

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



N° Ref :.....

Centre Universitaire
Abd elhafid boussouf Mila

Institut de Mathématiques et d'Informatique

Département d'Informatique

Mémoire préparé En vue de l'obtention du diplôme de Master

En : Informatique

**Spécialité : Sciences et Technologies de l'Information et de la Communication
(STIC)**

**Formal Verification and validation of object-
aspect oriented models, an approach based on
graph transformation**

**Préparé par : Kennouche Wiam
Merabet Nouhad**

Soutenue devant le jury :

Boufaghes Hamida	MAA	C. U. Abdelhafid Boussouf, Mila	Président
Aouag Mouna	MCB	C. U. Abdelhafid Boussouf, Mila	Rapporteur
Meghzili Said	MCB	C. U. Abdelhafid Boussouf, Mila	Examineur

Année universitaire :2023/2024

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

ACKNOWLEDGEMENTS

First and foremost, we thank **Allah** for the energy He has given us over these five years, as well as for the patience that allowed us to complete this work.

We would also like to particularly thank our supervisor, **Mrs. Aouag Mouna**, for her advice, encouragement, and trust, which enabled us to accomplish this work.

We also wish to express our deep gratitude to all the members of our jury for agreeing to examine this work.

We do not forget to thank all the teachers who contributed to our training over these five years.

Finally, we extend our most sincere thanks to all our families for their unwavering encouragement.

Wiam & Nouhad

Dedication

هو الله من يشكر آناء الليل وأطراف النهار، الحمد لله حبا وشكرا وامتنانا على البدء والختام
(وَأَخْرُ دَعَوَاهُمْ أَنْ الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ)
إلى الذي أقسمت أن أبدأ به كل أعمالي، إلى من علمني أن النجاح لا يأتي إلا بالصبر والإصرار، إلى
من علمني أن الدنيا كفاح سلاحها العلم والمعرفة (أبي الغالي رحمه الله)
إلى التي ظلت دعواتها تضم اسمي دائما، إلى من كانت الداعمة الأولى، أهديك هذا الإنجاز الذي لولاك
لم يكن، أهديك مراحلتي وانجازاتي كلها فالفضل والثناء للمولى ثم لكفاحك لأجلي (أمي الحبيبة)
إلى من قيل فيهم: (سنشدُّ عضدك بأخيك)
إلى الكتف الذي لا يميل والظل الذي احتمي به، إلى العمود الثابت في الحياة (أخي الغالي)
إلى فراشة بيتنا، إلى القلب النابض بصدق الحب والمشاعر، إلى من أمنت بقدراتي (أختي الحبيبة)
إلى ذلك الشخص الذي بفضلته أنا هنا اليوم، إلى من شجعتني لخوض هذه التجربة، إلى من أوقن على
أنني أستطيع الوصول إلى ما وصلت له اليوم ...
إلى مؤنستي ورفيقة عمري، داعمتي، بئر أسراري وغيمة قلبي (لينة)
إلى الحاملة لاسمي بين كفوف أديتها، لنجمتي المضيئة (ونام)
إلى رفيقة هذا المشوار، شريكة الليالي الصعبة (نهاد)
إلى أفراد عائلتي، إلى من حملوني بين طيات دعواتهم، إلى من كنت فرحتهم الأولى،
إلى جدتاي الغاليتان (شرطيوي زهية وعبيدة)،
إلى اعمامي واخوالي، عماتي وخالاتي،
خاصة إلى داعمتاي وصديقتاي (عمتي الغالية أمل) و (خالتي الحبيبة مريم)
إلى من وثقت في قدراتنا وشجعتنا لاتمام هذا العمل مشرفتنا (الأستاذة عواق منى)
وأخيرا وصلت لما تمنيت وحلمت وطمحت له، ها أنا اليوم اتممت أول ثمراتي راجية من الله تعالى ان
ينفعني بما علمني وان يعلمني ما أجهل ويجعله حجة لي لا علي
الحمد لله الذي ما نجحنا ولا تفوقنا الا برضاه، الحمد لله الذي ما اجتزنا دربا ولا تخطينا جهدا الا
بفضله واليه ينسب الفضل والكمال.

ونام

Dedication

الحمد لله الذي يسر البدايات وأكمل النهايات وبلغنا الغايات، الحمد لله ما تنهى درب ولا ختم جهدا ولا
تم سعي الا بفضل الله ليس بجهدى واجتهادى انما بفضلك وتوفيقك وكرمك...
أهدي هذا النجاح الى التي تحارب الضغوطات والكتب والأيام الصعبة
وتركز على أهدافها ومستقبلها بدون تراجع... أنا
إلى الذي زين اسمي بأجمل الألقاب، إلى الذي علمني أن الدنيا كفاح وسلاحها العلم والمعرفة... أبي
الغالي
إلى الشمعة التي كانت لي في الليالي المظلمات، سر قوتي ونجاحي ومصباح دربي... أمي الغالية
إلى من عشت معهم أجمل لحظات حياتي، إلى من شهدوا معي متاعب الدراسة وسهر الليالي... أخي
وأختي
إلى روح جدي الغالي الذي لطالما تمنى لحظة تخرجي، إلى الذي توسده التراب قبل أن يراني خريجة
(رحمه الله)
-إلى رفيقة مشواري التي قاسمتني لحظاته... ونأم
إلى من غرست فينا حب التميز، وأدت واجبها بكل اتقان وأمانة أستاذتنا عواق منى
إلى رفقاء الروح الذين شاركوني خطوات هذا الطريق، إلى من شجعوني على المثابرة وإكمال
المسيرة... ياسمين ورشا
إلهي لا تطيب اللحظات إلا بذكرك وشكرك... ولا تطيب الآخرة إلا بعفوك... ولا تطيب الجنة إلا
برؤيتك...

نهاد

Abstract

Modeling plays a fundamental role in the software development process. When Object-Oriented Modeling (OOM) presents certain limitations, these restrictions pose problems, especially with the rapid advancement of computing. Aspect-oriented modeling (AOM) offers solutions to these problems, although it lacks semantics. This is why formal modeling, which possesses this semantics, is important.

In this memory, we propose an approach to transform a detailed object-oriented sequence diagram into a detailed aspect-oriented sequence diagram, based on graph grammar. Then, we propose a method to transform the aspect-oriented sequence diagram into a Petri net. Our work begins with a single Meta modeling for the first approach, using graph grammar to achieve an aspect-oriented model. Then, we apply the second approach to the result of the first one, using two Meta modeling and the transformation, which results 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, detailed sequence diagram, graph grammar, Petri nets, MDA, AToMPM, TINA.

Résumé

La modélisation joue un rôle fondamental dans le processus de développement logiciel. Lorsque la Modélisation Orientée Objet (MOO) présente certaines limites, ces restrictions posent des problèmes, surtout avec l'avancement rapide de l'informatique. La Modélisation Orientée Aspect (MOA) propose des solutions à ces problèmes, bien qu'elle manque de sémantique. C'est pourquoi la modélisation formelle, qui possède cette sémantique, est importante.

Dans ce mémoire, nous proposons une approche pour transformer un diagramme de séquence détaillé orienté objet vers un diagramme de séquence détaillé orienté aspect, en se basant sur une grammaire de graphes. Ensuite, nous proposons une méthode pour transformer le diagramme orienté aspect vers un réseau de Petri. Notre travail commence par une seule méta-modélisation pour la première approche, qui utilise une grammaire de graphes pour aboutir à un modèle orienté aspect. Ensuite, nous appliquons la deuxième approche sur le résultat de la première, en utilisant deux méta-modélisations et des règles de transformation, ce qui aboutit à un réseau de Petri, nous utilisons l'outil de modélisation AToMPM. Finalement, nous faisons une analyse de propriété avec l'outil TINA.

Mots clés: modélisation orientée objet, modélisation orientée aspect, diagramme séquence détaillé, grammaire de graph, réseaux de pétri, MDA, AToMPM, TINA.

ملخص

يلعب التصميم دوراً أساسياً في عملية تطوير البرمجيات. عندما يقدم التصميم الموجه نحو الأشياء (OOM) بعض نقائص، وهذه النقائص تثير مشاكل، خاصة مع التقدم السريع في مجال الحوسبة. وقد استطاع تصميم المظهر الموجه (AOM) أن يقدم حلولاً لهذه المشاكل، على الرغم من أنه يفتقر إلى المعاني. لهذا السبب فإن التصميم الرسمية الذي يمتلك هذه الدلالات المهمة والضرورية.

في هذه المذكرة، نقترح نهجاً لتحويل مخطط تسلسل مفصل موجه للأشياء إلى مخطط مفصل المظهر الموجه، وذلك بالاعتماد على قواعد الرسم البياني. بعد ذلك نقترح طريقة لتحويل مخطط المظهر الموجه إلى شبكة بيترى. يبدأ عملنا بنمذجة وصفية واحدة (meta-model) للنهج الأول، تعتمد على استخدام قواعد الرسم البياني للوصول إلى نموذج المظهر الموجه. بعد ذلك، نطبق النهج الثاني على نتيجة الأول وذلك باستخدام نمذجتين وصفيتين وقواعد الرسم البياني، مما يؤدي إلى شبكة بيترى. نستخدم أداة النمذجة (AToMPM). في النهاية، نقوم بإجراء تحليل للخصائص باستخدام أداة TINA.

الكلمات الرئيسية: التصميم الموجه نحو الأشياء، تصميم المظهر الموجه، مخطط تسلسلي مفصل، قواعد الرسم البياني، شبكات بيترى، MDA، AToMPM، TINA.

Contents

GENERAL INTRODUCTION	1
1.INTRODUCTION:	1
2.PROBLEMATIC:	1
3.CONTRIBUTIONS:	2
4.MEMORY ORGANIZATION:.....	3
ASPECT-ORIENTED MODELING	1
INTRODUCTION:	4
1 THE MODEL:	4
2 MODELING:	4
2.1 <i>Goals of Modeling:.....</i>	<i>5</i>
2.2 <i>Types of Languages:.....</i>	<i>5</i>
3 OBJECT-ORIENTED MODELING:	5
4 LIMITATIONS OF THE OBJECT-ORIENTED APPROACH:.....	6
5 DEFINITION OF ASPECT-ORIENTED MODELING:	6
5.1 <i>The Purpose of Aspect-Oriented Modeling:</i>	<i>6</i>
5.2 <i>Basic Concepts:</i>	<i>6</i>
6 ASPECT-ORIENTED APPROACHES:	8
6.1 <i>Aspect oriented Requirements Engineering:</i>	<i>8</i>
6.2 <i>Aspect oriented Architecture:.....</i>	<i>8</i>
6.3 <i>Aspect oriented Design:.....</i>	<i>8</i>
6.4 <i>Aspect oriented Programming:</i>	<i>9</i>
6.5 <i>Verification of Aspect oriented Programs:.....</i>	<i>9</i>
6.6 <i>Aspect oriented Middleware:.....</i>	<i>9</i>
7 INCONVENIENTS OF ASPECT-ORIENTED MODELING:	9
CONCLUSION:.....	10
PETRI NETS	1
INTRODUCTION:	11
1 PETRI NETS:.....	11
2 BASIC CONCEPTS OF PETRI NETS:.....	11
2.1 <i>Formal Definition:.....</i>	<i>11</i>
2.2 <i>Graphical Definition:</i>	<i>11</i>
2.3 <i>Marking of a Petri net:</i>	<i>12</i>
2.4 <i>Matrix Representation of a Petri Net:</i>	<i>13</i>
3 THE EVOLUTION OF STATES IN A PETRI NET:	14
4 PETRI NET PROPERTIES:	15
4.1 <i>Liveliness:.....</i>	<i>15</i>
4.2 <i>Boundedness:.....</i>	<i>15</i>
4.3 <i>Reachability:.....</i>	<i>15</i>
4.4 <i>Blocking:.....</i>	<i>15</i>
5 CATEGORIES OF PETRI NETS:	16
6 THE TYPES OF PN:	19
6.1 <i>Ordinary Petri Net:</i>	<i>19</i>
6.2 <i>Generalized Petri Net:.....</i>	<i>19</i>
6.3 <i>Timed Petri Net:</i>	<i>19</i>
6.4 <i>Colored Petri Net:</i>	<i>19</i>

6.5 Continuous Petri Net:	20
CONCLUSION:	20
GRAPH TRANSFORMATION AND VERIFICATION	20
INTRODUCTION:	21
1 MODEL DRIVEN ARCHITECTURE (MDA):	21
1.1 Basic Concepts of MDA:	21
1.2 The MDA Approach:	21
1.3 Model Driven Architecture:	22
1.4 Types of MDA Approaches:	23
1.5 The tools of MDA:	25
2 MODEL TRANSFORMATIONS:	25
3 GRAPH TRANSFORMATIONS:	26
3.1 Graph Concepts:	26
3.2 Subclass of a Graph:	27
3.3 Graph Transformation Tools:	27
4 ATOMPM:	27
5 ANALYSIS OF PETRI NETS:	28
5.1 Analysis Techniques:	28
5.2 Analysis tools for PN:	28
6 TINA:	29
CONCLUSION:	29
PROPOSED APPROACHES	29
INTRODUCTION:	30
1 TRANSFORMATION OF DETAILED OBJECT-ORIENTED SEQUENCE DIAGRAMS INTO DETAILED ASPECT-ORIENTED SEQUENCE DIAGRAMS:	30
1.1 Meta-modeling:	30
1.2 The proposed graph grammar:	35
1.3 Explanatory example:	42
2 TRANSFORMATION OF ASPECT-ORIENTED DETAILED SEQUENCE DIAGRAMS INTO RDPS:	43
2.1 Model Transformation Process:	43
2.2 Meta-modeling:	45
2.3 The proposed graph grammar:	46
3 REPRESENTATION OF THE TINA TOOL:	49
4 EXPLANATORY EXAMPLE:	50
5 RESULTS AND DISCUSSION:	51
CONCLUSION:	53
CASE STUDIES	53
INTRODUCTION:	54
1 CASE STUDY ON BOOKING IN A TOURIST AGENCY:	54
1.1 Transformation from Object-Oriented to Aspect-Oriented:	54
1.2 Transformation from Aspect-Oriented to Petri Nets:	56
1.3 Verification of the Petri Net:	56
2 CASE STUDY ON THE MANAGEMENT OF A SHOPPING MALL:	57
2.1 Transformation from Object-Oriented to Aspect-Oriented:	57
2.2 Transformation from Aspect-Oriented to Petri Nets:	59
2.3 Verification of the Petri Net:	59

CONCLUSION:.....	60
CONCLUSION AND PERSPECTIVES	60
CONCLUSION:.....	61
PERSPECTIVES:	61
BIBLIOGRAPHIC REFERENCES:.....	62

List of Figures

1 Principle of Transformation of a Detailed Object-Oriented Sequence Diagram into a Detailed Aspect-Oriented Sequence Diagram.....	2
2 Principle of Transformation from a Detailed Aspect-Oriented Sequence Diagram to a Petri Net.....	2
1.1 Representation of Modeling.....	4
1.2 Classification of Languages or Methods.....	5
1.3 The Aspect-Oriented Modeling Process.	8
2.1 (a) Graphical representation of a place with one token, (b) transition.....	12
2.2 Example of a Petri net.....	12
2.3 Example of a Petri net.....	13
2.4-a Example of a Marked Petri Net.....	14
2.4-b Incidence Matrix.....	14
2.5 The Evolution of States in a Petri Net.....	14
2.6 Example of Petri Net Blocking.....	16
2.7 State Graph.	16
2.8 Event Graph.	16
2.9 conflict-free PN.	17
2.10 Free Choice PN	17
2.11 Simple PN.	17
2.12 Pure PN.....	17
2.13 Limited Capacity Petri Net.	18
2.14 Priority PN.....	18
2.15 Inhibitory Arcs in Petri Nets.	19
2.16 Petri Net (left) and Colored Petri Net (right).	20
3.1 The model transformation process in the MDA approach.....	22
3.2 The four abstraction levels for MDA.	23
3.3 The model transformation approaches.	24
3.4 Types of transformations and their main uses.	26
3.5 (a) Undirected graph G, (b) Directed graph, (c) Subgraph of G.	26
3.6 Presentation of the AToMPM tool.	27
3.7 Presentation of the TINA tool.....	29
4.1 Meta-model of the detailed sequence diagram.....	31
4.2 The tool generated for the detailed aspect-oriented sequence diagram.....	35
4.3 The motif of transformation of approach.....	35
4.4 Application of category 1 with Aspect.PJ == Actor and Aspect.Ad == linevieAct.....	36
4.5 The LHS of the first create rule(Python code).....	36
4.6 The RHS of the first create rule (Python code).....	37
4.7 Application of category 1 with Aspect.PJ == Control Zone et Aspect.AD == CreateDlg.....	37
4.8 Application of category 2 with Aspect.PJ == Control Zone, Entity Zone and Aspect.AD == Asynchrone2E.	38

4.9 The LHS of the first link rule (Python code).....	38
4.10 The RHS of the first Link rule (Python code).....	39
4.11 Application of category 2 with Aspect.PJ== Synchronone message et Aspect.AD== Cntrl.	39
4.12 Application of category 3 with Aspect.PJ== Zone Actor et Aspect.type== delet.....	40
4.13 The LHS of the first delete rule (Python code).....	40
4.14 The RHS of the first delete rule (Python code).....	41
4.15 Application of category 3 with Aspect.PJ== Destroy et Aspect.type== delet.....	41
4.16 Basic Authentication Model.	42
4.17 Authentication Aspect Model.	42
4.18 The weaver model.	43
4.19 Meta-model of Petri Nets.	45
4.20 The generated tool for Petri Nets.	46
4.21 The transformation pattern of the approach.	47
4.22 Application of Category 1 on the actor.	47
4.23 Application of Category 1 on the actor Activity Zone.	48
4.24 Application of Category 2 on the lifeline boundary.	48
4.25 Application of Category 2 on the synchronous message.	48
4.26 Application of Category 3 on "destroy".....	49
4.27 Application of Category 3 on "Entity".....	49
4.28 Representation of the TINA tool (Input)	50
4.29 Representation of the TINA tool (Output)	50
4.30 Authentication Petri Net.	51
4.31 Analysis of the Authentication Petri Net	51
5.1 The Basic Model.	54
5.2 The Aspect Model.	55
5.3 The Weaver.	55
5.4 The Petri Net of the reservation modification case.....	56
5.5 The result of the verification of the Petri Net for the reservation modification case in TINA.....	56
5.6 The Basic Model.	57
5.7 The Aspect Model.	58
5.8 The Weaver.....	58
5.9 The Petri Net of the promotion addition case.....	59
5.10 The result of verification of Petri Net for the promotion addition case with TINA	59

List of Tables

4.1 Expresses the semantics of this approach.....	45
4.2 Comparison of Approaches for Transforming into Aspect-Oriented Diagrams.....	52
4.3 Comparison of Petri Net Transformation and Verification Approaches.....	53

General Introduction

1.Introduction:

The modeling of computer systems is a crucial step in their design and verification. Although it can be challenging, precise and rigorous modeling is essential to ensure that systems meet the needs and requirements of their application domain.

Among the many modeling languages available, UML (Unified Modeling Language) stands out as one of the most widely used and recognized. UML is a standardized modeling language widely employed in the field of software engineering and computer systems.

In our work, we used modeling with UML 2.0 diagrams. UML 2.0 comprises thirteen types of diagrams, each dedicated to representing specific concepts of a software system. These types of diagrams are divided into two main groups: six structural diagrams and seven behavioral diagrams. The sequence diagram is a type of interaction diagram in UML 2.0 that illustrates the chronological sequence of messages exchanged between different objects interacting within a system. Three types of analysis classes are used: dialogues, controls, and entities [Pascal Roque, 2008].

Despite the many advantages of Object-Oriented languages, the management of cross-cutting concerns remains a challenge, and the aspect-oriented paradigm emerges as a complementary solution. The aspect-oriented paradigm was initially proposed in 1997 by Kiczales and his team. It has since become a popular approach for modeling and managing cross-cutting concerns in software.

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. Petri nets are graphical and mathematical modeling tools used to represent and analyze discrete systems. The transition from aspect-oriented modeling to formal methods aims to overcome the limitations of semi-formal approaches, such as ambiguity and verification difficulty. Formal methods offer a rigorous framework for the specification and verification of systems.

To transform aspect-oriented sequence diagrams into Petri nets, it is essential to follow a methodology to ensure the accuracy of the resulting model. 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 such as liveness, boundedness, and safety.

2.Problematic:

Despite the use of all the power of object-oriented languages, certain limitations persist, notably the difficulty in effectively managing cross-cutting concerns and excessive code duplication. To address these drawbacks, the aspect-oriented modeling approach offers a promising solution. Aspect-oriented modeling allows the separation of functional and non-functional concerns in models, using an integration mechanism to achieve a unified model. Although UML is a powerful tool for modeling, its lack of formal semantics limits its ability to provide rigorous guarantees about the models. Formal methods, such as Petri nets, are often used as a solution to this problem. They offer a formal and mathematical

General Introduction

representation for specifying, analyzing, and verifying systems, ensuring more precise and reliable models. These resulting models will be verified using the TINA verification tool.

3. Contributions:

- In this memory, we propose two contributions:
The first is a transformation of object-oriented sequence diagrams into aspect-oriented sequence diagrams based on the graph transformation paradigm. Our approach involves proposing a meta-model, a graph grammar, and rules for the transformation from object-oriented to aspect-oriented. We use the AToMPM meta-modeling tool. The transformation is performed automatically by applying the graph grammar to the input models, which is the basic model (MB), plus the aspect model (MA), which automatically generates the output model.

Figure (1) illustrates the principle of transforming an object-oriented sequence diagram into an aspect-oriented sequence diagram.

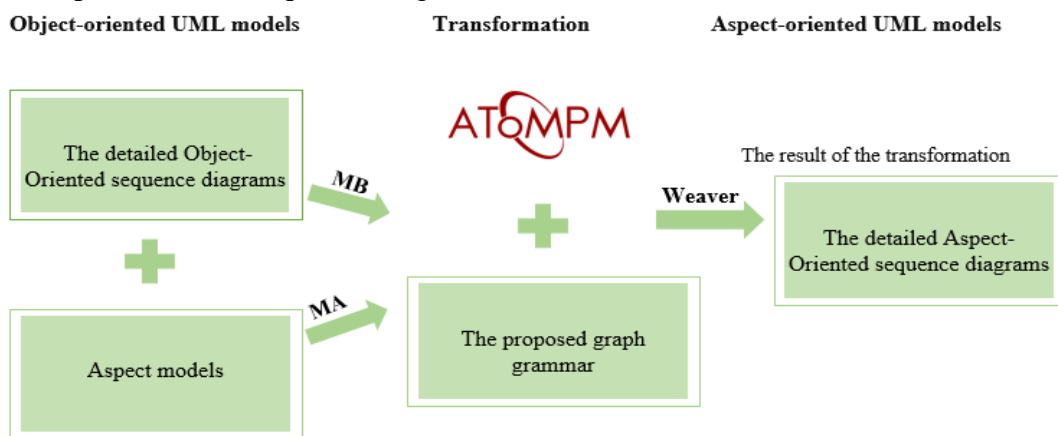


Figure 1: Principle of Transformation of a Detailed Object-Oriented Sequence Diagram into a Detailed Aspect-Oriented Sequence Diagram.

- The second contribution consists of the transformation of aspect-oriented sequence diagrams into Petri nets based on graph transformation. Our approach involves proposing two meta-models: one for aspect-oriented sequence diagrams and the other for Petri nets, along with a graph grammar and rules for the transformation from aspect-oriented to Petri nets. We use the AToMPM meta-modeling tool and have used the TINA tool for the verification of properties.

Figure (2) illustrates the principle of transforming an aspect-oriented sequence diagram into a Petri net.

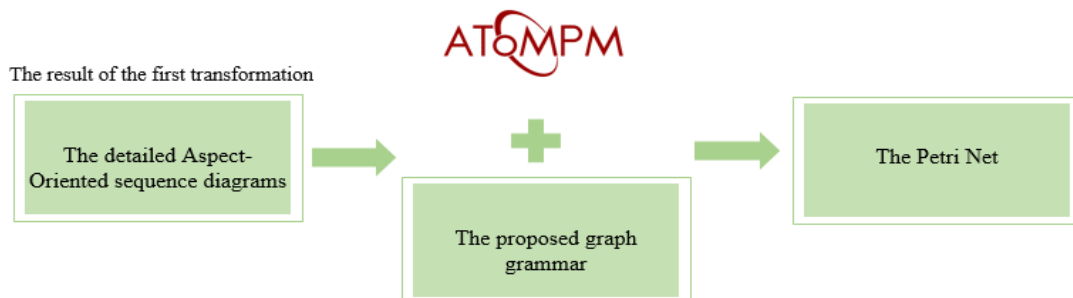


Figure 2: Principle of Transformation from a Detailed Aspect-Oriented Sequence Diagram to a Petri Net.

4.Memory organization:

We start with a general introduction addressing the importance of modeling computer systems for their design and verification. We emphasize the necessity of precise and rigorous modeling to ensure that systems meet the specific needs of their application.

Chapter 01: Aspect-Oriented Modeling

The first chapter is dedicated to aspect-oriented modeling. We will begin by defining the basic concepts of modeling, starting with the notions of model and modeling, as well as their importance in software development. Next, we will examine the different types of modeling, focusing on object-oriented modeling and its limitations. We will then discuss aspect-oriented modeling, describing its objectives, fundamental concepts, and associated techniques and technologies. Finally, we will discuss the main Inconvenients of this approach.

Chapter 02: Petri Nets

This chapter is dedicated to Petri nets. First, we will define Petri nets and present their basic concepts. Next, we will explore the state evolution of a Petri net, their properties, and the different categories of Petri nets. We will also discuss some specific types of Petri nets. Finally, we will examine the modeling of complex systems using Petri nets.

Chapter 03: Graph Transformation and verification

In the third chapter, we will discuss Model-Driven Architecture (MDA). We will begin by presenting the fundamental concepts of the MDA approach, including its architecture and various types. Next, we will introduce the tools available for applying MDA. We will then detail model transformation, a key concept in MDA. This will start with a definition of model transformation, followed by the different types of transformations, basic concepts of graphs, and graph transformation tools. In addition, we will introduce the AToMPM work environment. Finally, we briefly discussed the analysis techniques and tools for Petri nets, and we will focus on verification and validation techniques. The verification tool used is TINA.

Chapter 04: Proposed Approaches

In this chapter, we will present our two proposed approaches. The first approach aims to transform object-oriented sequence diagrams into aspect-oriented sequence diagrams. We have proposed a meta-model for the detailed sequence diagram, accompanied by a graph grammar that will be applied to the base model and the aspect model to obtain the aspect-oriented sequence diagram. In the second approach, we will take the result of the first transformation, which is an aspect-oriented sequence diagram, and apply a set of rules to transform it into a Petri net. Then, we will present the tool used to verify the obtained Petri nets. For both transformations, we will provide illustrative examples to explain the two approaches more clearly. The generated Petri nets from the example will be verified by TINA. Finally, we make a comparison between works related to our work and discuss this comparison.

