

Methodology for automated Petri Net model generation to support Reliability Modelling

by

Christina Latsou

Doctoral Thesis

Submitted in partial fulfilment of the requirements
for the award of

Doctor of Philosophy
of Loughborough University

December 2018

© by Christina Latsou 2018

Dedicated to my parents, Theodora and Christos,

and to Dimitris

Acknowledgments

I would like to thank my supervisors, Sarah Dunnett and Lisa Jackson, for their support, guidance and their confidence in me throughout my Ph.D. I am lucky to have had two committed supervisors, who in their different, yet complementary ways have guided and shaped this research. I would like to express my sincerest thanks to Sarah and Lisa for their contributions, and for their continued help and encouragement over the years. I learned from their insights a lot.

Special thanks to all my friends and fellow researchers in the office, with whom I have shared the good and bad times and all of those at Loughborough University who have directly or indirectly offered help, suggestions in bringing towards the completion of this research.

My sincerest thanks go to my family, particularly to my parents for their love, support and for believing in me. Finally, my deepest gratitude goes to my partner, Dimitris, for his understanding and unending encouragement throughout the last 3 years.

Abstract

As the complexity of engineering systems and processes increases, determining their optimal performance also becomes increasingly complex. There are various reliability methods available to model performance but generating the models can become a significant task that is cumbersome, error-prone and tedious. Hence, over the years, work has been undertaken into automatically generating reliability models in order to detect the most critical components and design errors at an early stage, supporting alternative designs. Earlier work lacks full automation resulting in semi-automated methods since they require user intervention to import system information to the algorithm, focus on specific domains and cannot accurately model systems or processes with control loops and dynamic features.

This thesis develops a novel method that can generate reliability models for complex systems and processes, based on Petri Net models. The process has been fully automated with software developed that extracts the information required for the model from a topology diagram that describes the system or process considered and generates the corresponding mathematical and graphical representations of the Petri Net model. Such topology diagrams are used in industrial sectors, ranging from aerospace and automotive engineering to finance, defence, government, entertainment and telecommunications. Complex real-life scenarios are studied to demonstrate the application of the proposed method, followed by the verification, validation and simulation of the developed Petri Net models. Thus, the proposed method is seen to be a powerful tool to automatically obtain the PN modelling formalism from a topology diagram, commonly used in industry, by:

- Handling and efficiently modelling systems and processes with a large number of components and activities respectively, dependent events and control loops.
- Providing generic domain applicability.
- Providing software independence by generating models readily understandable by the user without requiring further manipulation by any industrial software.

Finally, the method documented in this thesis enables engineers to conduct reliability and performance analysis in a timely manner that ensures the results feed into the design process.

Contents

List of Figures	xvii
List of Tables	xx
 CHAPTER 1 - Introduction	1
1.1 Introduction to Reliability Modelling	1
1.1.1 Analytical Reliability Modelling Methods	2
1.1.1.1 Fault Tree	2
1.1.1.2 Cause-Consequence Diagram	3
1.1.1.3 Reliability Block Diagram	3
1.1.1.4 Markov Method	4
1.1.1.5 Petri Net	5
1.1.2 Simulation Modelling Methods	5
1.1.3 Reliability Modelling, Implementation and Deficiencies	6
1.2 Introduction to Automated Reliability Model Generation	7
1.3 Industrial Representation of Systems	9
1.3.1 Introduction to System Modelling Tools	9
1.3.2 Summary of System Modelling Tools	13
1.4 Research Scope and Delimitations	14
1.5 Aim and Objectives	14
1.6 Thesis Layout	16
 CHAPTER 2 – Modelling Tools and Methods – Automated Reliability Modelling	19
2.1 Introduction	19
2.2 Industrial System Representation	20
2.2.1 Systems Modelling Languages Review: UML and SysML	20
2.2.1.1 UML/SysML Activity Diagram	25
2.3 Petri Net Modelling Review	28
2.4 Methods for Automation of Reliability Models	33
2.4.1 Introduction	33

2.4.2	Overview of Methods for Automated Reliability Modelling (Not Including PN)	34
2.4.2.1	Automated Generation of the Fault Tree Model	34
2.4.2.2	Automated Generation of Other Reliability Models	38
2.4.2.3	Summary of Methods	39
2.5	Automated Generation of the Petri Net Model	41
2.5.1	Review of Methods for Automated Petri Net Modelling	41
2.5.2	Summary of Methods for Automated Petri Net Modelling.	55
2.6	Review and Research Motivations	56
CHAPTER 3 – Methodology for the Automated Generation of a Petri Net Model		58
3.1	Introduction.....	58
3.2	Overview of Developed Methodology.....	59
3.3	Input – System Modelling.....	60
3.4	Algorithm – Java Database Programming.....	63
3.4.1	Transformation Rules	63
3.4.2	Database Introduction.....	64
3.4.2.1	Relational Database Management Systems Products Review	64
3.4.3	Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix.....	66
3.4.4	Algorithm – Java Database Programming – Petri Net Initial Marking Matrix.....	76
3.5	Automated Graphical Representation of a Petri Net Model	78
3.6	Summary	81
CHAPTER 4 – Application of the Automated Petri Net Model Generation Methodology to a Recycling IT Asset Process		82
4.1	Introduction.....	82
4.2	Process Description.....	82

4.3	Manual Development of the Petri Net Model for the Recycling IT Asset Process	84
4.4	Automated Mathematical Representation of the Petri Net Model for the Recycling IT Asset Process.....	86
4.4.1	Input – System Modelling	86
4.4.2	Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix.....	87
4.4.3	Algorithm – Java Database Programming – Petri Net Initial Marking Matrix.....	92
4.5	Automated Graphical Representation of the Petri Net Model for the Recycling IT Asset Process.....	93
4.6	Summary	94
CHAPTER 5 – Verification & Validation of Petri Net Model		95
5.1	Introduction.....	95
5.2	Petri Net Model Verification Methods	96
5.2.1	Static Verification Methods.....	96
5.2.2	Dynamic Verification Method.....	99
5.2.3	Comparison of PN Models (Bi-Simulation) for Verification	100
5.3	Verification of Automated Petri Net Development	100
5.4	Petri Net Model Validation Methods.....	105
5.4.1	Expert Intuition Validation Method.....	105
5.4.2	Real System Measurements Validation Method.....	105
5.4.3	Theoretical Results/Analysis Method.....	106
5.5	Validation of Automated Petri Net Development.....	106
5.5.1	Petri Net Model Simulation Algorithm	107
5.5.2	Process Input Data	109
5.5.3	Petri Net Model Visual Check.....	111
5.5.4	Petri Net Model Numerical Simulation and Performance Analysis	114
5.5.5	Performance Analysis Results and Discussion.....	116
5.6	Summary	118

CHAPTER 6 – Advanced Generic Methodology for the Automated Generation of a Petri Net Model	121
6.1 Introduction.....	121
6.2 Introduction of UML/SysML AD Additional Elements Notation.....	122
6.3 Input – System Modelling.....	125
6.3.1 Introduction.....	125
6.3.2 Uml/Sysml AD: Review of XMI Nested Elements.....	126
6.3.3 The Need of XMI Model Transformation using XSLT.....	133
6.3.3.1 First XMI Model Transformation using XSLT	134
6.3.3.2 Second XMI Model Transformation using XSLT	136
6.3.4 Application of the XMI Model Transformations to a Simple AD Example	137
6.4 Generic Algorithm – Java Database Programming	139
6.4.1 Transformation Rules	139
6.4.2 Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix.....	143
6.4.3 Algorithm – Java Database Programming – Petri Net Initial Marking Matrix.....	146
6.5 Summary.....	147

CHAPTER 7 – Application of the Generic Automated Petri Net Model Generation

Methodology to Real Life Scenarios.....	149
7.1 Introduction.....	149
7.2 Production System	149
7.2.1 Process Description	149
7.2.2 Automated Mathematical Representation of the Petri Net Model for the Production System.....	152
7.2.2.1 Input – System Modelling	152
7.2.2.2 Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix	153
7.2.2.3 Algorithm – Java Database Programming – Petri Net Initial Marking Matrix.....	156

