Back to the Model: UML Miner and the Power of Process Mining

Pasquale Ardimento[1][a], Mario Luca Bernardi[2][b], Marta Cimitile[3][c] and Michele Scalera[1][d]

[1]*Department of Informatics, University of Bari Aldo Moro, Via Orabona 4, Bari, Italy*

[2]*Department of Engineering, University of Sannio, Benevento, Italy*

[3]*Department of Law and Digital Society, Unitelma Sapienza University, Rome, Italy*

Keywords: UML Class Diagrams, Software Modeling, Process Mining, Conformance Checking, Computing Education.

Abstract: Comprehension of the Unified Modeling Language is essential for learners in the context of software modeling. However, current UML learning tools provide minimal guidance to novice modelers as they are insufficient in analyzing modeling behaviour adopted during the diagram creation process. In order to address this gap, we present an enhanced version of UML Miner, a plugin for Visual Paradigm, that systematically records and analyzes UML modeling activities through the use of Process Mining techniques. UML Miner tracks all modeling events, resulting in event logs that warrant conformance checking against expert modeling practices. This tool establishes flexible, yet structured learning pathways through Declarative Process Mining, supporting trace-based and event-based filtering, customized violation reports, and integration with external process mining tools. This work emphasizes the potential of process mining in computing education, demonstrating how conformance checking can strengthen UML modeling proficiency.

1 INTRODUCTION

Graphical modeling languages, such as the UML (Unified Modeling Language), are essential for developing, understanding, and communicating various views of a system. In software engineering and computing education, improving the quality of modeling practices among students and novice modelers has been a long-standing challenge. To address this, educators have increasingly turned to software modeling principles, tools, and environments, as well as learning analytics techniques like Process Mining (PM). These approaches enable the study of individual behaviors during software diagram modeling, allowing educators to provide personalized feedback based on the practices of more skilled modelers. Despite the availability of UML tools for educational purposes (Crahen et al., 2002; Foss et al., 2022; Alhazmi et al., 2021), most lack a focus on conformance checking, a process mining technique that compares actual process executions (recorded in event logs) against predefined process models representing expected behaviors. While recent advancements have introduced AI-powered UML modeling tools, such as ChatUML, DiagrammingAI, and AI UML Diagram Generator, these tools primarily aim to facilitate diagram creation and improve modeling efficiency. However, they do not address the need for feedback on the modeling process itself, particularly for novice modelers. Although process mining has been applied to analyze student behaviors in development activities (Ardimento et al., 2019c; Ardimento et al., 2019b; Ardimento et al., 2019a), its application to support novice UML modelers remains unexplored. To bridge this gap, we present an enhanced version of UML Miner, a plugin for the Visual Paradigm (VP) environment. UML Miner records event logs generated from a modeler's interactions with VP, enabling the analysis of their modeling process. By leveraging process discovery and conformance checking, the tool provides insights into how students approach complex modeling tasks and compares their behaviors to those of experienced modelers. A prototype of UML Miner was previously introduced at the Models 2023 conference (Ardimento et al., 2023), and this work extends its capabilities to offer more comprehensive support for novice UML modelers. The current version of the tool introduces several significant extensions compared to the previous version. These enhancements include:

[a] https://orcid.org/0000-0001-6134-2993
[b] https://orcid.org/0000-0002-3223-7032
[c] https://orcid.org/0000-0003-2403-8313
[d] https://orcid.org/0000-0002-2455-2032

Ardimento, P., Bernardi, M. L., Cimitile, M., Scalera and M.
Back to the Model: UML Miner and the Power of Process Mining.
DOI: 10.5220/0013500100003964
In *Proceedings of the 20th International Conference on Software Technologies (ICSOFT 2025)*, pages 207-214
ISBN: 978-989-758-757-3; ISSN: 2184-2833

