Journal of Theoretical and Applied Information Technology
31st August 2024. Vol.102. No. 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OBJECT-ASPECT ORIENTED MODELS TO PETRI NETS MODEL, AN APPROACH FOR THE TRANSFORMATION, ANALYSIS AND VERIFICATION OF SOFTWARE SYSTEMS

MOUNA AOUAG[1], NOUHAD MERABET[2], WIAM KENNOUCHE [3]

[1, 2, 3] Department of Computer Science, Abd elhafid Boussouf University Centre Mila, Algeria

E-mail:
[1]aouag.mouna@centre_univ_mila.dz,[2]nouhadmerabet58@gmail.com,[3]wiamkennouche@gmail.com

ABSTRACT

Object-Oriented Modeling (OOM) is a software design approach that structures and organizes code to accurately reflect reality, thereby facilitating its maintenance and evolution. However, it does have limitations in managing crosscutting concerns. Aspect-Oriented Modeling (AOM) provides solutions to these challenges, despite its lack of formal semantics. This underscores the importance of formal modeling, which does offer rigorous semantics.

In this paper, we propose an approach to transform a detailed object-oriented sequence diagram into a detailed aspect-oriented sequence diagram, based on graph transformation. Subsequently, we propose a method to transform the aspect-oriented diagram into a Petri net. Our work begins with a single meta-modeling for the first approach, using graph grammar rules to achieve an aspect-oriented model. Then, we apply the second approach to the result of the first one, using two meta-modeling and graph grammar rules, resulting in a Petri net. We use the AToMPM modeling tool. Finally, we perform a property analysis with the TINA tool.

Keywords: *Object-Oriented Modeling, Aspect-Oriented Modeling, Petri nets, AToMPM, TINA.*

1. INTRODUCTION

UML (Unified Modeling Language) is defined as a graphical and textual modeling language designed to understand and describe requirements, specify, and document systems. UML 2.0 consists of thirteen types of diagrams. Among them, the sequence diagram, a type of interaction diagram, illustrates the chronological sequence of messages exchanged between different objects interacting within a system [1]. Despite the numerous advantages of object-oriented languages, managing crosscutting concerns remains a challenge. The aspect-oriented paradigm emerges as a complementary solution, offering a more effective way to handle these concerns by separating them from the rest of the main logic.

We will transform from object-oriented to aspect-oriented UML models to achieve significant advantages in handling cross-cutting concerns. This transformation will reduce code duplication, enhance maintainability, and improve scalability.

Formal methods enable the verification of systems using mathematical notations and formal techniques. One of the most interesting formal methods is Petri nets. The transition from aspect-oriented modeling to formal methods aims to overcome the limitations of semi-formal approaches, such as ambiguity and verification difficulty. Once the transformation is completed, the resulting Petri net must be verified using a specialized tool such as TINA (Time Petri Net Analyzer). TINA is a software tool for the modeling and analysis of Petri nets, allowing for the formal verification of essential properties.

In this article, we propose two approaches and two tools. The first approach involves the automatic transformation of detailed object-oriented sequence diagrams into detailed aspect-oriented sequence diagrams, for which we have proposed a single meta-model and a graph grammar. The second approach transforms detailed aspect-oriented sequence diagrams into Petri nets, for which we have proposed two meta-models and a set of rules. The resulting Petri nets will be verified using the TINA verification tool.

The rest of the article is organized as follows. In Section 1 we start with a general introduction which gives a clear idea of the work. In Section 2, we discuss related works. In Section 3,

we present the basic concepts of our approaches. In Section 4, we describe our two approaches: the first transforms detailed object-oriented sequence diagrams into detailed aspect-oriented sequence diagrams, and the second transforms the latter into Petri nets. We also define the proposed meta-models and graph grammars, and conclude this section with a representation of the TINA tool and we make a comparison between works related to our work and discuss this comparison. In Section 5, we illustrate our approach using case studies. The final section concludes the article.

2. RELATED WORKS

Numerous research efforts have focused on object-oriented modeling and UML 2.0 diagrams, providing a deep understanding of fundamental software design concepts. Researchers have explored various approaches to represent the structures and behaviors of software systems using object-oriented models and UML 2.0 diagrams.

In [2], the authors update the formalization of sequence diagrams, extending causal semantics based on partial order theory to derive all possible valid traces for sequence diagrams with CF behavior modeling of distributed systems.

The study proposes a method for verifying the compliance of UML sequence diagrams. This approach aims to determine if the behavior described by a low-level model is contained within a high-level model, thus ensuring the conformity and coherence of UML sequence diagrams. Meanwhile, aspect-oriented modeling has emerged as a new paradigm in software engineering, offering techniques to manage crosscutting concerns and improve system modularity. Researchers have developed methods and tools to integrate aspects into object-oriented models, thereby enabling better separation of concerns and more effective management of crosscutting concerns [3].

In [4] the authors look at the feasibility and potential advantages of employing an aspect orientation approach in the software development lifecycle to ensure efficient integration of security. These notations are aimed at documenting and analyzing security in a software design model. It also proposes a model called the Aspect-Oriented Software Security Development Life Cycle (AOSSDLC), which covers arrange of security activities and deliverables for each development stage.

The paper [5] discusses the application of aspect-oriented UML use case diagrams and formal language AspectZ to part of a classic Aspect-Oriented Software Development (AOSD) case study, the Health-Watcher software system. In addition, this article proposes an extension of AspectZ to reach a new property for asymmetric AOSD which reacts after a schema successfully finishes, or not, showing messages for that situation, with an implicit join point; and a way for generalizing similar operations in a system using AspectZ.

In [6] the main idea is to emphasize the aspect-oriented approach as an effective means to improve the reliability and temporal performance of fault-tolerant systems. This research focuses on the probabilistic evaluation of critical fault-tolerant systems to enhance their reliability and availability from the design phase, emphasizing the importance of separating concerns, both functional (behavior) and non-functional (control), to improve response time. Efforts have been made to transform object-oriented models into aspect-oriented models, thus facilitating the transition to this innovative approach and providing more flexible and modular solutions for the development of complex software.

The thesis [7] addresses related works on aspect-oriented modeling, aiming to present an in-depth analysis of the advantages and disadvantages of object-oriented modeling, while introducing the concept of aspect-oriented modeling as a solution to the limitations of the object-oriented approach.

In [8], the authors propose a transformation model to convert object-oriented design patterns into aspect-oriented design patterns, offering a systematic approach to integrate the advantages of both programming paradigms. Thus, over time, formal models such as Petri nets are added, which contain the semantics and mathematics to facilitate understanding, provide semantics to these models, and also enable easier analysis and modeling.

In [9], the work presents the importance of Petri nets. It examines the structure of multicellular converters using control strategies to ensure better results and improve system performance and reliability.

The study [10] presents a Petri net model that uses mathematical and graphical modeling to determine in advance the structure of fault evolution. This model describes how a functional failure evolves from a fault-free state to a state with a specific level of failure. With the rapid development of computing, object-oriented and aspect-oriented models may encounter challenges in complex software and lack semantic clarity. Hence,

researchers propose transforming these models into formal models such as Petri nets to ease modeling and enhance semantic integration.

The work [11] demonstrates the transformation of UML 2.0 diagrams into Petri nets, as the latter serve as a powerful tool for analysis and verification. Similarly to UML, extensions have been suggested to utilize them as a modeling language for mobile agents.

The thesis [12] explores the significance of formal models in addressing UML deficiencies. This allows the conversion of stereotyped models into NestedNet Petri nets to conduct formal verifications of these models. After transforming into formal models, researchers perform the analysis of the properties of formal methods, such as the verification of Petri nets, to ensure that the priorities are aligned with the system requirements to guarantee their proper functioning.

In this article [13], the researchers present another way to transform State Machine Diagrams (UML SMD) into Colored Petri nets models and to prove certain structural properties within this transformation itself

The study [14] presents an analysis of the results obtained after transforming the graphs to perform a verification of the target model's acceptance in the company's complex and virtual systems, in order to avoid errors and operational issues.

In this work [15], timed Petri nets are applied to model the temporal behavior of workflow systems, using TINA as a tool to support the verification of activity deadlines.

In our work, we propose a new transformation approach that integrates multiple methodologies to address complex software systems. Our approach combines the transformation from object-oriented (OO) paradigms to aspect-oriented (AO) paradigms and from aspect-oriented paradigms to Petri nets. This integration aims to resolve the limitations of object-oriented modeling. (Duplication, enhances maintainability, and improves scalability) and to ensure software properties and facilitate system comprehension, given that Petri nets provide a broad and commonly understood formalism.
Aspect-oriented Modeling (AOM) serves as an intermediary, offering solutions for modeling object-oriented systems, particularly in complex scenarios. By leveraging aspect-oriented modeling (AOM), we can modularize cross-cutting concerns more effectively. Our transformation approach enhances this by converting aspect-oriented models into Petri nets, which provide a clear and formal method for system analysis and verification
Finally, we employ the TINA tool for verification to ensure the correctness of the system properties, including those related to aspect-oriented designs. This tool helps validate the properties of the woven net, which integrates the various concerns and modules identified during the transformation process.

This approach not only addresses the limitations of previous methodologies but also contributes to a more comprehensive, understanding and verification of complex systems

3. BACKGROUNDS

Centered In this section we present the basic concepts of aspect oriented modeling, Petri nets, graph transformation and analysis Petri nets.

3.1 Aspect-Oriented Modeling (AOM)

Aspect-Oriented Modeling (AOM) is an approach (among others) that aims to achieve this goal. In other words, the process of weaving (composition) aspects is divided into two phases. First, a detection phase identifies specific parts of a base model, followed by a composition phase to construct the model taking into account the aspect [7]. Aspect-oriented modeling is based on the following points:
Concern: A concern is an interest related to the development of a system, its operation, or any other issues that are essential or important to one of the participants in the system [16].
There are two types of concerns: cross-cutting concerns or aspect concerns, and non-cross-cutting concerns or base concerns [7] **[24] [25].**
Base Concern: It therefore represents a non-cross-cutting (functional) concern that can be effectively captured with traditional approaches such as Object-Oriented approach.
Aspect Concern: It therefore represents a cross-cutting (non-functional) concern that can be effectively captured with the Aspect-Oriented approach.
Join point (PJ): The location in the model where advices should be inserted [7].
Pointcut (PC): It describes the set of join points where the aspect should intervene and be introduced [16].
Advice: Represents a particular technical behavior of an aspect. In concrete terms, it is a code block that will be grafted into the application at the execution of a join point defined in a point cut [17].

Aspect: A module defining Advices and their points cuts [7].
Weaver: The aspect weaver is an operation that takes as input base modules and aspect modules, and aims to apply and attach aspects to the base modules at specific join points corresponding to the aspect's cut specification [16].
So, an Aspect = Point cut + Advice.
Point cut = Σ join points [7].

3.2 Petri Nets

The Petri net is a graphical and mathematical tool used to model and analyze discrete systems, such as those that operate concurrently, in parallel, or non-deterministically. As a graphical tool, it helps to understand and simulate dynamic and concurrent activities. As a mathematical tool, it allows for the analysis of the modeled system using graphs and algebraic equations. [18]
A Petri net is a four-tuple (P, T, IN, OUT) where
P = {P1, P2, ……, Pm} is a set of places.
T = {t1, t2, …….., tm} is a set of transitions.
P∪T ≠ ∅, P ∩ T= ∅.
IN: (P x T) ->N is an input function that defines directed arcs from places to transitions.
OUT: (P x T) –> N is an output function that defines directed arcs from transitions to places.
Pictorially, places are represented by circles and transitions by horizontal bars.
If IN (Pi, Tj) = k, where k > 1 is an integer, a directed arc from place Pi, to transition Tj is drawn with label k [**26] [27].**
If IN (Pi, Tj) = 0, no arc is drawn from Pi, to Tj. Similarly, if OUT (Pi, Tj) = k, a directed arc is included from transition Tj to place Pi, with label k
If k > 1 and without label if k = 1. If k = 0, no arc is included from TJ to Pi. [19], [20].

3.3 Graph Transformation

Graph transformation is the mechanism for specifying and applying transformations between graphs. The main idea behind this transformation is the modification of graphs based on rules.
A graph grammar evolves from Chomsky grammar on strings to graphs. It consists of a set of graph-rewriting rules. Each one has a graph at its Left Hand Side (LHS) and another graph at its Right Hand Side (RHS). [21]

3.4 Analysis Petri nets

As systems become more complex, ensuring the quality of the model becomes more challenging. To address this, various techniques are used for analysis, such as verification, validation, qualification, and certification. In our work, we primarily focus on verification and validation: [14] [22] **[28].**

Verification: answers the question "Are we building the model correctly?" Verification encompasses review, inspection, testing, automated proof, or other appropriate techniques to establish and document the compliance of development artifacts with predefined criteria. ISO 8402 defines verification as "confirmation through examination and provision of tangible evidence (information whose accuracy can be demonstrated, based on facts obtained through observation, measurement, testing, or other means) that specified requirements have been fulfilled."

Validation: involves assessing the suitability of the developed system in relation to the needs expressed by its future users. Validation seeks to answer the question "Are we building the right model?" ("is the right system being built?"). By definition, validation is "confirmation through examination and provision of tangible evidence that specific requirements for an intended specific use are satisfied. Multiple validations can be performed if there are different intended uses" (ISO 8402).

4. PROPOSED APPROACHES

In our approach, we performed a transformation from the detailed object-oriented sequence diagram to an aspect-oriented detailed sequence diagram. Subsequently, the result of the first approach was transformed into Petri nets.

4.1 Transformation of detailed object-oriented sequence diagrams into detailed aspect-oriented sequence diagrams

To translate a detailed object-oriented sequence diagram into a detailed aspect-oriented sequence diagram, we propose a single meta-model, and a graph grammar.

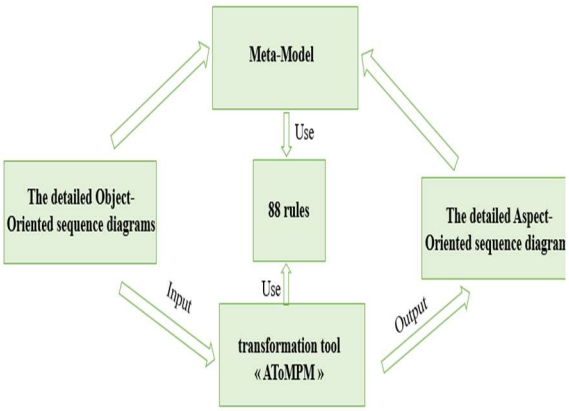
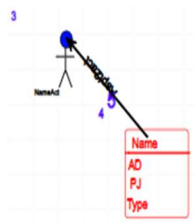
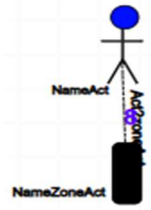
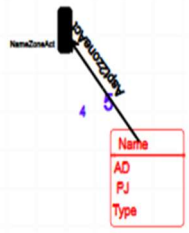
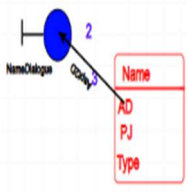
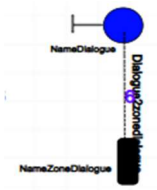
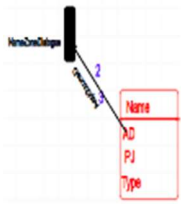
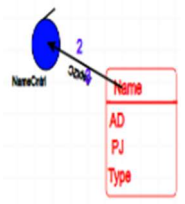
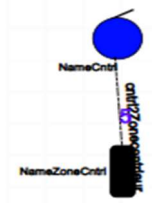
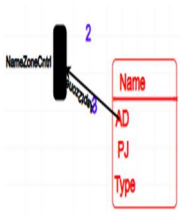
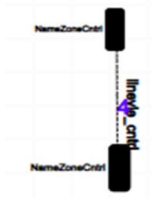
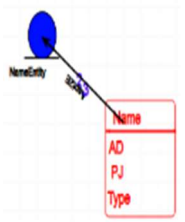
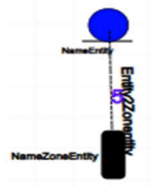
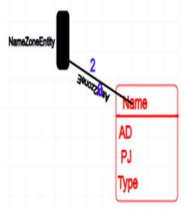
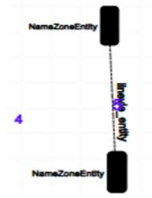


Figure 1: From Detailed Object-Oriented Sequence Diagrams to Detailed Aspect-Oriented Sequence Diagrams.