7.2.3	Automated Graphical Representation of the Petri Net Model for the Production System	157
7.3	Online Shopping Process	159
7.3.1	Process Description	159
7.3.2	Automated Mathematical Representation of the Petri Net Model for the Online Shopping Process	161
7.3.2.1	Input – System Modelling	161
7.3.2.2	Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix	161
7.3.2.3	Algorithm – Java Database Programming – Petri Net Initial Marking Matrix.....	167
7.3.3	Automated Graphical Representation of the Petri Net Model for the Online Shopping Process	167
7.4	Verification and Validation of Real-Life Scenarios	169
7.5	Summary	170
CHAPTER 8 – Conclusions and Future Work		171
8.1	Introduction.....	171
8.2	Conclusions.....	171
8.3	Contributions to Knowledge	173
8.4	Recommendations for Future Work.....	173
8.4.1	Automated Sub-PNs Construction followed by Simulation Analysis	173
8.4.2	Automated Reliability Analysis.....	174
8.4.3	Additional PN Model Features	175
8.4.4	Investigation of Inputs	175
8.4.5	Representation of PN Results into the UML/SysML AD	176
Bibliography		178
Appendix A - Simple Process Example (XMI File)		190
Appendix B – SQL CODE [A^T].....		192
Appendix C – SQL CODE [M₀]		200

Appendix D – Graphical Representation of PN Model	202
Part A – SQL Code for the Automated PN Generation.....	202
Part B – DOT File for the PN Model Generation (GraphViz Input).....	204
Appendix E – Recycling IT Asset Process Example (XMI File)	205
Appendix F	207
Part A – Validation – PN Visual Check (Token Game).....	207
Part B – Validation – PN Model Numerical Simulation	211
Part C – Validation – PN Model Performance Analysis	214
Appendix G.....	215
Part A – AD Examples in Chapter 6 (XMI Files)	215
Part B –XSLT Files	220
Part C – Java Code for the XMI Transformations.....	222
Part D – Outputs from XMI Model Tranformation of AD in Figure 6.4	224
Appendix H – Advanced Generic SQL Code [A^T].....	228
Appendix I – Advanced Generic SQL Code [M_0].....	249
Appendix J.....	252
Part A- Production System Example (XMI File)	252
Part B – Online Shopping Process (XMI File)	255
Appendix K – PN Mathematical Forms of the Production System and Online Shopping Process.....	263
Part A	263
Part B	264
Part C	265
Part D.....	267
Appendix L – Verification of the Production System and Online Shopping Process	268

List of Figures

Figure 2.1 Illustration of the Structure of Chapter 2.....	19
Figure 2.2 UML 2 Diagrams Taxonomy (OMG Unified Modelling Language (UML), Version 2.5, 2015)	21
Figure 2.3 SysML Diagrams Taxonomy (Object Management Group, 2006)	23
Figure 2.4 UML Activity Diagram Example (OMG Unified Modelling Language (UML), Version 2.5, 2015)	28
Figure 2.5 Inhibitor Arc	30
Figure 2.6 PN Firing Process with Inhibitor Arc	30
Figure 2.7 PN Firing Process	31
Figure 2.8 Reachability Matrix	38
Figure 2. 9 Simple Component Conversion into a Simple-Component CPN (Robidoux et al., 2010).....	48
Figure 3.1 Illustration of the Structure of Chapter 3.....	58
Figure 3.2 Methodology Steps for Automated PN Generation.....	59
Figure 3.3 AD for a Simple Process	61
Figure 3.4 Part of XMI File retrieved from the AD for the Simple Scenario.....	61
Figure 3.5 Model developed for the AD for the Simple Process.....	63
Figure 3.6 Flowchart for the steps followed for the Automated Generation of the Mathematical Representation of a PN Model	68
Figure 3.7 Part of XMI File retrieved from the AD.....	70
Figure 3.8 Part of XMI File retrieved from the AD.....	70
Figure 3.9 Flowchart for Step 2 for the Automated Graphical Representation of a PN Model	79
Figure 3.10 Automated Layout of the Petri Net Model for the Simple Example.....	80
Figure 4.1 UML AD of the Recycling IT Asset Process	84
Figure 4.2 PN Model developed for the UML AD for the Recycling IT Asset Process.....	85
Figure 4.3 Part of XMI File retrieved from the AD for Table 4.2 Generation	87

Figure 4.4 Part of XMI File retrieved from the AD for Table 4.3 Generation	88
Figure 4.5 PN Model for the Recycling IT Asset Process	93
Figure 5.1 Illustration of the Structure of Chapter 5	95
Figure 5.2 Reachability Graph Example (Aalst, 2011)	98
Figure 5.3 Petri Net developed in HiPS for the Recycling IT Asset Process	102
Figure 5.4 Structurally and Behaviourally Bounded Check in HiPS for the Recycling IT asset Process	104
Figure 5.5 Behavioural Liveness and Safeness Properties Check in HiPS for the Recycling IT asset Process	104
Figure 5.6 Flowchart for the Simulation Steps followed for the Recycling IT Asset Process	108
Figure 5.7 PN Model Extract from the Recycling IT Asset Process	111
Figure 5.8 PN Model Extract from the Recycling IT Asset Process	112
Figure 5.9 Simulation Results for the 1 st Path of the Recycling IT Asset Process for 2500 Simulations	115
Figure 6.1 Illustration of the Structure of Chapter 6	122
Figure 6.2 AD Example (Fowler, 2004)	127
Figure 6.3 XMI Extract for the AD in Figure 6.2	128
Figure 6.4 AD Example (MSDN Microsoft, 2017)	130
Figure 6.5 XMI Extract for the AD in Figure 6.4	130
Figure 6.6 AD Example (Sparx Systems, 2018)	131
Figure 6.7 XMI Extract for the AD in Figure 6.6	132
Figure 6.8 Part of the XMI File developed form the 1 st XMI Model Transformation	138
Figure 6.9 Part of the XML File developed form the 2 nd XMI Model Transformation	139
Figure 6.10 PN Model developed for the AD in Figure 6.2	139
Figure 6.11 PN Model developed for the AD in Figure 6.4	139
Figure 6.12 PN Model developed for the AD in Figure 6.6	140
Figure 6.13 AD (Central Buffer Node) and Corresponding PN Model (Pilone & Pitman, 2005)	140

Figure 6.14 AD (Data Store Node) and Corresponding PN Model (Pilone & Pitman, 2005)	140
Figure 6.15 AD (Call Behaviour Action) and Corresponding PN Model (Söding, 2009).....	141
Figure 6.16 AD (Activity Parameter Node) and Corresponding PN Model (Pilone & Pitman, 2005)	141
Figure 6.17 AD (Structured Activity Node) and Corresponding PN Model (Bock, 2005).....	142
Figure 6.18 Flowchart for the steps followed for the Generic Automated Generation of the Mathematical Representation of a PN Model	145
Figure 7.1 Production System (Villani et al., 2007)	150
Figure 7.2 UML AD for the Production System (Villani et al., 2007)	151
Figure 7.3 PN Model Automatically developed for the Production System	158
Figure 7.4 UML AD for the Online Shopping Process (Banas, 2012)	159
Figure 7.5 PN Model Automatically developed for the Online Shopping Process ...	168
Figure L.1 Structurally and Behaviourally Bounded Check in HiPS for the Production System.....	268
Figure L.2 Behavioural Liveness and Safeness Properties Check in HiPS for the Production System.....	268
Figure L.3 Structurally and Behaviourally Bounded Check in HiPS for the Online Shopping Process	268
Figure L.4 Behavioural Liveness and Safeness Properties Check in HiPS for the Online Shopping Process	268

List of Tables

Table 2.1 Notation and Description of Activity Diagram Control Nodes	26
Table 2.2 Notation and Description of Activity Diagram Nodes	27
Table 2.3 Notation and Description of Activity Diagram Edge	27
Table 2.4 Table of Methods for Automated Reliability Modelling (not including PN)	40
Table 3.1 Relationships between the AD and PN Notation and Symbols	63
Table 3.2 MySQL ‘node_xmi’ Table Extract	70
Table 3.3 MySQL ‘edge_place_xmi’ Table Extract	70
Table 3.4 MySQL ‘union_1’ Table Extract	71
Table 3.5 MySQL ‘unique_activities’ Table	72
Table 3.6 MySQL ‘union_node’ Table	72
Table 3.7 MySQL ‘union_node_table1’ Table Extract	73
Table 3.8 MySQL ‘union_node_table2’ Table	73
Table 3.9 MySQL ‘union_node_table2’ Table Extract	74
Table 3.10 MySQL ‘final_table’ Table	74
Table 3.11 MySQL ‘negative’ Table	75
Table 3.12 MySQL ‘positive’ Table	75
Table 3.13 MySQL ‘transpose_of_the_incidence_matrix’ Table	76
Table 3.14 MySQL Database ‘initial_marking Table	77
Table 4.1 Abbreviations and Full Names of Nodes and Edges from UML AD	84
Table 4.2 MySQL ‘node_xmi’ Table Extract	87
Table 4.3 MySQL ‘edge_place_xmi’ Table Extract	88
Table 4.4 MySQL ‘union_1’ Table Extract	88
Table 4.5 MySQL ‘union_node’ Table Extract	89
Table 4.6 MySQL ‘union_node_table1’ Table Extract	89
Table 4.7 MySQL ‘union_node_table2’ Table	90
Table 4.8 MySQL ‘final_table’ Table	90
Table 4.9 MySQL ‘Negative’ Table Extract	91