- **Capturing Complex Relationships and Behaviors**. To achieve this, we have enabled the capture of complex relationships and behaviors in UML Miner through the use of templates. These templates define constraints applied to activities within the modeling process, providing a structured approach to representing intricate dependencies. The semantics of these constraints are formalized using various logical frameworks, such as Linear-Time Temporal Logic, allowing precise reasoning about process behavior over time.
- **Enhanced Filtering Capabilities**, allowing users to dynamically filter modeling events based on specific analytical needs that may arise in the context of UML software modeling. The tool enables filtering across multiple dimensions, such as element type (e.g., classes, associations, attributes), modeling actions (e.g., creation, modification, deletion) and temporal aspects (e.g., sequence of edits, time spent on specific elements). Additionally, it supports both trace-based filtering (for entire modeling sessions) and event-based filtering (for specific modeling actions), ensuring a flexible and targeted approach to analyzing students' modeling behaviors.

The structure of this paper is organized as follows. Section 2 introduces the modeling language used to represent the modeling process in the tool. Section 3 describes the core functionalities of the tool, and Section 4 presents an example scenario. Section 5 concludes the paper, which provides directions for future research.

2 BACKGROUND: DECLARATIVE PROCESS MINING

PM enables the analysis of event logs generated during process execution, allowing insights into behavior and identification of inefficiencies (van der Aalst, 2016). In our tool, we adopted a *declarative* modeling approach, which is particularly suitable for software modeling, where different strategies may be valid as long as fundamental constraints are respected.

Unlike imperative models, which specify all possible execution paths, declarative models define *constraints* that govern allowed behaviors. We use the **Declare** language (Pesic et al., 2007), in which constraints are specified through reusable *templates*, formalized using temporal logics such as LTL. Declare supports various constraint types (Alman et al., 2020b; Alman et al., 2020a), including: **Existence**: ensuring an activity occurs at least once, **Relation**: expressing dependencies between two activities, **Negative relation**: specifying that an activity must not follow another, **Choice**: requiring the selection between alternative activities.

Constraints apply over traces, i.e., sequences of modeling events. When a constraint is activated, it imposes an obligation on subsequent events; it is *fulfilled* if respected and *violated* otherwise. If never activated, it is *vacuously satisfied* (Kupferman and Vardi, 2003). We also adopt the notion of *semantic vacuity* (Burattin et al., 2012), which ensures a constraint is meaningful only when it is both activated and fulfilled at least once.

3 UML MINER

UML Miner establishes a monitored modeling environment where all modeling activities are systematically captured and recorded in event logs. This ensures a detailed trace of the steps taken by students during the creation of class diagrams. It also enables the discovery of the actual process model adopted by each student, leveraging Declare Templates to reveal the sequence and logic of their actions. Furthermore, a conformance checking technique can be applied to compare a given process model with the corresponding event log, determining the degree of alignment between the two.

This environment facilitates both the discovery of actual process models and their evaluation in terms of conformance with a reference process, such as the one derived from the teachers' solution. Figure 1 illustrates the architecture of UML Miner, highlighting its core components:

- **Log Recorder**, which captures modeling activities;
- **Process Discoverer**, which reconstructs the process model;
- **Conformance Checker**, which assesses alignment with reference models;
- **Log Filtering**, which refines event logs for further analysis.
- **Violations Exporter**, which generates a customized report listing all adopted modeling behaviors, indicating for each whether it deviates from or aligns with a reference model, e.g. the instructor's reference model.

As shown in Figure 1, the tool is designed with maximum flexibility. Process discovery can be performed internally, using a declarative approach, or externally by exporting event logs to external process mining tools. These logs, formatted in the standard XES (eXtensible Event Stream) format, are compatible with a wide range of process mining tools. Similarly, conformance checking can also be conducted outside the VP environment, as long as the event logs conform to the defined schema for recording modeling activities. The flexibility of the system comes from its ability to integrate with external tools and processes. For example, by exporting event logs, users can apply any process discovery algorithm or representation language supported by external tools. Additionally, UML Miner supports conformance checking internally by generating outputs in various textual formats, while an extended version based on the RuM tool (Alman et al., 2021) provides graphical visualizations of both discovered process models and conformance checking results.