4.1.1 Model Transformation Process:

In this approach, the transformation is based on a set of transformation rules. These rules express the semantics in the following table:

Table 1: the semantics of approach 1

Source state and notation:	Description	Target state and notation
 **Actor**	We have : AD = ZoneAct. PJ = Actor. Type = Create.	 **Actor + Actor zone**
 **Actor activity zone**	We have : AD = ZoneAct. PJ = Actor activity zone. Type = Create.	 **2Actor activity zones**
 **Boundary**	We have : AD = ZoneDialog. PJ = Boundary. Type = Create.	 **Boundary + Boundary zone**
 **Boundary activity zone**	We have : AD = ZoneDialog. PJ =Boundary activity zone. Type = Create.	 **2 Boundary activity zones**
 **Control**	We have : AD = ZoneCntrl. PJ = Control. Type = Create.	 **Control + Control zone**
 **Control activity zone**	We have : AD = ZoneCntrl. PJ =Control activity zone. Type = Create.	 **2 Control activity zones**
 **Entity**	We have : AD = ZoneEntity. PJ =Entity. Type = Create.	 **Entity + Entity zone**
 **Entity activity zone**	We have : AD = ZoneEntity. PJ =Entity activity zone. Type = Create.	 **2 Entity activity zones**

4.1.2 The Meta-model of the detailed Sequence Diagram and Graphical Representation of Classes

The meta-model of detailed sequence diagrams: The meta-model for object-oriented sequence diagram consists of seventeen classes:(Seq, MA, Actor, Dialogue, Controleur, Entity, ZoneAct, ZoneDialogue, ZoneCntrl, ZoneEntity,Destroy,

Aspect, Operation, Ref, Opt, Alt, Loop) and forty-three associations Our meta-model for the UML 2.0 sequence diagram (see Figure 2) consists of the following classes:

Seq: This class represents the aspect-oriented detailed sequence diagram. It contains an attribute <Diagram> of type String displaying by default "detailed_sequence_diagram" and an attribute <Model> of type String displaying by default "basic_model" before the transformation. After the transformation, it is modified to "Weaver". Graphically, it is represented by a large blue rectangle.

MA: This class represents the aspect model. It contains an attribute <MA> of type String displaying by default "Aspect_Model". Graphically, it is represented by a large red rectangle.

Actor: This class represents the actor. It contains an attribute <NameAct> of type string. Graphically, it is represented by a simple shape such as a blue stick figure.

ZoneAct: This class represents the actor's activity zone. It contains an attribute <NomZoneAct> of type string. Graphically, it is represented by a small black rectangle.

Dialogue: This class represents the boundary. It contains an attribute <NameDialogue> of type string. Graphically, it is represented by a blue circle with two horizontal and vertical lines.

ZoneDialogue: This class represents the boundary activity zone. It contains an attribute <NameDialogueZone> of type string. Graphically, it is represented by a small black rectangle.

Contrôleur: This class represents the Controller. It contains an attribute <NameCntrl> of type string. Graphically, it is represented by a blue circle with two diagonal lines inside and outside the circle.

ZoneCntrl:This class represents the controller activity zone. It contains an attribute <NameCntrlZone> of type string. Graphically, it is represented by a small black rectangle.

Entity: This class represents the entity. It contains an attribute <NameEntity> of type string. Graphically, it is represented by a blue circle with a line at the bottom.

ZoneEntity: This class represents the entity activity zone. It contains an attribute <NameEntityZone> of type string. Graphically, it is represented by a small black rectangle.

Destroy: This class represents the destruction. It contains an attribute <D> of type string. Graphically, it is represented by a black cross.

Aspect: This class represents the aspect. It contains attributes <AD>, <PJ>, <Type>, and <Name>, all of type string. Graphically, it is represented by a class.

Operation: This class represents an operation. It inherits from the <Actor> class and contains an attribute <idop> of type string. Graphically, it is represented by a large black rectangle.

Ref: This class represents the reference operation. It inherits from the <Operation> class and contains attributes <Nameref> and <ref>, both of type string. Graphically, it is represented by a small black rectangle.

Opt: This class represents the optional operation and inherits from the < Operation> class. It contains an attribute <opt> of type string. Graphically, it is represented by a small black rectangle.

Loop: This class represents the loop operation and inherits from the < Operation> class. It contains attributes <N> and <Loop>, both of type string. Graphically, it is represented by a small black rectangle.

Alt: This class represents the alternative operation and inherits from the < Operation> class. It contains an attribute <Alt> of type string. Graphically, it is represented by a small black rectangle.

The graphical representation of the classes in the detailed Sequence Diagram meta-model:

Journal of Theoretical and Applied Information Technology
31st August 2024. Vol.102. No. 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

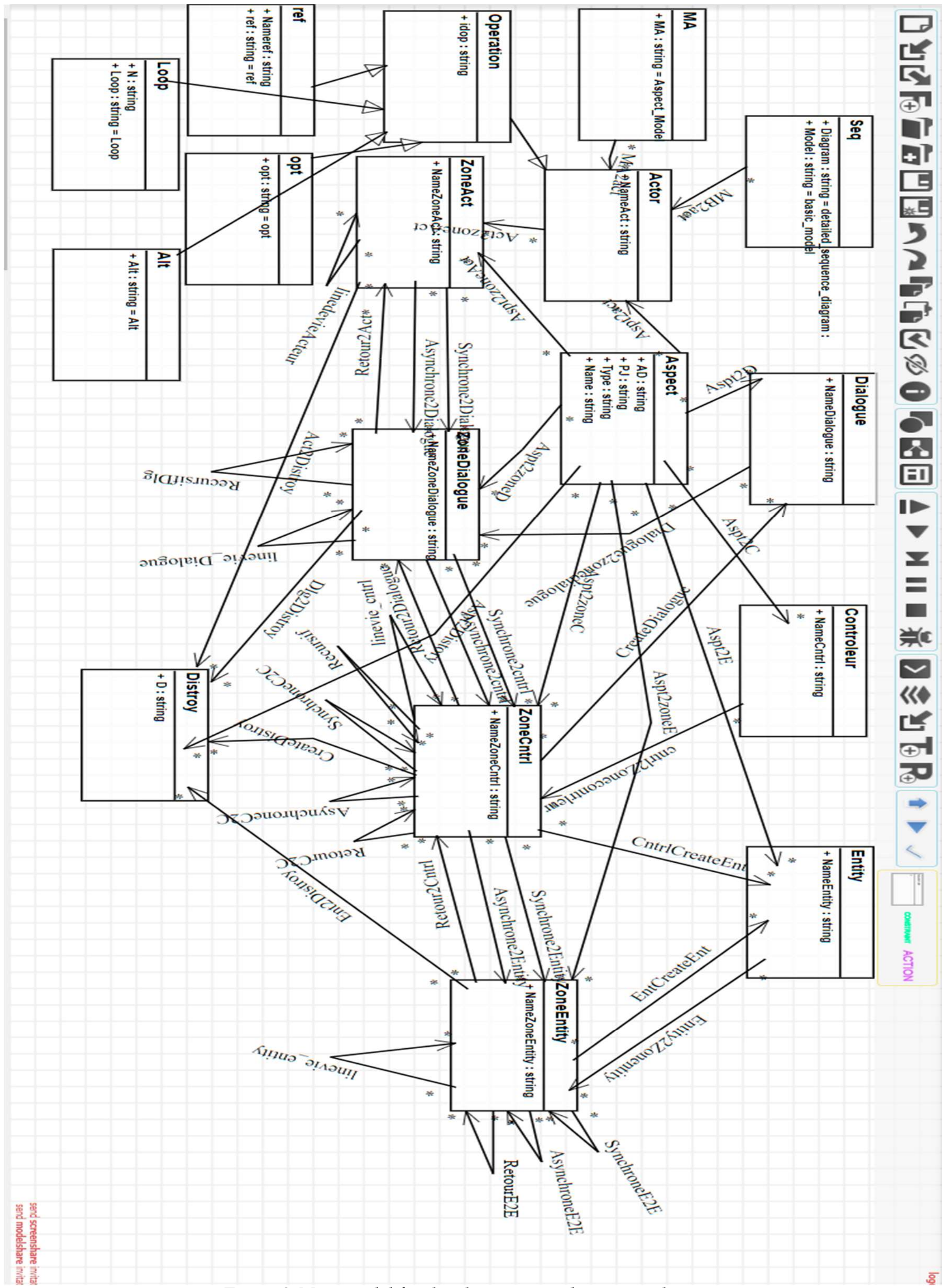


Figure 2:Meta-model for the object-oriented sequence diagram.

Journal of Theoretical and Applied Information Technology
31st August 2024. Vol.102. No. 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

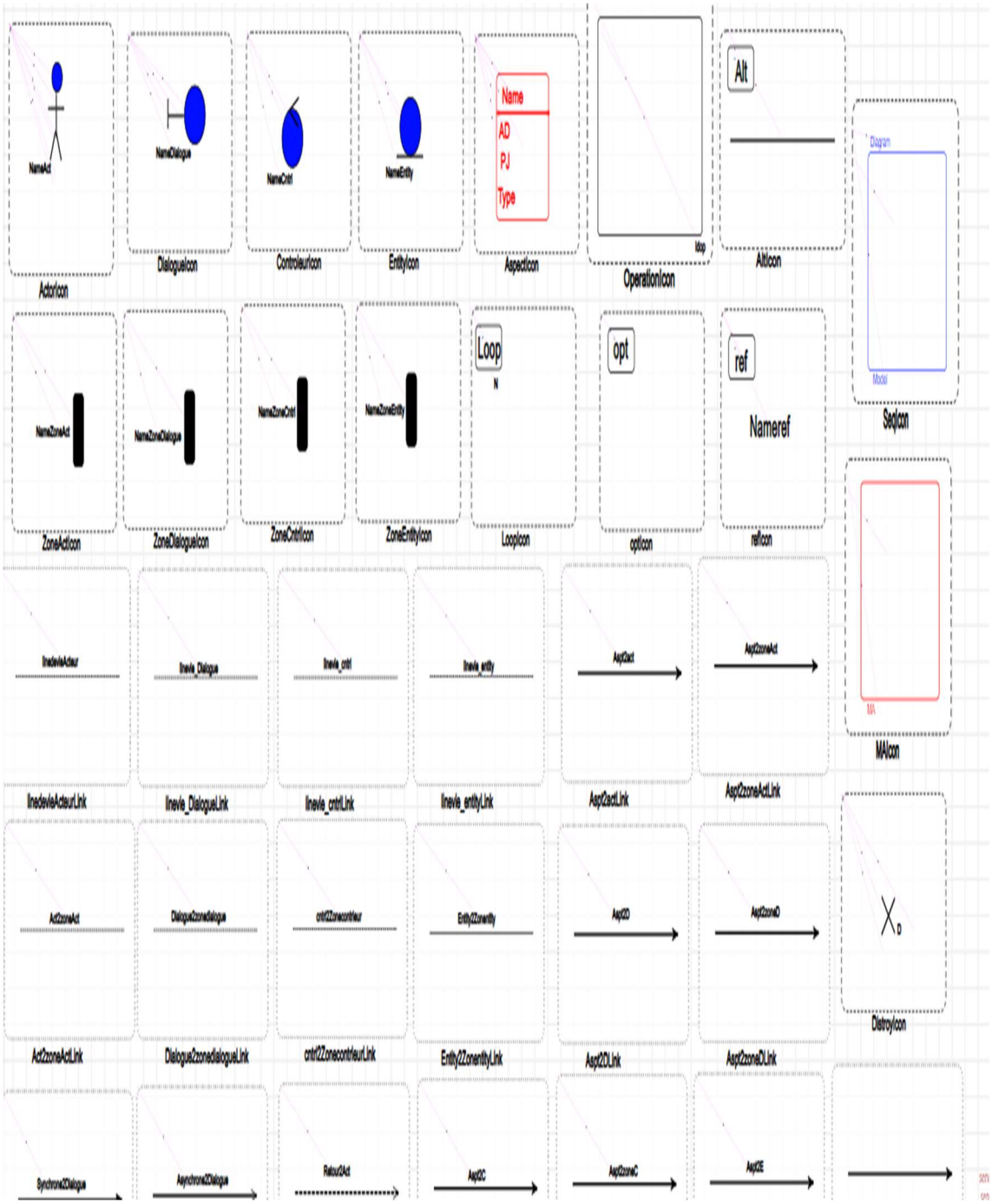


Figure 3: Graphical representation of the meta-model of the sequence diagram

Figure 4 illustrates the tool generated for manipulating detailed aspect-oriented sequence diagrams.

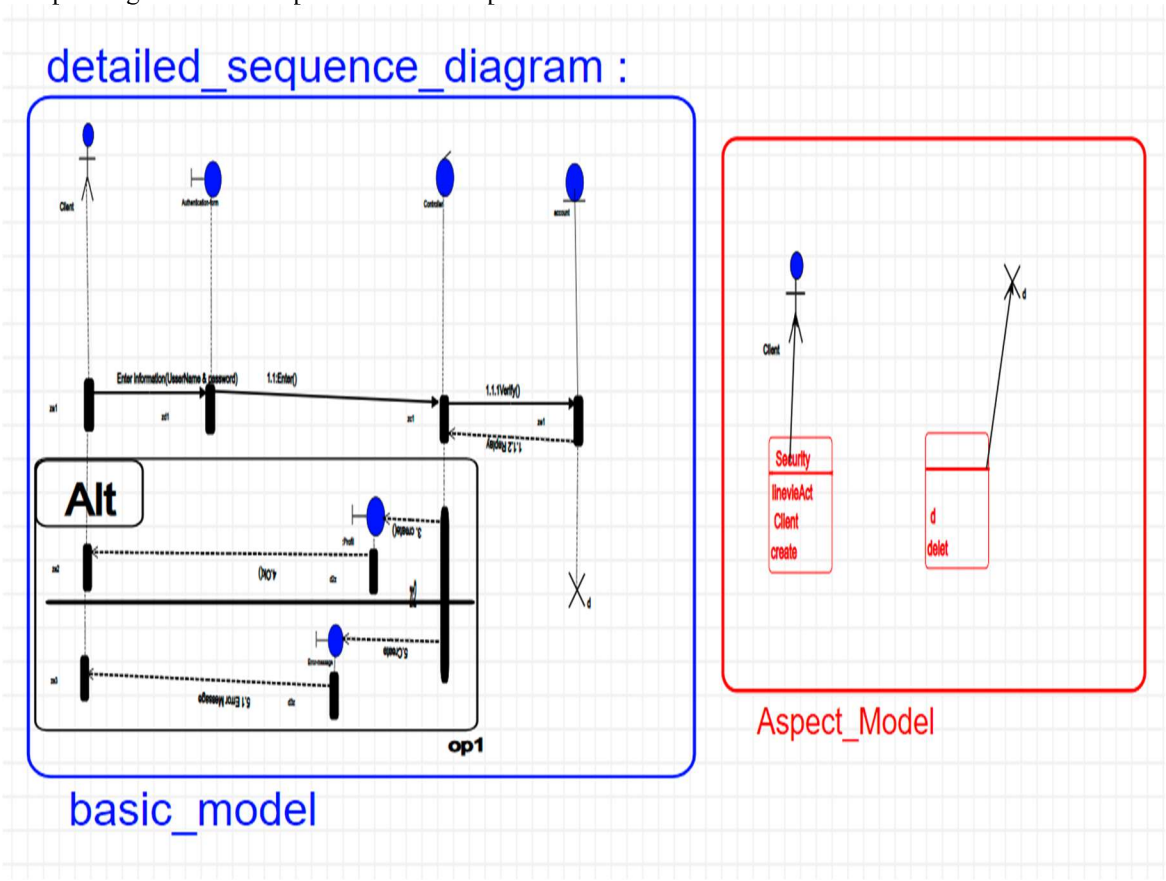


Figure 4: The generated tool for modeling the sequence diagram (AToMPM).

4.1.3 Our proposed graph grammar for transforming object-oriented detailed sequence diagrams into aspect-oriented detailed sequence diagrams

The main idea for transforming sequence diagrams into aspect-oriented sequence diagrams is to add aspects at the selected join points identified by the points cut found in the aspect model.