Table 4.10 Transpose of the PN Incidence Matrix	91
Table 4.11 MySQL ‘initial_maeking’ Table	92
Table 5.1 Abbreviations and Full Names of Places and Transitions included in the PN in Figure 5.3	102
Table 5.2 Probabilities for the PN developed for the Recycling IT Asset Process ...	109
Table 5.3 Activity Times for the PN developed for the Recycling IT Asset Process PN.....	110
Table 5.4 Interval Times for PN developed for the Recycling IT Asset Process	110
Table 5.5 Average Completion Time for Each Path of the Recycling IT Asset Process.....	116
Table 5.6 Average Time for each Activity Timed Transition of the Recycling IT Asset Process PN.....	117
Table 5.7 Average Time for each the Interval Timed Transition of the Recycling IT Asset Process PN	117
Table 5.8 Number of Visits to Places of the Recycling IT asset Process PN	118
Table 6.1 Notation and Description of Activity Diagram Object Nodes.....	123
Table 6.2 Notation and Description of Activity Diagram Actions (Nodes)	123
Table 6.3 Notation and Description of Activity Diagram Structured Activity Elements (Nodes)	124
Table 6.4 Notation and Description of Activity Diagram Notes (Nodes)	124
Table 6.5 Notation and Description of Activity Diagram Edges.....	125
Table 6.6 MySQL ‘node_xmi’ Table for XMI in Figure 6.3.....	129
Table 6.7 MySQL ‘node_xmi’ Table for the XMI in Figure 6.4.....	131
Table 6.8 MySQL ‘node_xmi’ Table for XMI in Figure 6.7.....	133
Table 6.9 Relationships between the AD and PN Notation and Symbols	143
Table 7.1 MySQL ‘union_node’ Table Extract	154
Table 7.2 MySQL ‘initial_node_table’	154
Table 7.3 MySQL ‘final_node_table’	155
Table 7.4 MySQL ‘final_table’ Extract	155
Table 7.5 MySQL ‘Transpose_of_the_PN_Incidence Matrix’ Extract.....	156
Table 7.6 MySQL ‘initial_marking’ Extract	157

Table 7.7 MySQL ‘union_node’ Table Extract	162
Table 7.8 MySQL ‘initial_node_table’	163
Table 7.9 MySQL ‘final_node_table’	164
Table 7.10 MySQL ‘exception_handler’ Table	164
Table 7.11 MySQL ‘datastore_table_a_b’ Table.....	164
Table 7.12 MySQL ‘central_buffer_table-a_b’ Table	164
Table 7.13 MySQL ‘in_outputValue_final’ Table	165
Table 7.14 MySQL ‘expansionNode_final’ Extract Table.....	165
Table 7.15 MySQL ‘final_table’ Extract	165
Table 7.16 MySQL ‘Transpose_of_the_PN_Incidence Matrix’ Extract.....	166
Table 7.17 MySQL ‘initial_marking’ Extract	167
Table K.1 MySQL ‘intial_marking’ Table for the Production System	264
Table K.2 MySQL ‘intial_marking’ Table for the Online Shopping Process	267

1 Introduction

1.1 Introduction to Reliability Modelling

The reliability of a system at time t , denoted by $R(t)$, is the probability that the system can perform a required function in time interval $[0, t]$ without a system failure occurring. In the context of process modelling, reliability, denoted by $R(A)$, is defined as the probability that the activities operate on users demand, following a discrete-time model (Cardoso, 2002). In this context, the reliability of an activity is given as the ratio of successful executions over scheduled executions. Reliability modelling, i.e. the prediction of the reliability of a component/activity or system/process prior to its implementation (Richardeau & Pham, 2013), has become one of the most important design considerations in industry. Reliability modelling should be applied at the earliest stages of the design effort in order to be effective given that failures can be avoided and mitigations put in place before they create greater financial and logistical problems later in the lifecycle.

The methods available to perform a reliability assessment of a system can be divided into two main categories, *analytical* and *simulation* methods. The *analytical* reliability modelling methods include various approaches from which an analyst should select the most suitable method for their given problem. The methods for failure analysis consist of *combinatorial models*, including Fault Trees (FTs), Cause-Consequence Diagrams (CCDs), Reliability Block Diagrams (RBDs), *state-space models*, including the subcategory of Markov approaches or alternative approaches such as the encoding of the state-space model in a Petri Net (PN) (Zille et al., 2010) and *hierarchical models* generated by the composition of combinatorial and state-space models, which are able to simplify the model and ease further analysis (Lanus et al., 2003). The *simulation* reliability modelling methods, such as the Monte Carlo method, simulate a modelled system via computer algorithms that rely on repeated random sampling to obtain numerical results.

1.1.1 Analytical Reliability Modelling Methods

In this section, analytical modelling methods are examined introducing the basics of combinatorial, state-space and hierarchical models.

1.1.1.1 Fault Tree

Fault Trees (FTs), a concept first introduced by H.A. Watson in the early 1960s, are the most widely used tool in assessing system reliability (Chew, 2010). These models, commonly used in the aeronautical and automotive industries, show a clear representation of the logic of a given system failure mode via the interconnections between the components failures. The construction of FTs typically follows a top-down approach, meaning that the model begins with the identification of the system failure mode, which becomes the *top event* of the tree, to be further analysed to component failures, represented by *basic events*. *Intermediate events* are also used to show events between the system failure and component failures. Gates link events logically. Therefore, Boolean logic gates such as AND, OR, PAND (priority AND) and NOT are used in the tree to decompose the top failure event into the component events that cause it. Multiple FTs should be created if a system has more than one failure mode.

Fault Tree models support both quantitative and qualitative evaluations. FT qualitative assessment can be characterised by the logical expression of the top event in terms of the basic events, using Boolean algebra rules to derive minimal cut sets. FT quantitative analysis is performed to evaluate the performance of a system, i.e. to determine the probability of occurrence of the top event of the FT in terms of the probabilities of the basic events. The system evaluation can be conducted using Boolean algebra techniques and probability rules, or computer software is available to perform the calculations.

The assumption of the modelling is that all the basic events are independent therefore, it is unable to capture the dynamic behaviour of real world applications including modelling of: sequence-dependent events, spare/repair/redundant components and priority of failure events. The Dynamic Fault Tree (DFT), introduced by Gulati (1996), is an enhancement of traditional FT including additional dynamic gates in order to enable the modelling of complex systems with dynamic characteristics. However, this formalism cannot cover all industrial cases, since it includes only a few

specific kinds of dependencies (Bouissou, 2006), and hence cannot provide a generic applicability.

1.1.1.2 Cause-Consequence Diagram

Cause-Consequence Diagrams (CCDs) combine two conventional reliability methods, the FTA (Fault Tree Analysis) method and the Event Tree Analysis (ETA) method. This method allows the modelling of sequential and dependent systems. The qualitative analysis of the CCD model is based on a list that includes the causes for each outcome condition (Vyzaite et al., 2005). The lists of component conditions can then be quantified using the probabilities of each event. Boolean algebra techniques and probability rules are used to perform the calculations. Despite the method addressing the dependency issue, generally, software packages for the analysis of CCDs cannot handle these dynamic characteristics, including loops and repair actions, and dependencies between systems components. Therefore, this method has very limited application to current systems in industry since it cannot provide advanced modelling capabilities.

1.1.1.3 Reliability Block Diagram

Reliability Block Diagrams (RBDs) are directed graphs enabling analysts to represent how components of a system are connected in a logical way. Contrary to FTA, RBDs are success-oriented illustrating how the component reliability contributes to the system success (Andrews & Moss, 2002). They are composed of a start and an end node on the left and on the right side, respectively. Between these models are all the remaining system components denoted by blocks. The blocks are connected in series, parallel or k-out-of-n depending on the logic of the system. Both a qualitative (yielding combinations of working components that contribute to the system working) and quantitative (yielding numerical performance measures) analysis can be carried out. RBDs do have some limitations rendering it not applicable to some cases, especially when the system includes complex structures, such as dependencies, standby redundancy or load sharing, which cannot be represented in a clear way.

The Dynamic Reliability Block Diagram (DRBD) was introduced by Distefano and Xing in 2006 as an extension of the traditional RBD, enhancing the model capabilities considering dependencies and system dynamics. Although the DRBD formalism

enhances the ability of the static RBD, it is rarely used in industry due to its high modelling complexity.