Chapter 05: Case Studies

We detail two case studies illustrating our transformation approach, which include a reservation system in a travel agency and the management of a shopping center, using the AToMPM transformation tool, the generated Petri nets will be analyzed by TINA.

We conclude with a general conclusion that highlights the contributions of this memory and outlines future perspectives.

Chapter1

Aspect- oriented Modeling

Introduction:

Despite harnessing the full power of object-oriented languages, certain limitations persist, notably the challenge of effectively managing cross-cutting concerns and excessive code duplication. To address these drawbacks, the approach of aspect-oriented modeling offers a promising solution. In this chapter, we started our exploration with a precise definition of the model and modeling, emphasizing their importance in software development, along with the various types of modeling. After that, we briefly discussed the object-oriented modeling, highlighting its limitations in handling cross-cutting concerns. We then clarify aspect-oriented modeling before delving into its objectives. Subsequently, we examine its core concepts and move on to associated techniques and technologies. Finally, we identify the key drawbacks of this approach to better understand its potentials challenges.

1 The Model:

A model is an abstraction of a system constructed for a specific purpose. It is an abstraction in the sense that each model contains a limited set of information about a system. It is constructed for a specific purpose in that the information, its content is chosen to be relevant to its intended use. It is then said that the model represents the system. [khalfaoui,2014]

2 Modeling:

Modeling is an essential component of human activity, begins when humans attempt to understand, interpret various phenomena of the world, and make predictions. In general, modeling can be seen as "the action of modeling or the result of this action," where modeling consists of "designing, elaborating a model to understand, act, achieve a goal." In the context of computing, modeling is understood as the construction of a schematic representation (the model) in a formal or semi-formal language, based on specifications given in natural language. [Ludovic,2009]

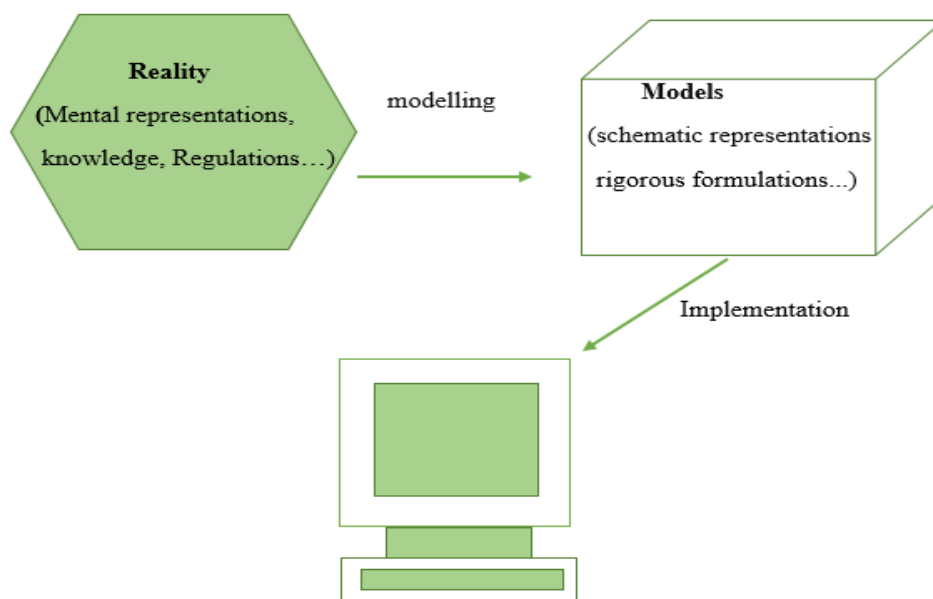


Figure 1.1: Representation of Modeling. [Dib et Saadaoui, 2023]

Chapter 1: Aspect-oriented Modeling

2.1 Goals of Modeling:

In order to understand better the functioning of the system and master its complexity with the assurance of its coherence, modeling is indispensable. Therefore, it allows: [Audibert,2008]

- Better distribution and automation of some tasks.
- Reduction of costs and delays.
- Ensuring a high level of quality and effective maintenance.

A model serves as a common, precise language known by all team members and it becomes as a result a privileged vector for communication.

2.2 Types of Languages:

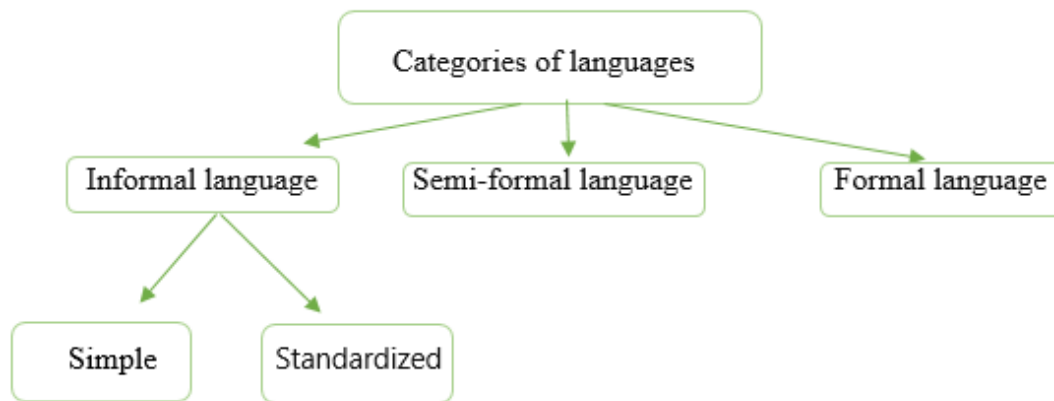


Figure 1.2: Classification of Languages or Methods.

Semi-Formal Language: A language that has a defined syntax to specify the conditions under which constructions are permitted, such as Entity-Relationship Diagram, Object Diagram.

Formal Language: A language that has rigorously defined syntax and semantics. There exists a theoretical model that can be used to validate a construction, such as Petri nets [Bahri,2010].

Informal Language: Informal language is a familiar way of communicating among people [Dib et Saadaoui,2023]. It includes two types: [Bahri, 2010]

- **Simple Language:** A language that does not have a complete set of rules to restrict a construction, such as natural language.
- **Standardized Language:** A language with a structure, format, and rules for composing a construction, such as structured text in natural language.

3 Object-Oriented Modeling:

Object-oriented modeling is often approached after an initial exposure to object-oriented programming (using a first programming language). Learners thus have basic knowledge of concepts such as classes, objects, encapsulation, and inheritance that they refine during the learning of OOM (Object-Oriented Modeling). [Ludovic,2009]

Chapter 1: Aspect-oriented Modeling

The object-oriented approach is classified in the category of semi-formal modeling. It represents a way of conceptualizing problems by applying models organized around real-world concepts. Object orientation views software as a collection of dissociated objects, and consequently, its fundamental concept is the object, which combines both a data structure and behavior. The functionality of the software emerges from the interaction between the different objects that constitute it. [Aouag,2014]

4 Limitations of the Object-Oriented Approach:

The limitations of the object-oriented approach are as follows: [Aouag,2014]

- The resolution model is difficult to read and understand, and it may lead to errors.
- Objects cannot handle deletion.
- Duplication of cross-cutting functionalities also exists in the application model.
- The model is neither structured nor organized coherently.
- Reusing models is complex.

Therefore, aspect-oriented emerges to improve and address the limitations of the object-oriented paradigm.

5 Definition of Aspect-Oriented Modeling:

The definition of aspect in the analysis and design phases has attracted the interest of several researchers. Several works have been proposed to define the aspect (as an entity) in the analysis and design phases. They focused on separating cross-cutting concerns (aspects) throughout the software development lifecycle, particularly at the design phase. Most existing approaches are based on extending the UML meta-model to define their aspect concepts and relationships. UML, the standard object modeling language, is most often used for defining an aspect modeling language. [LAOUAR,2013]

5.1 The Purpose of Aspect-Oriented Modeling:

Since its early years, the aspect-oriented approach has been used slowly in programming languages during the coding stage. However, the aspect-oriented paradigm now extends to upstream phases of software development and it is no longer limited to programming. Nowadays, at every phase of software development—requirements analysis, design, and even implementation—aspect-oriented approaches are available. Transitioning between phases while preserving previously known aspects remains a major, understudied challenge. An iterative approach focused on concerns aims to automate the transformation from an object-oriented requirements model to an aspect-oriented design model. [Dib et Saadaoui,2023]

5.2 Basic Concepts:

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. [Boubendir,2011]

There are two types of concerns: cross-cutting concerns or aspect concerns, and non-cross-cutting concerns or base concerns: [Aouag,2014]

- ❖ **Base Concern:** It represents a non-cross-cutting (functional) concern with traditional approaches such as object-oriented approach. [Zerara et Megrou,2020]

Chapter 1: Aspect-oriented Modeling

- ❖ **Aspect Concern:** It represents a cross-cutting (non-functional) concern with traditional approaches such as the object-oriented approach. [Zerara et Megrous,2020]

Aspect: Analogous to the object-oriented approach that provides explicit mechanisms for encapsulation and inheritance of objects, the aspect-oriented approach provides first-class explicit mechanisms that explicitly capture and encapsulate the structure of cross-cutting concerns. We call a module encapsulating a cross-cutting concern: an aspect. [Boubendir,2011]

An aspect is a software entity that captures a non-functional functionality. [Boualita et Laggoune,2021]

Join Point (PJ): A location in the model where advices should be inserted. [Aouag,2014]

There are several types of join points, depending on what the developer wishes to intercept. Thus, a join point can occur for: [Otmame Rachedi, 2015]

- A class
- An interface
- Method execution
- Access to a class variable
- Execution of a code block
- Execution of a language keyword (condition, loop, ...)

Point cut (PC): A point cut is used to select join points by specifying positions in the primary models to which the corresponding advice for the cross-cutting concern should be applied. [LAOUAR,2013]

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. [Otmame Rachedi, 2015]

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. [Boubendir,2011]

So, an **Aspect = Join points + Advice**. [Aouag,2014]

Point cut = Σ join points. [Aouag,2014]

In Figure (2.3), we present the aspect-oriented modeling process such as e1, e2, and e3 are states, are aspects, e1 and e2 in the aspect model are join points.

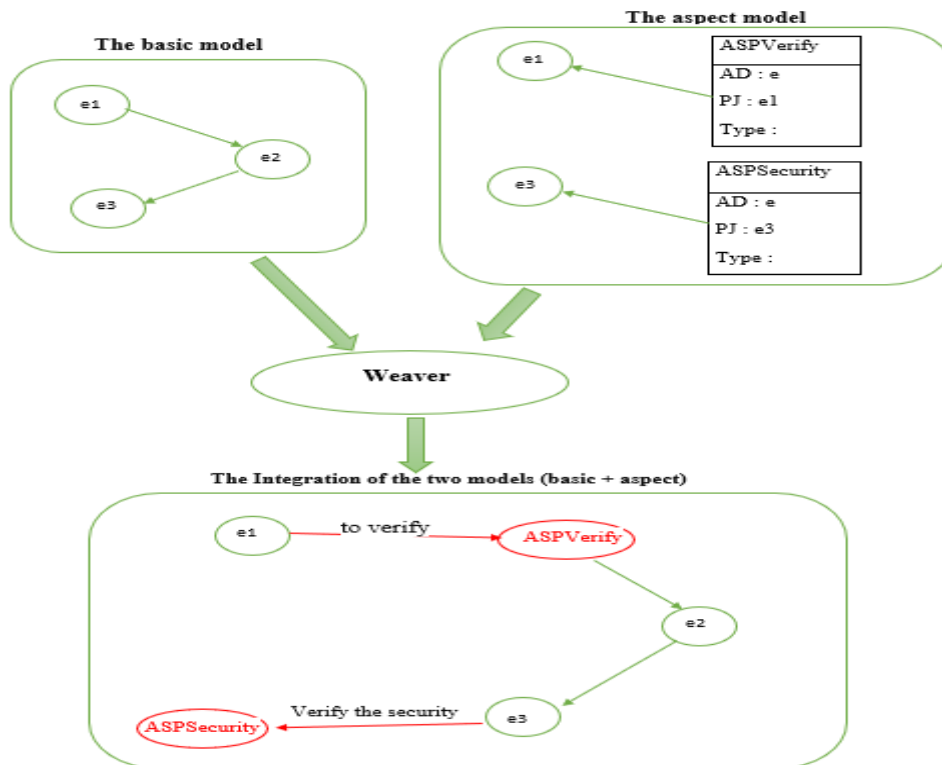


Figure 1.3: The Aspect-Oriented Modeling Process.

6 Aspect-Oriented Approaches:

6.1 Aspect oriented Requirements Engineering:

These approaches provide a representation of cross-cutting concerns present in requirement artifacts. They clearly acknowledge the importance of identifying and addressing cross-cutting concerns early on. Cross-cutting concerns can be both non-functional and functional requirements, also their early identification enables early analysis of their interactions. These approaches focus on the principle of composing all concerns to have the complete system under construction. This allows understanding the interactions and trade-offs between concerns. Composing requirements not only allows examining requirements as a whole but also detecting potential conflicts early on, so that the right decisions can be made for the next steps. [Boubendir,2011]

6.2 Aspect oriented Architecture:

An architectural aspect is an architectural module that has a significant influence on other architectural modules. Aspect-oriented architecture design approaches therefore describe the steps for identifying architectural aspects and their intertwined components. This information is used to redesign a given architecture while making the architectural aspects explicit. This is different from traditional approaches where architectural aspects are implicit information in the architecture specification. [Aouag,2014]

6.3 Aspect oriented Design:

These design approaches focus on the explicit representation of cross-cutting concerns using suitable design languages. Initially, designers simply used object-oriented methods and languages (such as UML) for designing their aspects. This proved challenging since

Chapter 1: Aspect-oriented Modeling

UML was not designed to provide constructs for describing aspects. The main contribution of aspect-oriented design was therefore to provide designers with explicit means to model systems by aspect. [Boubendir,2011]

In our work, we are interested in this approach.

6.4 Aspect oriented Programming:

Aspect orientation manifests at the programming level through aspect-oriented programming languages. Most of these aspect-oriented languages are existing (object-oriented) languages extended with aspect-oriented features to represent aspects, express point cuts, join points, and advices, etc. For example, AspectJ is based on Java. [Aouag,2014]

6.5 Verification of Aspect oriented Programs:

New challenges in software verification and validation techniques arise in the aspect-oriented approach to ensure that the desired functionality is satisfied by the system. Aspects could potentially compromise the reliability of a system to which they are woven, and may invalidate essential properties of the system that were correct before aspect weaving. To ensure the accuracy of aspect-oriented software, there is extensive research on the use of formal methods and testing techniques specifically tailored to aspects. [Boubendir,2011]

6.6 Aspect oriented Middleware:

Although middleware is not a step in the software lifecycle, it is an important and extensive area for aspect-oriented ideas. Many software developers have adopted middleware approaches to aid in the construction of large-scale distributed systems. Middleware facilitates the development of distributed software systems by supporting heterogeneity, hiding distribution details, and providing a set of specific services for a common domain. [Aouag,2014]

7 Inconvenients of Aspect-Oriented Modeling:

Aspect-oriented modeling has limitations that make its application elegant in all possible problem situations. [Aouag,2014]

-Transaction management, through cross-cutting, presents challenges to be isolated into a distinct aspect.

- AOM (Aspect-Oriented Modeling) is mostly appropriated for a large-scale software development projects.

-In distributed systems, aspect-oriented modeling poses particular challenges regarding testing and debugging. This it is due to side effects emanating from the dynamic injection of the model, which can in the worst-case scenarios, lead to semantic ambiguities in the control flow of an aspect-oriented model.

-Different aspects can effectively harm even the join points in weaving. Thus, AOM may violate the principle of encapsulation, albeit in a fairly systematic and well-controlled manner.

Chapter 1: Aspect-oriented Modeling

Conclusion:

In this chapter, we have provided an explanation of a model and modeling, followed by an overview of different types of modeling. We then explored object-oriented modeling and its limitations. Subsequently, we presented aspect-oriented modeling, starting with a precise definition and not neglecting its main objective. We moved into the basic concepts of aspect-oriented modeling, exploring the various techniques and technologies used in this field. Finally, we identified the main drawbacks associated with this approach. Aspect-oriented models lack's behavior verification tools, which are available in Petri nets, defined in the next chapter.

Chapter2

Petri Nets

Introduction:

Petri nets, which allow both a graphical representation and a formal mathematical description of systems, were introduced by Carl Adam Petri in his thesis entitled "Communication with Automata" at the University of Darmstadt in Germany in 1962 (Tran, 2005). In this chapter, we will explore a definition of Petri nets (PNs) and their basic concepts. Then, we will present the state evolution of a PN, their properties, the different categories of PNs. Finally, we will conclude with some specific types of PNs.

1 Petri nets:

The Petri net is a graphical and mathematical tool for modeling and analyzing discrete systems, particularly concurrent systems, parallel, non-deterministic, etc. As a graphical tool, it helps us to easily understand the modeled system, and furthermore, it enables us to simulate dynamic and concurrent activities. With its role as a mathematical tool, it allows us to analyze the modeled system through graph models and algebraic equations. [Hettab,2009]

2 Basic concepts of Petri nets:

2.1 Formal Definition:

Formally, a marked Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where: [GUERROUF,2011]

- ❖ $P = \{P_1, P_2, \dots, P_m\}$ is a non-empty finite set of places P .
- ❖ $T = \{t_1, t_2, \dots, t_n\}$ is a non-empty finite set of transitions T .
- ❖ $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs where:
 - $(P \times T)$ is the arc going from P to T .
 - $(T \times P)$ is the arc going from T to P .
- ❖ $W: F \rightarrow \{1, 2, 3, \dots\}$ is a weight function where:
 - $W(P, T)$: "Pre (p, t)" is the weight of the arc going from P to T .
 - $W(T, P)$: "Post (p, t)" is the weight of the arc going from T to P .
- ❖ $M_0: P \rightarrow \{0, 1, 2, 3, \dots\}$ is the initial marking.
- ❖ $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

2.2 Graphical Definition:

A Petri net is a bipartite directed valued graph. It has two types of nodes: [CHERIEF et MELIANI,2020]

- ✚ **Places:** graphically represented by circles. Each place contains an integer number (positive or zero) of tokens. These tokens are represented by black dots.
- ✚ **Transitions:** graphically represented by a rectangle or a bar. A transition that has no input place is called a source transition, and a transition that has no output place is called a sink transition.

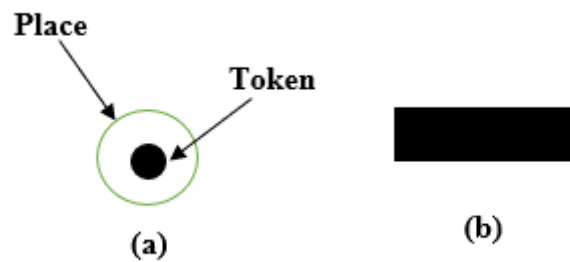


Figure 2.1: (a) Graphical representation of a place with one token, (b) transition. Places and transitions are connected by directed arcs where:

- An arc connects either a place to a transition or a transition to a place, but never a place to a place or a transition to a transition.
- Each arc is labeled with a value (or weight), which is a positive integer. An arc with a weight of k can be interpreted as a set of k parallel arcs. An arc without a label is an arc with a weight equal to 1.

Figure 2.2 illustrates the graphical notation of a Petri net:

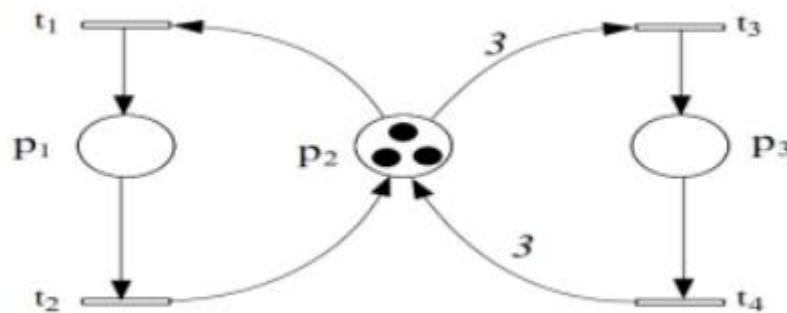


Figure 2.2: Example of a Petri net. [CHERIEF et MELIANI,2020]

2.3 Marking of a Petri net:

A PN is a bipartite graph, i.e. places and transitions alternate on a path made up of consecutive arcs. It is compulsory for each arc to have a node at each of its ends. (From a node k , place or transition, to a node h , transition or place, there is at most one arc). [David et Alia,2005]

Figure 2.3(a) represents a PN with 7 places, 6 transitions and 15 directed arcs. The set of places of a PN will be called P and its set of transitions will be called T . For the example in question, we thus have $P = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7\}$ and $T = \{T_1, T_2, T_3, T_4, T_5, T_6\}$. [David et Alia,2005]

Place P_3 is said to be upstream or an input of transition T_3 because there is a directed arc from P_3 to T_3 . Place P_5 is said to be downstream or an output of transition T_3 because there is a directed arc from T_3 to P_5 . In a similar way, a transition is said to be an input (upstream) or an output (downstream) of a place. A transition without an input place is a source transition. A transition without an output place is a sink transition. [David et Alia,2005]

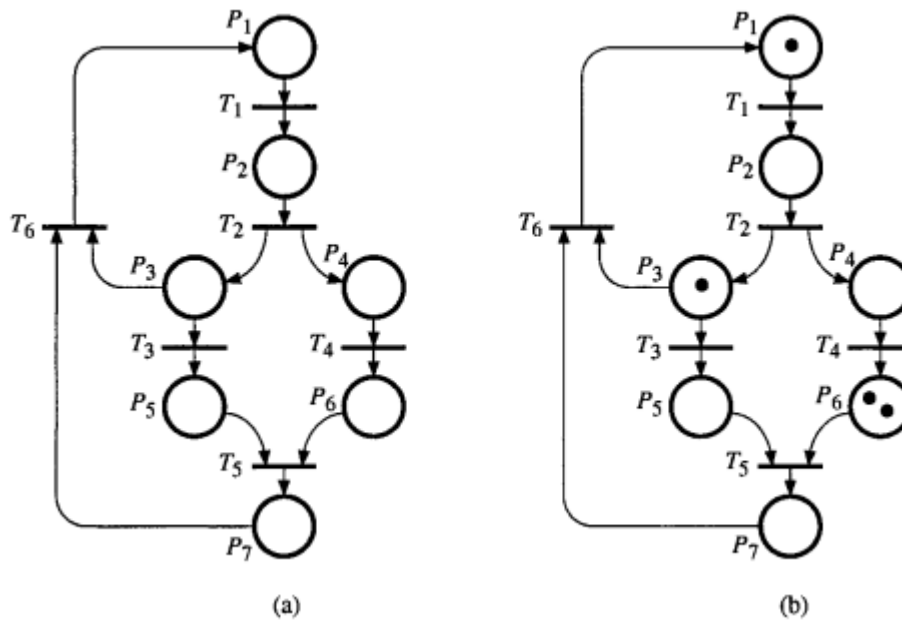


Figure 2.3: Example of a Petri net. [David et Alia,2005]

Figure 2.3(b) represents a marked Petri net. Each place contains an integer (positive or zero) number of tokens or marks. The number of tokens contained in a place P_i , will be called either $m(P_i)$ or m_i . For the example in question, we have $m_1 = m_3 = 1$, $m_6 = 2$ and $m_2 = m_4 = m_5 = m_7 = 0$. The net marking, m , is defined by the vector of these markings, i.e., $m = (m_1, m_2, m_3, m_4, m_5, m_6, m_7)$. The marking of the PN in Figure 2.3(b) is thus $m = (1, 0, 1, 0, 0, 2, 0)$. The marking defines the state of the PN, or more precisely the state of the system described by the PN. The evolution of the state thus corresponds to an evolution of the marking, an evolution which is caused by firing of transitions, as we shall see. [David et Alia,2005]

2.4 Matrix Representation of a Petri Net:

The matrix representation of a Petri net is used to facilitate the analysis and verification of Petri net models. Working with a graphical representation of a Petri net model is a challenging task compared to a matrix representation. [kerkouche,2011]

Consider a Petri net $R = (P, T, W)$ where $P = \{p_1, p_2, \dots, p_m\}$ and $T = \{t_1, t_2, \dots, t_n\}$. The pre-condition matrix, denoted as pre , is a $m \times n$ matrix with coefficients in N , such that $pre(i, j) = W(p_i, t_j)$, indicating the number of marks needed in place p_i for transition t_j to become enabled. Similarly, the post-condition matrix, denoted as $post$, is a $n \times m$ matrix such that $post(i, j) = W(t_j, p_i)$ containing the number of marks deposited in p_i when transition t_j fires. The matrix $C = post - pre$ is termed as the incidence matrix of the net (where m represents the number of places in a Petri net and n represents the number of transitions).

The marking of a Petri net is represented by a vector of dimension m with coefficients in N . The firing rule of a Petri net is defined as: $M'(p) = M(p) + C(p, t)$. The matrix representation of the Petri net in the following figure:

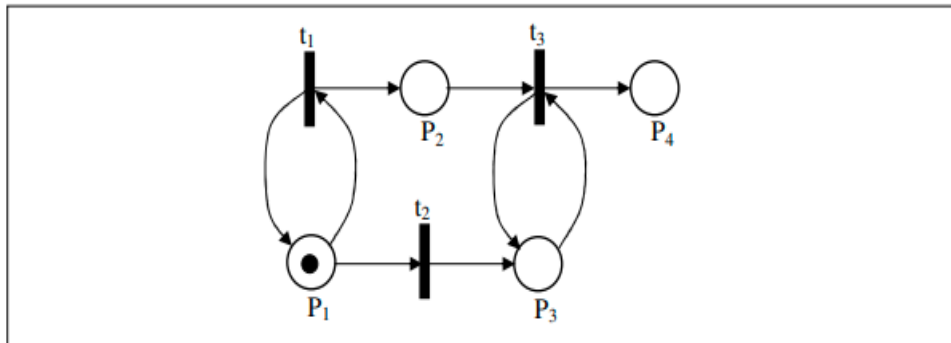


Figure 2.4-a: Example of a Marked Petri Net. [kerkouche,2011]

$$\begin{array}{l}
 P = \{p_1, p_2, p_3, p_4\} \\
 T = \{t_1, t_2, t_3\}
 \end{array}
 \quad
 \text{Pré} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}
 \quad
 \text{Post} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{C} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}
 \quad
 \text{M} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

La matrice d'incidence Le vecteur de marquage M

Figure 2.4-b: Incidence Matrix. [kerkouche,2011]

3 The Evolution of States in a Petri Net:

The evolution of the Petri net state corresponds to a marking evolution. The tokens, which indicate the state of the network at a given time, can move from one place to another through a firing or transition. In the case of networks known as simple arc or with equal weights of 1 (Figure 2.5), firing a transition involves removing one token from each input place of the transition and adding one token to each output place of that transition. [BAHRI,2010]

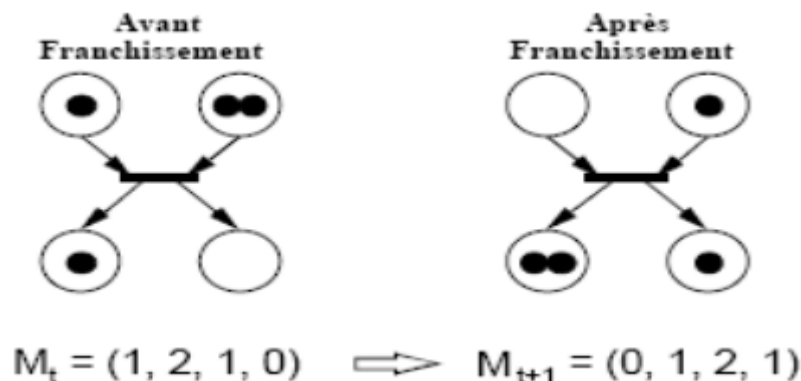


Figure 2.5: The Evolution of States in a Petri Net. [BAHRI,2010]

In general, the evolution of states in a simple marked Petri net follows the following rules: [BAHRI,2010]

- A transition is enabled or fireable when each of its input places has at least the number of tokens corresponding to the weight of the arc connecting it to the transition.