In this section, we present a graph grammar containing eighty-eight (88) rules, each expressing a particular case. These rules will be executed in an order determined by the motif (T). We decompose the 88 rules into three categories based on the type of aspect, which are defined as follows:

Aspect.type: the action of the aspect, whether it is creation, deletion, or linking between two components.
In the following figure, the pattern (T_O2A) of this graph grammar for this approach 1:

Journal of Theoretical and Applied Information Technology
31st August 2024. Vol.102. No. 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

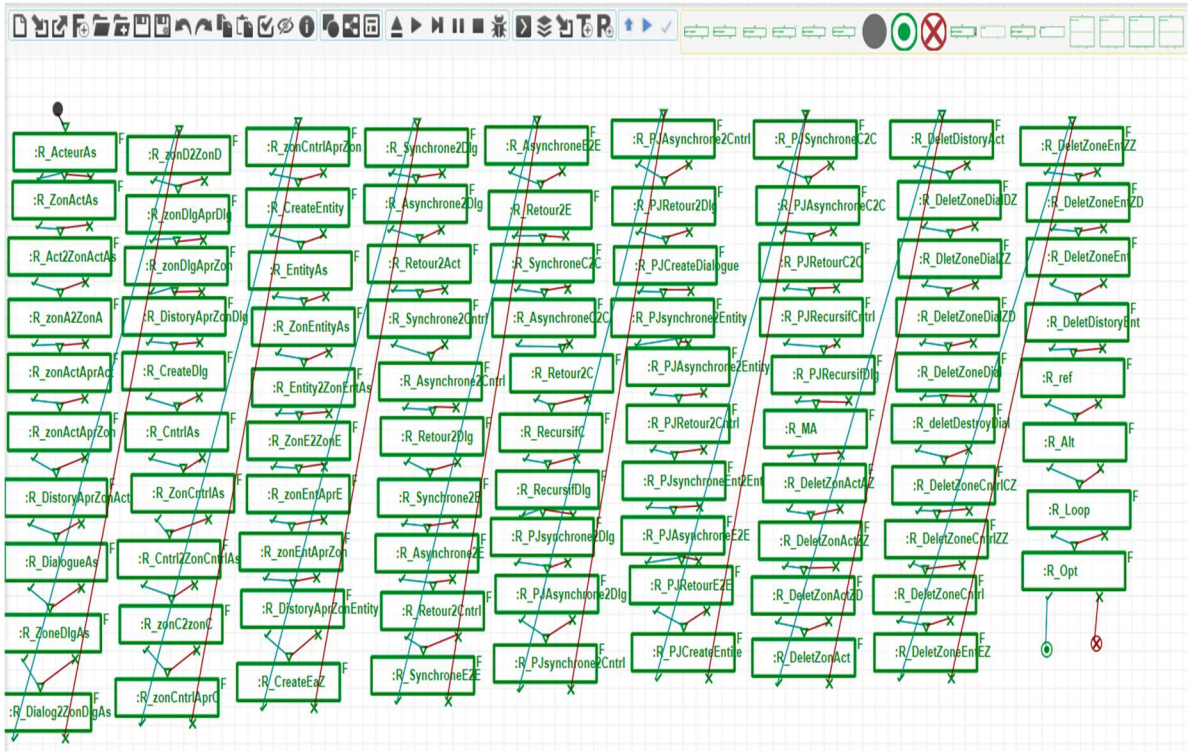


Figure 5: The motif of transformation of approach

Category 1: Rules that have the aspect type "create" are applied to create an aspect according to the PJ (the join point) and AD (advice). In the following figures, we depict the rules of the first category:

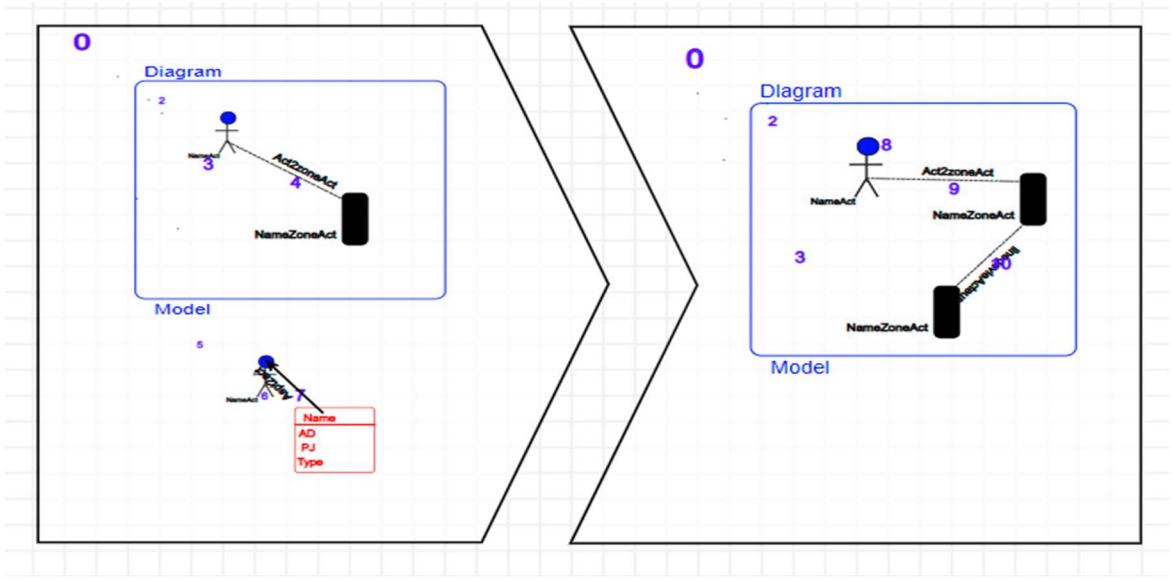


Figure 6: Application of category 1 with Aspect.PJ == Actor and Aspect.Ad== linevieAct.

In Figure 6, for the application of the first category, in this rule we positioned the aspect on the actor (the join point) and added an activity zone (advice) with the aspect type set to 'create'.

In Figures 7 and 8, the Python code for the first creation rule:

Journal of Theoretical and Applied Information Technology
31st August 2024. Vol.102. No. 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

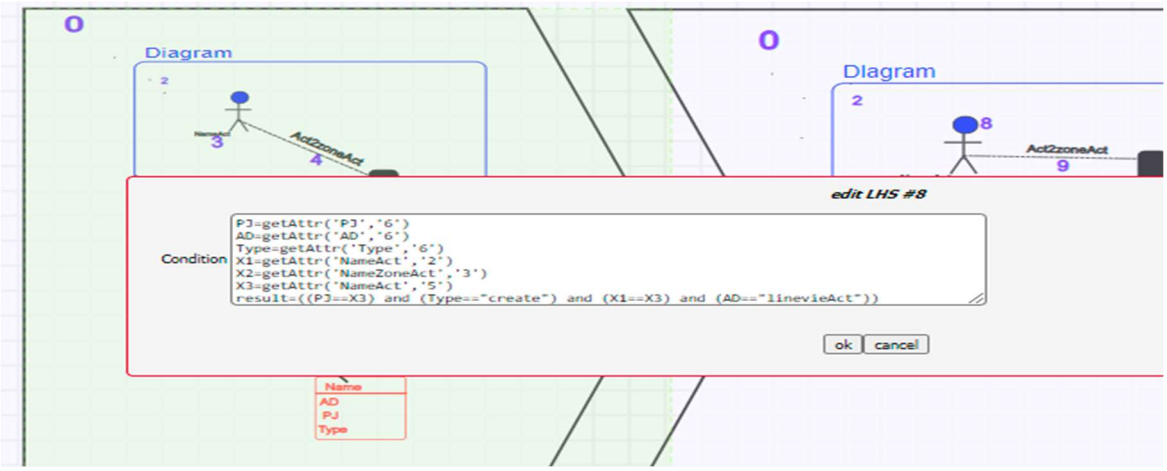


Figure 7: The LHS of the first create rule (Python code).

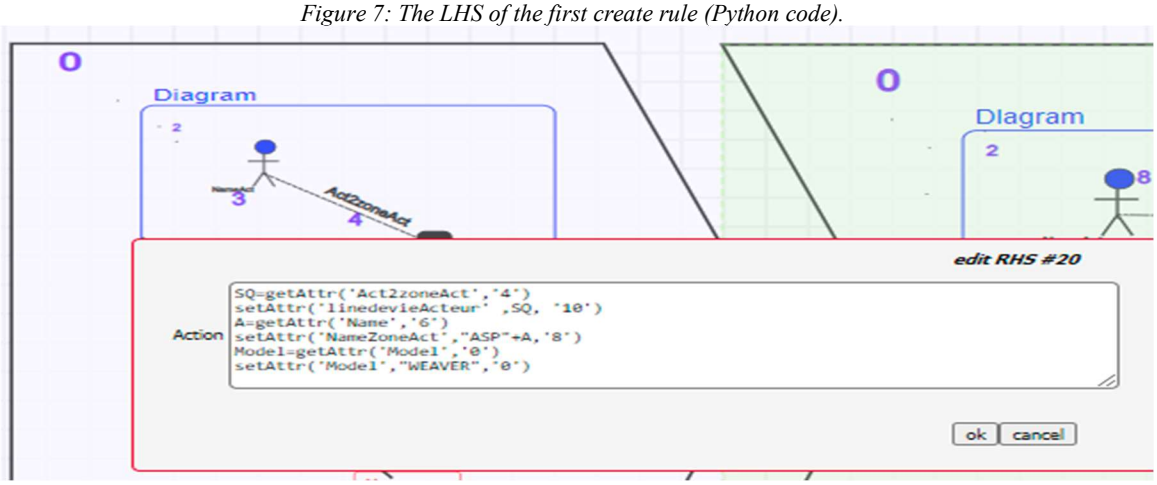


Figure 8: The RHS of the first create rule (Python code).

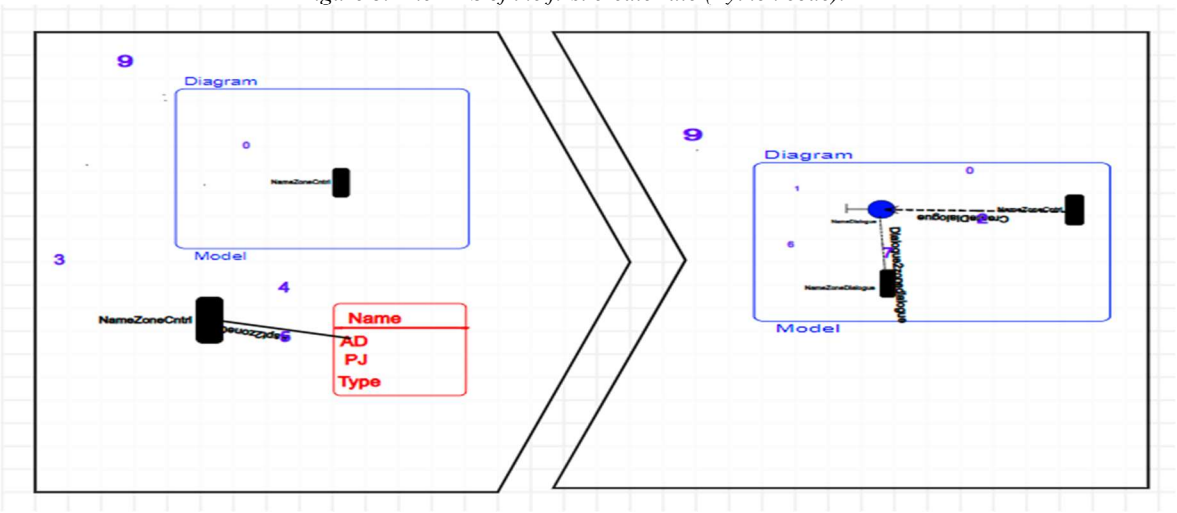


Figure 9: Application of category 1 with Aspect.PJ== Control Zone and Aspect.AD== CreateDlg.

In Figure 9, for the application of the first category, in this rule we positioned the aspect on the Controller zone (the join point) and added boundary (advice) with the aspect type set to 'create'.

Category 2: Rules with the aspect type "lien" are applied to add an aspect to a link between two objects.

In the following figure, we depict the rules of the second category:

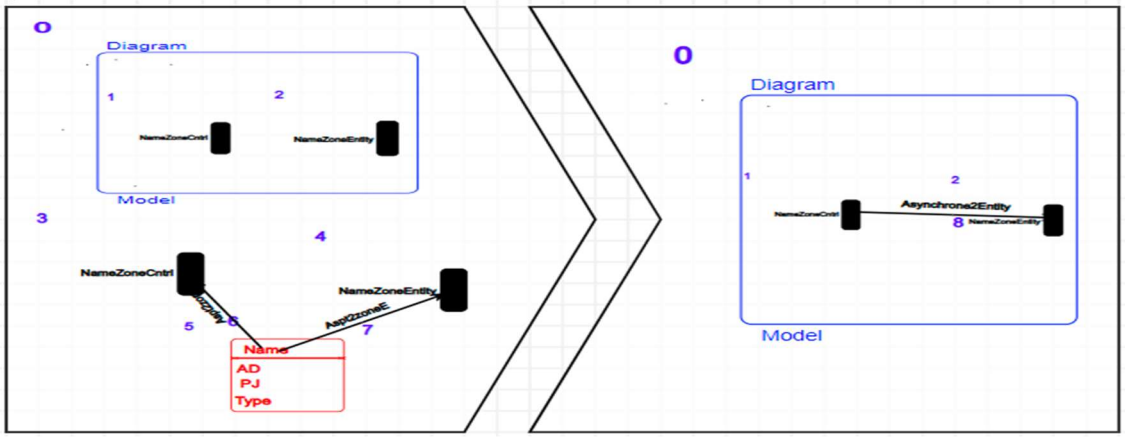


Figure 10: The Application of category 2 with Aspect.PJ== Control Zone, Entity Zone and Aspect.AD== Asynchrone2E

In Figure 10, for the application of the second category, in this rule we positioned the aspect on the Controller zone and entity zone (the join points) and added asynchronous message (advice) with the aspect type set to 'lien'

In Figures 11 and 12, the Python code for the first link rule:

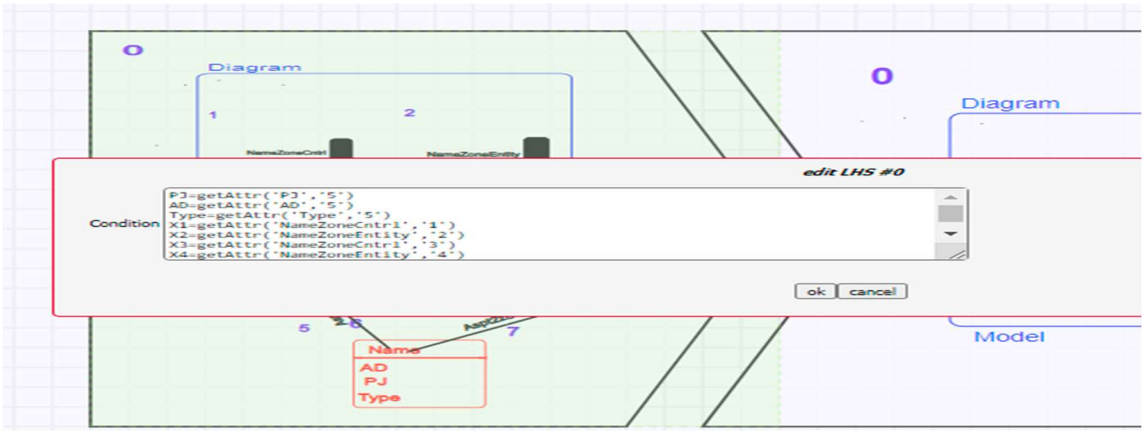


Figure 11: The LHS of the first link rule (Python code).

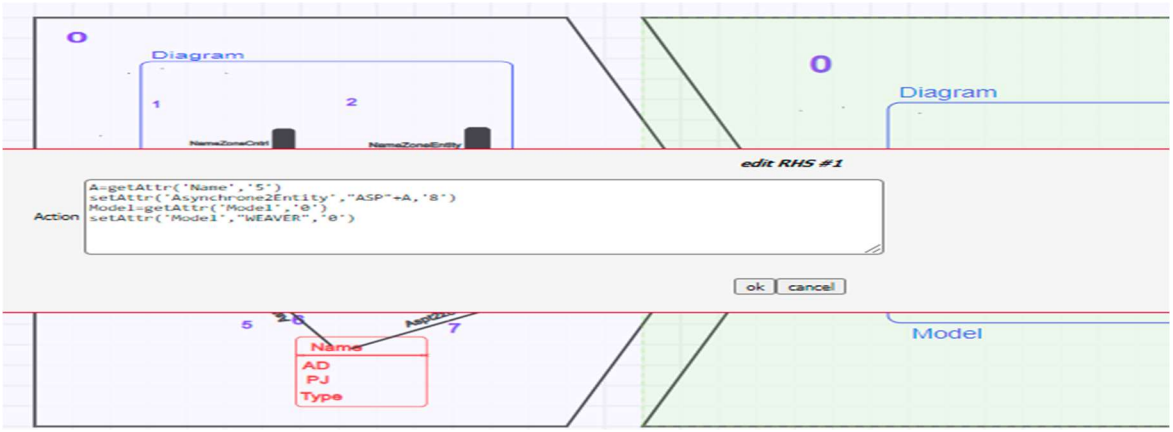


Figure 12: The RHS of the first Link rule (Python code)

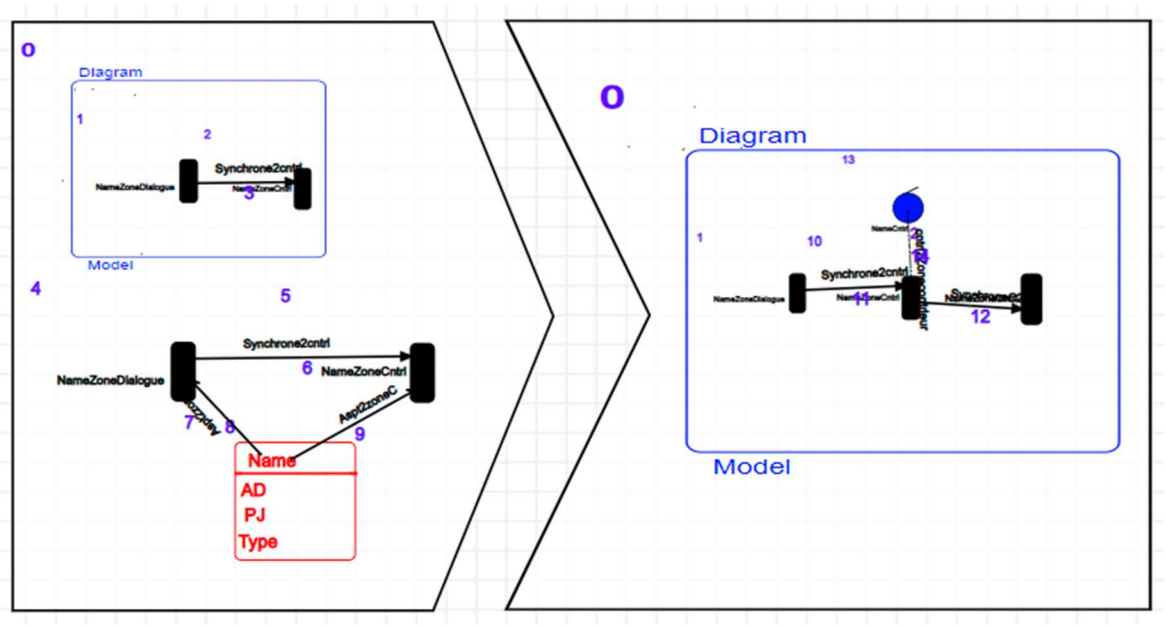


Figure 13: Application of category 2 with Aspect.PJ== Synchrone message and Aspect.AD== Cntrl

.In Figure 13, for the application of the second category, in this rule we positioned the aspect on the boundary zone and controller zone (the join points) and added a controller (advice) with the aspect type set to 'lien'.