1.1.1.4 Markov Method

The assumption of statistical independent components in complex systems, made in combinatorial modelling methods, may ease the reliability analysis but it can be an incorrect assumption leading to an underestimation of system unavailability. Therefore, the dependency between components is a major factor in modern systems. Failure dependencies have been studied from the late 1980's mostly on behalf of the nuclear power industry (Fleming & Kalinowski, 1983; Mosleh et al., 1988). During that period, mathematical modelling techniques such as Markov methods have been developed that enable time dependency to be considered facilitating the reliability analysis of complex systems.

Markov methods including Discrete Markov chains and Continuous Markov processes, are used to model stochastic processes. These methods can describe a system in time and space using probability laws. Discrete systems move from one state to another at set points in time, whereas continuous systems move from one state to another at any point in time. Discrete systems have a set of non-overlapping exhaustive states identified, where the systems must be in one of those at any given time, Continuous system states can degrade continuously between working and failed. These methods are applicable in modelling complex systems with dynamic properties such as dependent events (e.g. common-causes, standby redundant events and secondary failures) and repairs given the repairs rates are known (Villemeur, 1992). The basic assumption of Markov approaches is that the system behaviour in each state is memoryless/random. A lack of memory is defined by two characteristics: the future state is only dependent on the immediately preceding state and not on the full history; and the system should be stationary (Andrews & Moss, 2002). In Markov processes each component of the system is represented by a state and all the states are connected together by transitions. State transition diagrams are used to represent the structure of the system.

Although Markov methods are capable of handling systems with dependencies, they suffer the state-space explosion as the number of components increases. Hence, large systems are difficult to be modelled using Markov methods, as the final diagram can

be very large, difficult to build due to its complexity and computationally costly. However, many papers (for example Gulati & Dugan, 1997; Andrews & Ridley, 2001) have mentioned the application of Markov methods in combination with traditional approaches, such as Fault Trees. Small sections of the FT are analysed using Markov methods and the results from these Markov model sections can be fed into the combinatorial models (FTs) for a quantitative evaluation. However, the combination of these two methods cannot provide a generic applicability, since it presupposes that the dependencies can be isolated in a small enough section of a FT for a Markov model to be applied.

1.1.1.5 Petri Net

According to the Markov methods review, large complex systems are difficult to be modelled using Markov approaches, hence, alternative approaches have been developed such as the encoding of the state-space model in a Petri Net (Zille et al., 2010), that can handle all the aforementioned limitations.

The Petri Net (PN) model, introduced in the thesis of C.A. Petri in 1962, is a bipartite directed graph, including two types of nodes, places and transitions. The nodes are connected together with directed arcs.

In the area of applied mathematics, Petri Nets are of special interest since once the bipartite graph (PN structure) and the marking (PN behaviour) are defined the user can analyse, simulate and model numerically and graphically the PN, obtaining information about the behaviour of the system. This model has been applied to complex systems, workflows, networks and other cases handling efficiently complex structures such as loops, dynamic characteristics, dependent failures (Volovoi, 2004), phased missions (Mura & Bondavalli, 2001; Chew et al., 2008). Over the years, the standard PN has been extended and hence, several models with advanced capabilities such as Coloured Petri Nets (CPNs) (Jensen, 1990) have been developed enabling the modelling of real systems used in industry.

1.1.2 Simulation Modelling Methods

Complex industrial systems often introduce many dependencies or/and inconstant failure and repair rates, and hence, the application of analytical methods with the view to obtain performance measures such as system reliability, availability and

maintainability, becomes inaccurate. In such cases, the Monte Carlo method, a ubiquitous and flexible modelling method widely used for the behavioural analysis of industrial systems, is commonly used to computationally simulate complex models, using repeated random sampling and statistical analysis to obtain the required performance (Raychaudhuri, 2008). The construction of a computer model for performing a Monte Carlo simulation is based on the logic representation of the system's operation, with the view to identify all events that may occur, all activities that can be conducted and the correspondence relationship between these events and activities. If a graphical model, which can be represented with the help of analytical methods such as PNs, is available, it can provide useful information for the system operational flow that can then be used to develop the corresponding computer model, using computer languages such as MATLAB, Java, C++ and C#.

Monte Carlo simulation is carried out to conduct system performance analysis, by simulating the occurring system events and the performing system activities for a predefined period of time with the help of the computer model. The times related to model's events and activities can be generated by randomly sampling the corresponding probability distribution, such as exponential, Weibull or normal distributions, for the real system. Computer models are repeatedly simulated for a number of replications so system's performance can be obtained.

1.1.3 Reliability Modelling, Implementation and Deficiencies

The analysis of reliability models once constructed has been the main focus of analysts over the years and this can now be conducted systematically, using bespoke computer software, providing advanced qualitative and quantitative analysis results for a given system (Dugan et al., 2000). However, the model generation is still a manual process and requires considerable time and effort with the user needing to have experience and understanding of the method. The requirement to detect the most critical components and design errors at an early stage of design becomes more challenging due to the increase in complexity of today's systems. Hence, the need for a full automated reliability analysis including model construction, with the aim to save time, money and effort increases. Hence, to overcome the limitations of existing reliability model analysis requiring human-aided model construction, the automated generation of reliability modelling methods for complex systems is investigated.

1.2 Introduction to Automated Reliability Model Generation

Automating the generation of reliability models reduces the model construction time and cost, and minimises human error; therefore, over the last 40 years there have been several attempts to automate the construction of reliability models. In the case of Fault Trees two main techniques have been introduced in the 1970's that were considered as pioneering concepts, namely, the decision table method (Salem et al., 1977) and the digraph method (Lapp & Powers, 1977). Some of these techniques, such as the modified decision tables proposed by Henry and Andrews (1997) and the component-model based methods, are alternatives of the decision table and digraph methods.

As systems complexity increased due to complex structures such as loops and electric circuits, higher level automated methods such as Expert system methods (Xie et al., 1993) and others such as HiP-HOPS (Papadopoulos et al., 2001) and AltaRica (Point & Rauzy, 1999) were developed. More recent approaches have been proposed for the automated construction of other reliability models such as FMEA (Papadopoulos & Grante, 2005), Hazard and Operability (Zhao et al., 2005) and Petri Nets. In the case of Petri Nets, several semi-automated methods were introduced (Alhroob et al., 2010; Stockwell & Dunnett, 2013; Taibi et al., 2013).

Some of the methods reviewed for the automated reliability model generation present difficulties in handling complex systems such as the decision table methods that do not provide any facilities for the detection and classification of control loops and circuits. In many cases such as in the decision table methods, digraph methods and the modified decision table method, there is not a software package available but an algorithm which is applied manually. Additionally, some of the reviewed approaches enable the automated generation of reliability models for systems with certain types of characteristics, such as the modified decision table methods that focus on circuit systems, without providing a generic applicability. The most frequent shortcoming, found in the literature of the current methods for automated reliability modelling, is that the system representation used for input into the automation process is conducted manually by the user.

Therefore, the main deficiencies for the automated generation of reliability models identified in the literature are as follows:

- *The range and domain that the approach targets*: There are approaches that focus on specific domains such as mechatronics (Mhenni et al., 2014), without providing a general methodology applicable to any system.
- *The degree of automation*: Some efforts result in semi-automated reliability model generation, since the algorithm execution needs the intervention of analysts, carrying out one or more steps manually or the data is not derived automatically from the description diagram, but the user imports it manually.
- *The level of the system's complexity*: Although most approaches argue that they are applicable in complex cases, only a limited number of methods prove their semi-automated model generation of complex systems, including dynamic characteristics.

According to the deficiencies discussed in this section for the reviewed methods for automated reliability modelling, this research study can contribute to the literature, targeting the enhancement of the generation of reliability models from semi-automated to fully automated methods, using directly an industrial system representation.

Additionally, according to the review conducted for reliability models, Fault Trees and Petri Nets are singled out due to the common applicability of the former in industry for reliability modelling, and the dynamic capabilities of the latter for future implementation. Therefore, although FT is a widely applied reliability method in the industrial sector, Petri Net has been selected to be automatically generated due to the flexibility of this model to handle complex systems with loops and dynamic characteristics such as dependent and repair components. In addition to the automated generation of a PN, the model can also be simulated, enabling the detection of the most critical components and design errors existing in a system at an early design stage, contributing to the literature by improving decision-making and enabling different designs to be investigated in a short-time.

Therefore, the purpose of this research is the development of an algorithm that can accept as an input an industrial description diagram of a given system and automatically generate the corresponding Petri Net model.

1.3 Industrial Representation of Systems

Main key challenge in the automated construction of reliability models is the starting point of the automation process, which is the system representation as used in industry. This step corresponds to how systems should be modelled in order to develop comprehensive models that include high level of detail. Additional challenge in the automation process is that the selected modelling technique should provide a wide applicability covering various domains. Hence, in this section, various system modelling tools, used to represent systems, are discussed in order to identify the strengths and weakness of each method.