Chapter 2: Petri Nets

- The network can evolve only by firing one transition at a time, selected from among all those enabled at the time of selection.
- Firing a transition is indivisible and null duration

These rules introduce some indeterminism in the evolution of Petri nets, as they can pass through different states whose occurrence is conditioned by the choice of fired transitions. This functioning reflects well the real situations, where there is no priority in the succession of events.

4 Petri Net Properties:

4.1 Liveliness:

A transition t_j is said alive if, for any reachable marking, we can construct a firing sequence that includes transition t_j . A Petri net is said alive if all of its transitions are alive. [HAMRI, 2017]

4.2 Boundedness:

The boundedness of PN expresses that the number of states that can be taken by the system modeled by this Petri net are finite. When the Petri net is unbounded, the number of states is infinite, and this is due to the fact that some parameters of this system are unbounded. For example, if we model a queuing system using Petri nets, then the parameter "Queue size" can be unlimited, which introduces the unboundedness of the model. [MEDJANI,2020]

✓ **K-bounded place:**

A place $p \in P$ is called k -bounded for an initial marking M_0 if and only if: $\exists k \in \mathbb{N}, \forall M_0 \in A(M_0), M_0(p) \leq k$.

Where: $A(M_0)$ is the set of reachable markings. If $k = 1$, we say that place p is safe. [MEDJANI,2020]

✓ **K-bounded Petri net:**

A Petri net is called k -bounded (or bounded) for an initial marking M_0 if and only if all its places are k -bounded. The 1-bounded Petri net is called a binary Petri net (safe).

[MEDJANI,2020]

4.3 Reachability:

The verification of reachability in a marked Petri net involves determining whether a marking M_k can be reached from a marking M_0 . Marking M_k is reachable from M_0 if there exists a firing sequence that leads from marking M_0 to marking M_k . [Hamri,2017]

4.4 Blocking:

A marking M in a network (N, M_0) is called a "deadlock" marking if no transition is enabled from M . A network is said to be deadlock-free if every reachable marking from M_0 is not a "deadlock" marking. [Guerrouf,2011]

The marked Petri net represented in Figure 6.2 for blocking the marking:

$$M_4 = [1, 0, 0, 3]$$

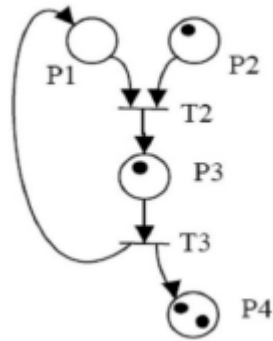


Figure 2.6: Example of Petri Net Blocking. [Guerrouf,2011]

5 Categories of Petri Nets:

a) **State Graph:** In this case, each transition has only one input place and one output place. [HADDOUCHE et DAHAMNA,2022]

Example: Transitions T1, T2, T3, T4, and T5 have one input place and one out place.

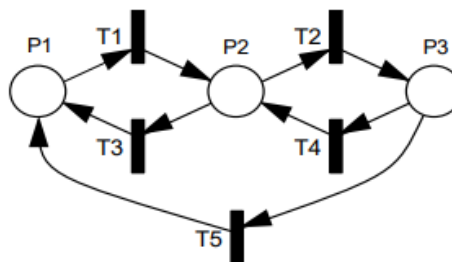


Figure 2.7: State Graph. [SAGGADI, 2007]

b) **Event Graph:** A Petri net is an event graph if and only if each place has exactly one input transition and one output transition. [LAOUAR,2013]

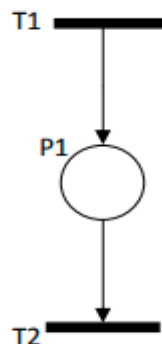


Figure 2.8: Event Graph. [LAOUAR,2013]

c) **Conflict-free Petri Net:** Each place is associated with only one output transition. [HADDOUCHE et DAHAMNA,2022]

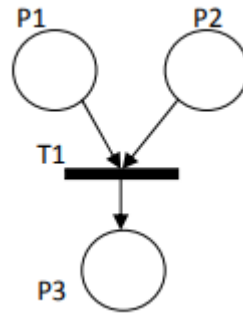


Figure 2.9: conflict-free PN. [LAOUAR,2013]

- d) **Free Choice Petri Net:** A Free Choice Petri Net is a network in which for any conflict $(K = (P_i, \{T_1, T_2, \dots, T_n\}))$ none of the transitions T_1, T_2, \dots, T_n has any other input place than P_i . [kerkouche,2011]

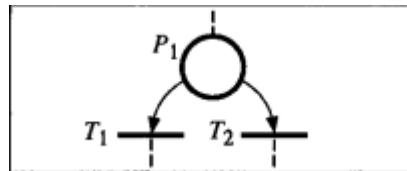


Figure 2.10: Free Choice PN. [David et Alia,2005]

- e) **Simple Petri Net:** is a Petri Net in which each transition can be involved in at most one conflict. In other words, if a transition T_1 and two conflicts $(P_1 \{T_1, T_2, \dots\})$ and $\{P_2, \{T_1, T_3, \dots\}\}$ exist, then the Petri Net is not simple. [Rene et Hassane, 2005]

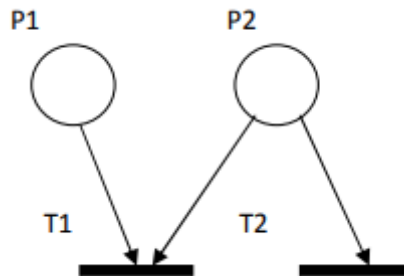


Figure 2.11: Simple PN. [LAOUAR, 2013]

- f) **Pure Petri Net:** In a Petri Net, a transition is pure if it has no place that acts both as input and output. If all transitions in the Petri Net are pure, then the Petri Net is pure. [SAGGADI, 2007]

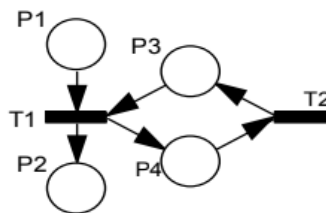


Figure 2.12: Pure PN. [SAGGADI, 2007]

- g) **Limited Capacity Petri Net:** In ordinary Petri Nets, the capacity of places is not limited. Here, a capacity is assigned, defined by a positive integer associated with places. Thus, the firing of a transition is conditioned by the capacity of the places downstream. This limitation can illustrate the capacity of a stock, for example. [HAOUES,2006].

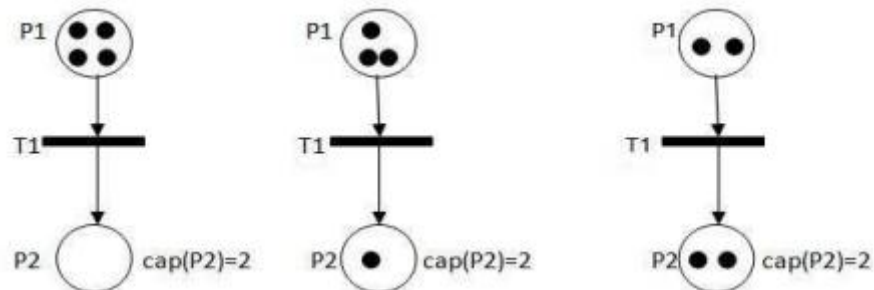


Figure 2.13: Limited Capacity Petri Net. [LAOUAR, 2013]

- h) **Priority Petri Net:** In this type of network, if we reach a marking where multiple transitions are enabled, we must fire the transition with the highest priority. In the following example, a Petri Net with priorities is illustrated. [LAOUAR,2013]

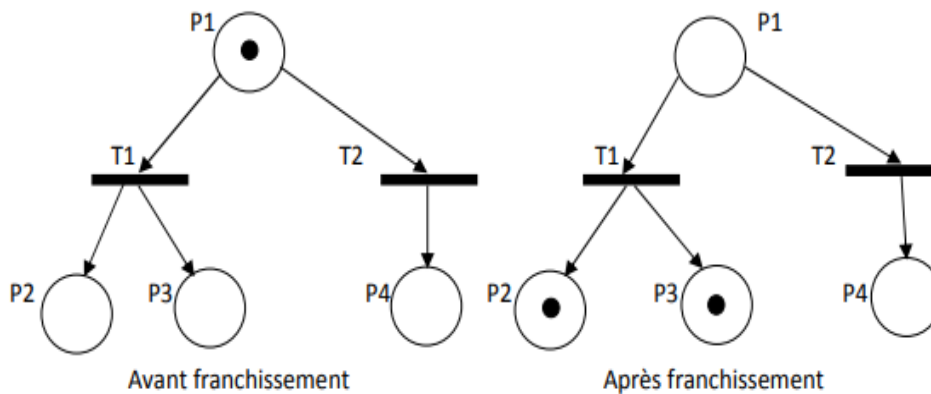


Figure 2.14: Priority PN. [LAOUAR, 2013]

- i) **Inhibitory Arcs in Petri Nets:** An inhibitory arc is a directed arc that starts from a place and ends at a transition (rather than the reverse). Its end point is marked by a small circle. The presence of an inhibitory arc between a place P_i and a transition T_j signifies that transition T_j is enabled only if place P_i contains no tokens. The firing of transition T_j involves removing a token from each place upstream of the transition except place P_i , and adding a token to each place downstream of the transition. [Aouag,2023]

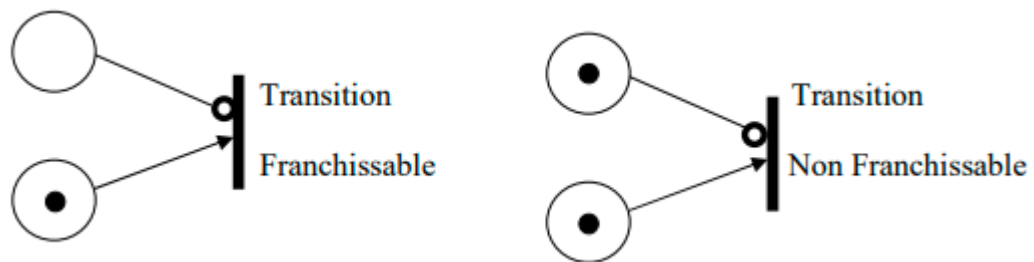


Figure 2.15: Inhibitory Arcs in Petri Nets. [Aouag,2023]

6 The types of PN:

6.1 Ordinary Petri Net:

The arc with weight $n = 1$ is an ordinary arc. So, if all arcs in a network are ordinary, the network will be called ORDINARY. The formal description of this model is defined by a 6-tuple: $P N = \langle P, T, M_0, A, Pre, Post \rangle$ with: [CHERIEF et MELIANI,2020]

- P , a finite set of places,
- T , a finite set of transitions,
- M_0 , the initial marking of the network,
- A , a finite set of arcs such that $P \setminus T = P \setminus A = T \setminus A = \emptyset$
- Pre , indicates how many tokens are consumed from a place to a transition.
- $Post$, indicates how many tokens are produced by a transition into the downstream place.

6.2 Generalized Petri Net:

In a Petri Net, each arc is assigned with a weight (positive integer associated with arcs). This weight indicates the number of tokens consumed or produced when a transition is fired. These weights are found in the incidence matrix. This is a simplification by aggregation of an ordinary Petri Net [HAOUES,2006]

6.3 Timed Petri Net:

In this model of Petri net, the duration of an activity is explicitly integrated. Timing can concern either the places (P-timed Petri nets) or the transitions (T-timed Petri nets) depending on the modeled events. [Bahri,2010]

6.4 Colored Petri Net:

The modeling of large systems is facilitated by the use of colored Petri nets. They are of great interest in modeling certain complex systems. The principle consists of representing information by place/mark sets. Each place's marks are associated with a color (or identifier). The transition of these marks can be performed in several ways depending on the colors associated with transitions. The relationship between transition colors and colored markings is defined by functions associated with arcs. [Aouag,2023]

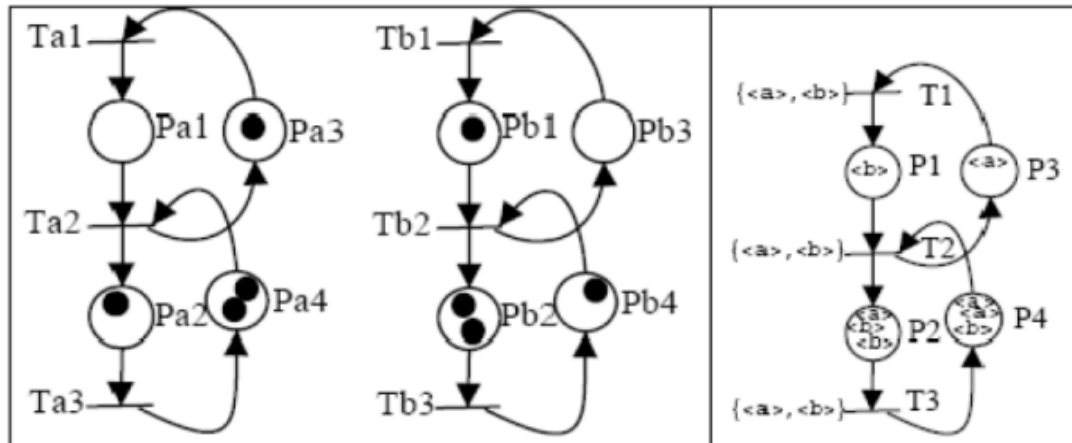


Figure 2.16: Petri Net (left) and Colored Petri Net (right).[Hettab, 2009]

6.5 Continuous Petri Net:

In a continuous Petri Net, the markings of places are no longer integers but positive real numbers. This type of Petri Net offers performance analysis in terms of through put. It is highly useful when the number of markings in a classical Petri Net becomes too large or to represent continuous processes. [HAOUES,2006]

Conclusion:

In this chapter, we have provided an explanation of Petri nets, followed by their basic concepts. We then explored the various properties and state evolution of Petri nets. Delving into the categories of Petri nets. Finally, we examined their different types of PNs. We emphasized that aspect-oriented models have shortcomings in terms of behavior verification tools, unlike Petri nets, whose tools are defined in the next chapter. The next chapter will focus on graph transformation.

Chapter3

Graph

Transformati-

on and

verification

Introduction:

Model-Driven Architecture (MDA), proposed and endorsed by the OMG (Object Management Group), is based on the use of models at different stages of the application development cycle. This approach primarily relies on high-level abstract representations of the system, which are the models. It emphasizes automating the system development process, including verification, which involves analyzing the system properties, thus ensuring the reliability of the modeled system. This approach enables formal verifications to be conducted and ensures the correct behavior of the system.

This chapter aims to provide fundamental concepts about Model-Driven Architecture, explaining the MDA approach and illustrating MDA architecture as well as its different types. Additionally, we discuss the available tools for MDA. Model transformation is a key concept in MDE (Model-Driven Engineering). We present a definition along with the various types of existing model transformations. Then, we are interested into graph transformation, starting with some basic concepts about graphs and graph transformation tools. Then, we introduce the ATOMPM tool. Following that, we conduct an analysis of Petri nets focusing on verification and validation. Finally, we present Petri Net analysis tools and concentrate on the Tina tool.

1 Model Driven Architecture (MDA):

Model-Driven Architecture, or MDA, is a software development approach proposed and endorsed by the OMG. It is a specific variant of Model-Driven Engineering. [ElMansouri,2009]

1.1 Basic Concepts of MDA:

- **System:** A system is a set of elements interacting with each other according to a certain number of principles and rules with the aim of achieving a goal. [Aouag,2023]
- The boundary of a system determines its membership criteria.
- The environment refers to the part of the world outside the system.
- Systems are often hierarchically organized using subsystems.
- ❖ **Model:** A model is an abstraction of reality; it emphasizes certain aspects of the system while ignoring others (it is a simplified representation). Its purpose is to facilitate a simpler study within a controlled context other than the actual context. [Hachichi,2013]
- ❖ **Meta-model:** A meta-model is both a model that defines the language of expression or the structure of a model, and a specification of the syntax and semantics of a system. [Aouag,2014]
- ❖ **Meta-Meta-Model:** A specification language for meta-models. [Zerara et Megrous,2020]

1.2 The MDA Approach:

- ✚ **The CIM (Computational Independent Model):** This is the requirements model. It describes the functional needs of the application independently of the details related to its implementation. The technical independence

Chapter 3: Graph Transformation and verification

of this model allows it to maintain its relevance over time. It is only modified if knowledge or business needs change. [Khelfaoui,2014]

- ✚ **The PIM** (Platform Independent Model) corresponds to the specification of the "business" part of an application, in accordance with a computer analysis seeking to meet business needs independently of the implementation technology. [Hettab,2009]
- ✚ **The PSM** (Platform Specific Model) is the most delicate phase of MDA. Code generation can begin from the analysis and design models. The main difference between a code model and an analysis and design model lies in the fact that the code model is tied to a specific platform. [Aouag,2014]

There are model transformations from the system to other models of the same system, As example: [Zerara et Magrous,2020]

- Transformations from CIM models to PSM
- Transformations from CIM models to PIM
- Transformations from PIM models to PSM
- Transformations from UML 2.0 diagrams to Petri nets

The following figure provides an overview of the model transformation process in the MDA approach:

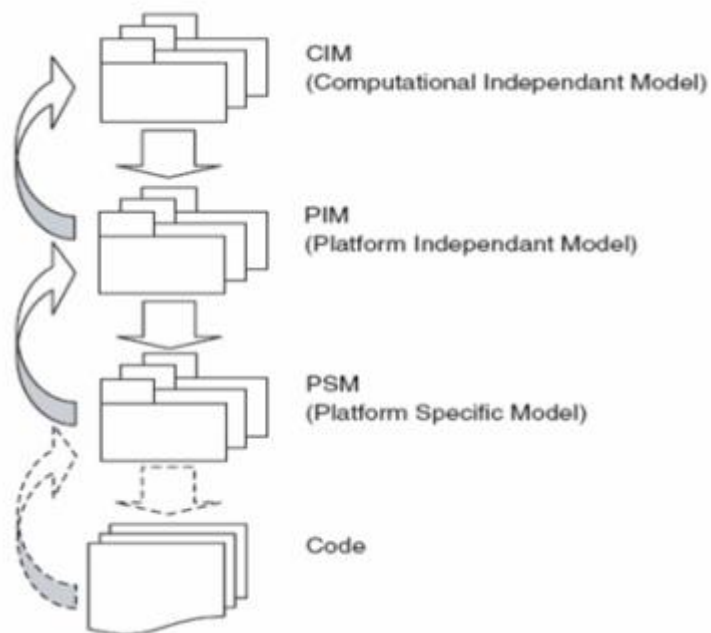


Figure 3.1: The model transformation process in the MDA approach. [Aouag,2014]

1.3 Model Driven Architecture:

The OMG has established four-level abstraction architecture as a general framework for integrating meta-models, based on the MOF (Meta-Object Facility) as depicted in Figure 3.2.

Chapter 3: Graph Transformation and verification

In this architecture, models at adjacent levels are connected by an instantiation relationship. [Bahri,2010]

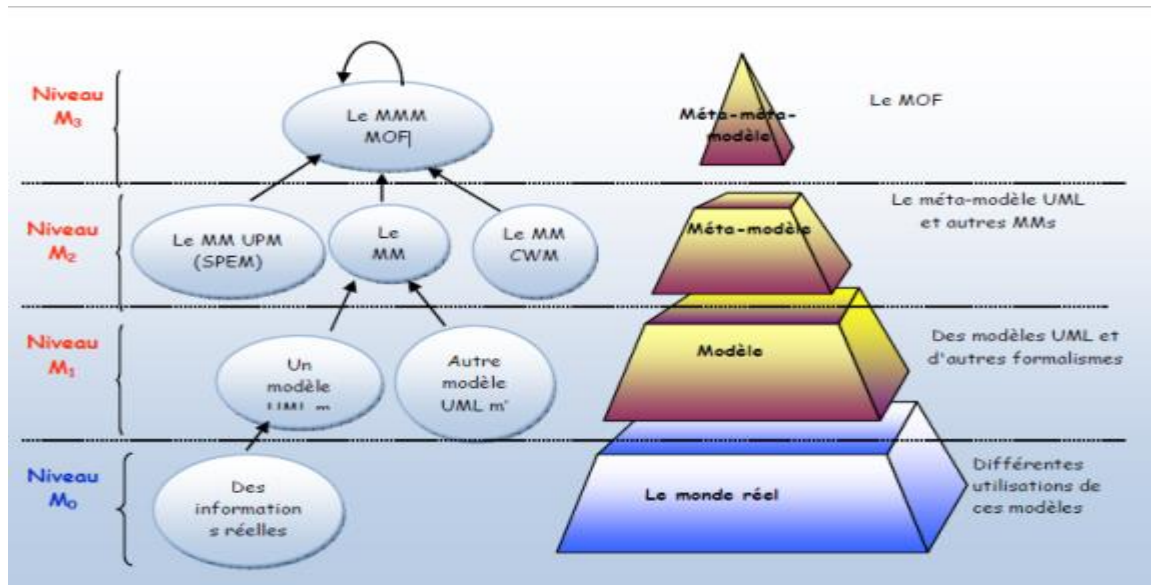


Figure 3.2: The four abstraction levels for MDA. [Dib et Saadaoui, 2023]

The four abstraction levels for MDA are as follows: [Bahri, 2010]

- **Level M0:** Instance level of models. It defines information for modeling real-world objects.
- **Level M1:** This level represents all instances of a meta-model. Models of M1 level must be expressed in a language defined at level M2. UML is an example of models of M1 level.
- **Level M2:** This level represents all instances of a meta-meta-model. It consists of languages for specifying information models. The UML meta-model, described in the UML standard and defining the internal structure of UML models, belongs to level M2.
- **Level M3:** This level defines a single language for specifying meta-models. The MOF, a reflective element of M3 level, defines the structure of all meta-models of M2 level.

1.4 Types of MDA Approaches:

According to the classification by Czarnecki and Helsen, model transformations can be grouped into two main categories: 'Model to Model' transformations and 'Model to Code' transformations (Figure 3.3). [BENDIAF, 2018]

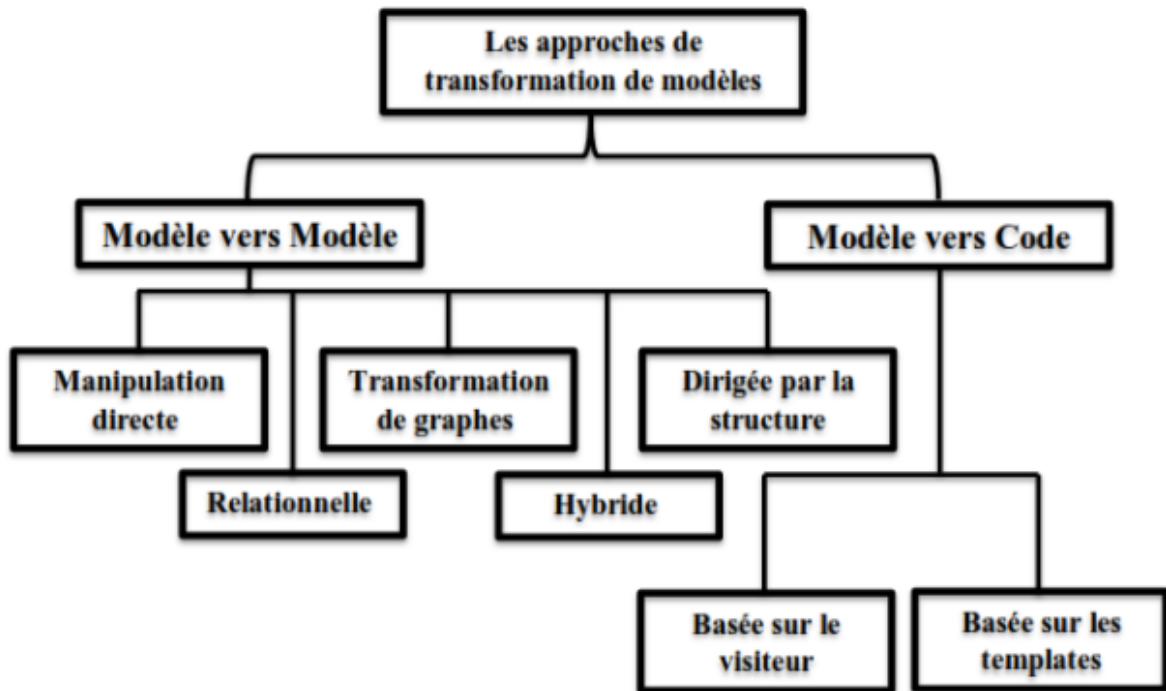


Figure 3.3: The model transformation approaches. [BENDIAF, 2018]

- ✚ **Model to Code Transformations:** There are two different approaches for model to code transformations: visitor-based approaches or template-based approaches.
 - **Visitor-based approaches:** involve traversing the model by adding elements (visitor's mechanisms) that reduce the semantic gap between the model and the target programming language. Code is generated by traversing the enriched model to create a text stream.
 - **Template-based approaches:** are currently the most widely used. The target code contains snippets of meta-code used to access information from the source model. The majority of commonly available MDA tools support this principle of code generation from models. Among the tools based on this principle are: OptimalJ, XDE (Extended Development Environment) (which also provide model-to-model transformation), JET (Java Emitter Templates), ArcStyler, AndromDA (Andromeda Model-Driven Architecture) (a code generator that relies on open technology such as Velocity for template writing), Acceleo, and XPand.
- ✚ **Model-to-Model transformations:** involve converting a source model into a target model, which can be instances of different meta-models. They offer more modular and easier-to-maintain transformations. In cases where there is a significant abstraction gap between PIMs and PSMs, it is easier to generate intermediate models than to go directly to the target PSM. Intermediate models can be useful for optimization or debugging purposes. Moreover, model-to-model transformations are

Chapter 3: Graph Transformation and verification

useful for computing different system views and their synchronization.
[HAMROUCHE, 2010]

The techniques for transformations of this type can be classified into five categories:
[BENDIAF,2018]

- Direct manipulation approaches.
- Relational approaches.
- Graph transformation-based approaches.
- Hybrid approaches.
- Structure-driven approaches.