Category 3: Rules with the aspect type "Delete" are applied respectively to delete an object automatically delete the relations associated with it. In the following figures, we depict the rules of the third category:

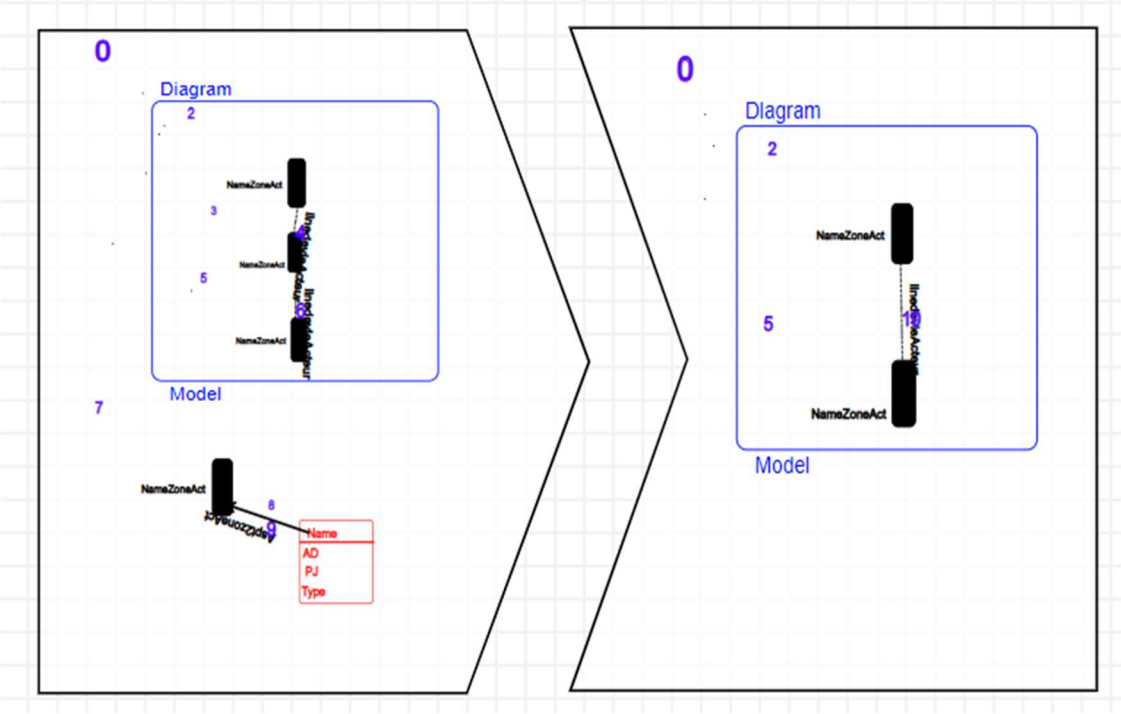


Figure 14: Application of category 3 with Aspect.PJ== Zone Actor and Aspect.type== delete

In Figure 14, for the application of the third category, in this rule we positioned the aspect on the actor zone (the join point) with the aspect type set to 'delet'.

In Figures 15 and16, the Python code for the first Delet rule:

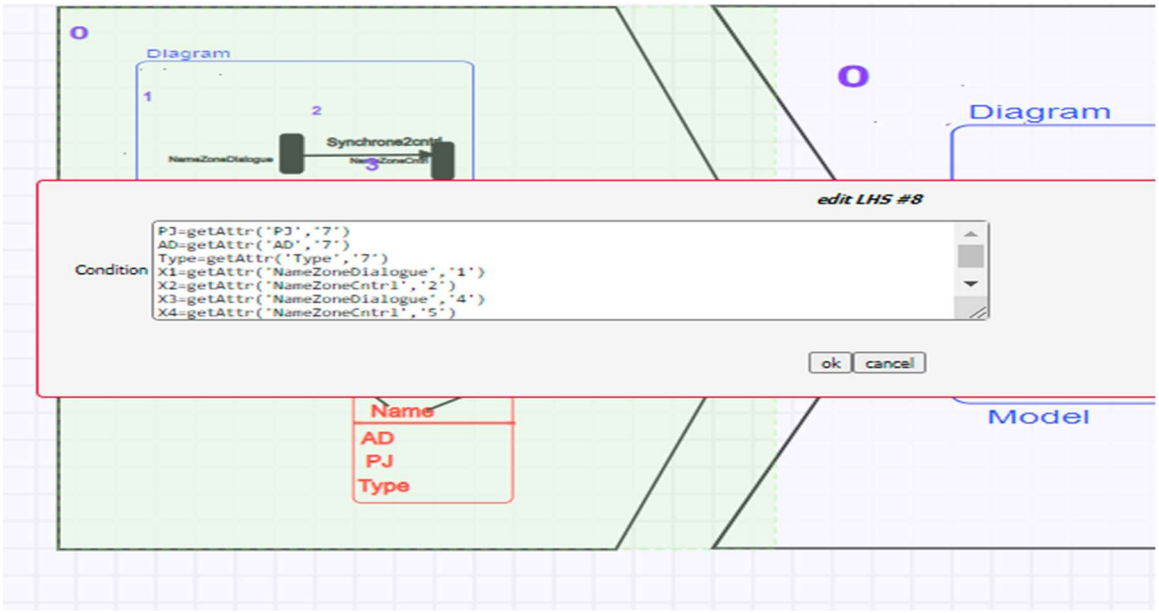


Figure 15: The LHS of the first delete rule (Python code).

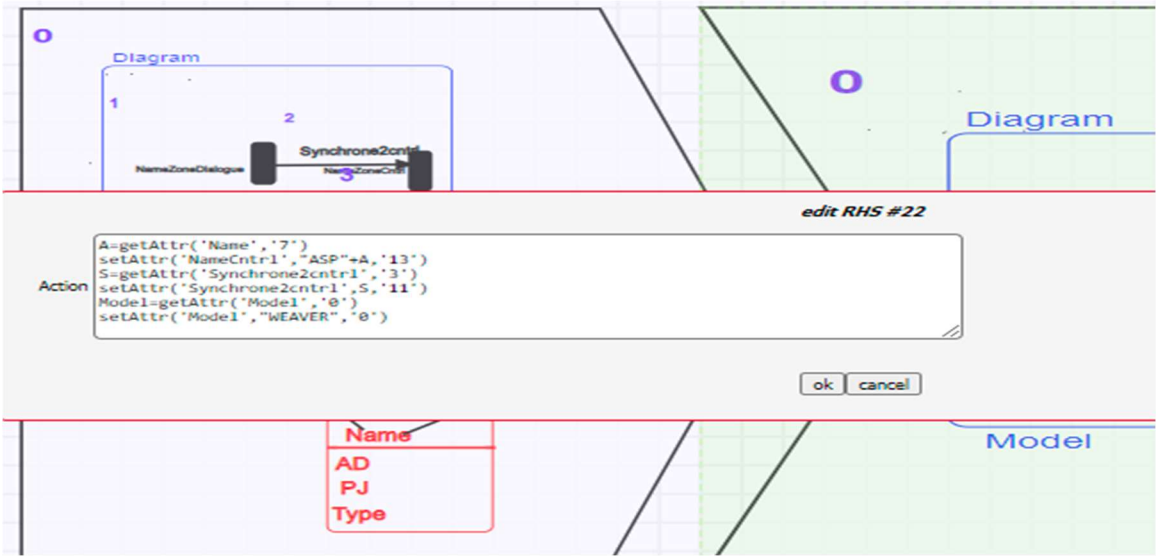


Figure 16: The RHS of the first delete rule (Python code).

In Figure 17, for the application of the third category, in this rule we positioned the aspect on destroy (the join point) with the aspect type set to 'delete'.

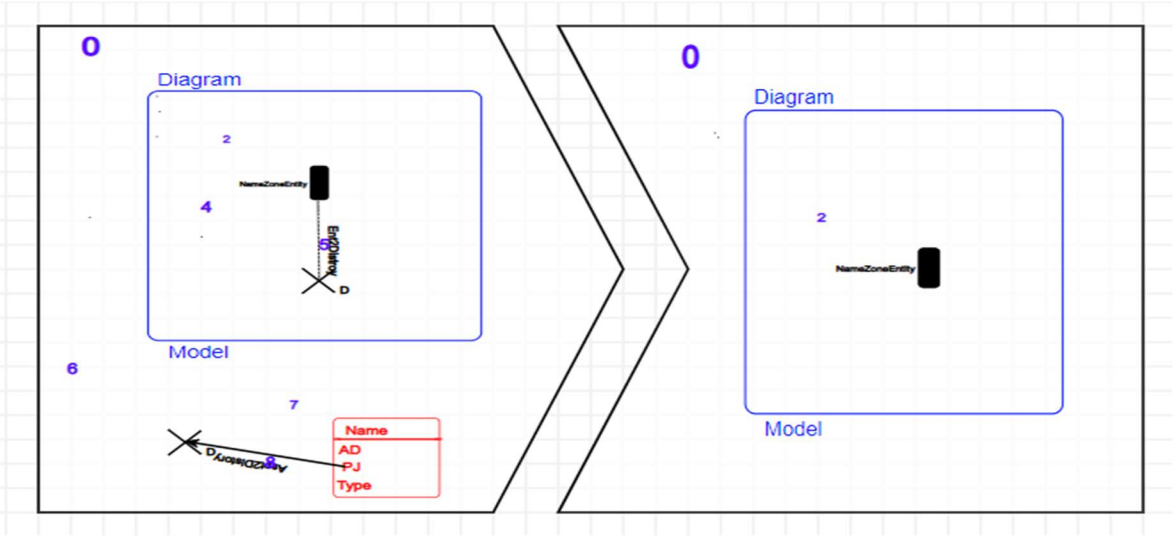


Figure 17: Application of category 3 with Aspect.PJ== Destroy and Aspect.type== delet

4.2 Transformation of Aspect-Oriented Detailed Sequence Diagrams into Petri Nets

In the second approach, we take the result of the first transformation, which is an aspect oriented detailed sequence diagram, as the input. Using the proposed graph grammar, we obtain a Petri net.

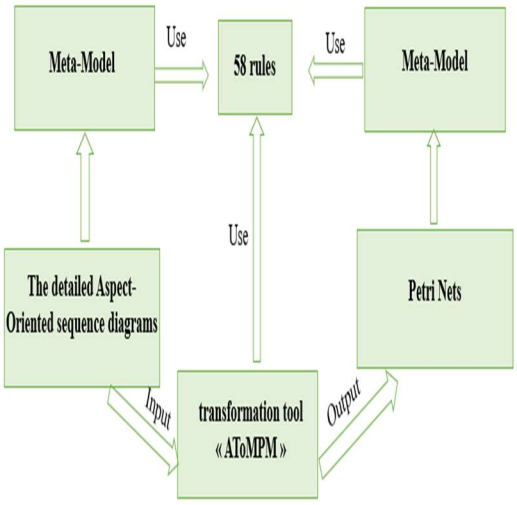


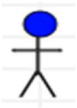
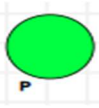
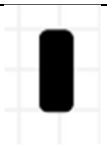
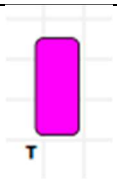
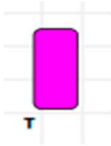
Figure 18: From Aspect-Oriented Sequence Diagram to Petri net.

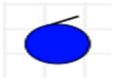
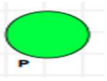
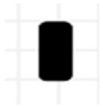
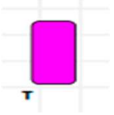
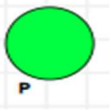
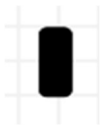
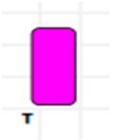
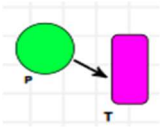
To translate a detailed aspect-oriented sequence diagram into a Petri net, we propose two meta-models, and a graph grammar.

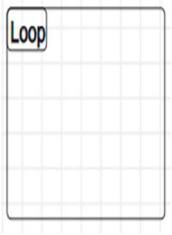
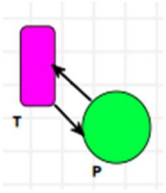
4.2.1 Model Transformation Process:

In this approach, the transformation relies on transformation rules. These rules express the semantics in the following table

Table 2: the semantics of this approach 2

Source state and notation:	Description	Target state and notation
 **Actor**	That transforms the actor into a place because there is no change in state.	 **Place**
 **Actor activity zone**	That transforms the actor's activity zone into a transition because it performs a state change.	 **Transition**
 **Boundary**	That transforms the boundary into a place because there is no change in state.	 **Place**
 **Boundary activity zone**	That transforms the boundary's activity zone into a transition because it performs a state change.	 **Transition**

 **Control**	That transforms the control into a place because there is no change in state.	 **Place**
 **Control activity zone**	That transforms the control's activity zone into a transition because it performs a state change.	 **Transition**
 **Entity**	That transforms the entity into a place because there is no change in state.	 **Place**
 **Entity activity zone**	That transforms the entity's activity zone into a transition because it performs a state change.	 **Transition**
 **Destroy**	That transforms the destroy into a place and a transition because it is a final state.	 **Place and transition**
 **diagram framework**	That transforms the diagram framework into a Petri net framework	 **Petri-Net framework**

 **Loop operation**	That transforms the Loop operation into a place and a transition as it's performed in a loop.	 **Place and transition**

4.2.2 The Meta-model of the Petri net and Graphical Representation of Classes

The meta-modeling of Petri Nets consists of three classes and three associations. In Figure 19, we present the meta-model for Petri Nets.

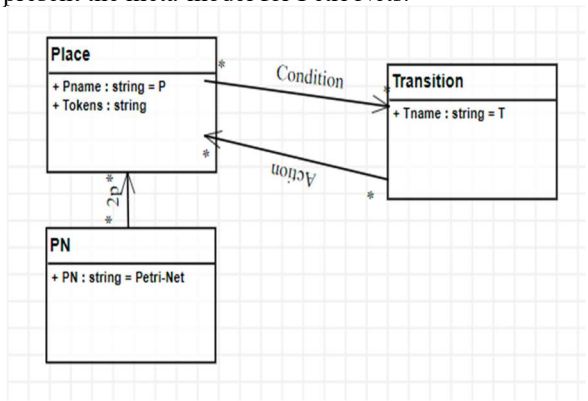


Figure 19: Meta-model of Petri Net.

Classes:

PN: This class represents the Petri Net framework. It contains an attribute <PN> of type string, displaying "Petri-Net" by default. Graphically, it is represented by a black rectangle.

Place: This class represents the place. It contains an attribute <Pname> of type string, displaying "P" by default, and an attribute <Tokens> of type string. Graphically, it is represented by a green circle.

Transition: This class represents the Transition. It contains an attribute <Tname> of type string, displaying "T" by default. Graphically, it is represented by a pink rectangle.

The graphical representation of the classes in the detailed Sequence Diagram meta-model

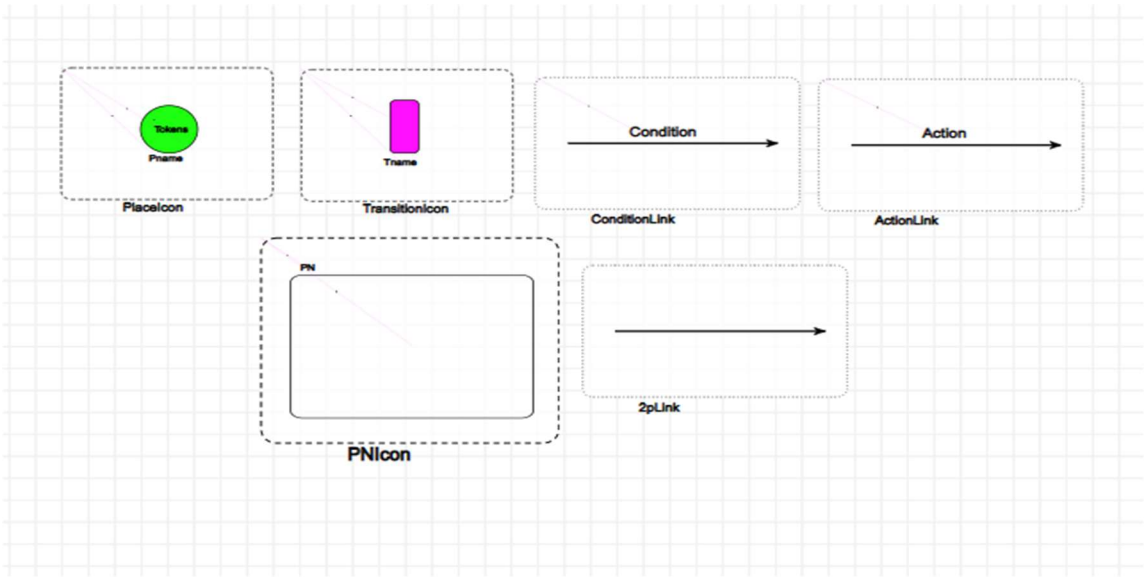


Figure 20: The Concrete Syntax of Petri Nets.