1.3.1 Introduction to System Modelling Tools

System modelling is the use of tools and techniques to conceptualise and construct systems in business and IT development. The modelling of a given system refers to the analytical description of its visual representation that depicts the flow of information/data from one component to another. Hence, generally, the system models from a reliability perspective can include the following:

- System topology, including the component interconnectivity.
- Multiplicity and failure modes of each component.

The first requirement in the list is extracted from the description diagram of the system given by industry, whereas the multiplicity and failure modes of the components are provided either by the industry that is responsible for the system design or the component manufacturers.

The modelling tool employed to generate a system description (topology), used as input into the automation process, should ideally satisfy some criteria which have been identified during the literature review of this research study and selected with the view to enable the modelling of complex systems. These criteria are as follows:

1. Enable the accurate and realistic modelling of a given system, providing a high level of expressiveness of the model towards timing concepts for reliability.
2. Provide a language with a rich variety of notations so as to enable modelling of various industrial systems and processes and, hence, offer a broad-spectrum applicability.

3. Store model information in a markup-language-based format, i.e. a computer language that uses tags to define elements within a document, such as XML. This markup-language-based format is required to define a set of rules for encoding topology information into a format for data interchange between modelling tools. This format should be simple, generic and usable to be easily manipulated for further analysis.
4. Be commonly applicable in industry.

There are several graphical tools used for system modelling to capture in a realistic way the representation of any scenario. They provide simplicity, usefulness and flexibility to the user, aiming to increase the modelling accuracy and contribute in decision-making. The system modelling tools can be categorised in terms of their current domains of use, as follows:

- **Software design modelling tools**, mostly applicable in embedded and real-time software systems to design, implement and test software components such as network and internet services, graphics engine, computer hardware, processors, buses, memory, user interface, system utilities, services and others.
- **Business Process Modelling (BPM) tools**, mostly applicable in enterprise and business processes to design, implement and test business aspect of processes and activities.
- **Mechanical modelling tools**, mostly applicable to design, implementation and testing of the engineering aspects of systems and components.

The **software modelling tools**, used to design, implement, and test software components, enabling the software representation and analysis of components of real-time embedded systems, are the following: *Architecture Analysis and Design Language (AADL)*; *Unified Modelling Language (UML)*; and *System Modelling Language (SysML)*.

Architecture Analysis and Design Language (AADL) was released in 2004 by the Society of Automotive Engineers (SAE) as an aerospace standard AS5506 and finds common applicability in industry in a wide spectrum of disciplines such as in avionics, automotive, aerospace and autonomous systems (Feiler & Lewis, 2004). AADL, a textual and graphical modelling language, employs formal and high-level

concepts enabling precise description and architectural analysis of the structural and behavioural aspects of highly complex systems, conducting performance, schedulability and reliability (Feiler et al., 2006). This language provides a well-defined syntax, and hence the user can describe complex scenarios including data inputs and outputs, interactions between the systems components and timing requirement properties of components, if available (Feiler et al., 2004).

AADL consists of software, hardware and system component abstractions enabling the specification and performance analysis of complex real-time embedded systems, complex systems of systems and specialised performance capability systems (SAE ASD AS-2C Subcommittee, 2004). The structure of systems is described as an assembly of software elements, such as data, threads, processes, etc., that are mapped onto computational hardware components, such as processors, memory, bus and devices (Feiler et al., 2006). Additionally, the AADL core has been enhanced with the AADL Error Model Annex, a textual model which was standardised in 2006 and is defined manually by the user complementing the description capabilities of AADL. The AADL Error Model Annex provides features able to describe dependability-related characteristics of AADL models such as faults, failure modes, repair policies and error propagations (Wang, 2017).

Unified Modelling Language (UML) language was developed by Brooch, Jacobson and Rumbaugh at Rational Software in 1994-1996, but since 1997 it has been adopted as a standard by the Object Management Group (OMG), an international, open membership, not-for-profit technology standards consortium.

UML is a general-purpose modelling language that provides a standard way to represent the design of a given system. In 2005, UML version 2.0 advances the successful UML version 1.5 by adding more precise definitions of its abstract syntax rules and semantics, a more modular structure of the language and an enhanced capability for modelling large-scale systems (OMG UML 2: Infrastructure, 2005). Following this update, *System Modelling Language (SysML)* was introduced and defined as a subnet of UML 2 with some dialectical extensions. SysML, developed by the Object Management Group (OMG), International Council on Systems Engineering (INCOSE), a systems engineering professional society, and Application Protocol 233 (AP233 consortium), is a STEP-based data exchange standard that supports the needs

of the systems engineering community. This language is used for the specification, analysis, design, verification and validation of systems and systems of systems.

These two industry standard modelling languages, UML and SysML, are characterised as critical enablers for Model Driven Systems Engineering that allow the user to model the structure, behaviour and architecture of complex systems and business processes, providing a wide variety of notations (representation of meaning) enhancing the effective expressiveness of systems (Hause, 2006; Kapos et al., 2014). UML and SysML also support model and data interchange via XML Metadata Interchange (XMI) and the evolving AP233. However, the wide variety of the diagrams and profiles application can cause a large learning curve for the users.

The main **Business Process Modelling (BPM) tools**, identified for defining and outlining business processes, information flows, data stores and systems (SPARX Systems, 2018), are the following: *flowcharts*; *Event-driven Process Chain (EPC) flowchart*, *Business Process Model and Notation (BPMN)*, *Unified Modelling Language (UML) Activity Diagram (AD)* and *Integrated DEfinition for Function modelling (IDEF)*.

According to a review conducted for these tools, the *flowchart* provides limited modelling capabilities (Heller, 1997) as well as informal and ambiguous syntax; the *EPC flowchart* lacks a formal syntax, restricting its applicability spectrum (Aalst, 1999), and the graphical representation of the *IDEF* tool is not as user-friendly as the representation of other BPM tools, such as BPMN and UML 2 AD (Tangkawarow & Waworuntu, 2016). Therefore, the three first BPM tools introduced above have initially been rejected, since they do not meet the criteria, identified in section 1.3.1 for the representation of a system topology.

The *BPMN diagram* and the *UML AD* are the most commonly applied BPM techniques, sharing many characteristics and elements that have the same semantic meaning. Johansson et al. (2007) performed a literature review regarding BPM tools used in industry mentioning that these two tools concentrate the most benefits such as high expressiveness, simplicity and directness. However, the BPMN application entails the danger of missing information regarding the flow of physical objects, processes and data since it lacks the accurate modelling of business processes (Bao,

2010; Khabbazi et al., 2013) compared to the UML AD notation that provides a wide variety of elements such as receiving and sending signal, central buffer and data store elements that enable the modelling of any business process. Additionally, the BPMN is mainly used for the modelling of business processes, whereas the AD brings the benefits of UML enabling the modelling of both business processes and engineering systems, which cover the focus of this research study. Hence, it was concluded that the UML AD is more suitable than the BPMN for this research study.

The main **mechanical modelling tools** identified for the development of a system topology, are the following: Computer Aided Design (CAD), Piping and Instrumentation Diagrams (P&IDs), used to provide a detailed graphic description regarding the industrial process equipment interconnections of pipeline systems (Walker, 2009), and Simulink (MATLAB). However, the primary purpose of CAD and P&IDs is simulation and analysis of mechanical aspects of a design, using physical quantities i.e. mass, stress, strain, etc., dynamic characteristics and others (Friedenthal et al., 2011). Similarly, the primary purpose of Simulink is simulation, automated code generation and continuous test and verification (MathWorks, 2017). Hence, for these reasons the mechanical modelling tools were found unsuitable and have not been taken forward as a starting representation for further automated reliability modelling.

1.3.2 Summary of System Modelling Tools

From the review conducted for the software design modelling tools, it is concluded that AADL, UML and SysML meet the criteria defined in section 1.3.1 for the industrial representation of system topology, which would be used as a starting point in the methodology for the automated generation of PN models. Following the review for these powerful software design-modelling languages, it is identified that they provide two different views of the same system. AADL is usually used for the design and analysis of embedded systems, allowing the development of the low-level view for the software engineer (De Saqui-Sannes & Hugues, 2012), where in contrast, UML and SysML are mostly used at the early stages of system engineering, allowing the high level view for the system engineers. Regarding the fields of their applications, AADL is mostly used in computer science, whereas UML and SysML in business processes and engineering systems (de Niz, 2007; Evensen & Weiss, 2010).

