W. Complak, A. Wojciechowski, A. Mishra, D. Mishra, Use cases and object modelling using argouml, in: On the Move to Meaningful Internet Systems: OTM 2011, Hersonissos, Crete, Greece, October 17-21, 2011. Proceedings, Springer, 2011, pp. 246–255.

[16] E. Planas, J. Cabot, How are uml class diagrams built in practice? a usability study of two uml tools: Magicdraw and papyrus, Computer Standards & Interfaces 67 (2020) 103363.

[17] M. Ozkaya, Are the uml modelling tools powerful enough for practitioners? a literature review, IEt software 13 (2019) 338–354.

[18] L. Khaled, A comparison between uml tools, in: 2009 Second International Conference on Environmental and Computer Science, 2009, pp. 111–114. doi:10.1109/ICECS.2009.38.

[19] J. Kurt, M. K. Lars, Coloured petri nets modeling and validation of concurrent systems (2009).

[20] Object Management Group (OMG), OMG BPMN by Examples, 2010. URL: http://www.omg.org/spec/BPMN/2.0/examples/PDF.

[21] M. Chinosi, A. Trombetta, Bpmn: An introduction to the standard, Computer Standards & Interfaces 34 (2012) 124–134.