1.5 The tools of MDA:

To achieve such efficiency, various conceptual tools are available. The Model Driven Architecture technology is supported by the OMG, which also provides UML and CORBA (Common Object Request Broker Architecture). These tools are: [Hettab,2009]

- a) **UML**, widely used elsewhere, which facilitates easy implementation of MDA by providing familiar support.
- b) **XMI** (XML Metadata Interchange), which offers a formalism for structuring XML (eXtensible Markup Language) documents in such a way that they can represent application metadata in a compatible manner.
- c) **MOF** (Meta Object Facility), a specification that enables the storage, access, manipulation, and modification of metadata. MOF allows for a unified expression of meta-models, whether they are subsequently used as UML profiles or not.
- d) **CWM** (Common Warehouse Metamodel), database for meta-data.

2 Model Transformations:

Model transformation is a key concept in MDE. It involves making models productive (operational). Indeed, the interest lies in transforming a model M_a into a model M_b , whether the respective meta-models MMA and MMb are identical (endogenous transformation) or different (exogenous transformation).
[AMROUNE,2014]

In other words, a model transformation is defined by the operation of generating one or more target models from one or more source models, in accordance with a transformation definition. [AMROUNE,2014]

In the literature, three types of transformations can be distinguished: [Kerkouche,2011]

- 1) **Les Vertical transformations:** The source and target of a vertical transformation are defined at different levels of abstraction. A transformation that decreases the level of abstraction is called a refinement. A transformation that increases the level of abstraction is called an abstraction.
- 2) **Horizontal transformations:** A horizontal transformation modifies the source representation while retaining the same level of abstraction. The modification can involve adding, modifying, deleting, or restructuring information.
- 3) **Oblique transformations:** An oblique transformation integrates both horizontal and vertical transformations. This type of transformation is notably used by compilers, which perform optimizations on the source code before generating executable code.

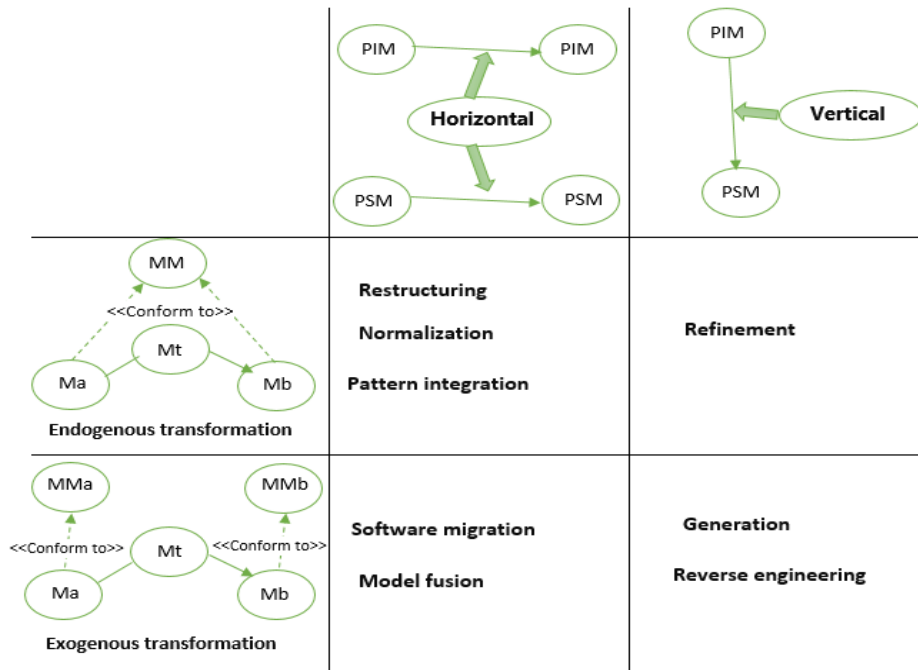


Figure 3.4: Types of transformations and their main uses. [Aouag,2014]

3 Graph Transformations:

Before we begin discussing graph transformation, let's cover some basic concepts of graphs. [ElMansouri,2009]

3.1 Graph Concepts:

- A graph consists of nodes that are connected by edges.
- Two nodes connected by an edge are adjacent.
- The number of nodes present in a graph is called the order of the graph.
- The degree of a node is the number of edges incident to that node.
- A subgraph of a graph G is a graph G' composed of certain nodes of G , such that all the edges connecting these nodes are also present in G .

There are two types of graphs: undirected graphs (where nodes are connected by edges) and directed graphs (where nodes are connected by arcs, which are edges with a direction).

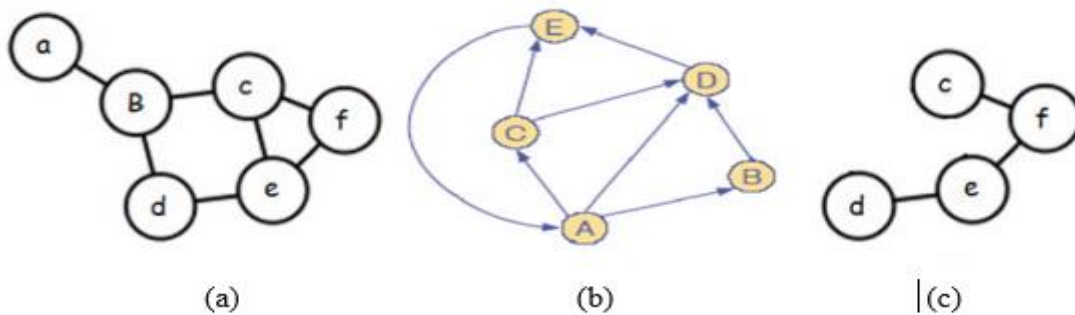


Figure3.5: (a) Undirected graph G , (b) Directed graph, (c) Subgraph of G . [ElMansouri,2009]

Chapter 3: Graph Transformation and verification

3.2 Subclass of a Graph:

- ✚ A **labeled graph**: is a directed graph in which the arcs are assigned labels. If all labels are positive numbers, it is called a weighted graph. [Kerkouche,2011]
- ✚ An **attributed graph**: is a graph that can contain a predefined set of attributes. [Kerkouche,2011]

3.3 Graph Transformation Tools:

Among the many tools available for graph transformation, such as AGG (Attributed Graph Grammar), TGG (Triple Graph Grammars), Fujaba (From UML to Java And Back Again), and AToM³ ((A Tool for Multi-formalism and Meta-Modeling), we have selected AToMPM for its distinctive advantages. [Syriani et al,2013]

- Performs transformations, controls, and processes models.
- Works on the web, independent of operating systems and platforms.
- Philosophy: explicit modeling, high level of abstraction, adapted formalisms and processes, autonomous

4 AToMPM:

AToMPM ("A Tool for Multi-Paradigm Modelling") is a (Meta) modelling workstation that enables language developers to create domain-specific visual languages, and domain experts to use these languages. A language is defined by its abstract syntax in a meta-model, its one or more concrete syntaxes, which define how each element of abstract syntax is visualized, and its semantic definitions, either operational (a simulator) or translational (by mapping them onto a known semantic domain). AToMPM supports model transformations for modeling semantics. [AToMPM Documentation]

AToMPM is developed to satisfy three main functionalities:

- Meta-modelling.
- Model transformation.
- Execution of transformation on a model.

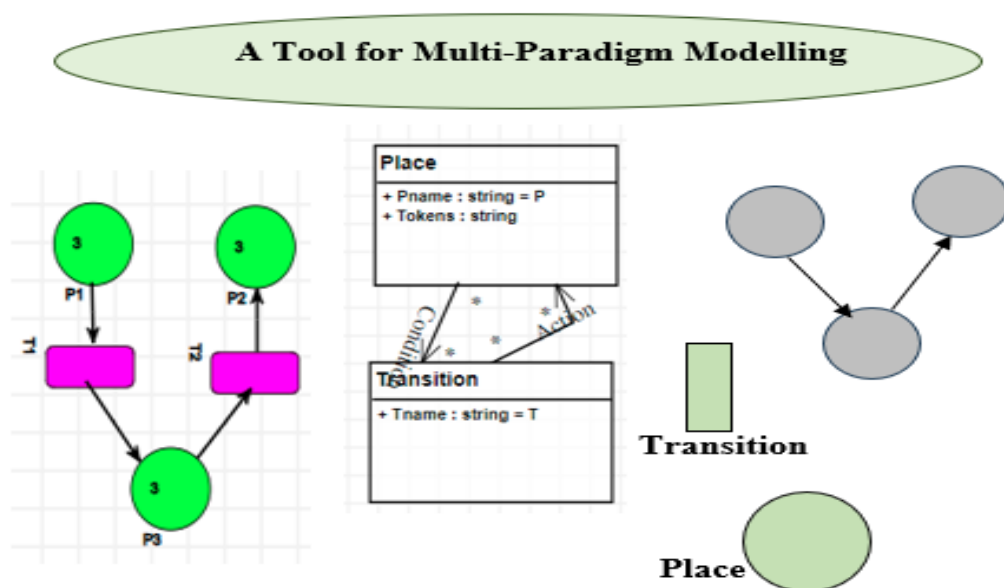


Figure 3.6: Presentation of the AToMPM tool.

5 Analysis of Petri nets:

The transformation from aspect-oriented diagrams to formal models, such as Petri nets, often suffers from a lack of verification to preserve the important properties of the system and to ensure that the model transformation is correct, as well as to check if they meet the system requirements. There are several Properties to Verify: [Aouag,2023]

- Termination of the transformation.
- Preservation of the semantics of the source model.
- Confluence.
- Invariants.
- Completeness.
- Absence of deadlock & infinite loop...

5.1 Analysis Techniques:

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: [Elmansouri,2009]

- ✚ **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].

5.2 Analysis tools for PN:

Among the many analysis tools available, such as PEP (Programming Environment based on Petri nets) and INA (Integrated Net Analyzer), we have selected TINA for its distinctive advantages:

- TINA is easy to use thanks to its user-friendly and easy-to-understand interface.
- Providing users with great flexibility to customize the tool to their specific needs.
- TINA provides advanced features for editing and analyzing Petri networks, making it a powerful tool for complex analysis tasks.

6 TINA:

(Time Petri Net Analyzer) is a toolbox for editing and analyzing Petri Nets. It supports features such as inhibitor and read arcs, Time Petri Nets with priorities and stopwatches, and an extension of Time Petri Nets with data handling, known as Time Transition Systems. [TINA Documentation]

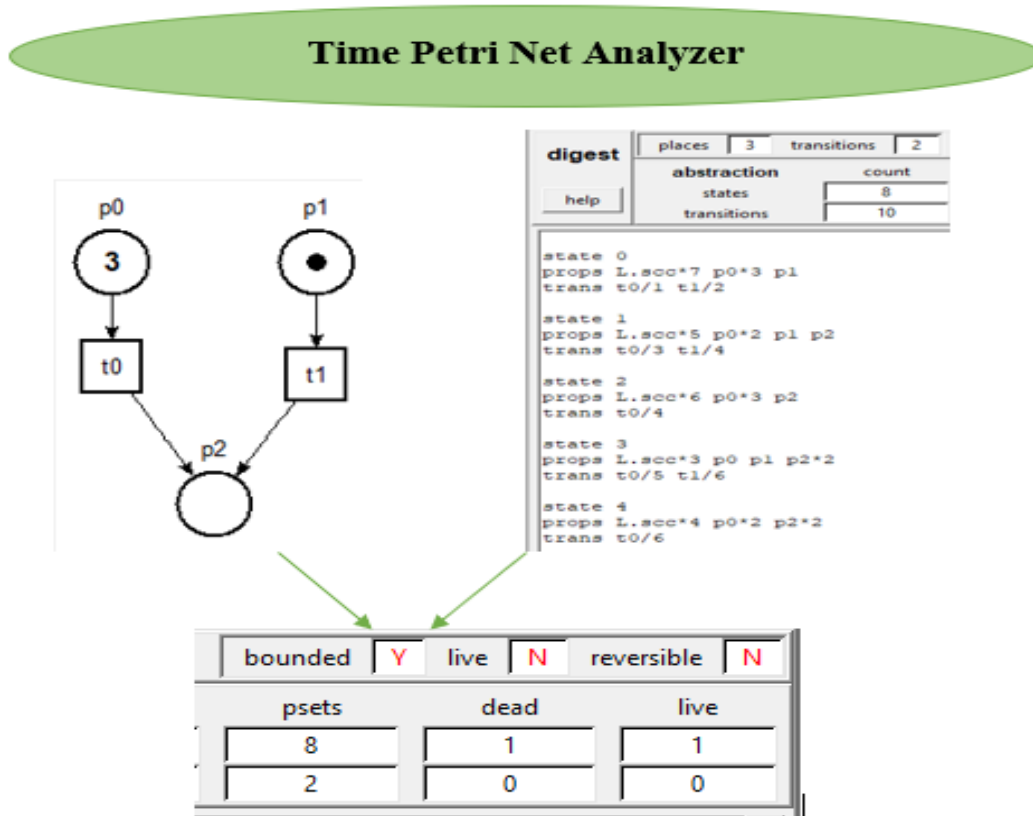


Figure 3.7: Presentation of the TINA tool.

Conclusion:

In this chapter, we have explored the key concepts of Model-Driven Engineering. By detailing the Model-Driven Architecture approach along with the four-level architecture of MDA, and examining its various variants, we have also looked into the tools available to support this methodology. Model transformation has been highlighted as a central element of MDE, with an overview of the various forms of model transformations. Finally, we have covered the basic concepts and relevant tools in the field of graph transformation. These insights provide a solid foundation for understanding and effectively applying the principles of MDE in software development. Thus, we presented the tool used in the implementation of our work, AToMPM. Next, we introduce the techniques for analyzing Petri nets and the analysis tools, concluding with the presentation of the tool used in our work.

The next chapter (Chapter 4) presents our model transformation approach that is based on the concepts introduced previously.

Chapter4

Proposed

Approaches

Introduction:

In this chapter, we present our approach aimed at transforming a source model into a target model based on graph transformation. The chapter contains two approaches:

- ✚ Transformation of detailed object-oriented sequence diagrams into detailed aspect-oriented sequence diagrams.
- ✚ Transformation of detailed aspect-oriented sequence diagrams into Petri nets.

We begin our approach with meta-modeling that includes classes and associations. Next, we propose a graph grammar for each approach to transform an object-oriented model into an aspect-oriented model. The target model of the first approach serves as the basis for the second approach, which results in Petri nets. In this transformation, we use the modeling tool AToMPM. Additionally, we used the TINA tool to verify the properties of the transformation results. Finally, we make a comparison between works related to our work and discuss this comparison.

1 Transformation of detailed object-oriented sequence diagrams into detailed aspect-oriented sequence diagrams:

In this approach, we transform a detailed object-oriented sequence diagram into a detailed aspect-oriented sequence diagram by combining the base model of the detailed sequence diagram with the aspect model. This results in weaving (the composite model, which is the detailed aspect-oriented sequence diagram). We perform this transformation using a single meta-modeling. The transformation method takes as input a source model that includes the base detailed sequence diagram and the aspect model. After executing the graph grammar rules, we obtain the detailed aspect-oriented sequence diagram as output.

1.1 Meta-modeling:

The meta-modeling of the aspect-oriented detailed sequence diagram contains seventeen classes, forty-three associations, and five inheritances. In Figure 4.1, we present the meta-model for aspect-oriented detailed sequence diagrams.

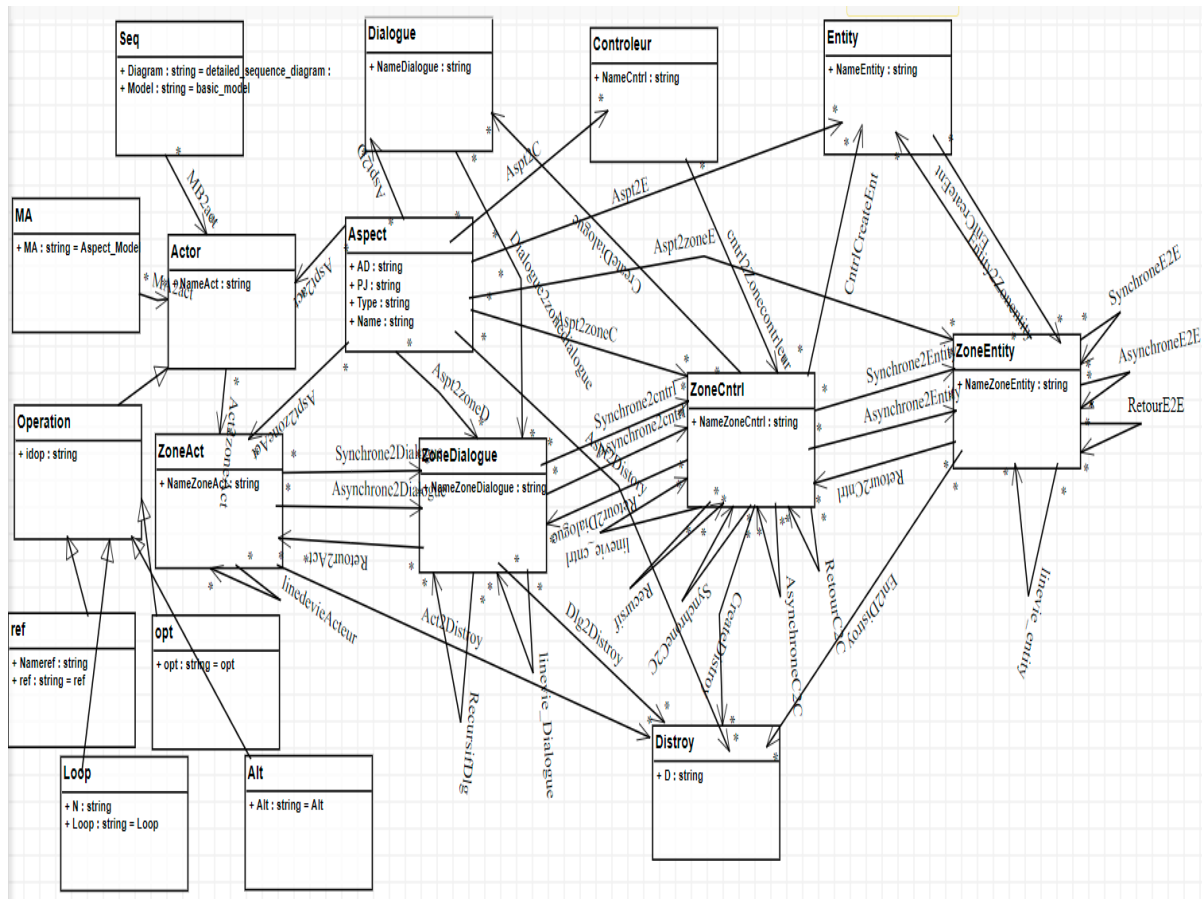


Figure 4.1: Meta-model of the detailed sequence diagram.

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.

Chapter 4: Proposed Approaches

- **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.

Associations:

- **MA2act:** connects class <MA> and class <Actor>, It represents an invisible link.
- **MB2act:** connects class <Seq> and class <Actor>, It represents an invisible link.
- **Aspt2act:** connects class <Aspect> and class <Actor>. It contains an attribute <Aspt2act> of type string.
- **Act2zoneAct:** connects class <Actor> and class <ZoneAct>. It contains an attribute <Act2zoneAct> of type string.
- **linedevieActeur:** connects class <ZoneAct> and class <ZoneAct>. It contains an attribute <linedevieActeur> of type string.
- **Aspt2zoneAct:** connects class <Aspect> and class <ZoneAct>. It contains an attribute <Aspt2zoneAct> of type string.

- **Retour2Act:** connects class <ZoneDialogue> and class <ZoneAct>. It contains an attribute <Retour2Act> of type string.
- **Aspt2D:** connects class <Aspect> and class <Dialogue>. It contains an attribute <Aspt2D> of type string.
- **CreateDialogue:** connects class <ZoneCntrl> and class <Dialogue>. It contains an attribute <Aspt2D> of type string.
- **Aspt2C:** connects class <Aspect> and class <Controller>. It contains an attribute <Aspt2C> of type string.
- **Aspt2E:** connects class <Aspect> and class <Entity>. It contains an attribute <Aspt2E> of type string.
- **CntrlCreateEnt:** connects class <ZoneCntrl> and class <Entity>. It contains an attribute <CntrlCreateEnt> of type string.
- **EntCreateEnt:** connects class <ZoneEntity> and class <Entity>. It contains an attribute <EntCreateEnt> of type string.
- **Aspt2zoneD:** connects class <Aspect> and class <ZoneDialogue>. It contains an attribute <Aspt2zoneD> of type string.
- **Synchrone2Dialogue:** connects class <ZoneAct> and class <ZoneDialogue>. It contains an attribute <Synchrone2Dialogue> of type string.
- **Asynchrone2Dialogue:** connects class <ZoneAct> and class <ZoneDialogue>. It contains an attribute <Asynchrone2Dialogue> of type string.
- **RecurisifDlg:** connects class <ZoneDialogue> and class <ZoneDialogue>. It contains an attribute <RecurisifDlg> of type string.
- **linevie_Dialogue:** connects class <ZoneDialogue> and class <ZoneDialogue>. It contains an attribute <linevie_Dialogue> of type string.
- **Retour2Dialogue:** connects class <ZoneCntrl> and class <ZoneDialogue>. It contains an attribute <Retour2Dialogue> of type string.
- **Dialogue2zonedialogue:** connects class <Dialogue2zonedialogue> and class <ZoneDialogue>. It contains an attribute <Dialogue2zonedialogue> of type string.
- **Aspt2zoneC:** connects class <Aspect> and class <ZoneCntrl>. It contains an attribute <Aspt2zoneC> of type string.
- **cntrl2Zonecontrleur:** connects class <Controller> and class <ZoneCntrl>. It contains an attribute <cntrl2Zonecontrleur> of type string.
- **Synchrone2cntrl:** connects class <ZoneDialogue> and class <ZoneCntrl>. It contains an attribute <Synchrone2cntrl> of type string.
- **Asynchrone2cntrl:** connects class <ZoneDialogue> and class <ZoneCntrl>. It contains an attribute <Asynchrone2cntrl> of type string.
- **linevie_cntrl:** connects class <ZoneCntrl> and class <ZoneCntrl>. It contains an attribute <linevie_cntrl> of type string.
- **Recurisif:** connects the class <ZoneCntrl> and the class <ZoneCntrl>. It contains an attribute <Recurisif> of type string.
- **SynchroneC2C:** connects the class <ZoneCntrl> and the class <ZoneCntrl>. It contains an attribute <SynchroneC2C> of type string.
- **AsynchroneC2C:** connects the class <ZoneCntrl> and the class <ZoneCntrl>. It contains an attribute <AsynchroneC2C> of type string.

- **RetourC2C:** connects the class <ZoneCntrl> and the class <ZoneCntrl>. It contains an attribute <RetourC2C> of type string.
- **Retour2Cntrl:** connects the class <ZoneEntity> and the class <ZoneCntrl>. It contains an attribute <Retour2Cntrl> of type string.
- **Aspt2zoneE:** connects the class <Aspect> and the class <ZoneEntity>. It contains an attribute <Aspt2zoneE> of type string.
- **Synchrone2Entity:** connects the class <ZoneCntrl> and the class <ZoneEntity>. It contains an attribute <Synchrone2Entity> of type string.
- **Asynchrone2Entity:** connects the class <ZoneCntrl> and the class <ZoneEntity>. It contains an attribute <Asynchrone2Entity> of type string.
- **linevie_entity:** connects the class <ZoneEntity> and the class <ZoneEntity>. It contains an attribute <linevie_entity> of type string.
- **RetourE2E:** connects the class <ZoneEntity> and the class <ZoneEntity>. It contains an attribute <RetourE2E> of type string.
- **SynchroneE2E:** connects the class <ZoneEntity> and the class <ZoneEntity>. It contains an attribute <SynchroneE2E> of type string.
- **AsynchroneE2E:** connects the class <ZoneEntity> and the class <ZoneEntity>. It contains an attribute <AsynchroneE2E> of type string.
- **Entity2Zonentity:** connects the class <ZoneEntity> and the class <ZoneEntity>. It contains an attribute <Entity2Zonentity> of type string.
- **Act2Distroy:** connects the class <ZoneAct> and the class <Distroy>. It contains an attribute <Act2Distroy> of type string.
- **Dlg2Distroy:** connects the class <ZoneDialogue> and the class <Distroy>. It contains an attribute <Dlg2Distroy> of type string.
- **Aspt2Distory:** connects the class <Aspect> and the class <Distroy>. It contains an attribute <Aspt2Distory> of type string.
- **CreateDistroy:** connects the class <ZoneCntrl> and the class <Distroy>. It contains an attribute <CreateDistroy> of type string.
- **Ent2Distroy:** connects the class <ZoneEntity> and the class <Distroy>. It contains an attribute <Ent2Distroy> of type string.

Inheritance: is a relationship between two classes. Graphically, it is represented by an arrow.

Figure 4.2 illustrates the tool generated for manipulating detailed aspect-oriented sequence diagrams.

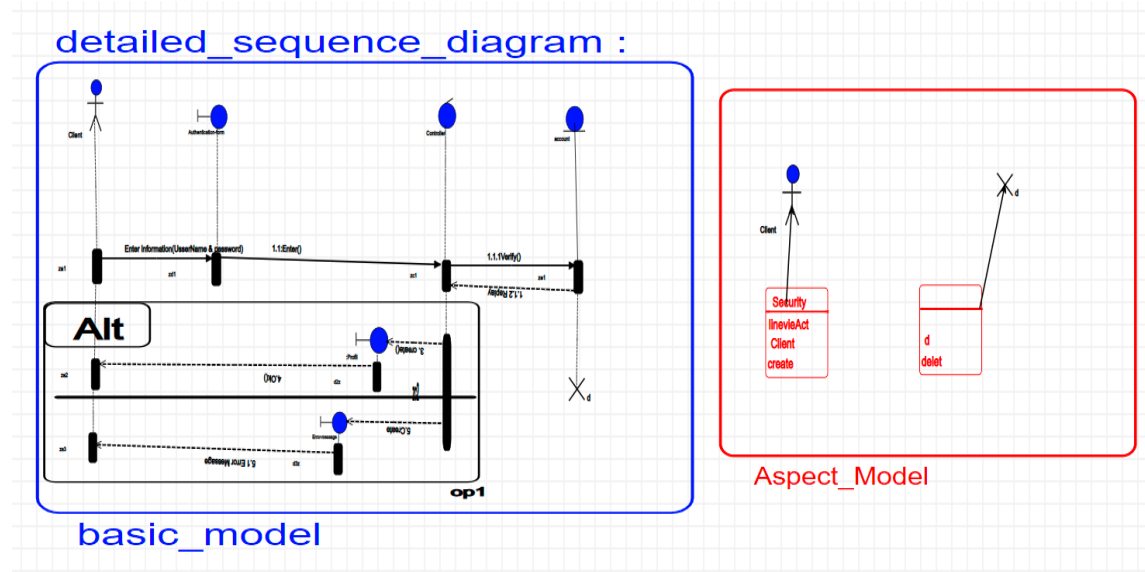


Figure 4.2: The tool generated for the detailed aspect-oriented sequence diagram.