Therefore, referring to the software design modelling tools, UML and SysML were concluded to be more suitable for this study, since AADL restricts the spectrum of applications focusing mainly on computer systems modelling. On the contrary, the UML and SysML diagrams, commonly used in industry at the early modelling stages, support modelling of general-purpose systems engineering applications, including systems and processes, which is also the focus of this research study. Thus, if the Model-based Systems Engineering (MBSE) approach, which allows for the reuse of system specifications, enhances quality of specification and design and improves communications between members of the development team, is used in system design, then it can be expected that behavioural models (in the form of UML/SysML diagrams) are a priori available. So, an automation to PN model could then save time if PNs are the basis of reliability analysis for the system.

To conclude, UML and SysML, which are attracting growing interest as system level visual languages, are increasingly applied for modelling and analysis of complex systems from a reliability and safety perspective. The development of a UML/SysML diagram from which the PN can be automatically constructed is a more efficient approach than constructing the PN directly, as these languages increase model availability, facilitating the modelling of systems across various domains and developing comprehensive models while maintaining sufficient level of detail. Therefore, the UML (including UML 2 AD) and SysML diagrams, which satisfy all the criteria defined in section 1.1.3, are the most suitable modelling tools to represent efficiently the topology of industrial systems, which can be used as a starting point for the automated generation of PN models and, hence, they are taken forward.

1.4 Research Scope and Delimitations

The scope of this research is the development of a methodology, implemented in computer software, which takes as an input the descriptions of real-life industrial scenarios and automatically generates the corresponding PN models. The input information is to be provided by a modelling tool, commonly applied in industry, which can be used to extract topological information from the real-life industrial scenarios and source it to the algorithm.

The delimitations of this research are identified as follows:

1. The PN model, generated following the automation procedure, will enable via the simulation the prediction (estimation) of the: (i) average time each path of which the model consists requires to be completed; (ii) average time for each transition; (iii) most common visited places in each path; and (iv) the paths resulted most in failure and the nodes most involved with route to failure.
2. The proposed methodology is predominantly aimed to be applied to industrial business processes rather than engineering systems. However, a limited number of systems is to be considered.
3. The systems and processes considered in this research provide the topology information, by describing how the components and activities of which they consist connect together. The examined systems may include a large number of components/activities, dependent events and loops.
4. The components/activities, of which the examined systems/processes consist, are described in terms of timed and probabilistic data. The timed data represents the time a component/activity requires to either complete an action or move from one component/activity to the following. The probabilistic data shows the pass and fail probabilities of components/activities through the various flow paths included in a system/process.
5. The PN model automatically generated, by applying the proposed methodology, can be: (i) verified, showing that the initial UML/SysML diagram, which depicts the system/process representation and lacks formal syntax and semantics, is correct; and (ii) simulated. According to the results obtained from the PN simulation, predictions can be made about the PN model behaviour and, by extension, of the initial system or process. These predictions may consider the performance assessment and the detection of any limiting factors/deficiencies of the examined system/process. Then, these predictions can be used to suggest modifications to be made in the developed PN, supporting the quality of the decision-making process and, hence, enhancing the system's/process' performance. Thus, modellers can help decision-makers to make more credible decisions according to company's requirements, maximise the profit, reputation and lifetime of businesses, by considering minimum execution time and cost of systems/processes, efficient usage of staff resources and equipment, and others.

1.5 Aim and Objectives

The aim of this research study is to automatically generate Petri Net models by applying an algorithm that accepts as an input the industrial description diagram of a system/process and generates the corresponding Petri Net model. The developed algorithm, will be able to handle complex systems/processes including a large number of components/activities and characteristics such as dependencies and control loops. The potential to apply the proposed methodology more broadly to processes will be explored. To achieve this aim, the following objectives will be accomplished:

1. *Identify the most suitable UML or SysML diagram*: to be used as a starting point for the automated PN model generation. The selected system/process description diagram, used to extract from it the behavioural aspects of system components/process activities and import this information into an algorithm, needs to be applicable in complex systems/processes.
2. *Perform a detailed literature review of Petri Net model*: identifying model's characteristics, strengths and weaknesses, other PN formalisms and types of model analysis.
3. *Review the automated model generation methods*: that exist in literature, highlighting their benefits and limitations so that the most feasible method for the automated Petri Net model generation can be developed.
4. *Develop a methodology for the automated Petri Net model generation*: using as a starting point the most suitable UML or SysML diagram according to the findings identified in objective 1, and dealing with the limitations identified in the current automated reliability model generation methods according to the findings identified in objective 3.
5. *Validate and verify the Petri Net model*: developed in objective 4, to prove the correctness, completeness and consistency of the PN model and, by extension, of the automation procedure.
6. *Extend the proposed methodology providing advanced applicability and scalability*: developing an algorithm that enables the automated generation of PNs for systems/processes represented by the chosen UML or SysML diagram. The proposed methodology should be applied both to industrial complex systems and processes including loops and dependencies, in order to prove its advanced applicability and scalability.

1.6 Thesis Layout

The thesis is structured as follows:

Chapter 2 gives a detailed review of UML and SysML software design modelling tools, since one or more diagrams from these tools will be used as a starting point for industrial system representation; an evaluation of the diagrams is carried out and the most suitable to model complex industrial systems/processes is selected. The Petri Net model is also reviewed, identifying its strengths and capabilities. Additionally, in this chapter, a review of past work focused on the automated generation of reliability models using FTs, CCDs, RBDs and Markov chains, is conducted. A recent literature review for the various approaches developed for the automated Petri Net generation is also performed and a summary of the methods is conducted, where research motivations of this research are identified.

Chapter 3 gives a detailed overview of the methodology proposed for the automated generation of PN models. The database concept is introduced and a review of the database tools is undertaken to identify the suitable tool for this research study. The steps followed for the methodology development, which uses as a starting point an industrial system representation and generates the mathematical and graphical representations of PN models, are thoroughly discussed.

Chapter 4 presents a case study in which the proposed methodology from Chapter 3 is demonstrated by its application to an end of life manufacturing process, including a discussion about the resulting model and its limitations.

Chapter 5 provides an overview of the methods used for the verification and validation of Petri Net models. The PN automation procedure is verified and validated via evaluation of the PN model obtained from the case study in Chapter 4. This is achieved, by checking: (i) the model's structural and behavioural properties; (ii) the system's behaviour playing the token game; (iii) the PN model's quality performing numerical simulation; and (iv) if there exist any limitations or incorrect/omitted logic by conducting performance analysis.

Chapter 6 extends the automated PN model generation discussed in Chapter 3 by considering the transformation of any element included in the UML/SysML diagram, selected in Chapter 2, to the corresponding Petri Net element. Hence, in this chapter,

new transformation rules are introduced. The structure of XMI files obtained from the selected UML/SysML diagram is also examined focusing on the structure of nested elements included in these documents and how they are loaded into the database. The need to transform the structure of XMI documents arises so that the information included in them can be properly loaded into the database to be further analysed. XML-related terms and definitions are also introduced. The steps followed for this advanced generic methodology development as well as the newly introduced transformations rules are discussed in detail.

Chapter 7 applies the proposed methodology outlined in Chapter 6 to two complex real-life scenarios. The complexity includes: (i) using all the elements available in the selected modelling diagram; (ii) a large number of components/activities; (iii) dependent events; and (iv) control loops to demonstrate the capability of this advanced methodology for the automated mathematical and graphical representation of PNs. Thus, in this chapter, the methodology's efficiency is enhanced by proving its applicability in both complex systems and processes.

Chapter 8 drawn some conclusions from the methodology proposed in the thesis; presents the contributions to knowledge; and provides recommendations for future work.

2 Modelling Tools and Methods – Automated Reliability Modelling

2.1 Introduction

In this chapter, a review of UML and SysML modelling languages is conducted and the most suitable diagram(s) to be used as a starting point for automated PN model generation is/are selected. The Petri Net model is also reviewed; identifying its characteristics, strengths, applicability as well as the model's extended formalisms. Additionally, literature reviews are conducted on automated reliability model generation methods. More explicitly, a review of the attempts at automating the process initially focusing on the automation of reliability models such as Fault Trees, Cause-Consequence Diagrams, Reliability Block Diagrams and Markov Chains; and additionally investigating attempts that target the automated generation of PN models. The current PN model deficiencies in automation are also identified and the research motivations of this thesis are highlighted. The layout of this chapter is illustrated in Figure 2.1.