Figure 21 illustrates the generated tool for manipulating Petri Nets:

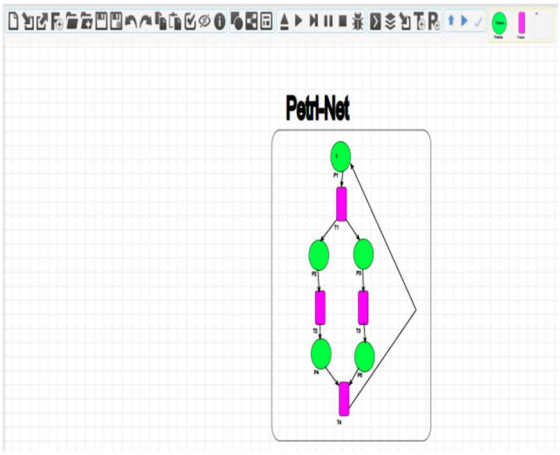


Figure 21: The generated tool for Petri Nets.

4.2.3 The proposed graph grammar for transforming aspect-oriented detailed sequence diagrams into Petri Nets

In this section, we present a graph grammar containing fifty-six (56) rules, each expressing a particular case. These rules will be executed in an order determined by the motif (T). We decompose the 56 rules into three categories based on the transformation of objects, which are defined as follows:

Creation: For transforming, the object and aspect in the aspect-oriented detailed sequence diagram into a PN

Linking: to connect the PN objects and aspect from the aspect-oriented detailed sequence diagram.

Deletion: for removing the aspect-oriented detailed sequence diagram that conclude the transformation.

In the following figure, the pattern (T_OA2RDP) of this graph grammar for this approach is represented:

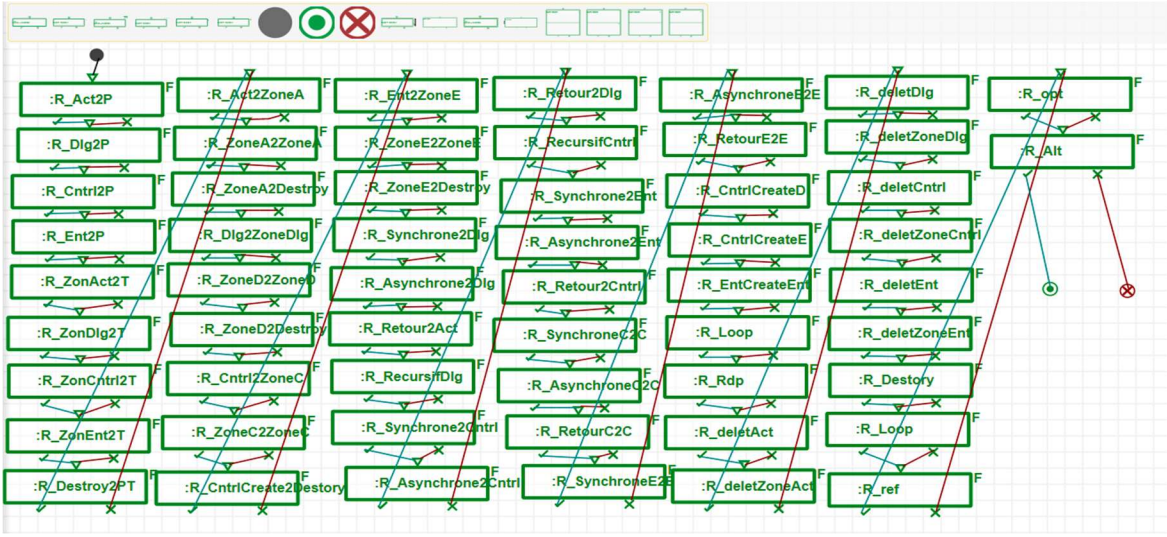


Figure 22: The transformation pattern of the approach.2.

Category1: the creation rules are applied to create a PN (Petri Nets) based on an aspect-oriented sequence diagram.

In the following figures, we illustrate the rules of the first category:

The Figures 24 and 25 represent the Python code for the rule of the first category on the actor:

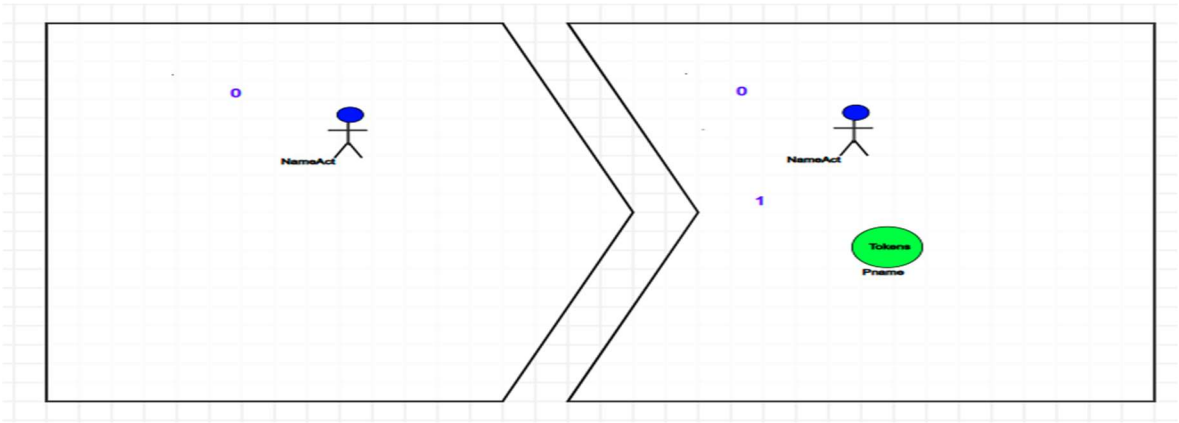


Figure 23: Application of Category 1 on the actor

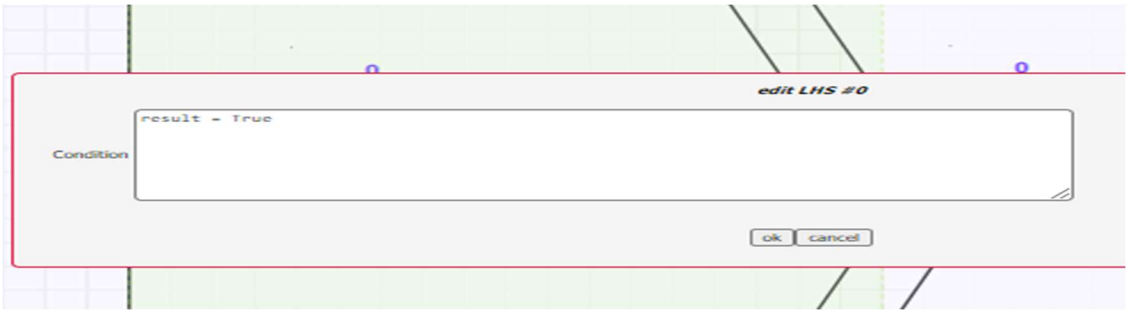


Figure 24: The LHS of the first rule on the actor (Python code).

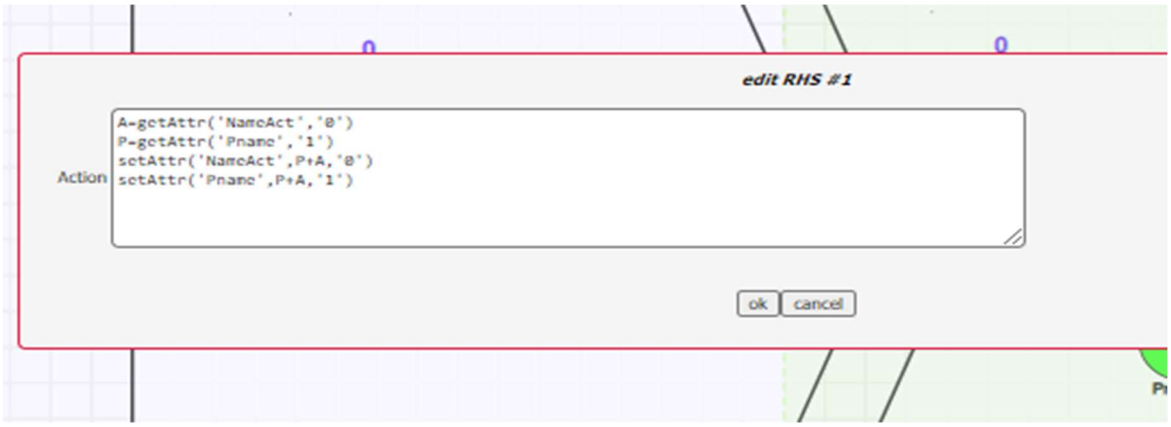


Figure 25: The RHS of the first rule on the actor (Python code).

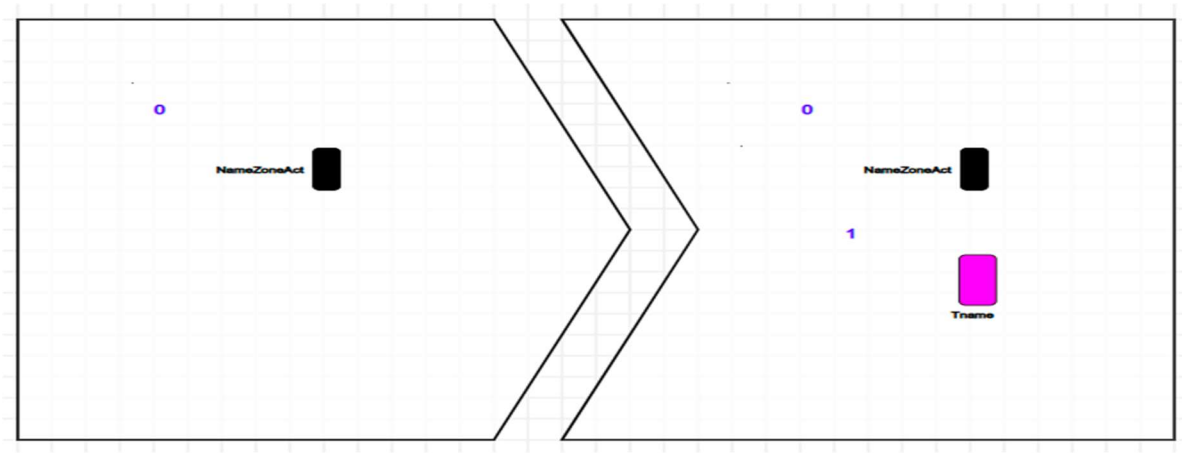


Figure 26: Application of Category 1 on the actor Activity Zone

.

Category2: The linking rules are applied to connect Petri net based on the linking of object and aspect from the aspect-oriented detailed sequence diagram.

In the following figures, we represent the rules of the second category:

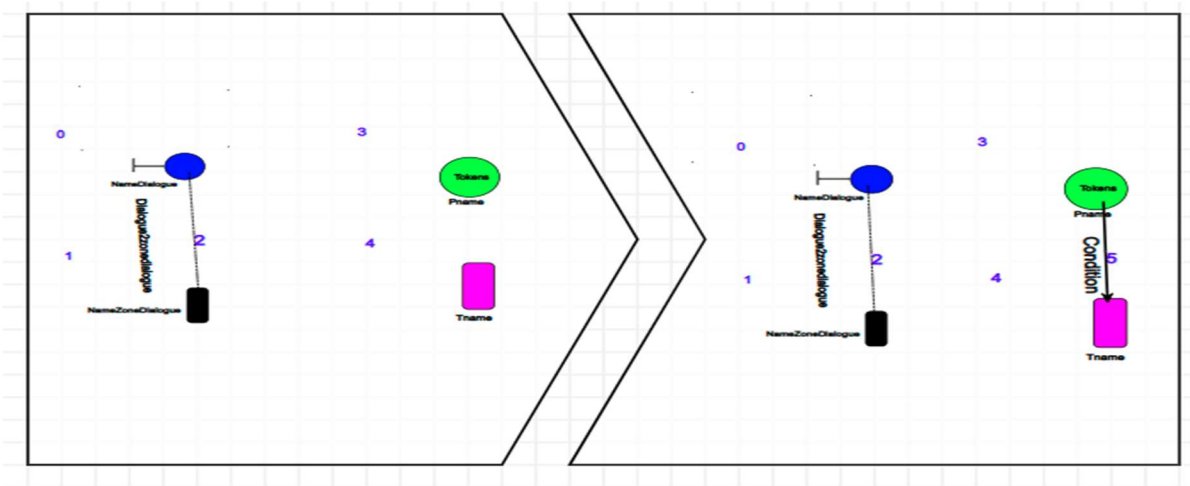


Figure 27: Application of Category 2 on the lifeline boundary

.The Figures 28 and 29 represent the Python code for the rule of the second category on the on the lifeline boundary:

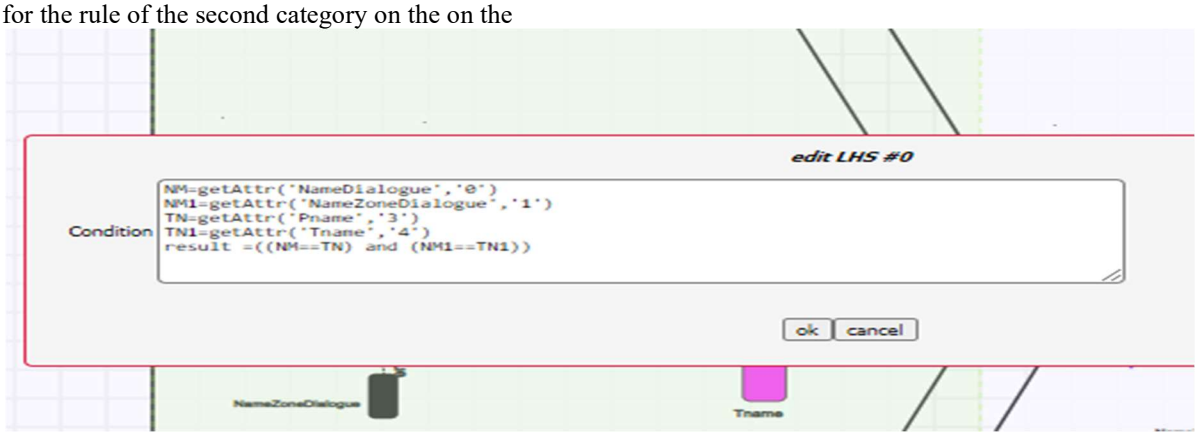


Figure 28: The LHS of the rule on the lifeline boundary (Python code).