3.1 Log Recorder

The **Log Recorder** is a core component of UML Miner, designed to capture and record all modeling events performed across UML diagrams within the VP environment. Performing as a background daemon, it operates continuously and autonomously, ensuring uninterrupted monitoring of modeling activities. The Log Recorder initiates automatically upon VP startup and remains active until VP is terminated, without interfering with the user's modeling tasks. For each VP project, UML Miner generates a unique event log that comprehensively documents all working sessions associated with the project. A *working session* is defined as the sequence of modeling events that occur between the opening and closing of a project. The Log Recorder begins to capture events immediately upon project creation and stores them in a dedicated event log file. If a project is reopened, the tool appends a new trace to the existing log file, preserving a complete history of the project's modeling activities.

To facilitate structured analysis, the Log Recorder categorizes modeling events into the following types.

- **ProjectEvent:** Captures high-level events related to the VP project, such as creating or deleting diagrams, adding or removing UML model elements, opening or saving the project, and renaming the project. For example, adding a new UML diagram or removing a class from the project triggers a ProjectEvent.
- **DiagramEvent:** Tracks events specific to individual UML diagrams, such as adding or removing UML model elements within a diagram. This ensures granular tracking of diagram-level modifications.
- **ModelElementPropertyEvent:** Records changes to the properties of UML model elements, whether at the project or diagram level. Properties can be *atomic* (e.g., class visibility) or *composite* (e.g., class attributes with properties like name, visibility, type modifier, and multiplicity). Atomic properties are intrinsic to model elements and assume default values if not explicitly defined, whereas composite properties are user-defined and include unique identifiers (IDs). For instance, modifying a class attribute logs both the property and its updated value, along with metadata specific to the property type.
- **ModelsRelationshipEvent:** Documents events involving relationships between UML model elements, such as associations, aggregations, generalizations, or "extends" relationships between use cases. Each relationship event logs specific properties, including the relationship ID, the source model element ID, and the target model element ID.
- **ExtensionMechanismEvent:** Captures events related to UML extension mechanisms, such as stereotypes and tagged values applied to model elements. Since a single model element may have multiple stereotypes or tagged values, these are stored in a vector for efficient retrieval and analysis.

The Log Recorder ensures comprehensive event tracking, even in scenarios where a UML model element is added to a project but not yet included in any diagrams. However, to maintain focus on meaningful modeling activities, the tool deliberately excludes stylistic or formatting-related information (e.g., diagram layout or element colors) from the event logs. This design choice ensures that the logs remain concise and relevant for process mining and conformance checking tasks.

3.2 Defining UML Event Log Structure

To ensure interoperability and enable external analysis, the modeling event data collected by UML Miner are serialized according to the IEEE XES standard. The event log encodes trace-level and event-level information related to the UML modeling process.

Figure 2 shows the metamodel representing the structure of UML modeling event logs.

Each **Log** consists of multiple **Trace** elements. A trace corresponds to a modeling session and includes metadata such as:

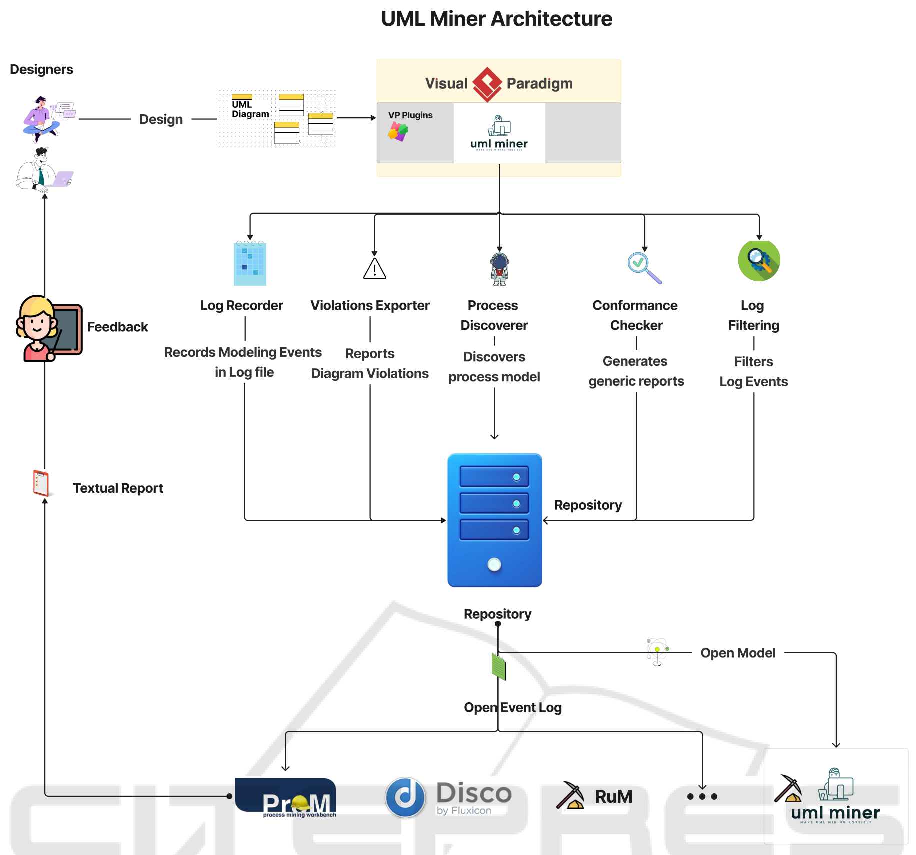


Figure 1: UML Miner architecture.

- `concept:name` (trace name),
- `identity:id` (trace/session identifier),
- `AuthorName` (student or modeler),
- `ProjectName`,
- `CaseTimestamp` (timestamp of the session).

Each trace is composed of a sequence of **Event** elements, where each event corresponds to a specific modeling action performed by the user. For every event, the log records:

- the name and instance of the activity (`concept:name`, `concept:instance`),
- the timestamp of execution (`time:timestamp`),
- information about the UML element involved in the activity, including its ID, type, and name (`UMLElementId`, `UMLElementType`, `UMLElementName`),
- information about the diagram context, such as its ID, type, and name (`DiagramId`, `DiagramType`, `DiagramName`).

Depending on the type of event, additional attributes may also be present:

- `UMLElementChildren`, for events involving hierarchical elements like packages,
- `PropertyName` and `PropertyValue`, for events related to property updates (e.g., renaming an element),
- `RelationshipFrom`, `RelationshipTo`, and their IDs, for events involving relationships such as associations or generalizations.

This structure allows the log to describe the sequence, timing, and context of modeling operations performed in the UML editor. The metamodel also provides a foundation for trace-based analysis, process discovery, and conformance checking. The representation is flexible enough to include custom classifiers and extensions relevant to UML modeling semantics.

3.3 Process Discoverer

The behavior recorded in an event log serves as the basis for further analysis. This behavior is then structured and interpreted by extracting process models from the event log, a procedure known as process

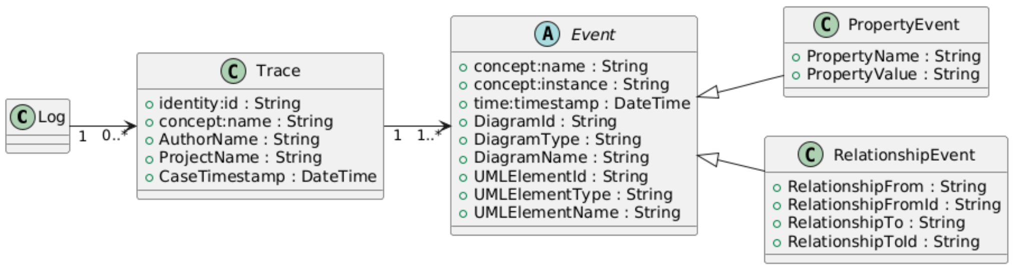


Figure 2: Metamodel representing the structure of the UML Event Log.

discovery. The process models discovered for UML class diagram modeling are represented as a set of constraints, where each constraint specifies a particular aspect of the modeling process. These constraints describe either the cardinality of a modeling activity (e.g., the number of times a class or attribute is defined in a trace) or the relationships between different modeling activities (e.g., how the creation of a class requires or prohibits the definition of a relationship with another class).