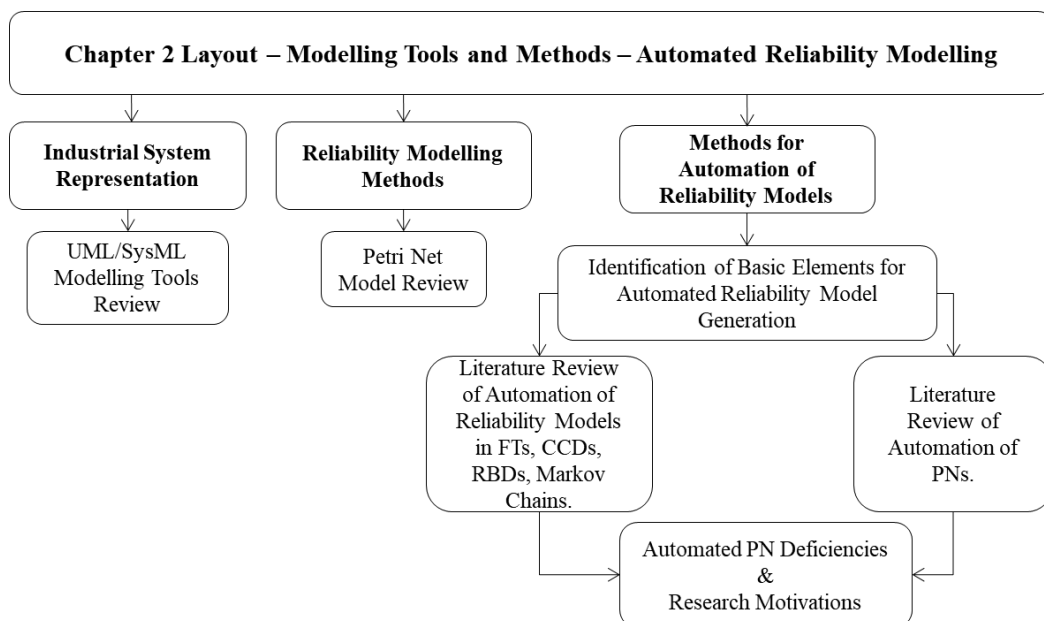


Figure 2.1 Illustration of the Structure of Chapter 2

2.2 Industrial System Representation

The system topology, derived from the initial representation of a system, constitutes the basic input for the automated generation of reliability models. Hence, the detailed topological representation of an industrial system is considered as a vital tool for the automated generation of a reusable and versatile reliability model. The system topology focuses on the structural and behavioural description of the system being modelled. The structural description includes information about the inputs and outputs to and from each component and the way in which the components are connected with each other, whereas the behavioural information corresponds to the role and behaviour that each component plays in the entire system.

According to the findings in Chapter 1, section 1.3, regarding the industrial representation of systems, UML and SysML diagrams have been concluded to be the most suitable to model an industrial system. These languages have been selected due to: (i) their modelling capabilities that allow the accurate representation of a given system/process; (ii) the wide variety of notations they provide to describe a broad-spectrum of industrial systems and processes; (iii) the data interchange via XMI format; and (iv) their wide industrial applicability to model systems and processes. In this section, the UML 2 and SysML are reviewed in order to identify the most suitable diagram(s) to be used as a starting point for the automated construction of a Petri Net model.

It is stated in this section that once we talk about systems it covers processes as well and the systems consists of components, whereas the processes of activities.

2.2.1 Systems Modelling Languages Review: UML and SysML

UML version 2 (OMG UML 2: Infrastructure, 2005) is the latest version of UML and it consists of 14 diagrams. Figure 2.2 shows the taxonomy of these 14 diagrams. According to this taxonomy, two diagram groups can be identified: the structural consisting of seven sub-diagrams and the behavioural also consisting of seven sub-diagrams. The terms “*structural*” and “*behavioural*” refer to the static and dynamic aspects of systems. The UML 2 behavioural diagrams describe how a system works, showing the dynamic behaviour of components included in a system. The dynamic behaviour is described as a series of changes (operations) of the system over time,

tracking how this system will act in a real-world environment and observing the effects of an operation/event, including its results.

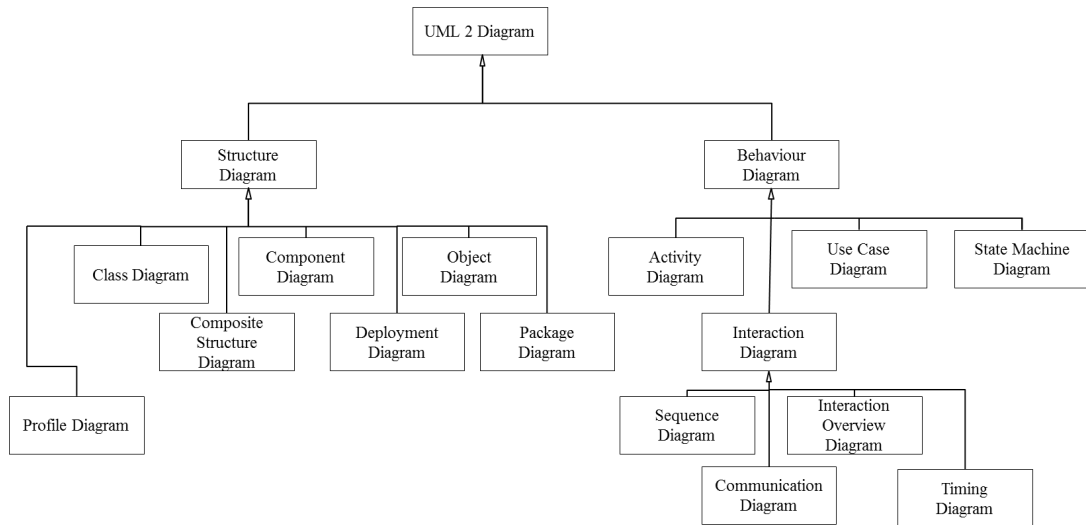


Figure 2.2 UML 2 Diagrams Taxonomy (OMG Unified Modelling Language (UML), Version 2.5, 2015)

A brief description of the diagrams is as follows:

- **Class Diagram:** It describes the structure in a system, showing the system's classes, their attributes, operations and the relationships between classes. A class, the building block of this diagram, is represented as a rectangle divided into three parts: the *name* of the class is placed in the first part; the *attributes* of the class (values attached to instances of the class) in the second; and the *operations* of the class (method/function that can be executed by an instance of a class) are listed in the third partition.
- **Component Diagram:** It shows how the system is split up into components and describes the dependencies among these components. The term “component” refers to a set of classes that can represent independent systems/subsystems that interfere with the rest of the system.
- **Object Diagram:** It shows a complete/partial view of the model's structure at a specific time.
- **Composite Structure Diagram:** It presents the internal structure of a class.

- **Deployment Diagram:** It models the allocation of nodes that can be either hardware or software and shows how software elements and artifacts, i.e. products of the software development process, such as design models, source files, design documents, etc. are mapped to those nodes.
- **Package Diagram:** It shows the arrangement and organisation of model elements, representing both the structure and dependencies between the packages that made up a model. This diagram is used to make the UML 2 diagrams simpler and easier to understand.
- **Profile Diagram:** It is used to add new building blocks with properties, which can be used to user's specific domain. This diagram provides three types of mechanisms: *stereotypes*, which are used to introduce new building blocks, which are created for a specific domain, *tagged values* used to specify any desired attribute values and *constraints*, which are used to specify conditions that should be satisfied all the time.
- **Activity Diagram:** It graphically shows the flow of activities and actions that show a step within an activity.
- **Use Case Diagram:** It describes the system functionality, representing interactions between the actor and the system. An actor can be either a *human user* of the modelled system such as a customer, supplier, passenger, etc., or other systems/hardware using services of the system such as authority, bank and others.
- **State Machine Diagram:** It is used to describe the state transitions and actions of the system/components.
- **Sequence Diagram:** It graphically represents how and in what order the collaborating parts are interconnected in the system.
- **Interaction Overview Diagram:** It is used in a similar way as the Activity Diagrams showing an overview of the control flow of nodes presented as interaction diagrams. Interaction diagrams can include sequence, communication and timing diagrams.
- **Communication Diagram:** It shows the message flow between objects and the interactions between classes.

- **Timing Diagram:** It shows how the state/value of objects changes throughout a given period of time.

SysML, officially issued by the OMG in 2007, was defined as a subnet of UML 2 with some extensions compared to UML. The SysML diagrams, which can be grouped into three types: diagrams borrowed from UML 2 reused without modification; modified constructs from UML 2; or new modelling constructs defined for SysML, as shown in Figure 2.3. The first group includes four diagrams that focus on the behavioural aspects of the system. The second group includes the requirement diagram in which the system requirements are described based on the needs of the customer. Finally, the third group consists of four diagrams that focus on the structural aspects of the system.