1.2 The proposed graph grammar:

In this approach, we present a grammar composed of eighty-eight (88) rules. These rules will be executed in an order determined by the pattern (T). Each rule consists of two parts: a left-hand side (LHS) and a right-hand side (RHS). These rules are then divided into three categories based on the type of aspect, which are defined as follows:

Aspect.type: the action of the aspect is a creation, deletion, or connection between two objects.

In the following figure, the pattern (T_O2A) of this graph grammar for this approach:

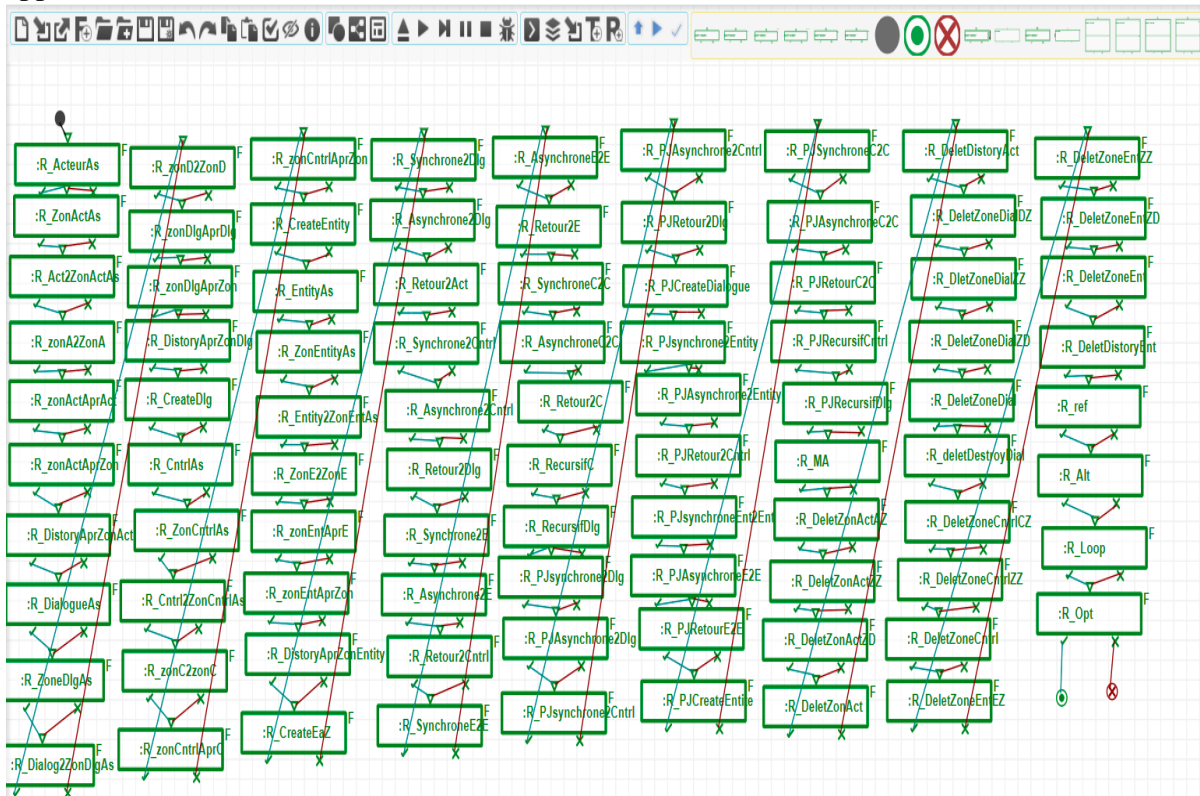


Figure 4.3: The motif of transformation of approach.

Chapter 4: Proposed Approaches

Category 1: Rules that have the aspect type "create" are applied to create an aspect according to the PJ and AD.

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

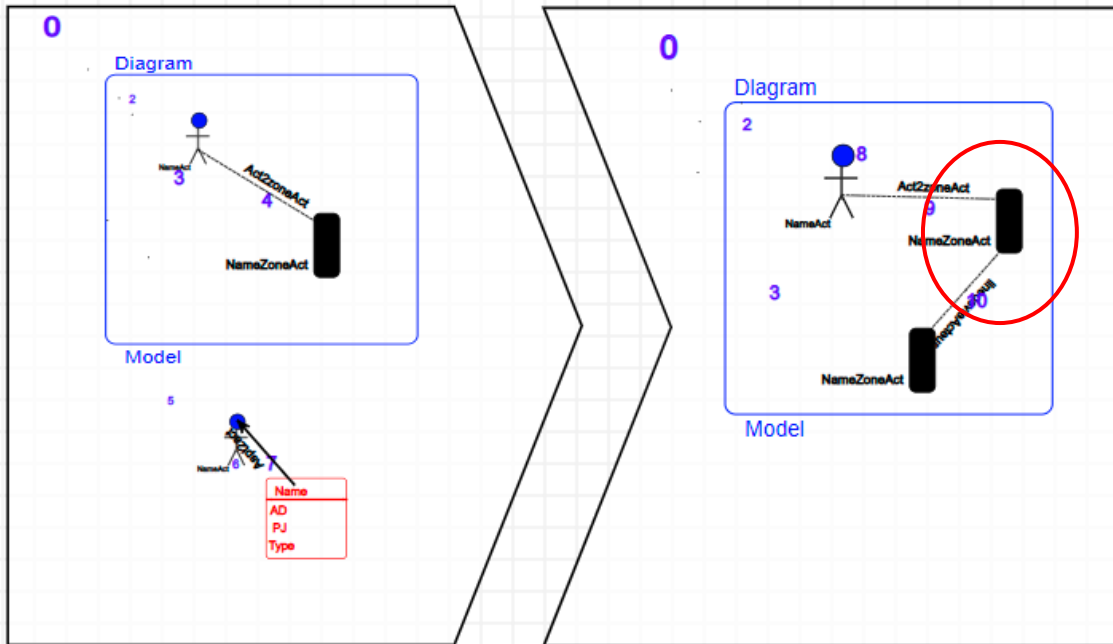


Figure 4.4: Application of category 1 with Aspect.PJ == Actor and Aspect.Ad== linevieAct.

In Figure 4.4, 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 4.5 and 4.6, the Python code for the first creation rule:

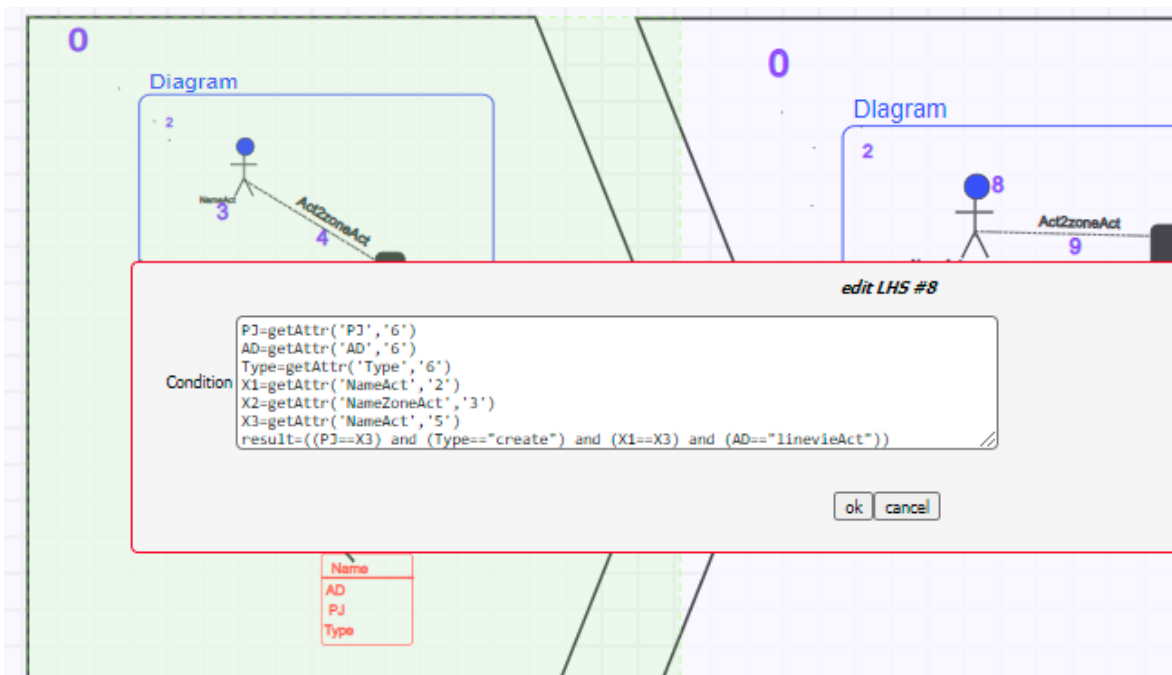


Figure 4.5: The LHS of the first create rule(Python code).

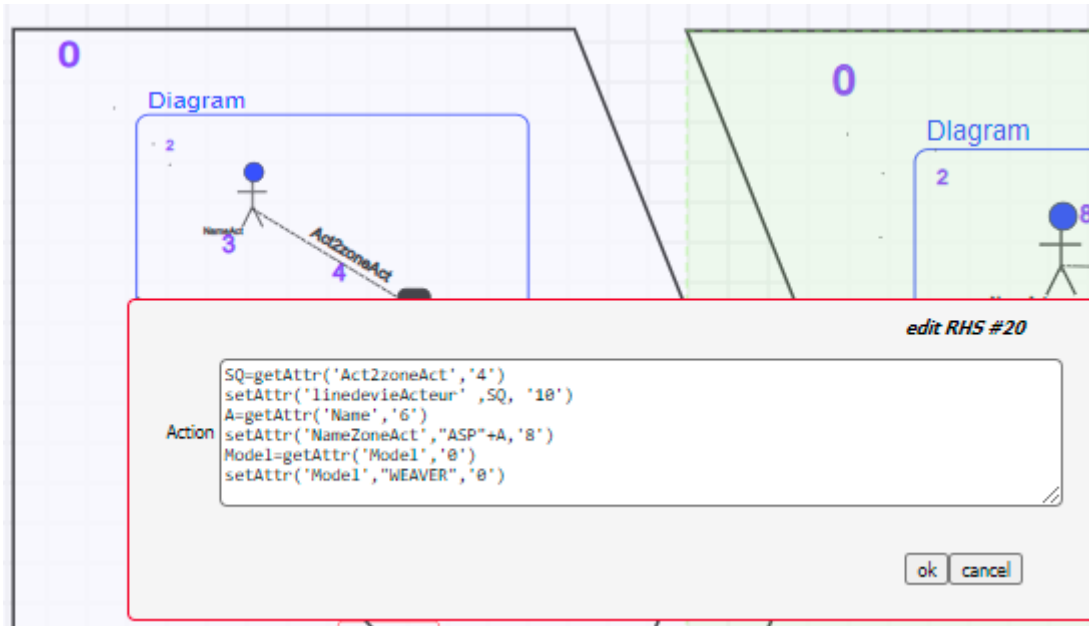


Figure 4.6: The RHS of the first create rule (Python code).

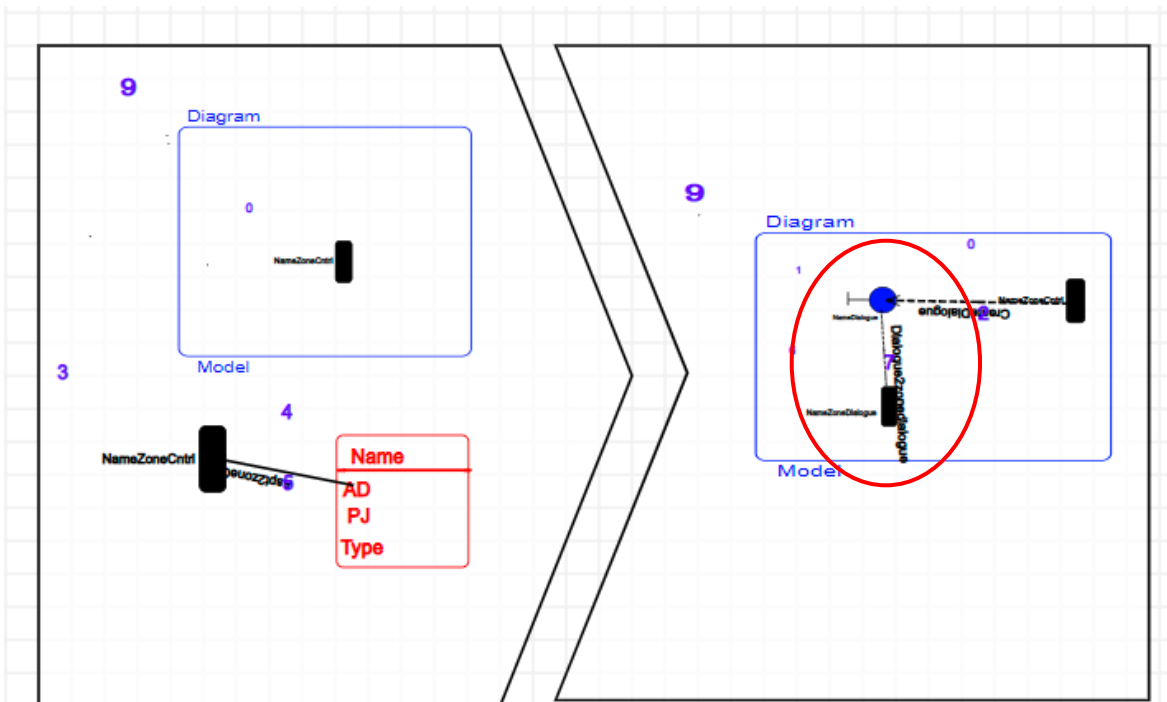


Figure 4.7: Application of category 1 with Aspect.PJ== Control Zone et Aspect.AD== CreatedDlg.

In Figure 4.7, 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:

Chapter 4: Proposed Approaches

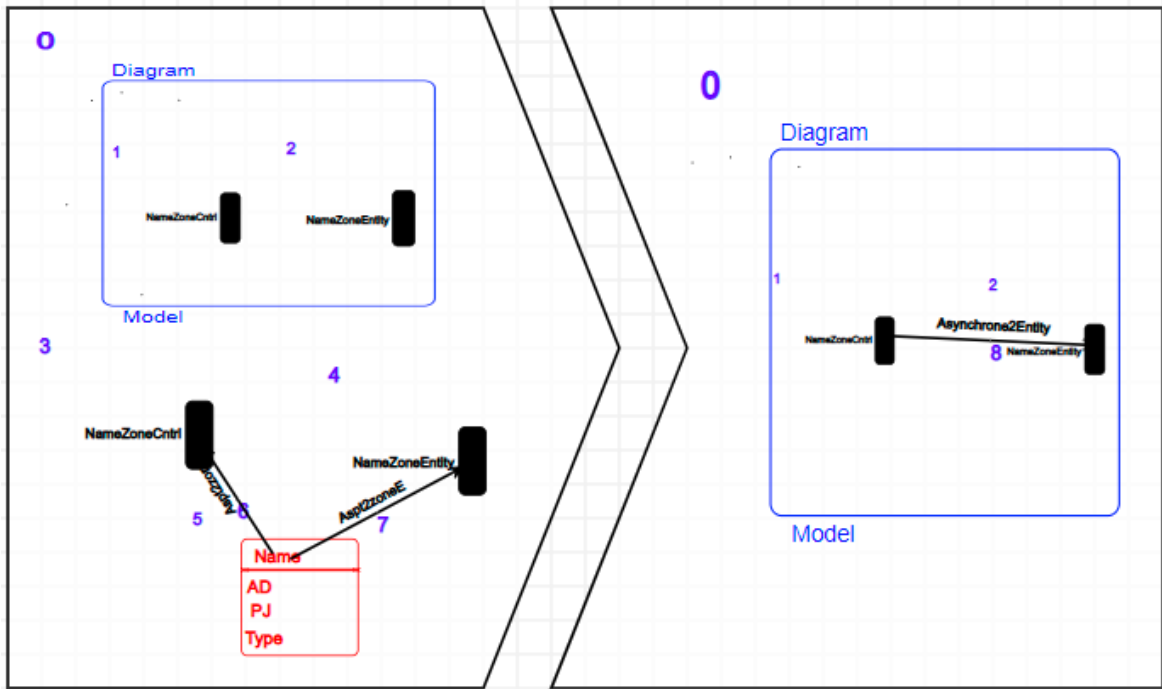


Figure 4.8: Application of category 2 with Aspect.PJ== Control Zone, Entity Zone and Aspect.AD== Asynchrone2E.

In Figure 4.8, 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 4.9 and 4.10, the Python code for the first link rule:

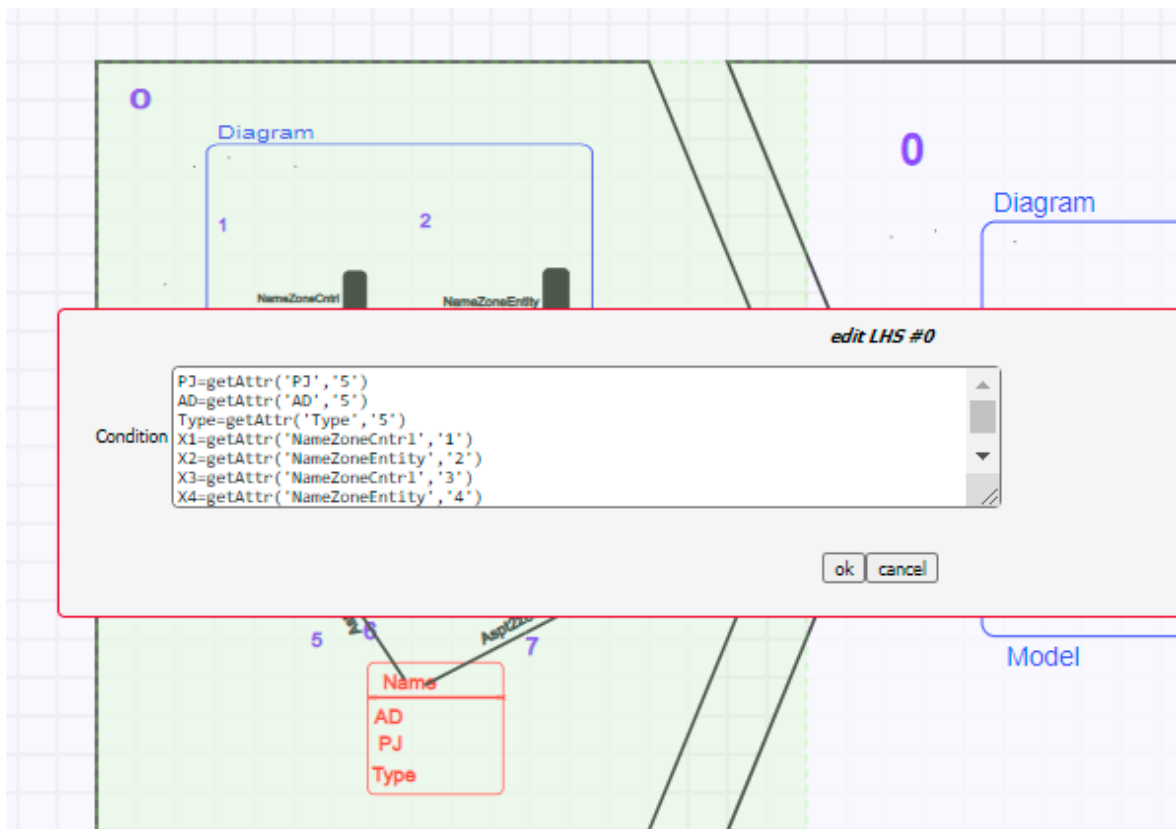


Figure 4.9: The LHS of the first link rule (Python code).

Chapter 4: Proposed Approaches

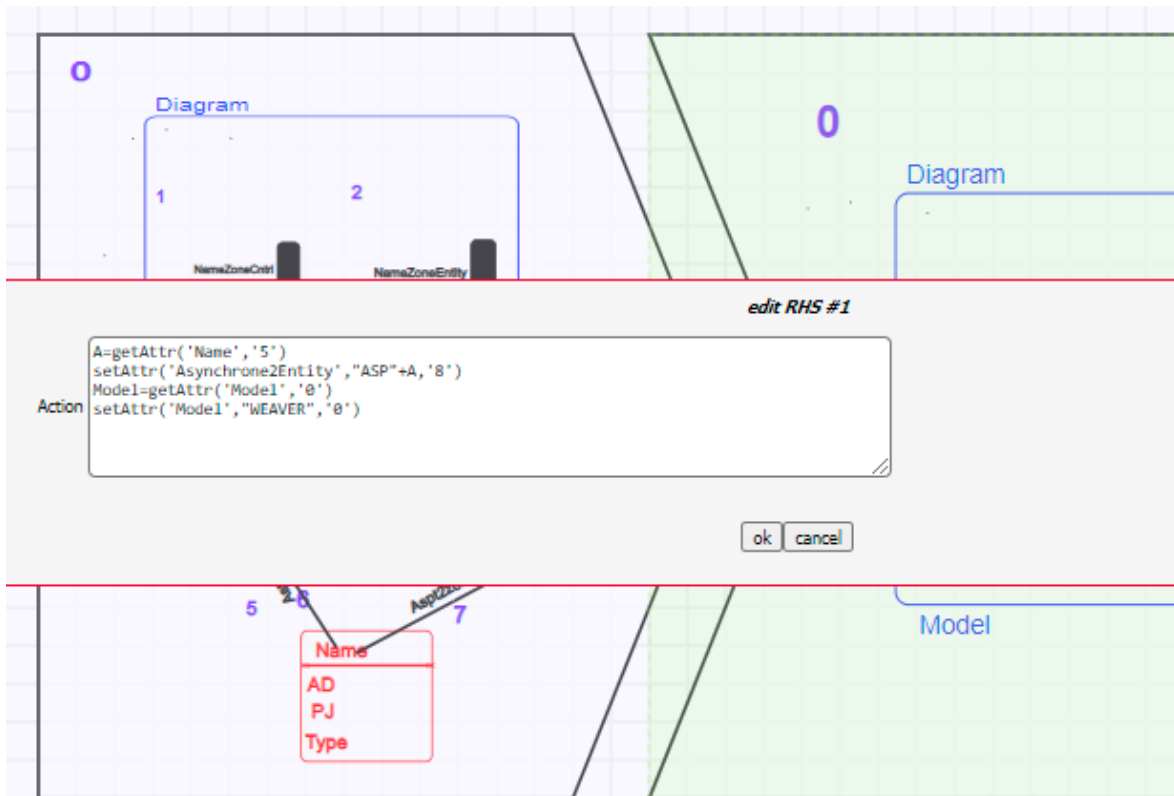


Figure 4.10: The RHS of the first Link rule (Python code).

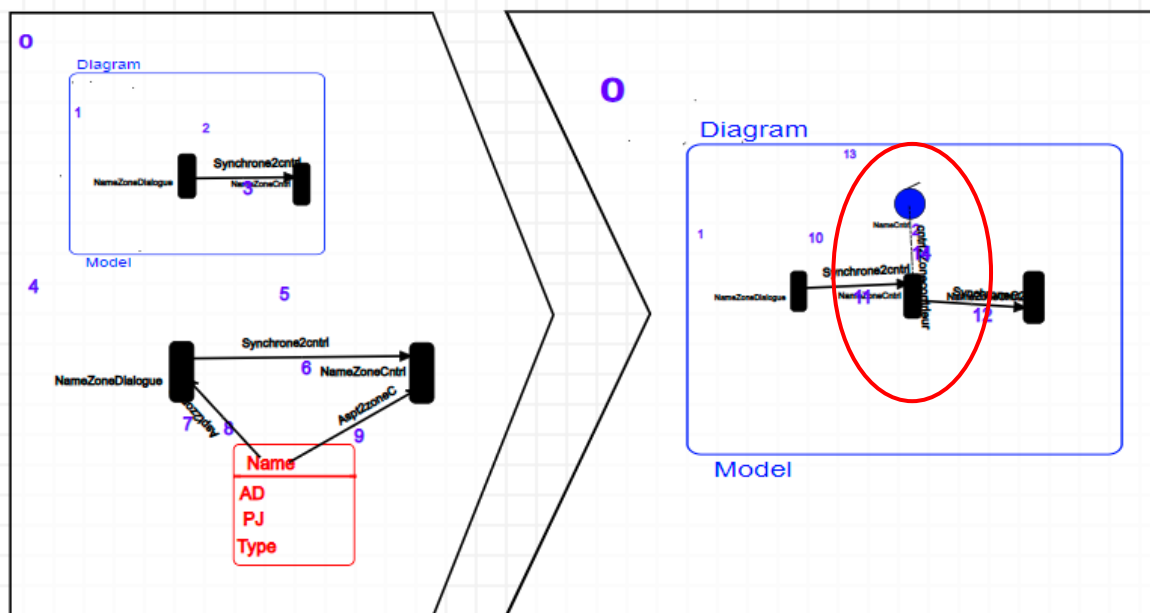


Figure 4.11: Application of category 2 with Aspect.PJ== Synchron message et Aspect.AD== Cntrl.

In Figure 4.11, 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'

Chapter 4: Proposed Approaches

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:

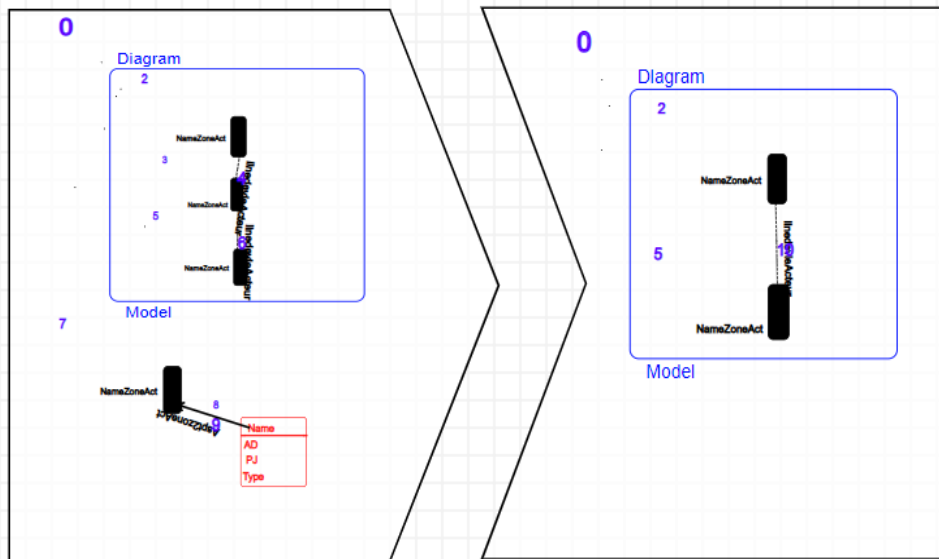


Figure 4.12: Application of category 3 with Aspect.PJ== Zone Actor et Aspect.type== delete.