In the context of class diagram, the modeling events include activities such as the creation of a class, the definition of an operation, the addition of an attribute, or the establishment of a relationship between classes. For instance, a constraint might specify that a class must always have at least one attribute, or that creating an association between two classes must follow the definition of those classes.

The discovered models can be exported in two formats: the Declare format and a textual format. The textual format represents the entire model as a set of natural language sentences, making it accessible to users who may not have prior knowledge of the Declare framework. For example, a textual representation could state, "Every class must include at least one attribute," or "If a class defines an operation, it must eventually define at least one attribute." Conversely, the Declare format is designed to allow these models to be utilized in conformance checking tasks within tools like UML Miner.

Declare uses templates, which are parameterized temporal logic patterns, to define constraints. These templates are equipped with a graphical notation to facilitate the representation of process maps and can be expressed in natural language, eliminating the need for teachers to be familiar with formal logic. Declare constraints are instantiated templates, where template parameters are replaced with specific modeling activities for class diagram creation.

The following list illustrates some examples of templates used for class diagram modeling. A complete list and a complete description of these templates can be found in (Pesic, 2008):

- **Unary: Existence templates:** These apply conditions to the occurrence of single modeling activities. For example, `EXISTENCE(CreateClass)` ensures that at least one class is created in the process.
- **Binary: Relation templates:** These define relationships between pairs of modeling activities. For instance, `RESPONSE(CreateClass, DefineAttribute)` mandates that if a class is created, at least one attribute must be defined for it at some point later. The binary templates partition their parameters into an *activation* and a *target*. For example, in `RESPONSE(CreateClass, DefineAttribute)`, the activation is `CreateClass`, while the target is `DefineAttribute`.
- **Negative templates:** These impose restrictions. For instance, `NOTRESPONSE(CreateClass, DefineOperation)` states that creating a class must not be immediately followed by defining an operation for that class.
- **Choice templates:** These specify that it is necessary to choose between several activities. For example, `CHOICE(CreateClass, DefineAttribute, DefineMethod)` indicates that after creating a class, the modeler must choose between defining an attribute or defining a method for that class, but not both simultaneously. The choice enforces flexibility in the design process, ensuring that only one of these actions is selected.

By combining textual and graphical representations, the Declare framework provides a flexible approach to analyzing the processes involved in class diagram modeling. The textual format aids in comprehending the high-level rules governing the modeling process, while the graphical Declare notation supports detailed analysis and conformance checking tasks, such as assessing whether a student's modeling process aligns with expected behavioral rules.

3.4 Conformance Checker

The Conformance Checker in UML Miner is specifically designed to detect and analyze discrepancies between the class modeling process followed by students and the reference process model defined by the teacher. This capability relies on the implementation of the Declare Analyzer method (Burattin et al., 2016), which enables thorough conformance checking. The analysis can be performed at two levels: trace-level conformance and constraint-level conformance.

The Declare Analyzer requires two key inputs:

- An event log: This log contains all the recorded modeling activities for a given project, as captured by UML Miner.
- A process model in Declare language: This model defines a set of constraints that describe the expected behavior of the modeling process.

The conformance checking algorithm iterates through all traces in the event log. For each trace, it evaluates each constraint by iterating through all events within the trace. Specific template-dependent operations are then invoked to calculate the number of violations (deviations from the model) and fulfillments (compliances with the model). The Declare Analyzer identifies both fulfilled and violated constraints, thereby pinpointing events that adhere to the model and those that deviate from it.