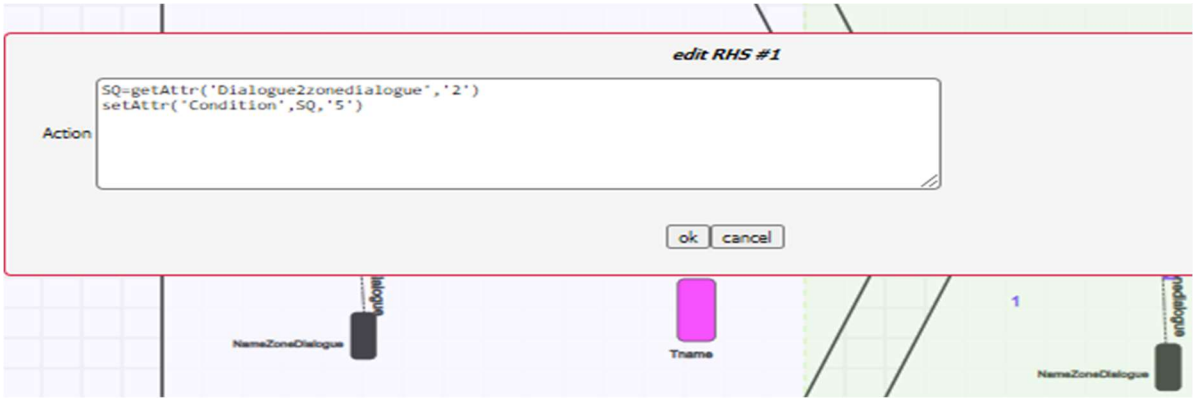


Figure 29: The RHS of the rule on the lifeline boundary (Python code)

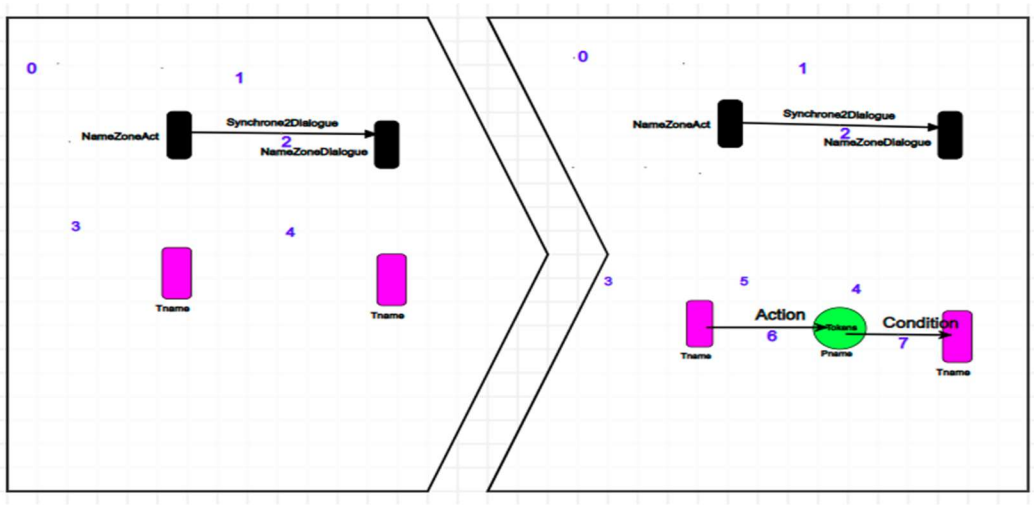


Figure 30: Application of Category 2 on the synchronous message.

Category3: the deletion rules are applied to remove objects and aspects from the aspect-detailed sequence diagram after the transformation.

In the following figures, we represent the rules of the third category:

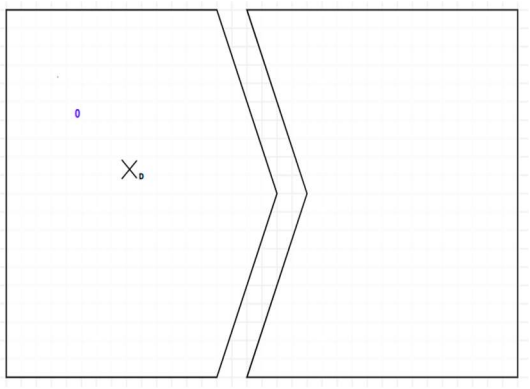


Figure 31: Application of Category 3 on "destroy".

The Figures 32and 33 represent the Python code for the rule of the third category on the on destroy:

Figure 32: The LHS of the rule on destroy (Python code).

Figure 33: The RHS of the rule on destroy (Python code).

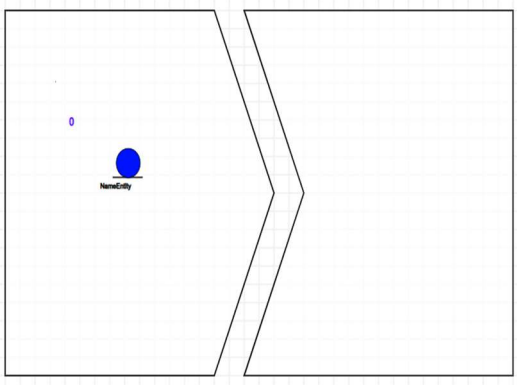


Figure 34: Application of Category 3 on "Entity".

4.3 Representation of the TINA tool

The figure 35 and 36 illustrates the tool used to verify the Petri nets obtained from the transformation:

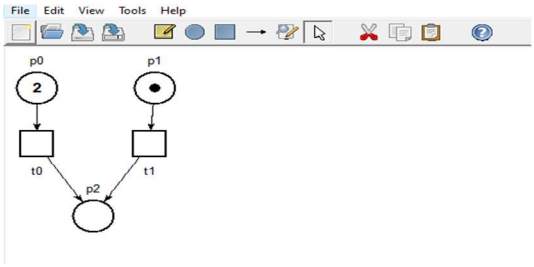


Figure 35: Representation of the TINA tool (Input).

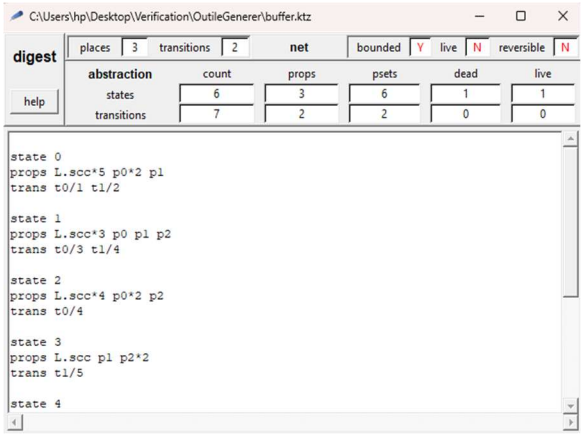


Figure 36: Representation of the TINA tool (Output).

4.4 Results and Discussion

4.4.1 Results:

In this paper, we propose two new approaches. There are several previous studies that are related to our approaches: The works related to the first approach, which involves transforming object-oriented diagrams into aspect-oriented diagrams, include the work of [7] titled "Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes". In addition, the work of [23] titled "La génération d'un outil de transformation des diagrammes UML 2.0 vers les diagrammes orientés aspect, basée sur la transformation de graphes". We have conducted a comparison with the works related to our research in Table 3.

Table 3: Comparison of Approaches for Transforming into Aspect-Oriented Diagrams.

The comparison work / Comparison Points	[7]	[23]	Our Approach
Design Model	Class diagram, Activity diagram, and Communication diagram.	State-transition diagram.	Detailed sequence diagram.
Join point	Any part of the model.	Any part of the model.	Components of detailed sequence diagram.
Advice	Any part of the model.	Any part of the model.	Components of detailed sequence diagram.
Aspect	A graphical model contains the join points and advices to be added.	A graphical model contains the join points and advices to be added.	A graphical model contains the join points and advices to be added.
Graph	Based on graph transformation.	Based on graph transformation.	Based on graph transformation.
Graph grammar	By given execution order.	From a graph grammar and by given execution order.	From a graph grammar and a (T) pattern.
Modeling tool	ATOM³	ATOM³	ATOMPM

The works related to the second approach, which involves transformation into Petri nets for verification purposes, include [29] with "specifing and Verifing Aspect-Oriented Systems in Rewriting Logic", [30] with his work "Based Aspect-oriented Petri Nets in Software Engineering", [31] with "Détection des préoccupations transversales par l'analyse formelle de concepts des diagrammes de sequence" and [32] with "A Petri net-based approach for supporting aspect-oriented modeling". We have conducted a comparison with the works related to our research in Table 4.

Table 4: Comparison of Petri Net Transformation and Verification Approaches.

The comparison works / Comparison Points	[29]	[30]	[31]	[32]	Our Approach
Transformation Approach	**Specifing and Verifing Aspect-Oriented Systems**	**The Petri net methodology with object-oriented technology.**	**Formal concept analysis of UML diagrams.**	**A Petri net-based method to support aspect-oriented modeling (AOM).**	**Modeling and verification of UML2.0 diagrams**
Design Model	**Aspect-Oriented Systems**	**Object-Oriented Petri Nets**	**UML class and sequence diagrams.**	**woven net**	**Detailed sequence diagram**
Graph	**Based on Maude Language**	**Colored Petri Net and the Object-Oriented Petri Net**	**Modeling the migration of crosscutting concerns.**	**Aspect Nets**	**Based on graph transformation**
Verification Method	**Transformation of the base/aspects modules into ordinary Maude modules.**	**Using the Petri net structure to verify real-time control software.**	**Formal concept analysis and method call orders.**	**XML Technology and Java Programming**	**Aspect-oriented transformation to Petri nets**
Modeling Tool	**Maude**	**Object-Oriented Petri Nets (OOPN).**	**Identification of candidate aspects.**	**Structural Analysis.**	**ATOMPM**
Verification Tool	**AO-Maude**	**Integrate SAADT (IDEF0) models and hierarchical colored Petri nets.**	**FCA Tool**	**Object Petri Net (OPN).**	**TINA**

4.4.2 Discussion

In the first approach, the focus is on detailed sequence diagrams, providing a finer granularity for aspect-oriented transformations. It specifies these elements within the components of the detailed sequence diagram, thereby allowing a more precise localization of variation points. Additionally, our method integrates a (T) pattern to structure the transformation, which could offer extra flexibility in defining the transformations. Our approach also stands out by using ATOMPM, a tool that could provide additional or enhanced functionalities tailored to our specific needs. In conclusion, our method proposes specific improvements, including finer granularity with detailed sequence diagrams and the use of a potentially more suitable modeling tool, ATOMPM. These distinctions can lead to more precise and flexible transformations, better meeting the specific requirements of certain modeling projects.

In the second approach, we rely on the use of detailed sequence diagrams, allowing for a finer and more precise analysis. Furthermore, we take it a step further by transforming them into aspect-oriented diagrams for Petri nets, thereby providing a more nuanced perspective on verification. To achieve this, we use ATOMPM, a tool specifically adapted to our method. In summary, our approach offers a more detailed and specific method for modeling and verifying aspect-oriented diagrams into Petri nets, enhancing the accuracy and scope of verification.

We have chosen comparison criteria from the previous tables because they form the central elements of our analysis. To mitigate threats to the validity of our study, we carefully selected criteria that cover the essential aspects of our field of study. For example, in the context of aspect-oriented transformation, we considered elements such as join points, advice, and the aspect itself, as they are crucial for evaluating the effectiveness of the transformation. The use of a new and different tool is also an important criterion, as it allows us to assess the innovation and effectiveness of the proposed approaches.

Regarding transformation with Petri nets and verification, we identified key criteria such as the transformation approach, design model, as well as the transformation and verification tools. These criteria are essential for ensuring the validity of Petri nets and allow for a rigorous comparison of the respective contributions. We selected these criteria based on their relevance and their ability to cover a broad spectrum of critical aspects, thereby reducing potential biases and increasing the reliability of our conclusions.

5. CASE STUDY

In this section, we implemented our transformation method on two case studies. The first case study focuses on reservation in a travel agency, and the second on managing a shopping center

5.1 Case Study on Booking in a Tourist Agency

5.1.1 Transformation from Object-Oriented to Aspect-Oriented

To demonstrate our approach, we applied it to booking in a tourist agency. We employed the detailed sequence diagram to represent the base model. Then, we introduced the aspect model representing the following three aspects: **Security**, **VerifyInformation**, and **DeleteDestroy.**

Security: This aspect allows verifying authentication security, positioned on the client.

VerifyInformation: This aspect verifies whether the information entered by the client is correct or not, positioned on the actor zone za2 and the boundary zone zd2.

DeleteDestroy: This aspect facilitates deleting the reservation database once its usage is completed, positioned on the destroy.

Base and Aspect Models for the Detailed Sequence Diagram

- **Basic Model**

In Figure 37, we present the basic model for the detailed sequence diagram

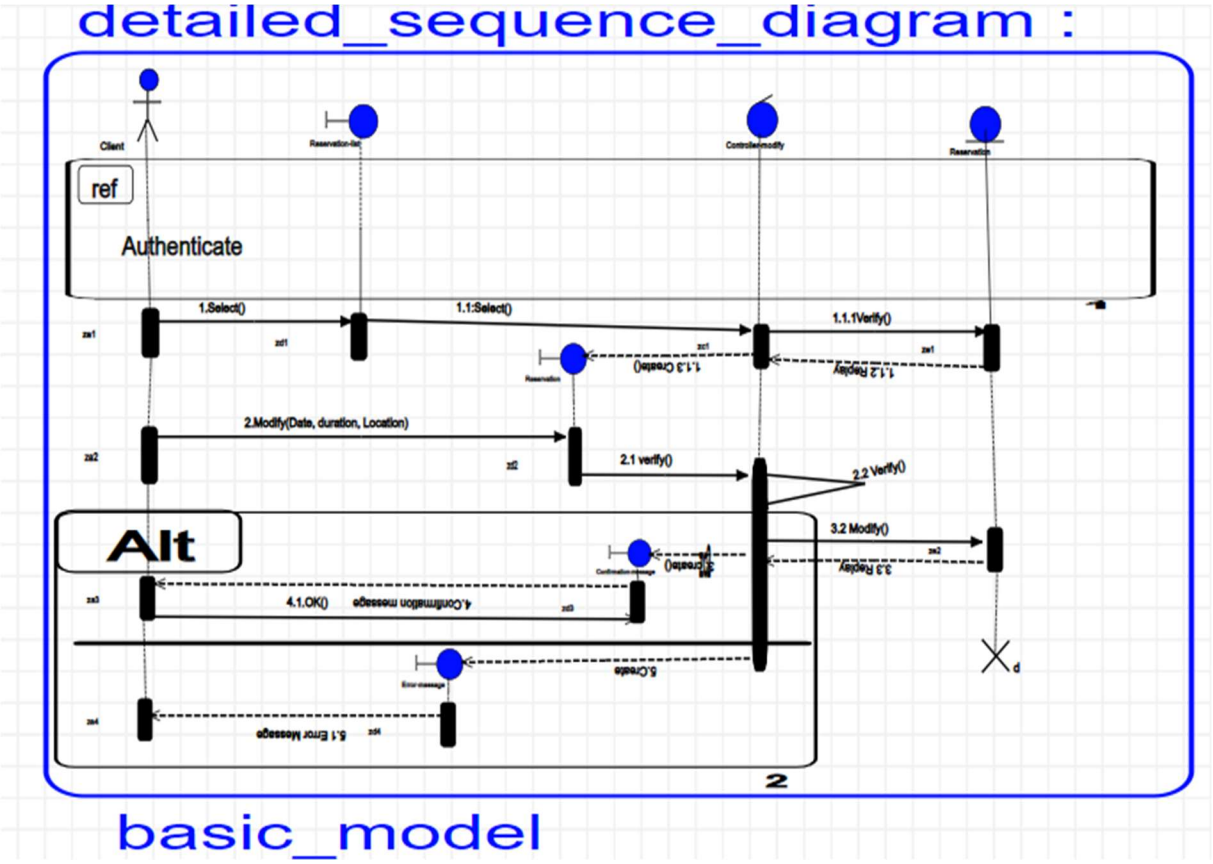


Figure 37: The Basic Model.

- **The Aspect Model**

In Figure 38, we present the aspect model of the detailed sequence diagram.

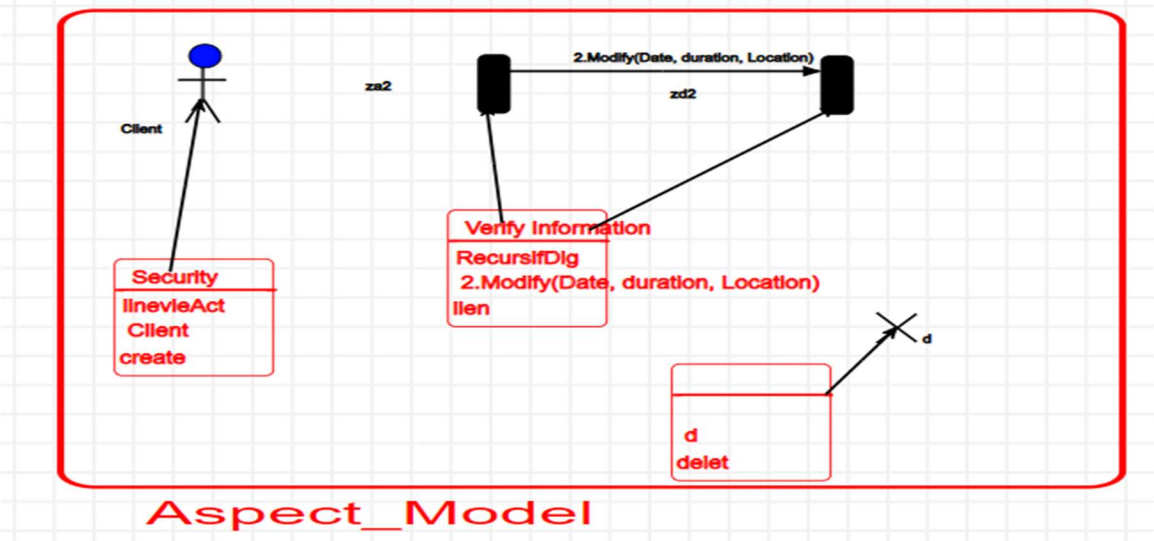


Figure 38: The Aspect Model.

- **Composite Model for the Detailed Sequence Diagram**

In Figure 39, we present the composite model, this model results from the integration of the basic model and the aspect model, which is a detailed aspect-oriented sequence diagram where we add:

- The actor zone to the client.
- Recursive messages to the boundary zone zd2.
- And remove the destroy.