In Figure 4.12, 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 'delete'

In Figures 4.13 and 4.14, the Python code for the first Delete rule:

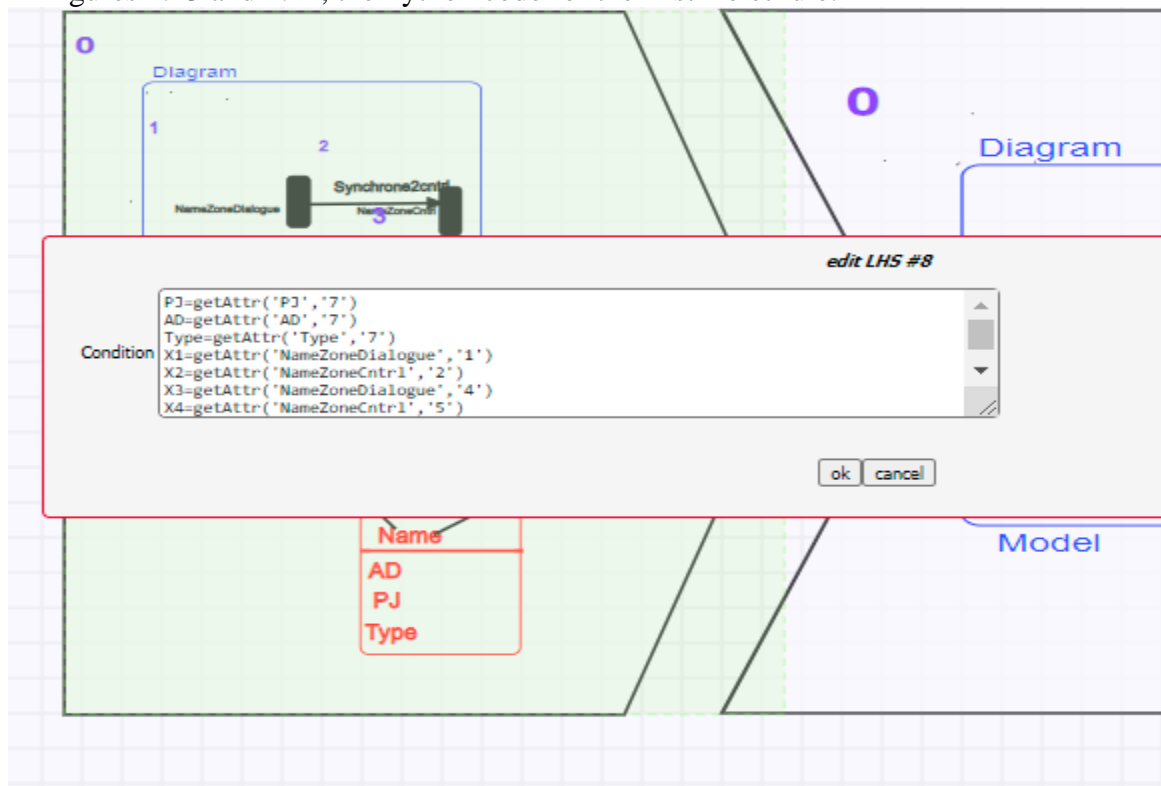


Figure 4.13: The LHS of the first delete rule (Python code).

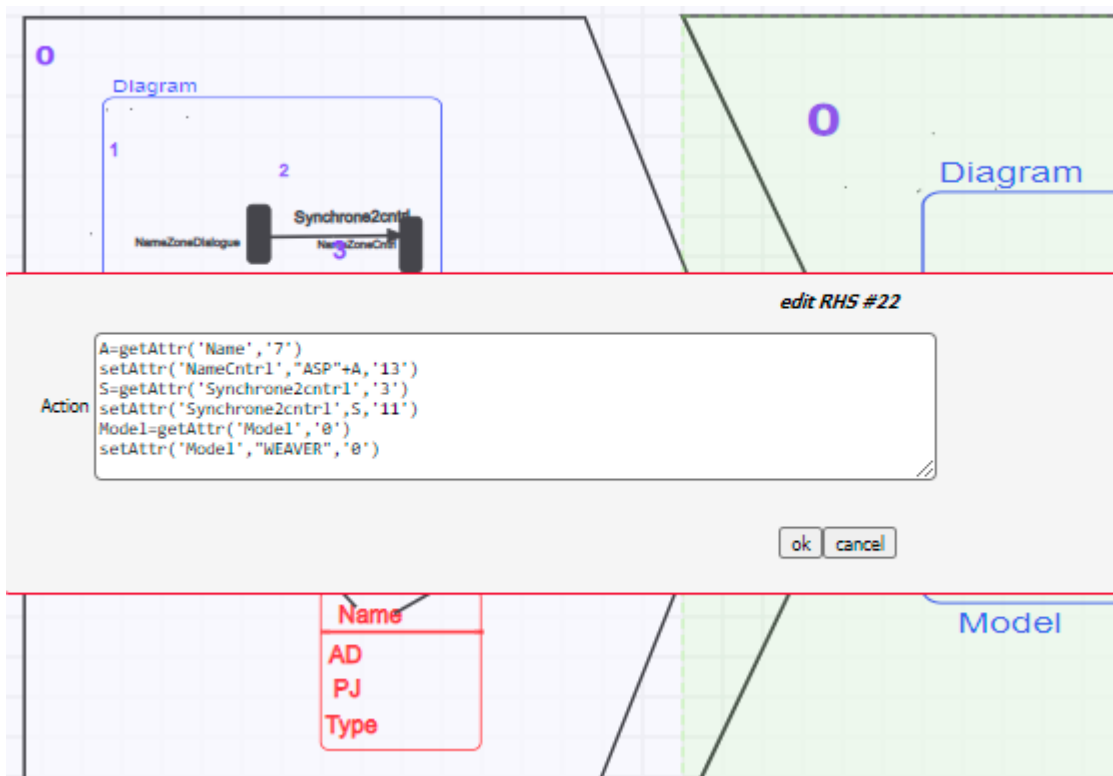


Figure 4.14: The RHS of the first delete rule (Python code).

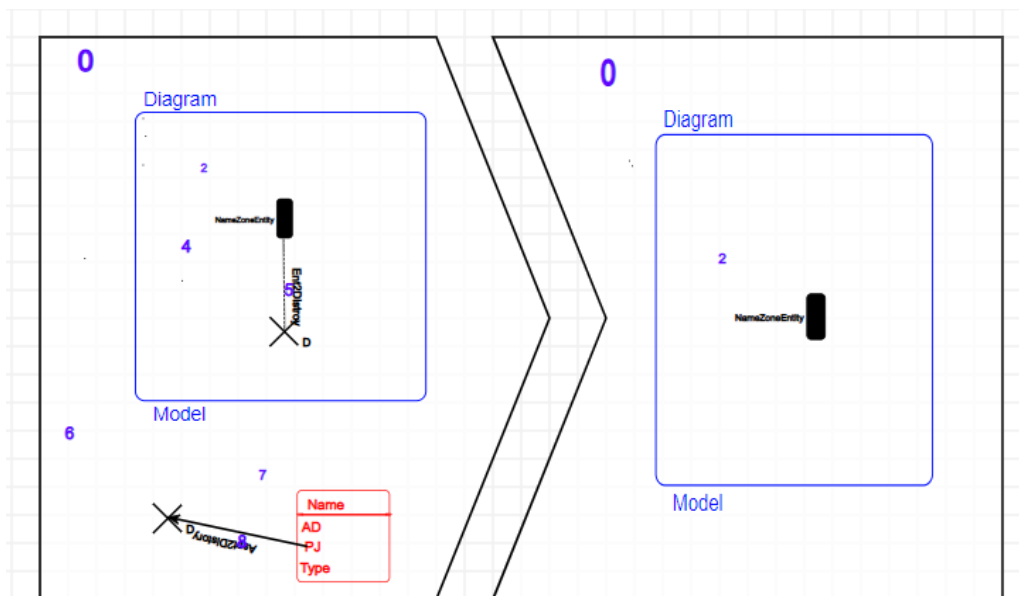


Figure 4.15: Application of category 3 with Aspect.PJ== Destroy et Aspect.type== delet.

In Figure 4.15, 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 'delet'

Chapter 4: Proposed Approaches

1.3 Explanatory example:

To highlight our approach, we have chosen to explore a case study example to demonstrate the transformation steps (the rules). We applied our approach to a detailed sequence diagram of the authentication case. We proposed a base model as well as an aspect model for this diagram. The aspect model includes three aspects:

- **Dialogue-Script:** This aspect allows verifying the information entered in the authentication form
- **Verify Security:** for information security.
- **Deleting** the "destroy" object.

In Figure (4.10), we present the base model.

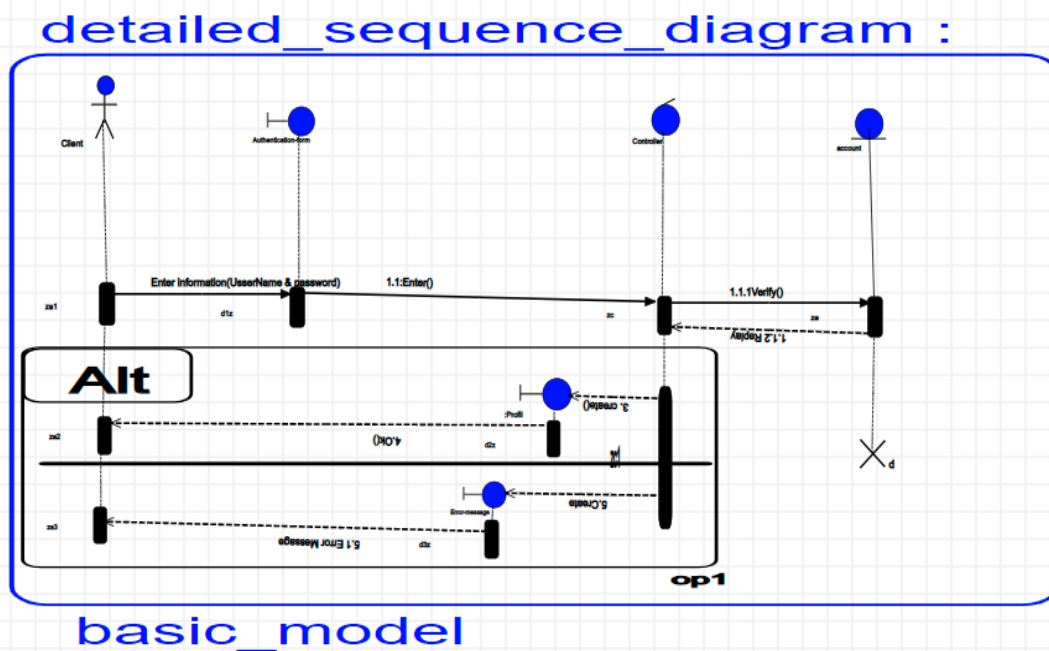


Figure 4.16: Basic Authentication Model.

In Figure (4.11), we present the aspect model of authentication.

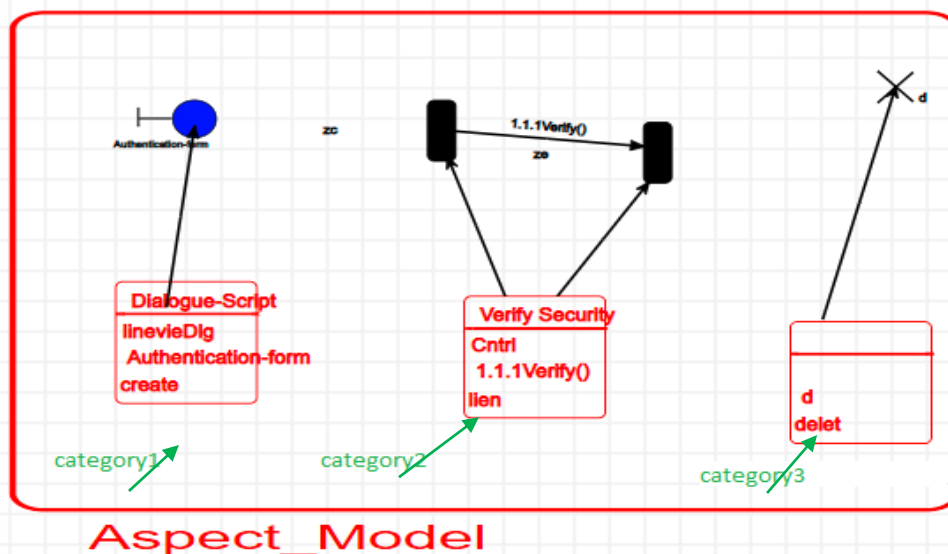


Figure 4.17: Authentication Aspect Model.

Chapter 4: Proposed Approaches

The integration between the base model (an Object-Oriented Sequence Diagram) and the aspect model is achieved by executing T_O2A, which contains the proposed graph grammar. Thus, we obtain the composite model represented in Figure (4.12).

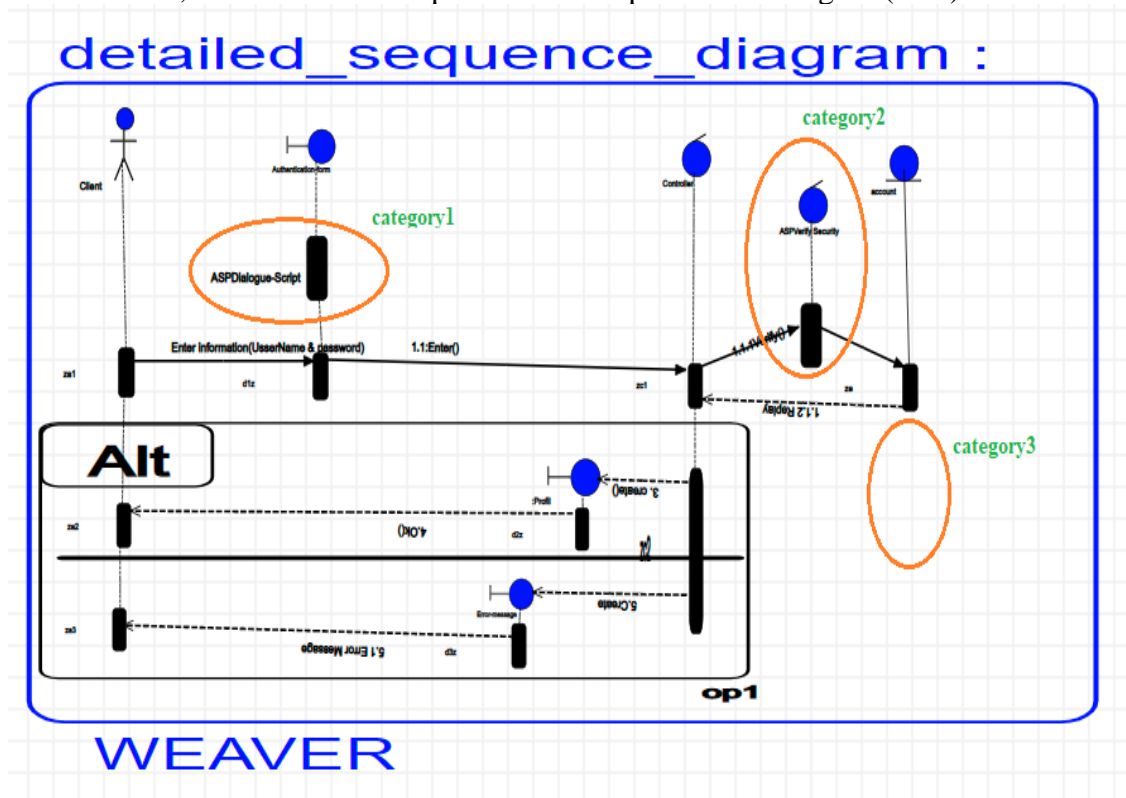


Figure 4.18: The weaver model.


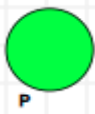


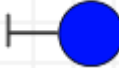
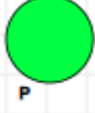



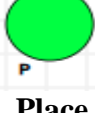



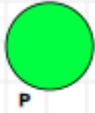



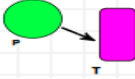
2 Transformation of Aspect-Oriented Detailed Sequence Diagrams into RDPs:

In this approach, we perform the transformation of an aspect-oriented detailed sequence diagram into PN (Petri Nets) by composing the source model (the composite model containing the aspect-oriented detailed sequence diagram). This yields the target model (the Petri Nets). We conduct this transformation through meta-modeling. The transformation method takes as input a source model that includes the aspect-oriented detailed sequence diagram. After applying the rules of graph grammar, we obtain PN as output.

2.1 Model Transformation Process:

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

Chapter 4: Proposed Approaches

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

Chapter 4: Proposed Approaches

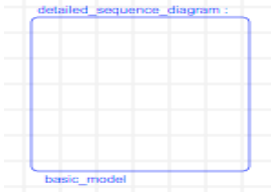
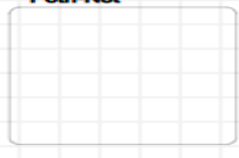
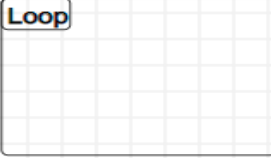
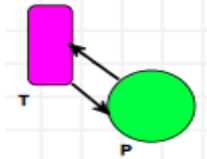
 <p>diagram framework</p>	<p>That transforms the diagram framework into a Petri net framework</p>	 <p>Petri-Net framework</p>
 <p>Loop operation</p>	<p>That transforms the Loop operation into a place and a transition as it's performed in a loop.</p>	 <p>Place and transition</p>

Table 4.1: Expresses the semantics of this approach.

2.2 Meta-modeling:

The meta-modeling of Petri Nets consists of three classes and three associations. In Figure 4.13, we present the meta-model for Petri Nets.

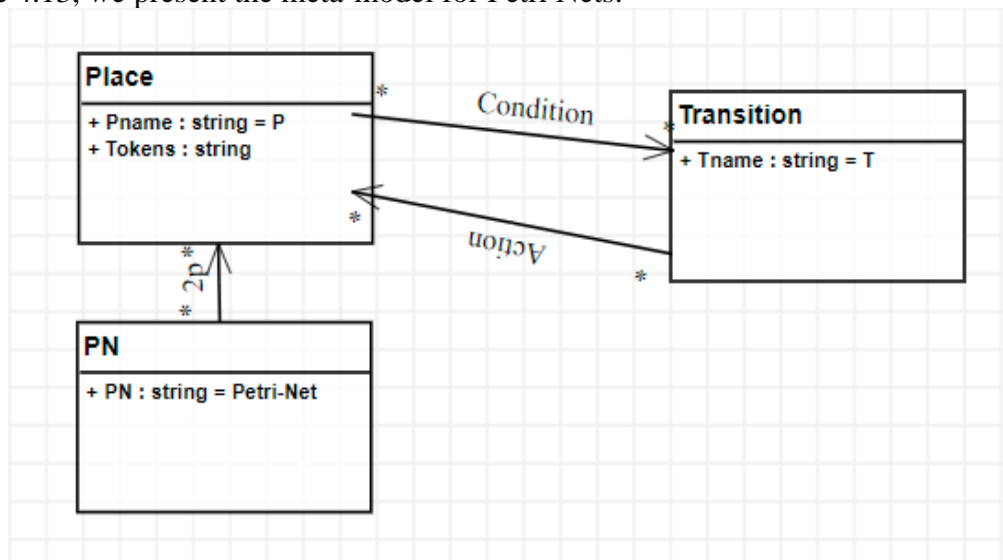


Figure 4.19: Meta-model of Petri Nets.

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.

Associations:

- **Condition:** Connects the <Place> class and the <Transition> class. It contains an attribute <Condition> of type string.

Chapter 4: Proposed Approaches

- **Action:** Connects the <Transition> class and the <Place> class. It contains an attribute <Action> of type string.
- **2p:** Connects the <PN> class and the <Place> class. The link is invisible.

Figure 4.14 illustrates the generated tool for manipulating Petri Nets:

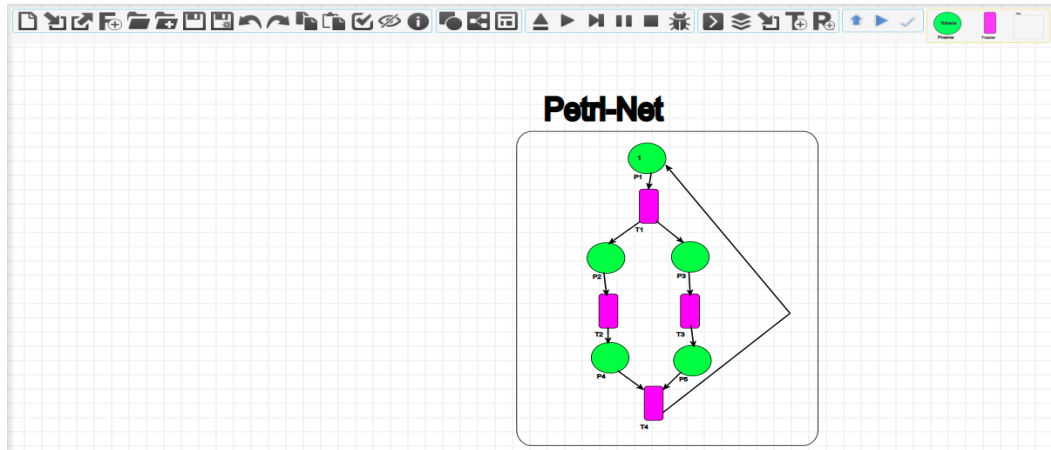


Figure 4.20: The generated tool for Petri Nets.

2.3 The proposed graph grammar:

In this approach, we introduce a grammar consisting of fifty-six (56) rules. These rules will be executed in an order determined by the motif (T). Each rule consists of two parts: a left-hand side (LHS) and a right-hand side (RHS). These rules are then divided into three categories based on the transformation of objects, 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:

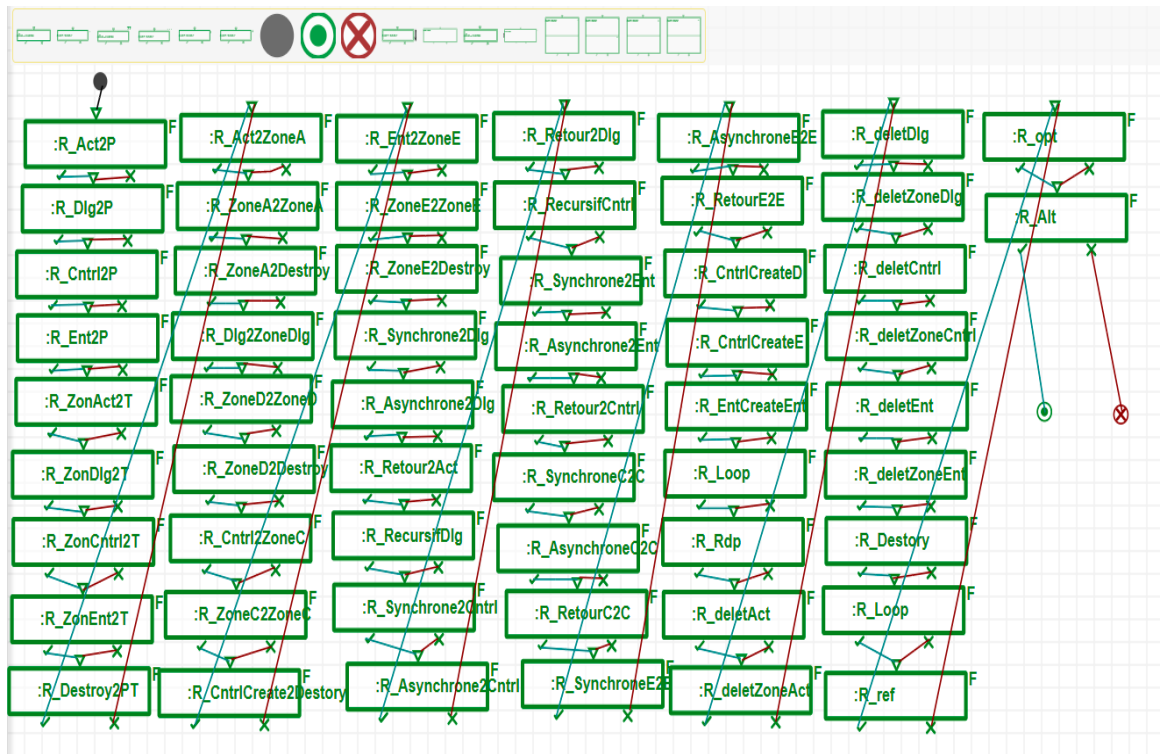


Figure 4.21: The transformation pattern of the approach.

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

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

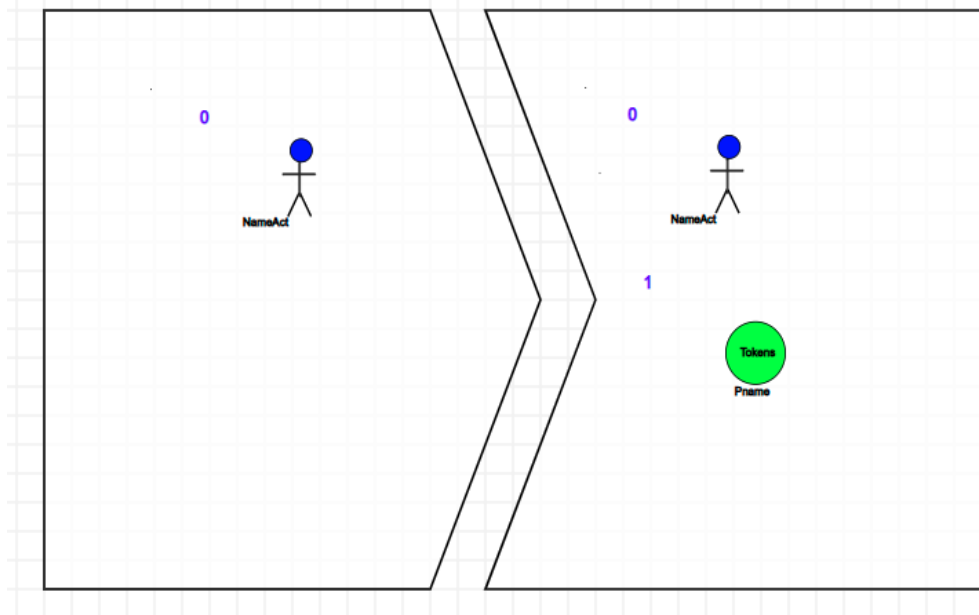


Figure 4.22: Application of Category 1 on the actor.