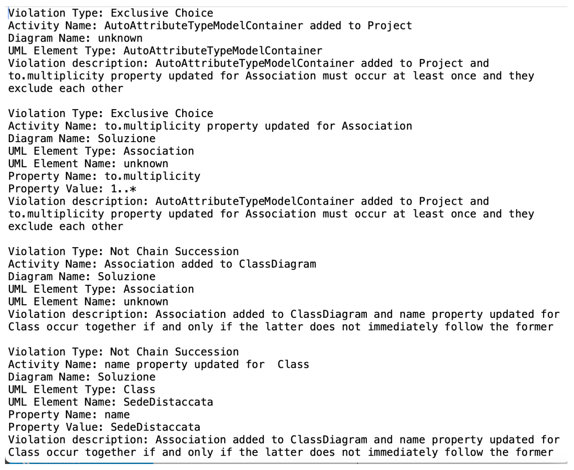
The conformance-checking results are presented at three granular levels:

- **Per event log:** Provides an overall assessment of how well the entire project aligns with the reference model.
- **Per trace:** Offers detailed insights into specific sequences of modeling events, identifying the exact deviations and fulfillments within individual sessions.
- **Per constraint:** Delivers a clear explanation of how each defined constraint has been upheld or violated, ensuring high explainability for individual process rules.

Figure 3 displays a segment of a conformance-checking report for student 1. The report lists constraint violations and fulfillments against the reference model, offering detailed insights into the trace, constraint, activities, and related properties. Below is an excerpt from the report:

Each row in the report corresponds to an evaluation result, providing the following key information:

- Trace ID: Unique identifier of the modeling session.



```
Violation Type: Exclusive Choice
Activity Name: AutoAttributeTypeModelContainer added to Project
Diagram Name: unknown
UML Element Type: AutoAttributeTypeModelContainer
Violation description: AutoAttributeTypeModelContainer added to Project and
to.multiplicity property updated for Association must occur at least once and they
exclude each other

Violation Type: Exclusive Choice
Activity Name: to.multiplicity property updated for Association
Diagram Name: Soluzione
UML Element Type: Association
UML Element Name: unknown
Property Name: to.multiplicity
Property Value: 1..*
Violation description: AutoAttributeTypeModelContainer added to Project and
to.multiplicity property updated for Association must occur at least once and they
exclude each other

Violation Type: Not Chain Succession
Activity Name: Association added to ClassDiagram
Diagram Name: Soluzione
UML Element Type: Association
UML Element Name: unknown
Violation description: Association added to ClassDiagram and name property updated for
Class occur together if and only if the latter does not immediately follow the former

Violation Type: Not Chain Succession
Activity Name: name property updated for  Class
Diagram Name: Soluzione
UML Element Type: Class
UML Element Name: SedeDistaccata
Property Name: name
Property Value: SedeDistaccata
Violation description: Association added to ClassDiagram and name property updated for
Class occur together if and only if the latter does not immediately follow the former
```

Figure 3: Excerpt from a conformance checking report for the creation of a class diagram.

- Constraint Name: Name of the MP-Declare constraint being checked.
- Activity Details: The specific activity or sequence of activities evaluated.
- Result Type: Indicates whether the activity fulfilled or violated the constraint.
- Diagram and Element Details: Information on the UML diagram and elements associated with the activity, such as diagram type, element name, and element type.
- Property Changes: Details of any property modifications, including property name and new value.

Using these detailed conformance-checking results, UML Miner provides actionable insights into the alignment of the modeling process with predefined constraints, facilitating the identification and resolution of inconsistencies.

3.5 Log Filtering

This component offers several types of basic filters, organized into five categories: attributes, performance, endpoints, followers, and timeframes. This functionality allows users to apply a combination of trace-based filters (which filter entire sequences of activities) and event-based filters (which target specific events within traces). In the context of UML modeling, this feature has been specifically adapted for class diagrams, enabling filtering based on parameters such as Diagram ID, Diagram Type, Diagram Name, UML Element ID, UML Element Type, and UML Element Name. This feature is particularly useful for teachers, as it enables them to focus on specific concepts or constructs explained during lectures. For example, it can isolate activities related to key class diagram components or concepts discussed in class. Additionally, it allows the removal of irrelevant information from the logs, retaining only the data deemed essential for a particular analysis. Instructors can also use it to focus on specific sessions, such as the first or the last modeling session. Events can be filtered by timestamp, type, and element to focus analysis.