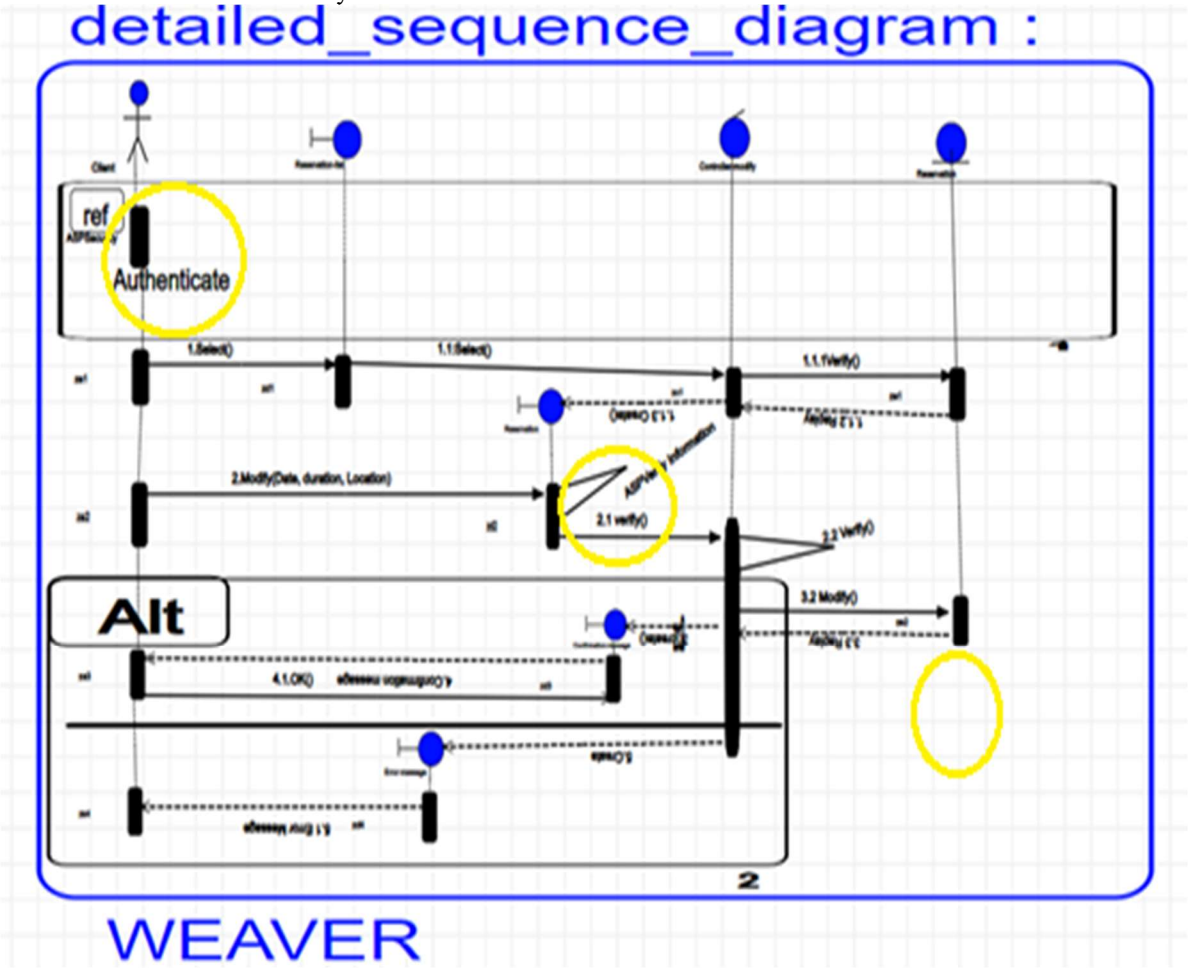


Figure 39: The Weaver

5.1.2 Transformation from Aspect-Oriented to Petri Nets

In the second approach, we used the detailed aspect-oriented sequence diagram obtained from the first transformation of the reservation modification case. We then applied the proposed graph grammar to this diagram to generate the corresponding Petri net.

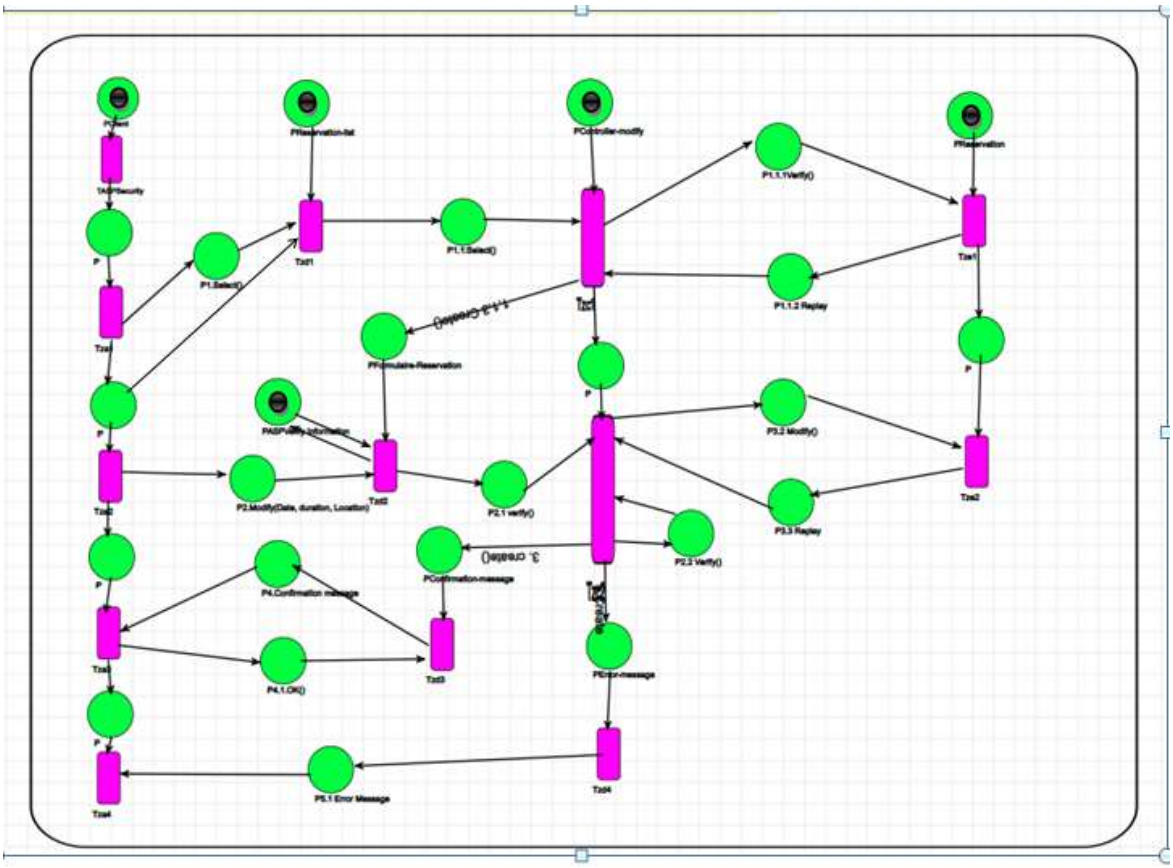


Figure 40: The Petri Net of the reservation modification case

5.1.3 Verification of the Petri Net

Finally, we perform a verification on the result of the second approach (on the Petri net) of the reservation modification case:

In the following figure, we present the result of verification:

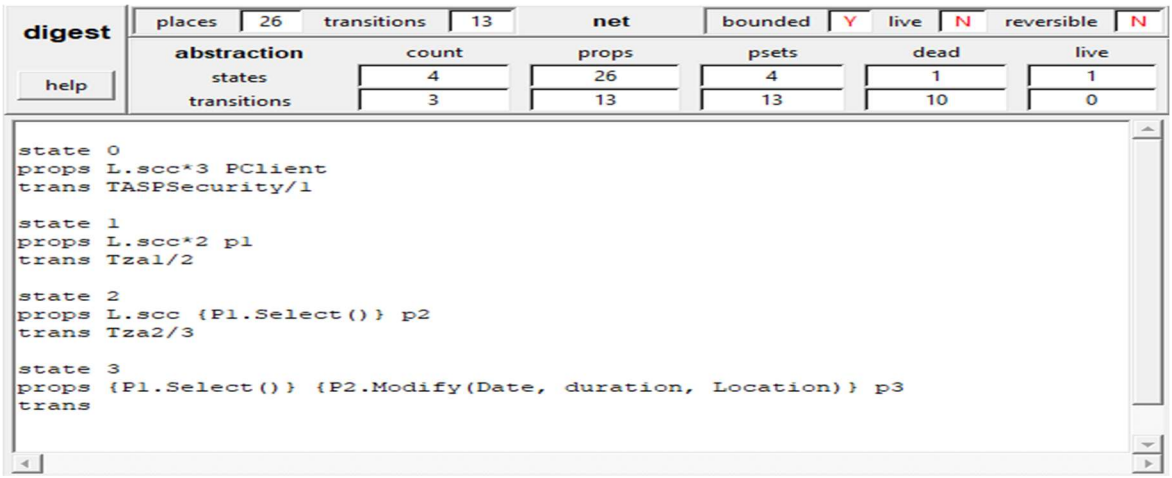


Figure 41: The result of the verification of the Petri Net for the reservation modification case in TINA.

- **Based on the results:**

Bounded = Y: The network is bounded.

Live = N: The network is not live.

Reversible = N: The network is not reversible

5.2 Case Study on the management of a shopping mall

5.2.1 Transformation from Object-Oriented to Aspect-Oriented

To demonstrate our approach, we applied it to mall management (Adding a promotion). We employed the detailed sequence diagram to represent the base model. Then, we introduced the aspect model representing the following four aspects: decision evaluation, update, product availability and Validate Information.

A decision evaluation: Allows making decisions based on the entered information, positioned on the controller zone zc1.

Update: This aspect updates the entries in the database table by adding a promotion, positioned on the entity promotion.

Product availability: Verifies the availability of products before adding a promotion, positioned on the controller zone zc1.

Validate Information: Validates the information for adding a promotion, positioned on the actor zone za1 and the boundary zone zd1

Base and Aspect Models for the Detailed Sequence Diagram

- **Basic Model**

In Figure 42, we present the basic model for the detailed sequence diagram.

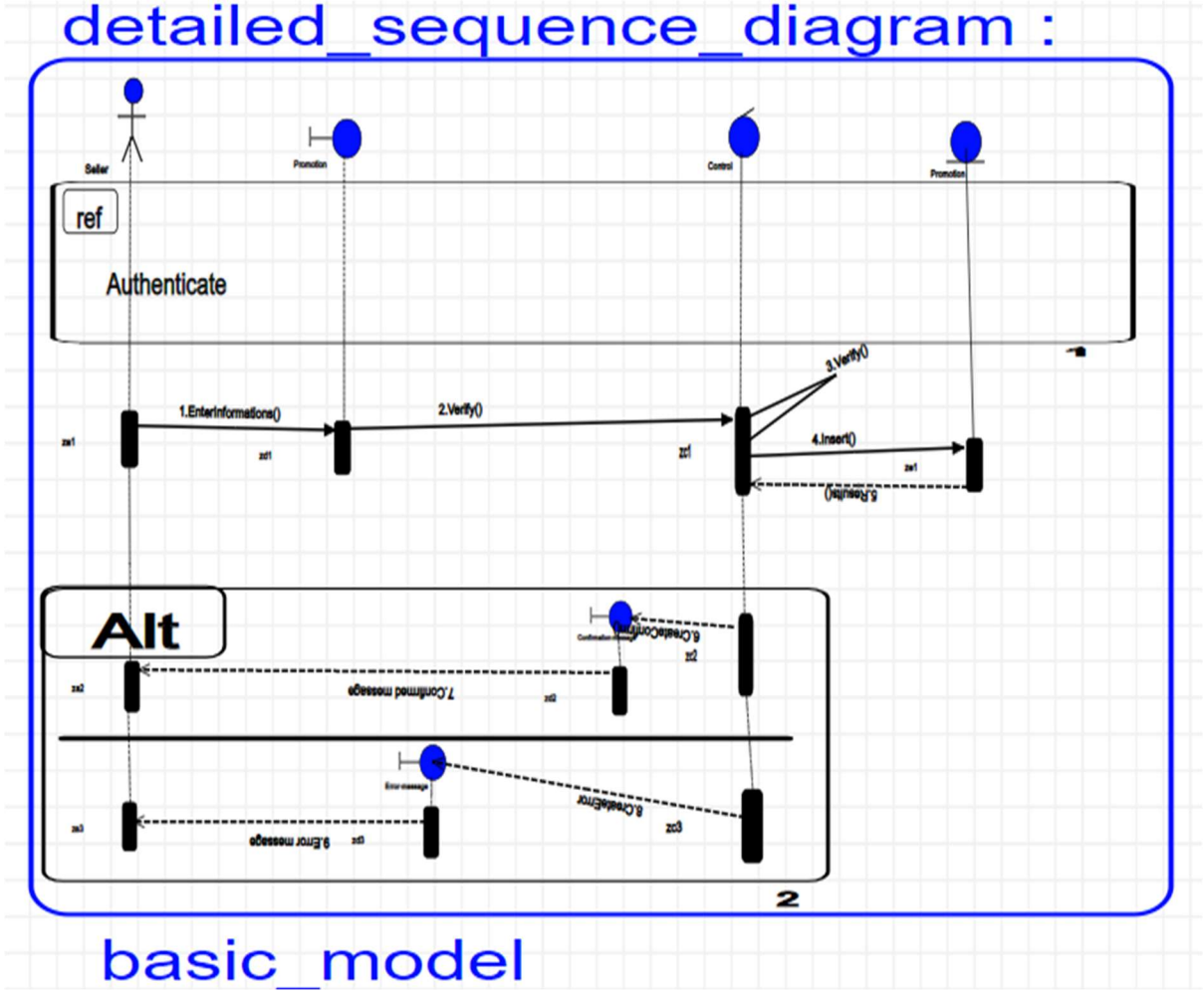


Figure 42: The Basic Model.

- **The Aspect Model**

In Figure 43, we present the aspect model of the detailed sequence diagram.

Journal of Theoretical and Applied Information Technology
31st August 2024. Vol.102. No. 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

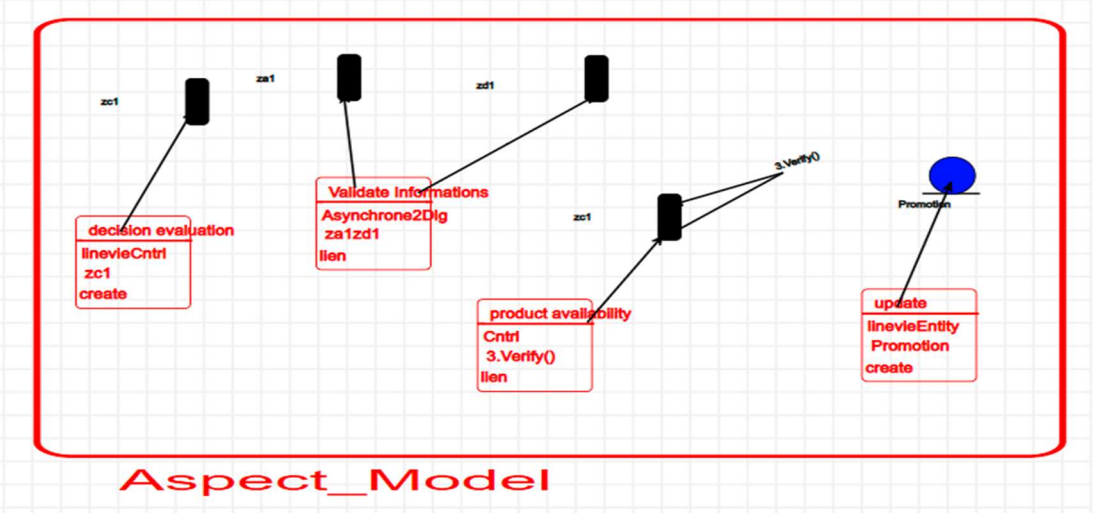


Figure 43: The Aspect Model

- **Composite Model for the Detailed Sequence Diagram**

In Figure 44, we present the composite model, which is an aspect-oriented detailed sequence diagram where we add:

- Controller zone to zc1.
- Entity zone to the entity promotion.
- Controller with his zone to zc1.
- Asynchronous Message to the actor zone za1 and the boundary zone zd1.