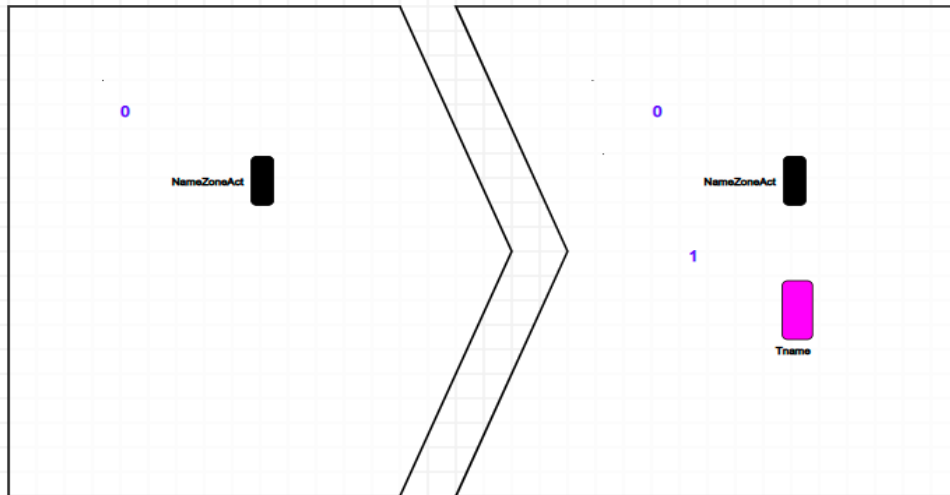


Figure 4.23: Application of Category 1 on the actor Activity Zone.

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:

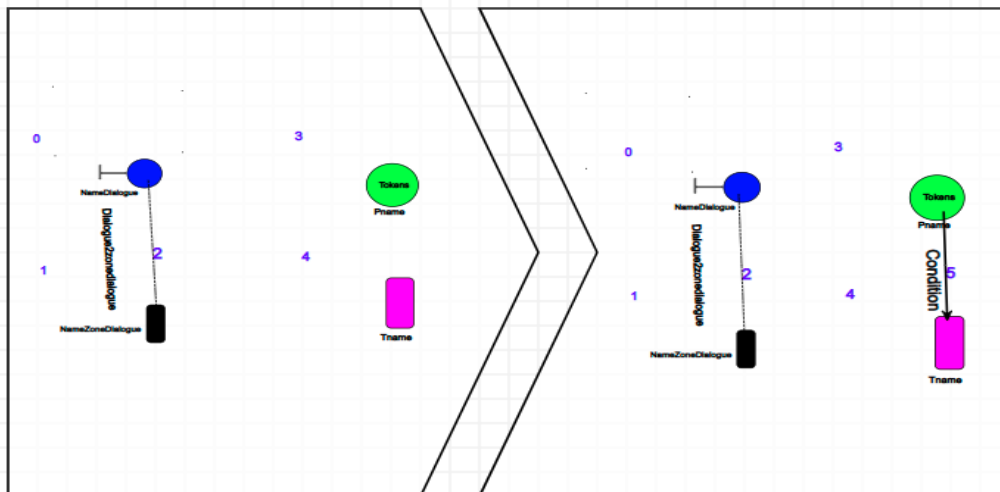


Figure 4.24: Application of Category 2 on the lifeline boundary.

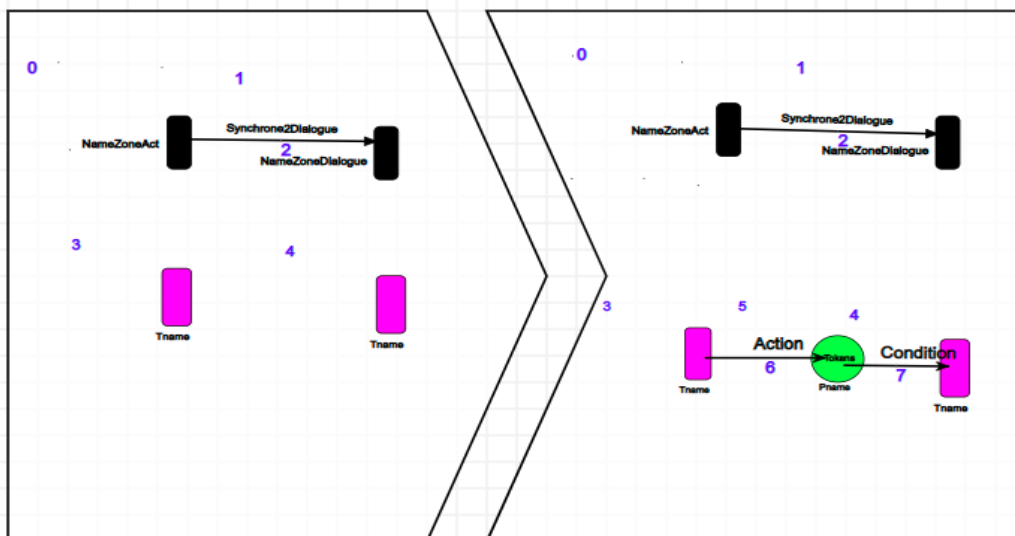


Figure 4.25: Application of Category 2 on the synchronous message.

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:

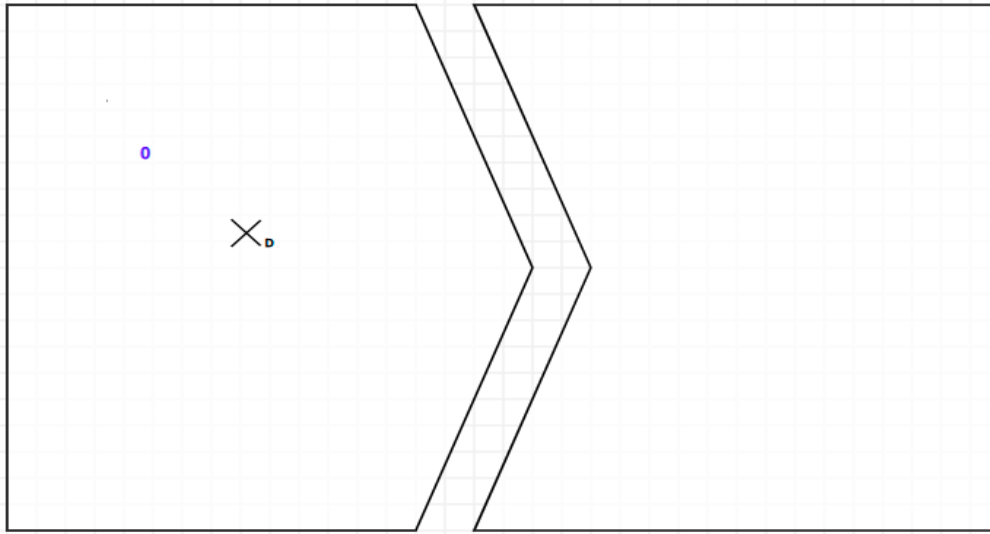


Figure 4.26: Application of Category 3 on "destroy"

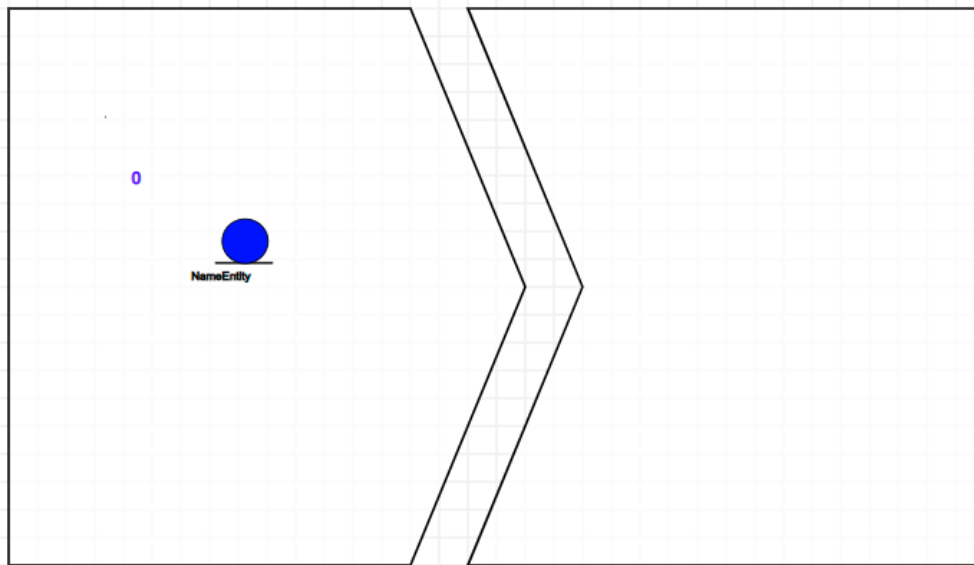


Figure 4.27: Application of Category 3 on "Entity"

3 Representation of the TINA tool:

The figure 4.27 and 4.28 illustrates the tool used to verify the Petri nets obtained from the transformation.

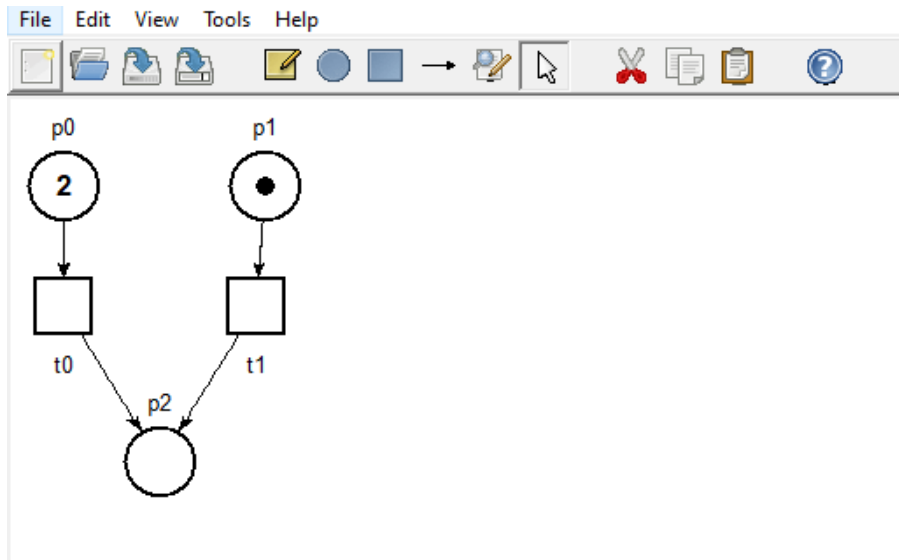


Figure 4.28: Representation of the TINA tool (Input).

digest		places	transitions	net	bounded	live	reversible
		3	2		Y	N	N
abstraction		count	props	psets	dead	live	
states		6	3	6	1	1	
transitions		7	2	2	0	0	

```

state 0
props L.scc*5 p0*2 p1
trans t0/1 t1/2

state 1
props L.scc*3 p0 p1 p2
trans t0/3 t1/4

state 2
props L.scc*4 p0*2 p2
trans t0/4

state 3
props L.scc p1 p2*2
trans t1/5

state 4

```

Figure 4.29: Representation of the TINA tool (Output).

4 Explanatory Example:

To illustrate our approach, we chose to explore an example case study to demonstrate the transformation steps (the rules). We applied our approach to an aspect-oriented detailed sequence diagram of the authentication case. In Figure (4.18), we present the composite model (Weaver) of the authentication case after executing T_OA2RDP, which contains the proposed graph grammar. We obtain the Petri net, represented in Figure (4.30).

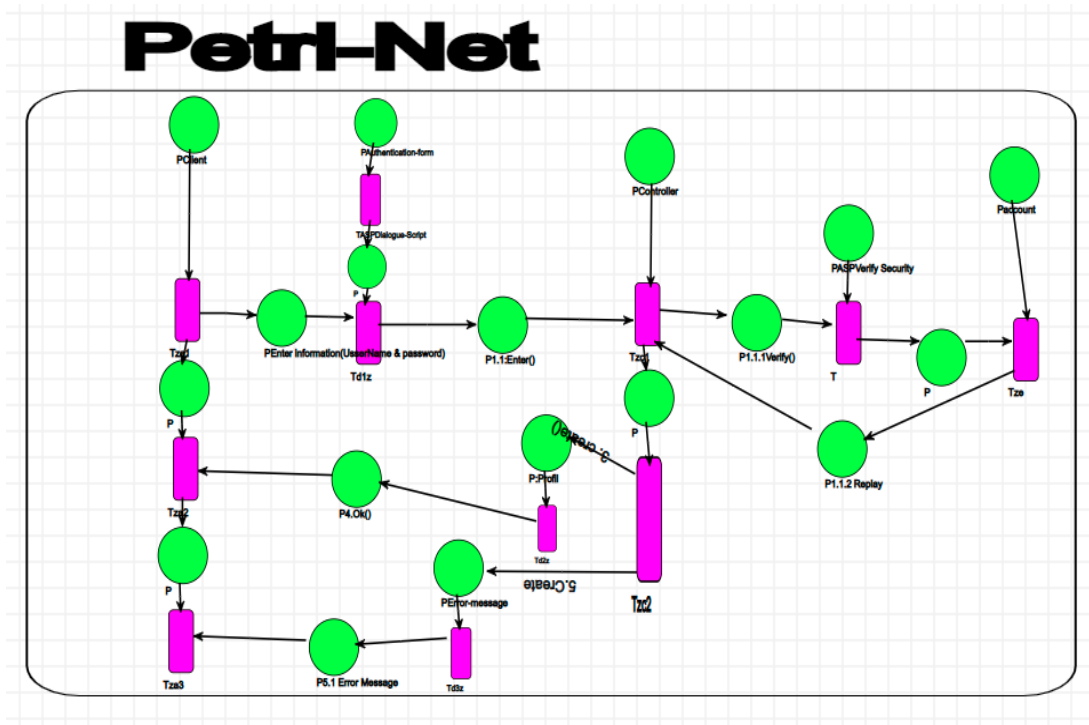


Figure 4.30: Authentication Petri Net.

Figure 4.30 shows the result of the verification of the Authentication Petri Net in the TINA tool.

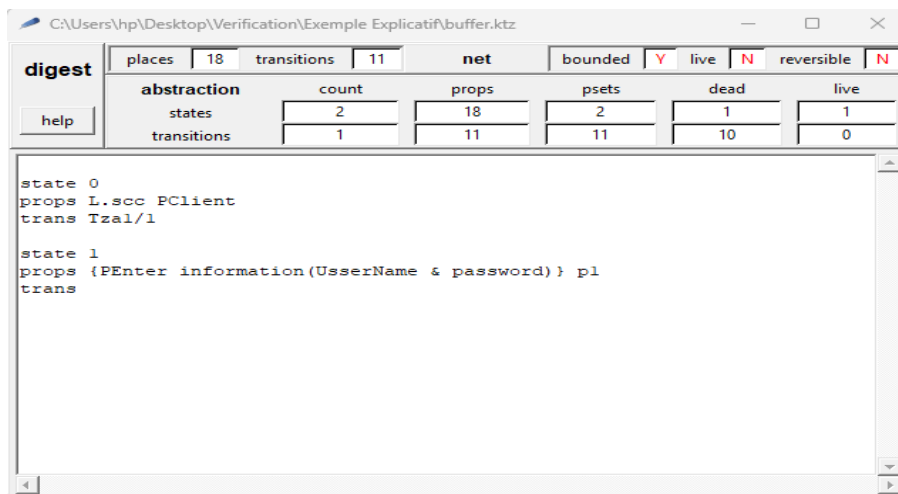


Figure 4.31: Analysis of the Authentication Petri Net.

5 Results and Discussion:

a) Results:

In this memory, 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 M. Aouag (2014) titled "Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes". In addition, the work of A. Zerara (2020) titled "La génération d'un outil de transformation des diagrammes UML 2.0 vers

Chapter 4: Proposed Approaches

les diagrammes orientés aspect, basée sur la transformation de graphes". We have conducted a comparison with the works related to our research in Table 4.2.

The comparaison work Comparison Points	M.Aouag,2013	A. Zerara,2020	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	ATOM ³	ATOM ³	ATOMPM

Table 4.2: Comparison of Approaches for Transforming into Aspect-Oriented Diagrams.

- ✚ The works related to the second approach, which involves transformation into Petri nets for verification purposes, include M. Bouarioua (2013) with his approach based on graph transformations for generating analyzable Petri net models from UML diagrams, and R. El Mansouri (2009) with his work on modeling and verifying business processes in virtual enterprises using a graph transformation-based approach. We have conducted a comparison with the works related to our research in Table 4.3.



The comparaison work Comparison Points	M.Bourioua,2013	R. Elmansouri, 2009	Our Approach
Transformation Approach	Modeling and verification of UML diagrams	Modeling and verification of business processes	Modeling and verification of UML2.0 diagrams

Chapter 4: Proposed Approaches

Design Model	Simple sequence diagram, State transition diagram	Simple sequence diagram, Business processes	Detailed sequence diagram
Graph	Based on graph transformation	Based on graph transformation	Based on graph transformation
Verification Method	Transformation to Petri nets	Transformation to Petri nets	Aspect-oriented transformation to Petri nets
Modeling Tool	ATOM ³	ATOM ³	ATOMPM
Verification Tool	TINA	INA	TINA

Table 4.3: Comparison of Petri Net Transformation and Verification Approaches

b) 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.

Conclusion:

In this chapter, we have proposed two approaches to transform a source model into a target model based on graph transformations. The two approaches are:

- Transformation from object-oriented detailed sequence diagrams to aspect-oriented detailed sequence diagrams.
- Transformation from aspect-oriented detailed sequence diagrams to Petri nets.

We have proposed a meta-model for the input model and the output model for the first approach, as well as two meta-models for the second approach. Then, we have formulated a graph grammar to perform the transformation from an object-oriented detailed sequence diagram to an aspect-oriented detailed sequence diagram. The result is then transformed into Petri nets using the AToMPM modeling tool. The obtained Petri nets are then verified using the TINA verification tool. Finally, we discuss our work in relation to other works related to our research.

Chapter 5

Case Studies

Introduction:

In this chapter, we implemented our transformation method on two case studies. The first one aimed to convert detailed object-oriented sequence diagrams into detailed aspect-oriented sequence diagrams. Subsequently, we applied a second approach to transform the obtained diagrams (the detailed aspect-oriented sequence diagrams) into Petri nets for verification. The first case study focuses on reservation in a travel agency, and the second on managing a shopping center.

1 Case Study on Booking in a Tourist Agency:

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 (5.1), we present the basic model for the detailed sequence diagram.

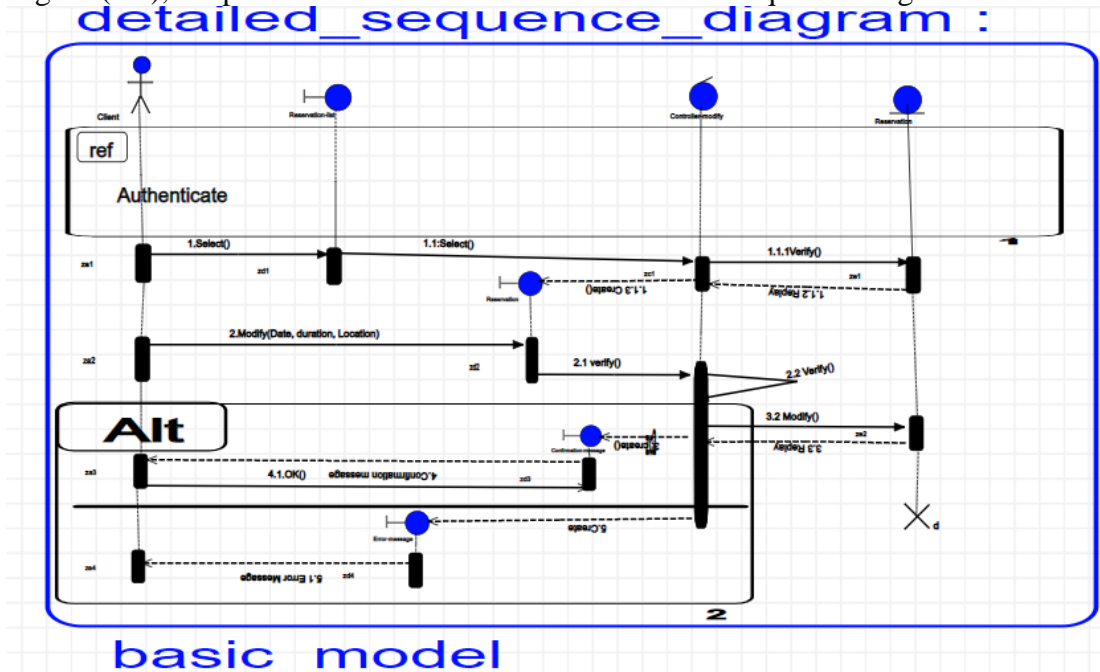


Figure 5.1: The Basic Model.

- **The Aspect Model:**

In Figure (5.2), we present the aspect model of the detailed sequence diagram.

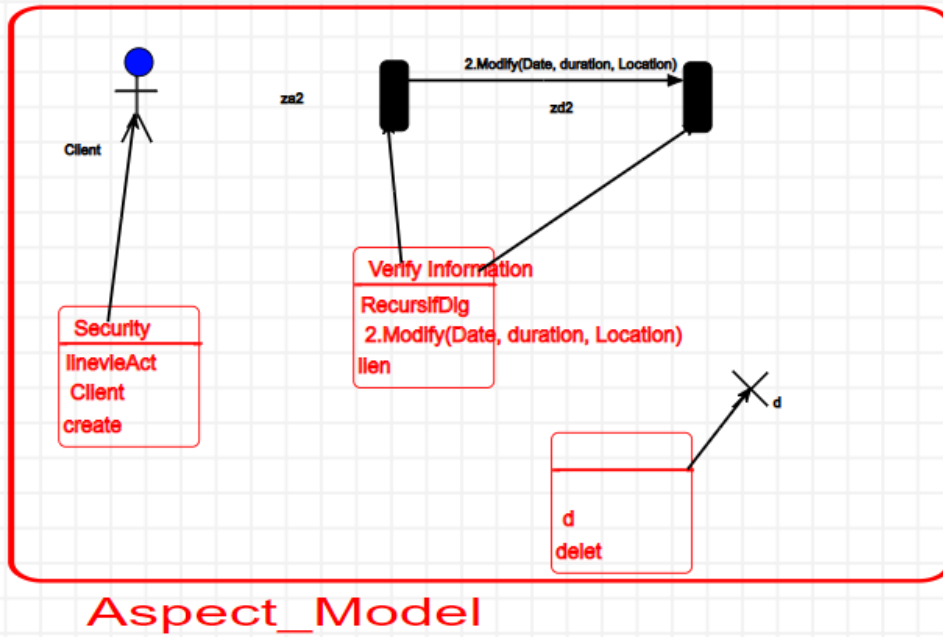


Figure 5.2: The Aspect Model.

❖ **Composite Model for the Detailed Sequence Diagram:**

In Figure 5.3, 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.

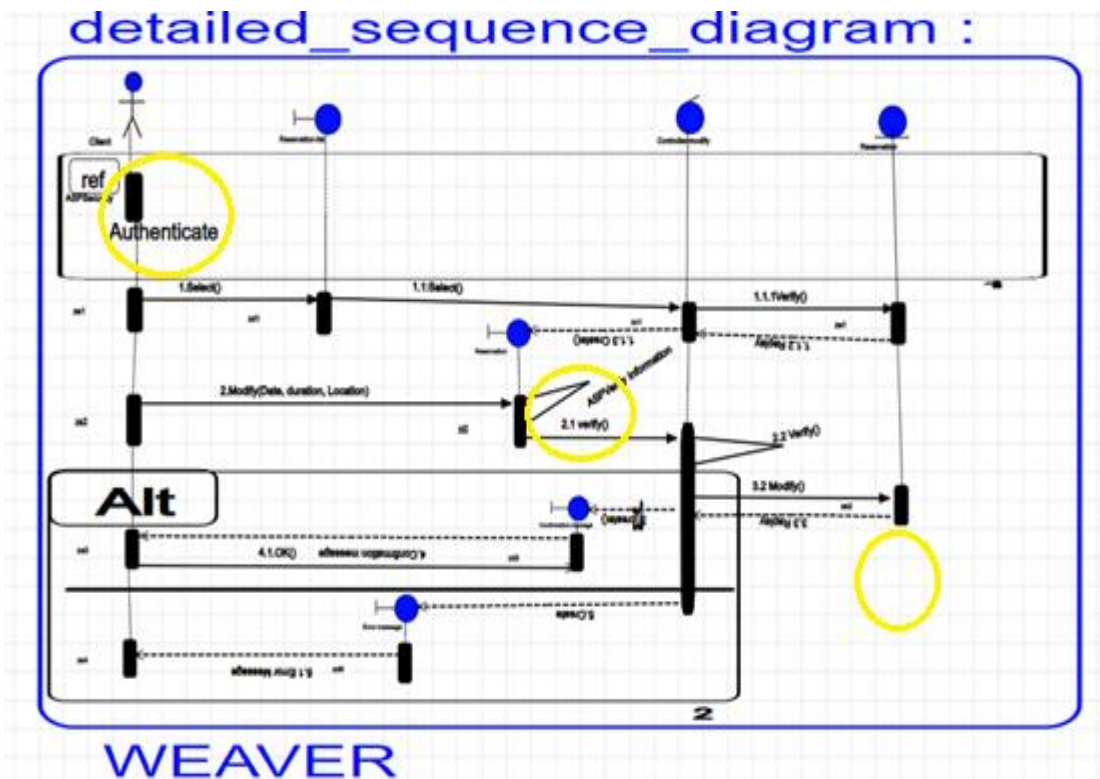


Figure 5.3: The Weaver.

Chapter 5: Case Studies

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.