4 EXAMPLE SCENARIO

To illustrate the use of *UML Miner*, consider the following modeling task assigned in an Object-Oriented Programming course: *Design a class diagram for a library management system.* The task includes core entities such as `Book`, `Author`, `Member`, and `Loan`, with structural constraints such as one-to-many associations (for example, an author can write multiple books) and attribute requirements (e.g., each `Book` must have a `Title` and an `ISBN`).

Step 1: Recording the Modeling Behavior

As students begin modeling the diagram in the VP environment, every modeling action is automatically captured by the **Log Recorder**. For example:

- `CreateClass: Book`
- `CreateClass: Author`
- `AddAssociation: Book ↔ Author`
- `SetAttribute: Book.title`

These activities are collected as event logs and grouped into *traces*, one for each modeling session.

Step 2: Structuring the Log

Each event is encoded in compliance with the **XES** standard, including:

- **Trace attributes:** session ID, project ID, student ID, and timestamp.
- **Event attributes:** activity name, element type and ID, diagram context, and any modified properties.

This standardized structure ensures that modeling sessions are portable and analyzable across multiple Process Mining tools.

Step 3: Discovering the Actual Process

The collected logs are analyzed by the **Process Discoverer**, which reconstructs the actual modeling process followed by the student using Declare templates. For example, the system might detect the following behavioral constraints:

- `EXISTENCE(CreateClass: Book)`
- `RESPONSE(CreateClass: Author, AddAssociation: Book ↔ Author)`
- `NOTRESPONSE(AddAssociation, SetNavigability)`

These constraints describe how actions relate to one another, exposing students' modeling strategies beyond the final diagram alone.

Step 4: Comparing with the Reference Process

The instructor provides a reference process model derived from expert solutions. The **Conformance Checker** then compares each student trace against the reference model and identifies violations. For instance, the following deviation was found:

[Class `Book` created], [Class `Author` created], [Association added between `Book` and `Author`], [Navigability property not set]

This violation highlights a missing activity in the expected modeling flow. Although the final diagram includes the association, the omission of the navigability definition reveals a gap in the student's modeling practice.

Step 5: Filtering for Focused Feedback (Log Filtering)

To support targeted feedback, instructors can use the **Log Filtering** functionality to isolate traces involving specific modeling concepts or behaviors, such as:

- All traces involving *aggregation relationships*.
- Traces where `attributes` were created or deleted.
- The first modeling session only.

This enables educators to analyze only relevant data and to tailor feedback based on specific constructs discussed in class.

This process-oriented example demonstrates how UML Miner captures and analyzes modeling behavior in depth. By making explicit connections between modeling actions and expected practices, the tool enhances instructional support and promotes reflective learning in UML modeling tasks. Additional resources, including software, example assignments, and guidelines for creating consistent class diagram tasks, are available at https://sites.google.com/view/uml-miner/.

5 CONCLUSIONS AND FUTURE WORK

This work extends our effort to provide tailored feedback for identifying gaps in students' modeling practices. The proposed tool detects sequences of activities in UML class diagram design that reveal incorrect use of inheritance, enabling educators to deliver targeted support, such as focused tutorials. Moreover, it supports real-time assessment, facilitating timely interventions during the modeling process. Future work includes validating the approach across different UML diagrams and improving RAG-LLM functionalities to provide natural language feedback. Integrating real-time feedback will also enable students to self-correct, promoting active learning. Longitudinal studies may help assess the long-term impact of these interventions on modeling skill development and teaching strategies.

REFERENCES

Alhazmi, S., Thevathayan, C., and Hamilton, M. (2021). Learning UML sequence diagrams with a new constructivist pedagogical tool: SD4ED. In Sherriff, M., Merkle, L. D., Cutter, P. A., Monge, A. E., and Sheard, J., editors, *SIGCSE '21: The 52nd ACM Technical Symposium on Computer Science Education, Virtual Event, USA, March 13-20, 2021*, pages 893–899. ACM.