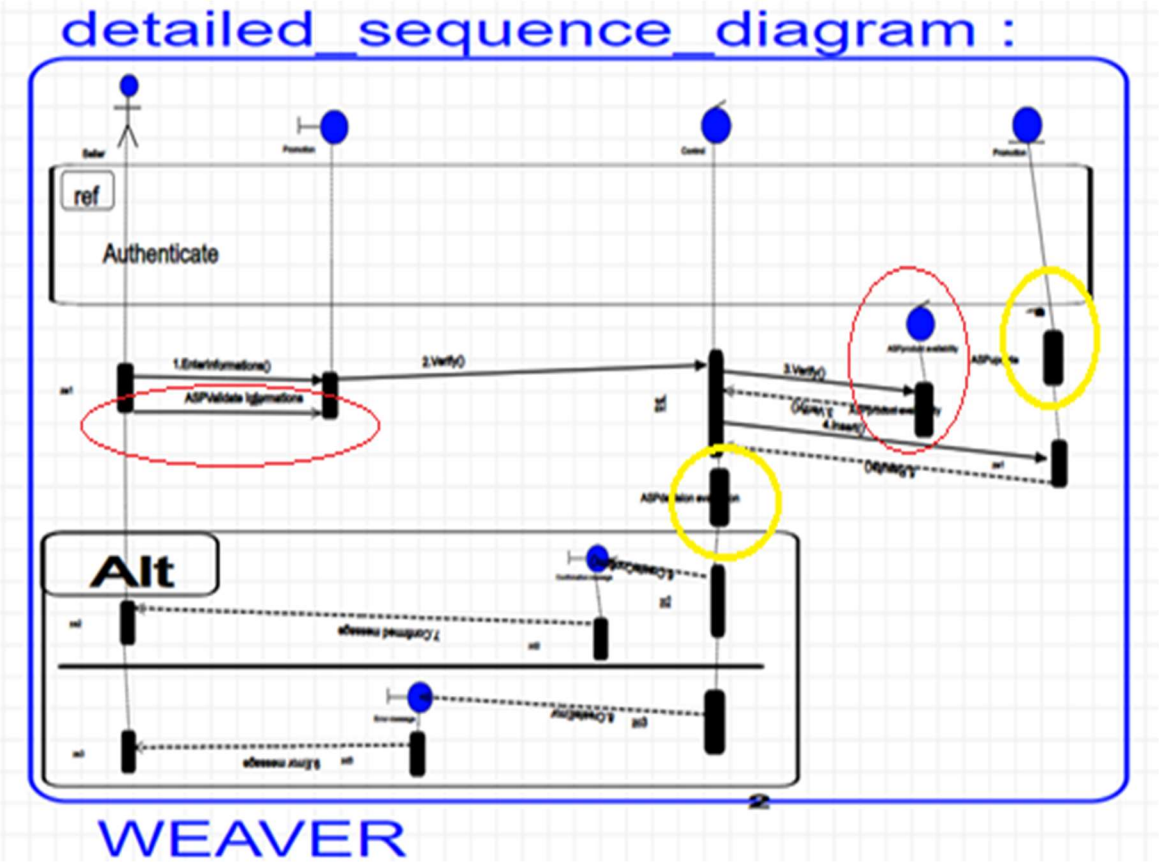


Figure 44: The Weaver

5.2.2 Transformation from Aspect-Oriented to Petri Nets

We transformed the composite model, which is an aspect-oriented sequence diagram of the promotion addition case, into a Petri net.

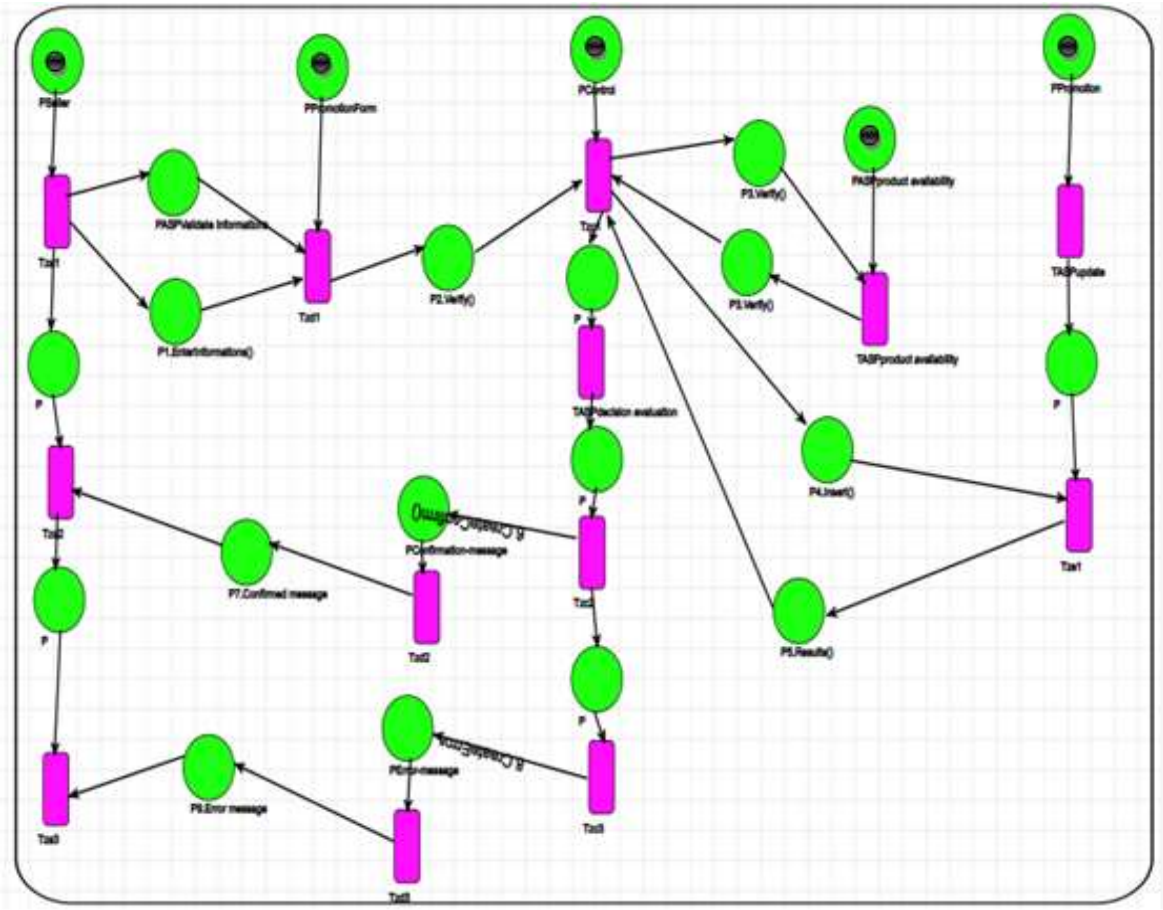


Figure 45: The Petri Net of the promotion addition case.

5.2.3 Verification of the Petri Net

Finally, we perform a verification on the result of the second approach (on the Petri net) of the promotion addition case:

In the following figure, we present the result of verification:

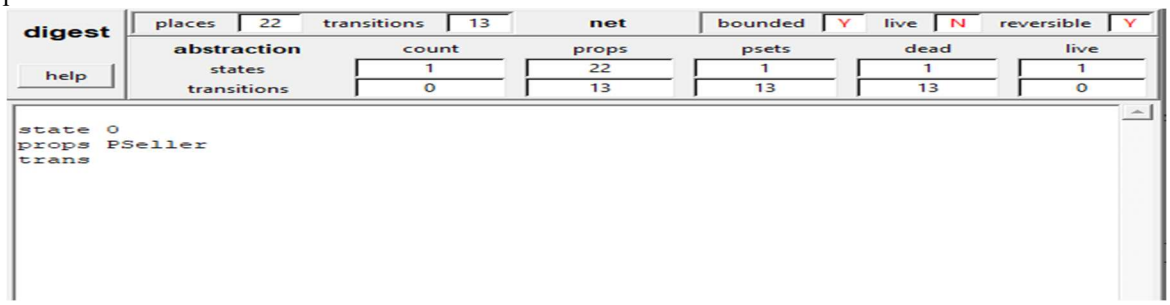


Figure 46: The result of verification of Petri Net for the promotion addition case with TINA.

- **Based on the results:**

Bounded = Y: The network is bounded.

Live = N: The network is not live.

Reversible = Y: The network is reversible

6. CONCLUSION AND PERSPECTIVES:

In this article, we have proposed two approaches. The first aims to generate aspect-oriented sequence diagrams from object-oriented sequence diagrams. This approach relies on meta-modeling (meta-models). We defined a single meta-model and subsequently presented a set of rules for the transformation process. In the second approach, we introduced two meta-models and a graph grammar to transform aspect-oriented sequence diagrams into Petri nets. This transformation was performed using the AToMPM modeling tool. Finally, we conducted a verification of the priorities of Petri nets such as liveness, boundedness, reachability, and reversibility using the TINA Petri nets analysis tool.

As perspectives, we propose to:

Study in detail how to transform the interaction frames of the detailed sequence diagram, whether alt, ref, loop, or opt, into Petri nets.

To develop a fully automated approach and generalize it to other types of UML diagrams, we plan to continue transforming other aspect-oriented UML diagrams into Petri nets using graph transformation and the AToMPM tool. Additionally, we intend to transform the aspect-oriented UML models into BPMN (Business Process Model and Notation) models, and subsequently convert these BPMN models into Petri nets.

Concurrently, we will work on integrating our approach into existing software development tools to facilitate its use by practitioners in the field. LOTOS (Language Of Temporal Ordering Specification) is a formal specification language widely used for the verification and validation of Petri nets. We propose in our future works to use LOTOS to verify our Petri nets. In addition, we can propose using the TGG (Triple Graph Grammar) modeling tool for model transformations, as it is considered the best among the others.

As strong points, we propose to:

In our study, we not only achieved the set objectives but also validated our hypotheses and determined the properties of Petri nets based on a detailed aspect-oriented sequence diagram.

We have innovated by transforming a detailed sequence diagram using an aspect for the first time, an approach still rare in the field of modeling.

As weak points, we propose to:

Time is very limited, and we sometimes end up with certain methods being incorrect. Therefore, few works on the aspect-oriented paradigm towards the formal

The lack of important data or the difficulty in collecting data for aspects oriented modeling.

REFERENCES:

[1] Pascal roques, Uml 2 par la pratique, Paris : Eyrolles, , 2008, pp : 246.

[2] Fatma Dhaou, Ines Mouakher, Christian Attiogbe and Khaled Bsaies. A Causal Semantics for UML2.0 Sequence Diagrams with Nested Combined Fragments, Proceedings of 12th International Conference on Evaluation of Novel Approaches to Software Engineering Porto, France, April 28 – 29,2017,pp.47-56.

[3] Faiz UL Muram, Huy Tran and Uwe Zdun. A Model Checking Based Approach for Containment Checking of UML Sequence Diagrams, Proceedings of 23rd Asia-Pacific Software Engineering Conference (APSEC), 6-9 Dec. 2016, pp.73-80

[4] Aws A. Magableh and Anas M. R. AlSobeh. Securing Software Development Stages using Aspect-Orientation Concepts, international Journal of Software Engineering & Applications (IJSEA), Vol.9, No.6, November 2018, pp.57-71.

[5] Cristian Vidal Silva, Rodrigo Saens, Carolina Del Río, and Rodolfo Villarroel, Aspect-Oriented Modeling: Applying Aspect-Oriented UML use cases and Extending aspect-Z. Computing and Informatics, Vol. 32, 2013, pp: 573–593.

[6] Khalid Bouragba, Hicham Belhadaoui, Mohamed Ouzzif, and Mounir RIFI, Approche orientée aspect pour l'amélioration de la fiabilité et de la performance temporelle d'un système tolérant aux fautes. Revue Méditerranéenne des Télécommunications, vol2, n°1,January 2012,pp.20-28

[7] Aouag, Mouna, Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes. Doctoral thesis, University of Mentouri, Constantine, 2014, pp. 23-25.

[8] Mohamed Lamine Berkane, Mahmoud Boufaida, Un Modèle de transformation des patrons de conception de l'Orienté Objet vers l'Orienté Aspect. 2nd Conférence Internationale sur l'Informatique et ses Applications, Proceedings of the 2nd Conférence Internationale sur l'Informatique et ses Applications (CIIA'09), Saida, Algeria, May 3-4, 2009.

[9] Sana OTHMAN, Modélisation et commande à base d'une représentation par réseau de Pétri d'un filtre actif parallèle avec un onduleur multicellulaire série. Doctoral thesis, University of Gabès, 2021, pp .73-106

[10] Minh Toàn VÕ, Assessment of heat pump operating faults coupled with building energy simulation using Petri net model, Doctoral thesis, University of, Rochelle France, 2021,pp.28-42.

[11] GUERROUF FAYÇAL, Une Approche de Transformation des Diagrammes d'Activités d'UML Mobile 2.0 vers les Réseaux de Petri. master's thesis, University of é El Hadj Lakhdar – BATNA, 2009, pp .03-75

[12] BAHRI, Mohamed Redha, Une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles. Doctoral thesis, University of Mentouri, Constantine, .2011, pp.8-75

[13] Said Meghzili, Allaoua Chaoui, Martin Strecker, Elhillali Kerkouche, On the Verification of UML State Machine Diagrams to Colored Petri Nets Transformation Using Isabelle/HOL. Proceedings of IEEE International Conference on Information Reuse and Integration (IRI), San Diego, CA, USA 04-06 August, 2017, pp.419 – 426.

[14] ElMansouri, R. Modélisation et Vérification des processus métiers dans les entreprises virtuelles : Une approche basée sur la transformation de graphes, Doctoral thesis, University of Mentouri, Constantine, 2009, pp. 40-41.

[15] Pedro M. Gonzalez del Foyo et José Reinaldo Silva, Using Time Petri Nets for Modeling and Verification of timed Constrained Workflow Systems, ABCM Symposium Series in Mechatronics, Vol. 3, University of São Paulo, Brazil.2008,pp.471-478.

[16] Boubendir, Amel, Un cadre générique pour la détection et la résolution des intéractions entre les aspects. Doctoral thesis, University of Mentouri, Constantine, 2011, pp. 19-27.

[17] Otmane rachedi, Soumeya, Apports des Approches de Séparation Avancée des Préoccupations : Une Etude Comparative Fondée sur les Modèles de Conception, Doctoral thesis, University of Badji Mokhtar Annaba, 2015, pp. 18

[18] S Dunnett , L Jackson and M Whiteley Simulation of polymer electrolyte membrane fuel cell degradation using an integrated Petri Net and 0D model, Reliability Engineering & System Safety, April 2020, Vol:196,pp.106741.

[19] N.Viswanadham, Y.Narahari and Timotht L.Johnson, Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models. IEEE Transactions on

Robotics and Automation, Vol 6, N° 6, December 1990), pp. 713 – 723.

[20] Sangita Kansal, Payal Dabas, An Introduction to Signed Petri Net, Journal of Mathematics, Volume 2021, N° 1, 16 June 2021,pp.1-8

[21] Raida Elmansouri, Said Meghzili, Allaoua Chaoui, Aissam Belghiat and Omar Hedjazi, Integrating UML 2.0 Activity Diagrams and Pi-calculus for Modeling and verification of Software Systems using TGG. Jordanian Journal of Computers and Information Technology, Vol 6, N° 4, Jordanian, December 2020, pp.326-344.

[22] Remigiusz Wi´sniewski , Marcin Wojnakowski and Zhiwu Li, Design and Verification of Petri-Net-Based Cyber-Physical Systems Oriented toward Implementation in Field-Programmable Gate Arrays—A Case Study Example, journal Energies 2023, Vol 15, N° 23 pp.16-67.

[23] Zerara Ahmed, Megrous Fares, La génération d'un outil de transformation des diagrammes UML 2.0 vers les diagrammes orientés aspect, basée sur la transformation de graphes , master memory, Abd elhafid Boussouf University Centre Mila, Algeria, .2020, pp.3-25.

[24] Anas Mohammad Ramadan AlSobeh, OSM: Leveraging model checking for observing dynamic behaviors in aspect-oriented applications, Journal of Communication and Media Technologies, Vol 13, N° 4, October 2023, pp.1-18.

[25] Fernando Pinciroli, Jose Luis Barros Justo, Raymundo Forradellas, Systematic mapping study: On the coverage of aspect-oriented methodologies for the early phases of the software development life cycle, Journal of King Saud University –Computer and Information Sciences, Vol 34, N° 6, June 2022, pp. 2883-2896.

[26] GokhanGelen and YaseminIçmez, Task planning and formal control of robotic assembly systems: A Petri net-based approach, Ain Shams Engineering Journal, Vol 15, N° 7, July 2024, pp.1-12.

[27] Faming lu, Fenghua lv, Minghao cui, Yunxia bao, Qingtian zeng, Petri Net Unfolding-Based Detection and Replay of Program Deadlocks, Open IEEE Access Journal, Vol 12, April 2024, pp. 53716.

[28] MoezKrichen, Vérification et validation formelles pour l'Internet des objets, HAL open Science, 5 Jan 2024, pp.11

[29] Amina Boudjedir, Toufik Benouhiba, Djamel Meslati, Specifing and Verifing Aspect-Oriented Systems in Rewriting Logic, International Conference on Advanced Aspects of Software Engineering ICAASE, , Constantine, Algeria, 2-4 November, 2014,pp. 36 – 42.

[30] Wensong HUa , Xingui Yanga ,Ke Zuoa, Based Aspect-oriented Petri Nets in Software Engineering, 2011 International Conference on Physics Science and Technology (ICPST 2011), Physics Procedia 22, 2011,pp. 646 – 650.

[31] Fairouz Dahi, Nora Bounour, Détection des préoccupations transversales par l'analyse formelle de concepts des diagrammes de séquence, Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées (ARIMA Journal), vol 18. August 2014, pp. 19-35

[32] Lianwei Guan, Xingyu li, Hao hu, Jian lu,Petri net-based approach for supporting aspect-oriented modeling, 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering, Nanjing, China, 17-19 June, 2008,pp. 413–423.