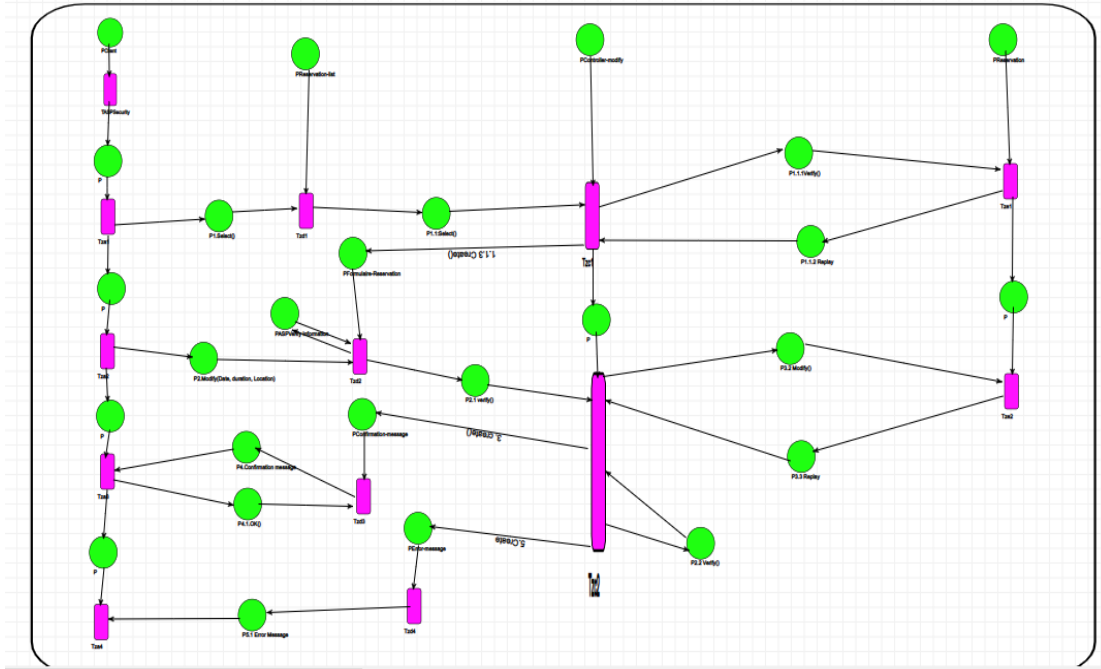


Figure 5.4: The Petri Net of the reservation modification case.

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:

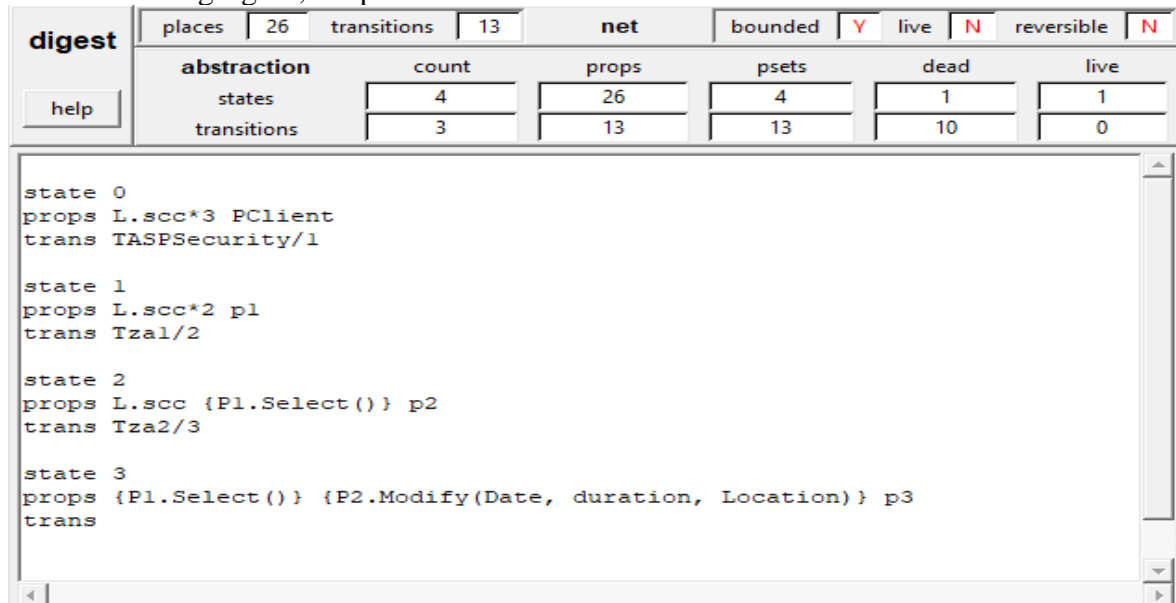


Figure 5.5: The result of the verification of the Petri Net for the reservation modification case in TINA.

✚ Based on the results:

- Bounded = Y: The network is bounded.
- Live = N: The network is not live.
- Reversible = N: The network is not reversible
- From the "dead" column, we deduce that there exists a dead state and 10 transitions are unreachable.

2 Case Study on the management of a shopping mall:

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.
- b) **update**: This aspect updates the entries in the database table by adding a promotion, positioned on the entity promotion.
- c) **product availability**: Verifies the availability of products before adding a promotion, positioned on the controller zone zc1.
- d) **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 (5.6), we present the basic model for the detailed sequence diagram.

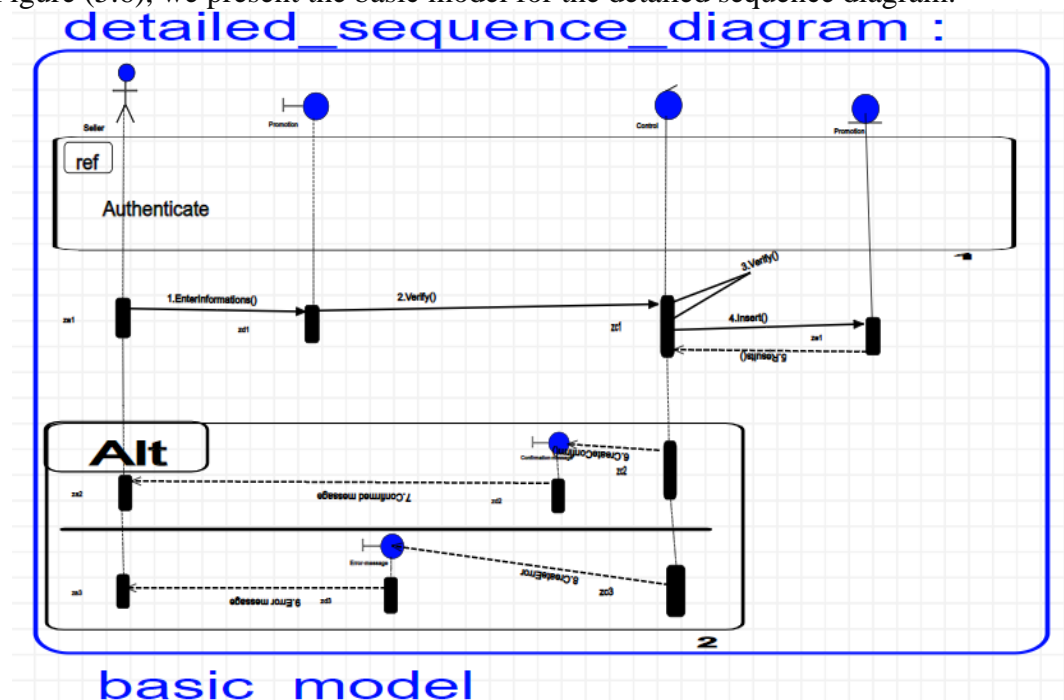


Figure 5.6: The Basic Model.

▪ The Aspect Model:

In Figure (5.7), we present the aspect model of the detailed sequence diagram.

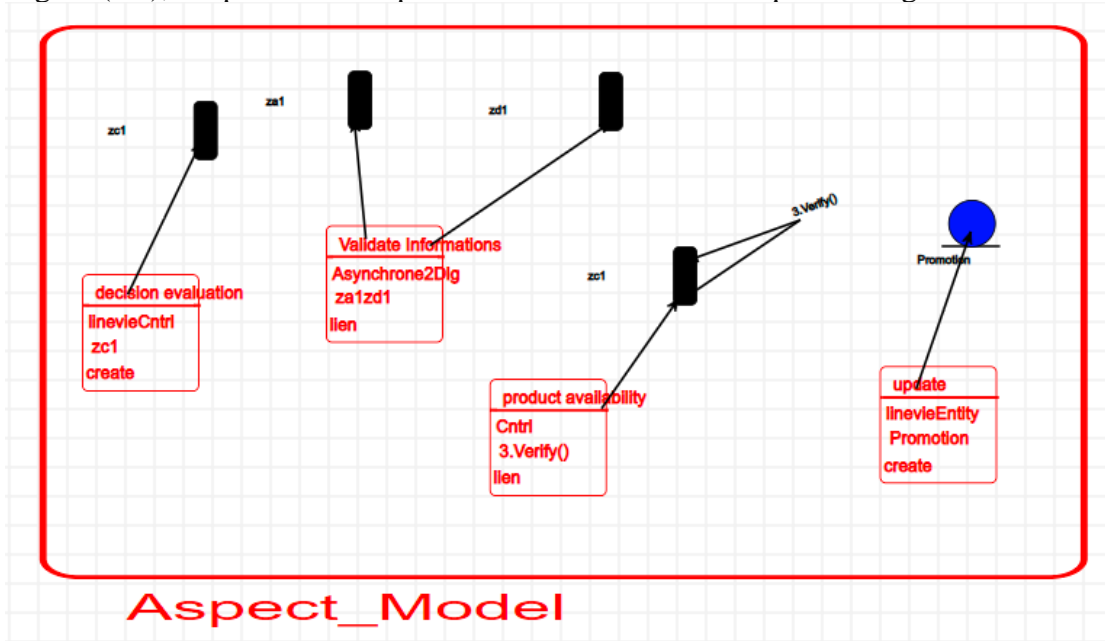


Figure 5.7: The Aspect Model.

❖ Composite Model for the Detailed Sequence Diagram:

In Figure (5.8), 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.

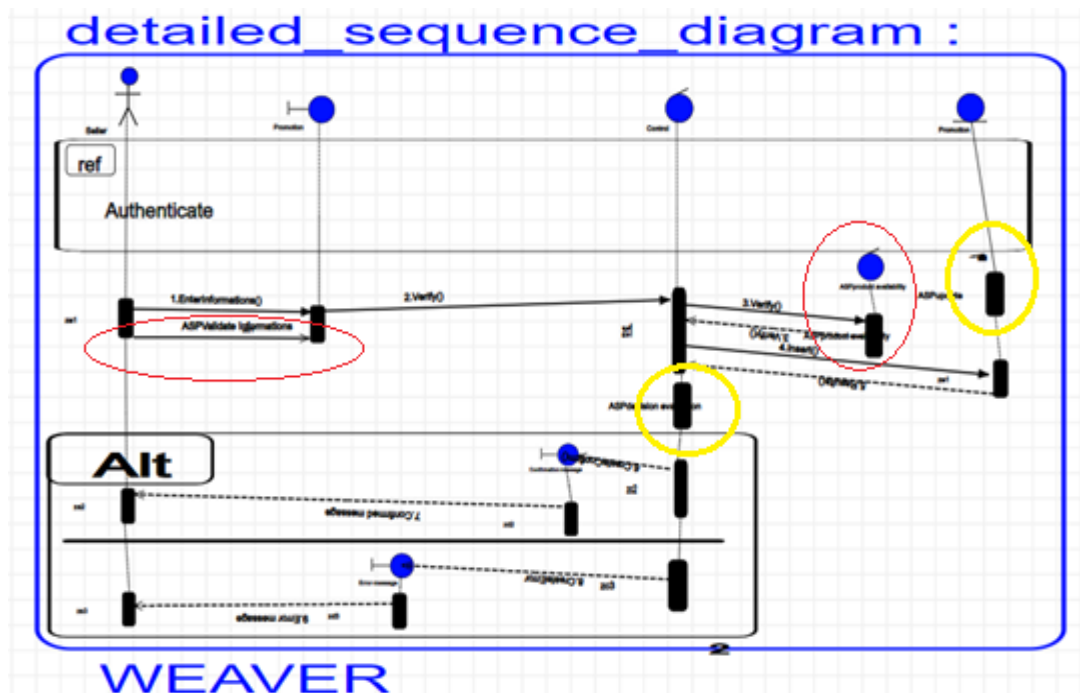


Figure 5.8: The Weaver.

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.

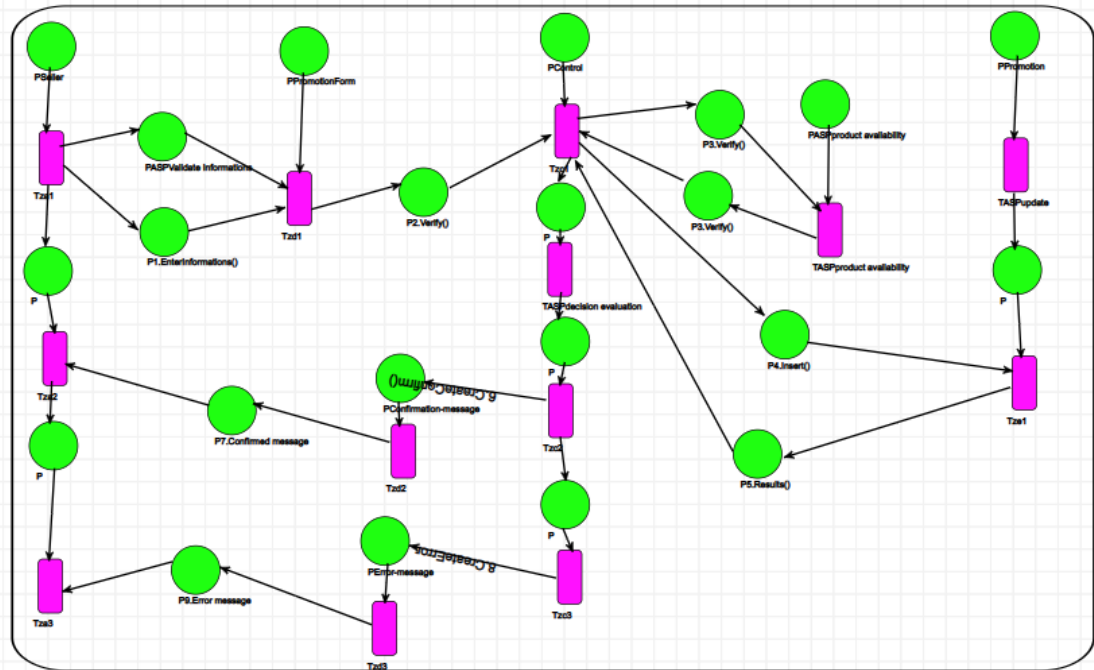


Figure 5.9: The Petri Net of the promotion addition case.

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:

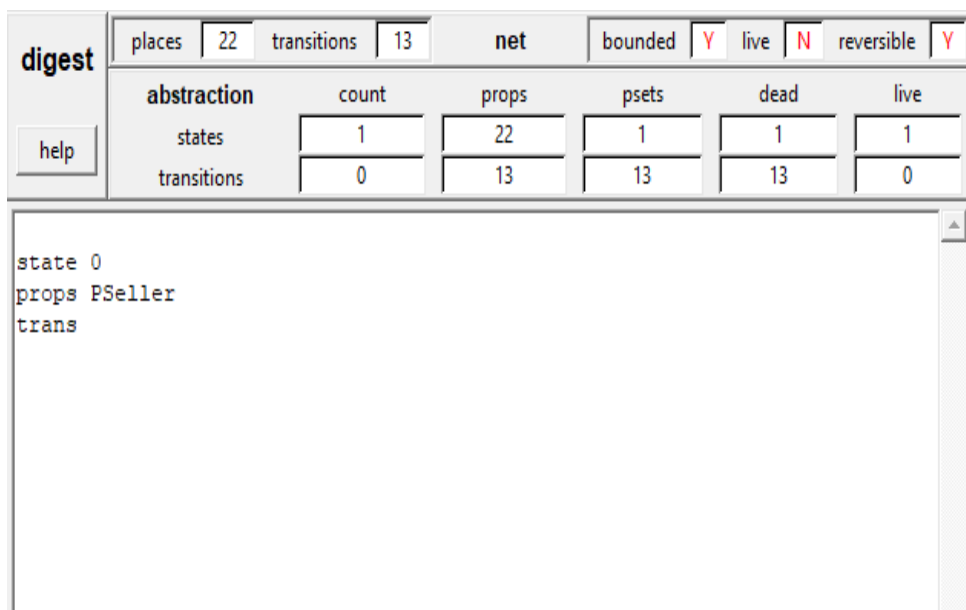


Figure 5.10: The result of verification of Petri Net for the promotion addition case with TINA.

Chapter 5: Case Studies

✚ Based on the results:

- Bounded = Y: The network is bounded.
- Live = N: The network is not live.
- Reversible = Y: The network is reversible
- From the "dead" column, we deduce that there exists a dead state and 13 transitions are unreachable.

Conclusion:

In this chapter, we applied our transformation method to two case studies: the first concerning reservations in a travel agency, and the second related to the management of a shopping center. For each case study, we first converted the detailed object-oriented sequence diagrams into detailed aspect-oriented sequence diagrams. Then, we used the resulting diagrams to apply the proposed graph grammar and generate the corresponding Petri nets. Finally, we performed a verification on the Petri nets for each case.

Conclusion and Perspectives

Conclusion and Perspectives

Conclusion:

Object-oriented modeling has long been favored in the field of software engineering for the design and development of software systems. Despite its clear advantages in representing system structures and behaviors, this approach does have certain limitations. One of its main shortcomings lies in its ability to effectively manage cross-cutting concerns. In light of these challenges, aspect-oriented modeling emerges as a promising solution. By introducing the notion of aspects, this approach allows for better separation of concerns and more efficient management of cross-cutting aspects, leading to clearer and more modular software design, reducing complexity, and improving system maintainability.

In this memory, we proposed an approach based on graph transformation to generate aspect-oriented sequence diagrams from object-oriented sequence diagrams. This approach relies on meta-modeling, with the definition of a meta-model for the sequence diagram. Then, we presented a set of rules for the transformation, carried out using the AToMPM modeling tool.

After performing the transformation of object-oriented sequence diagrams into aspect-oriented sequence diagrams, we applied our approach to several representative case studies. Finally, we proposed an extension of our approach by suggesting the transformation of the obtained aspect-oriented sequence diagram model into formal models, such as Petri nets, to enable system verification and validation. This approach aims to enhance the reliability and robustness of the developed software, thereby contributing to ensuring its quality and compliance with specified requirements. Our proposed approach involves defining a meta-model for Petri nets and proposing a graph grammar. We take as input the result of the first transformation (the aspect-oriented sequence diagram). Then, we apply this graph grammar to obtain the corresponding Petri net. Then, we make a comparison between works related to our work and discuss this comparison. Finally, we applied our approach to case studies.

Verification is essential to ensure that the transformed Petri net respects the essential properties, thereby ensuring the accuracy and reliability of the modeled system. To conclude our work, we verified the generated Petri nets using the TINA tool.

Perspectives:

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.

Bibliographic References:

- [AMROUNE,2014] AMROUNE, Mohammed. (2014). VERS UNE APPROCHE ORIENTÉE ASPECT D'INGÉNIERIE DES BESOINS DANS LES ORGANISATIONS MULTI ENTREPRISES. Thèse de Doctorat. UNIVERSITÉ DE TOULOUSE. Pp : 50
- [Aouag,2023] Course support Aouag Mouna 2023 Disponible sur : https://elearning.centre-univ-mila.dz/a2024/pluginfile.php/69978/mod_resource/content/0/Chapitre%2002_%20Les%20R%C3%A9seaux%20de%20P%C3%A9tri%20_RDP.pdf (Consulté le 01/06/2024)
- [Aouag,2014] Aouag, Mouna. (2014). Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes. Thèse de doctorat, Université de Mentouri, Constantine. Pp :11-33
- [Audibert,2008] <https://www.fichier-pdf.fr/2011/05/08/cours-uml/preview/page/14/>
- [AToMPM Documentation] [AToMPM Documentation — AToMPM 0.10.0 documentation](#) (Consulté le 05/05/2024)
- [BAHRI,2011] BAHRI, Mohamed Redha. (2011). Une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles. Thèse de doctorat, Université Mentouri Constantine. Pp: 32-68
- [Boualita et Laggoune,2021] Boualita, Salim. Et Laggoune, Fouad. La génération d'un outil de transformation des modèles orientés aspect vers des modèles formels. Une approche basée sur la transformation de graphes. Mémoire de master. Centre universitaire Mila. Pp :6
- [Boubendir,2011] Boubendir, Amel. (2011). Un cadre générique pour la détection et la résolution des interaction entre les aspects. Thèse de doctorat, Université de Mentouri, Constantine. Pp: 19-36
- [BENDIAF,2018] BENDIAF, Messaoud. (2018). SPÉCIFICATION ET VÉRIFICATION DES SYSTÈMES EMBARQUÉS TEMPS RÉEL EN UTILISANT LA LOGIQUE DE RÉÉCRITURE. Thèse de Doctorat. UNIVERSITE MOHAMED KHIDER BISKRA. Pp: 78-79
- [CHERIEF et MELIANI,2020] CHERIEF,Saliha. Et MELIANI,Toufik. (2020). Une Approche De transformation et D'analyse Des Diagrammes D'activités. Mémoire de Master. Université Djilali Bounaama-Khemis Miliana. Pp: 62-74
- [David et Alia,2005] Rene David et Hassane Alia."Discrete, Continuous, and Hybrid Petri Nets". New York: Springer Berlin Heidelberg. 2005. 524p.
- [Dehimi,2014] Dehimi,Nardjess,Tissilia.(2014). Un Cadre Formel pour La Modélisation et L'analyse Des Agents Mobiles. Thèse de Doctorat. Université de Mentouri Constantine. Pp : 66-67
- [Dib et Saadaoui,2023] Dib, Wiam.et Saadaoui, Amani. Spécification formelle des modèles orientés aspect, une approche basée sur la transformation de graphes et le langage Maude. Mémoire de master. Center universitaire Mila. Pp :4-9
- [ElMansouri,2009] ElMansouri, R. (2009). Modélisation et Vérification des processus métiers dans les entreprises virtuelles : Une approche basée sur la transformation de graphes. [En ligne]. Available: <https://bu.umc.edu.dz/theses/informatique/ELM5432.pdf>. [Accès le 1 juin 2024].
- [Guerrouf,2011] GUERROUF FAYÇAL. (2011). Une approche de transformation des Diagrammes d'Activités d'UML Mobile 2.0 vers les Réseaux de Petri. Memoire de magistere. Université El Hadj Lakhdar BATNA. Pp :28-32

Bibliographic References

- [**HACHICHI,2013**] HACHICHI, Hiba. (2013). Test formel des systèmes temps réel: Approche de transformation de graphes. Thèse de Doctorat. Université Mentouri de Constantine. Pp : 25
- [**HADDOUCHE, DAHAMNA,2022**] HADDOUCHE, MEBARKA. Et DAHAMNA, NOURA ELALDJA. (2022). La transformation automatique des diagrammes d'états-transitions vers les réseaux de Petri. Mémoire de Master. Université Mohamed El Bachir ElIbrahimi de Bordj Bou Arreridj. Pp15
- [**Hamri,2017**] HAMRI, Hakima. (2017). Contribution à la commande des systèmes à événements discrets soumis à des contraintes temporelles. Thèse de doctorat. UNIVERSITÉ MOULOUD MAMMARI DE TIZI-OUZOU. Pp :10
- [**HAMROUCHE,2010**] HAMROUCHE, Houda. (2010). Une Approche de transformation des diagrammes D'activité d'UML vers CSP basée sur la transformation de graphes. Memoire de magistere. UNIVERSITE 20 AOUT 1955 SKIKDA. Pp : 42-43
- [**HAOUES,2006**] HAOUES, MOHAMMED. (2006). L'UTILISATION CONJOINTE DES RÉSEAUX DE PETRI STOCHASTIQUES ET DES PROCESSUS DE MARKOV POUR LA MODÉLISATION, L'ANALYSE ET L'ÉVALUATION DES PERFORMANCES D'UN SYSTÈME DE PRODUCTION : LIGNE D'EMBOUITISSAGE DE L'ENTREPRISE B.A.G BATNA. Mémoire de Magister. UNIVERSITÉ EL-HADJ LAKHDAR BATNA. Pp: 27
- [**Hettab,2009**] HETTAB ABDELKAMEL. (2009). De M-UML vers les réseaux de pétri « Nested Nets »: une approche basée sur la transformation de graphes. Thèse de doctorat. Université de mentouri Constantine. Pp:58-74
- [**Kerkouche,2011**] Kerkouche, Elhillali. (2011). Modélisation Multi-Paradigme : Une Approche Basée sur la Transformation de Graphes. Thèse de doctorat. Université de Mentouri, Constantine. Pp: 9-52
- [**Khalfaoui,2014**] khalfaoui, Khaled. (2014). Une Approche de Spécification des Changements des Besoins Basée Transformation de Graphes. Thèse de doctorat, UNIVERSITE MOHAMED KHIDER BISKRA. Pp : 21-25
- [**Laouar,2013**] Laouar, Adnane. (2013). Une approche de transformation de diagrammes d'activités oriente aspects vers les réseaux de pétri colores basée sur la transformation de graphes. Mémoire de master. Université de constantine2. Pp :16-33
- [**Ludovic,2009**] Ludovic, Auxepaules. (2009). Analyse des diagrammes de l'apprenant dans un EIAH pour la modélisation orientée objet - Le système ACDC. Thèse de doctorat. Université du Maine USA. Pp :19-28
- [**MEDJANI,2020**] MEDJANI, Djedjiga. (2020). Analyse des Performances d'un Système de Gestion de Stocks à Produits Périssables par les RdPSG. Mémoire de Master. Université Abderahmane Mira de Béjaia. Pp : 31-32
- [**Otmane Rachedi,2015**] Otmane rachedi, Soumeya. (2015). Apports des Approches de Séparation Avancée des Préoccupations: Une Etude Comparative Fondée sur les Modèles de Conception. Thèse de doctorat. Université de Badji Mokhtar Annaba. Pp :17-18
- [**Pascal Roque, 2008**] Pascal roques, Uml 2 par la pratique, Paris : Eyrolles, 2008, 246p.
- [**SAGGADI,2007**] SAGGADI, Samira. (2007). Optimisation des temps d'attente des systèmes flexibles de production basée sur les réseaux de Petri. Mémoire de Magister. Université M'hamed Bougara Boumerdès. Pp :25
- [**Syriani et al,2013**] Eugene, Syriani. Hans, Vangheluwe. Raphael, Mannadiar. Conner, Hansen. Simon Van, Mierlo. And Huseyin, Ergin. AToMPM: A Web-based Modeling Environment [en ligne]. 2013.vol 1080. Disponible sur : <https://ceur-ws.org/Vol-1115/demo4.pdf> (Consulté le 05/05/2024)
- [**TINA Documentation**] [The TINA toolbox Home Page - TIme petri Net Analyzer - by LAAS/CNRS](#) (Consulté le 01/06/2024)

Bibliographic References

[Zerara et Megrous,2020] Zerara, Ahmed. Et Megrous, Fares. (2020). La génération d'un outil de transformation des diagrammes UML2.0 vers les diagrammes orientés aspect basé sur la transformation de graphes. Mémoire de master. Centre universitaire Mila. Pp :23-28