Alman, A., Ciccio, C. D., Haas, D., Maggi, F. M., and Mendling, J. (2020a). Rule mining in action: The rum toolkit. In Ciccio, C. D., Depaire, B., Weerdt, J. D., Francescomarino, C. D., and Munoz-Gama, J., editors, *Proceedings of the ICPM Doctoral Consortium and Tool Demonstration Track 2020 co-located with the 2nd International Conference on Process Mining (ICPM 2020), Padua, Italy, October 4-9, 2020*, volume 2703 of *CEUR Workshop Proceedings*, pages 51–54. CEUR-WS.org.

Alman, A., Di Ciccio, C., Haas, D., Maggi, F. M., and Nolte, A. (2020b). Rule mining with rum. In *2020 2nd International Conference on Process Mining (ICPM)*, pages 121–128. IEEE.

Alman, A., Di Ciccio, C., Maggi, F. M., Montali, M., and van der Aa, H. (2021). Rum: Declarative process mining, distilled. In Polyvyanyy, A., Wynn, M. T., Van Looy, A., and Reichert, M., editors, *Business Process Management*, pages 23–29, Cham. Springer International Publishing.

Ardimento, P., Aversano, L., Bernardi, M. L., Carella, V. A., Cimitile, M., and Scalera, M. (2023). UML miner: A tool for mining UML diagrams. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2023 Companion, Västerås, Sweden, October 1-6, 2023*, pages 30–34, Västerås, Sweden. IEEE.

Ardimento, P., Bernardi, M. L., Cimitile, M., and Maggi, F. M. (2019a). Evaluating coding behavior in software development processes: A process mining approach. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 84–93.

Ardimento, P., Bernardi, M. L., Cimitile, M., and Ruvo, G. D. (2019b). Learning analytics to improve coding abilities: a fuzzy-based process mining approach. In *2019 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2019, New Orleans, LA, USA, June 23-26, 2019*, pages 1–7. IEEE.

Ardimento, P., Bernardi, M. L., Cimitile, M., and Ruvo, G. D. (2019c). Mining developer's behavior from web-based IDE logs. In Reddy, S., editor, *28th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2019, Naples, Italy, June 12-14, 2019*, pages 277–282. IEEE.

Burattin, A., Maggi, F. M., and Sperduti, A. (2016). Conformance checking based on multi-perspective declarative process models. *Expert Systems with Applications*, 65:194–211.

Burattin, A., Maggi, F. M., van der Aalst, W. M. P., and Sperduti, A. (2012). Techniques for a posteriori analysis of declarative processes. In Chi, C., Gasevic, D., and van den Heuvel, W., editors, *16th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2012, Beijing, China, September 10-14, 2012*, pages 41–50. IEEE Computer Society.

Crahen, E., Alphonce, C., and Ventura, P. (2002). Quickuml: A beginner's uml tool. In *Companion of the 17th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA '02, page 62–63, New York, NY, USA. Association for Computing Machinery.

Foss, S., Urazova, T., and Lawrence, R. (2022). Learning UML database design and modeling with autoer. In Kühn, T. and Sousa, V., editors, *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS 2022, Montreal, Quebec, Canada, October 23-28, 2022*, pages 42–45. ACM.

Kupferman, O. and Vardi, M. Y. (2003). Vacuity detection in temporal model checking. *Int. J. Softw. Tools Technol. Transf.*, 4(2):224–233.

Pesic, M. (2008). *Constraint-based workflow management systems : shifting control to users*. Phd thesis 1 (research tu/e / graduation tu/e), Industrial Engineering and Innovation Sciences. Proefschrift.

Pesic, M., Schonenberg, H., and van der Aalst, W. M. P. (2007). DECLARE: full support for loosely-structured processes. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA*, pages 287–300. IEEE Computer Society.

van der Aalst, W. M. P. (2016). *Process Mining - Data Science in Action, Second Edition*. Springer.