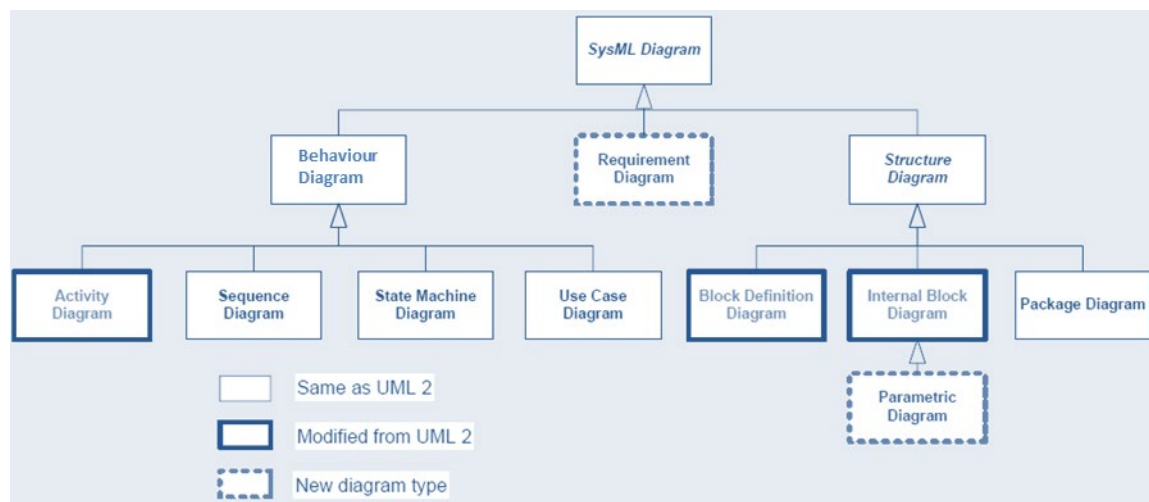


Figure 2.3 SysML Diagrams Taxonomy (Object Management Group, 2006)

A brief description of the diagrams modified or new to SysML is given below:

- **Requirement Diagram:** It captures system requirements and their relationships with other elements.
- **Activity Diagram:** has been slightly modified in SysML by adding some extensions such as definition of the rate and probability of the control/object flows, definition of the flow rate (discrete or continuous) and also control to support disabling of already executed actions.
- **Block Definition Diagram:** It describes the hierarchy and classifications of system and components. In this diagram the classes used in UML 2 Class Diagram are replaced with blocks and additional ports for physical flows are

introduced showing the flow of data or energy that can go through a block (input/output).

- **Internal Block Diagram:** It is used to describe the internal structure of the system. This diagram replaces UML 2 Composite Structure Diagram by replacing classes with blocks and introducing flow ports.
- **Parametric Diagram:** It represents constraints such as performance, reliability and mass properties by using mathematical relations.

Regarding the development of the SysML diagrams, the SysML specification does not define at which point of a system's life cycle they have to be created or the order which these diagrams are constructed. Additionally, the types of diagrams, which are required to be developed for the complete system representation, are not dictated. This implies that methodological decisions need to be taken by the corresponding project team in order to achieve the project's objectives (Delligatti, 2013).

To capture the full integrated system model must address multiple aspects of a system (Friedenthal et al., 2011), and hence for the complete representation of industrial systems, modelled using SysML, both *structural* diagrams such as Block Definition Diagram (BDD) and Internal Block Diagram (IBD) and *behavioural* diagrams such as Activity Diagram (AD) and State Machine (STM) Diagram, need to be developed to statically and dynamically represent a system. The order which these diagrams are constructed is not formally defined.

Regarding the structural diagrams, the BDD and the IBD which provide complementary views of a block, are two of the most commonly used in industry. A BDD that defines a block, i.e. a system component, and its properties and an IBD specifies the internal structure of a block, making clear all the relations between the SysML blocks. However, behavioural diagrams need to be constructed, since a complex system is not only the collection of its components and their structural architecture, but also its behaviour, which is derived from the collaboration of its components (Karban et al., 2011). The behavioural aspect of systems helps the user to understand how systems act in a real-world environment, observing the effects of operations/events, including their results. One of the most widely used in industry behavioural diagrams is the AD, which dynamically shows the actions undertaken by the system as well as system/components data, control flows and physical effects.

This diagram can express control logic of complex systems better than other behavioural diagrams such as Sequence and State Machine diagrams (Delligatti, 2013), providing readability, and hence is usually used to communicate with stakeholders and other team members. ADs can be created at any point in the system life cycle, since they are not tied to any particular stage (Delligatti, 2013).

Ultimately, following this SysML review, the BDD in conjunction with the IBD and the AD are concluded to be the most suitable diagrams to be used as a starting point for the automated PN model generation. These two potential diagrammatical options provide adequate information for the topology, i.e. components and interconnections, of a system that is the requirement input for PN model generation. Comparing these diagrams, the AD has been selected to be the initial point of the automation procedure of PN models because it can model both industrial systems and processes, whereas the BDD and IBD are mostly used for systems representation. Additionally, the applicability of the AD to current industrial system representation is widened since all the definitions of ADs used in UML can also be applied to SysML. Finally, ADs are readily understandable by all business stakeholders, providing an implicit and flexible graphical system representation, whereas the representation of IBDs once used to model complex systems can become complex. Hence, for these reasons the AD has been taken forward.




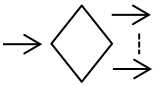
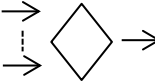
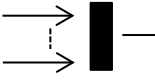
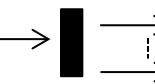
2.2.1.1 UML/SysML Activity Diagram

In this section, basic elements used in UML/SysML ADs are initially reviewed and then a simple AD example is introduced to show the applicability of this diagram. Definitions and terminologies adopted in this work used in UML Specification 2.5 Standard.

An AD is used to describe industrial systems and processes flow, expressing how actions are taken, what they do, when they take place, where they take place and how they affect other actions around them, i.e. their effects. An AD consists of **nodes** such as the initial, activity final, opaque action, decision, etc. and **edges** such as the control flow edges, used to link nodes together. The concept of this diagram is based upon the flow of tokens. These tokens can represent data, information, items, energy, etc. Tables 2.1 and 2.2 review the basic AD node notations, whereas the most commonly used AD edge is described in Table 2.3.

Table 2.1 includes the **control flow nodes** used in Activity Diagrams to coordinate the flows between other nodes. In an AD, the token flow starts using an *activity initial node* and is terminated by an *activity final* or a *flow final node*. The difference between these two terminal nodes is that the former represents the completion of all flows existing in a process, whereas the latter the completion of a flow, destroying all tokens arriving at it without affecting any other flow existing in the activity. Additionally, as seen in Table 2.1, *decision* and *merge* nodes share the same notation: a diamond shape, whereas similarly *fork* and *join* nodes use the same notation: a bar (vertical/horizontal). *Fork* nodes that synchronise outgoing concurrent flows and *join* nodes that synchronise incoming concurrent flows, have been introduced in the AD to enable modelling of parallel activities.

Table 2.1 Notation and Description of Activity Diagram Control Nodes

Nodes		
Notation		Description
Name	Symbol	
Activity Initial Node		Represents the beginning of a process/workflow.
Activity Final Node		Represents the completion of a process/workflow.
Flow Final Node		Represents the completion of one flow exists in a process.
Decision Node		Represents the branching of two or more activity flows.
Merge Node		Represents the merging of two or more activity flows.
Join Node		Represents the combination of two concurrent activities, reintroducing them into the flow as one.
Fork Node		Represents the separation of one activity flow into two concurrent activities.

Apart from the control nodes, additional fundamental AD nodes such as *action* and *activities* are reviewed in Table 2.2. The execution of an *action* node shows the execution of a part of a process in the modelled system, i.e. action nodes are contained

in an AD. There are various kinds of actions in UML2/SysML. Among them, the opaque action node, which is considered as one of the most commonly deployed in ADs, is a type of action that can be used to represent implementation of information or a temporary placeholder before other actions are chosen. The execution of an *activity* node shows the executions of actions included within it, i.e. an activity node contains an AD.

Table 2.2 Notation and Description of Activity Diagram Nodes

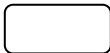
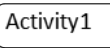
Nodes		
Notation		Description
Name	Symbol	
Action		This node cannot be further decomposed within the activity. An action node that represents a single step within an activity can have input and output control flow edges.
Activity		Represents a behaviour composed of actions.

Table 2.3 shows the most commonly used AD edge, the *control flow* edge, which connects the nodes together and enables the flow of tokens.

Table 2.3 Notation and Description of Activity Diagram Edge

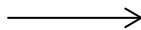
Edges		
Notation		Description
Name	Symbol	
Control Flow		This edge illustrates the control flow within an activity. Within the control flow, an incoming arrow starts a single step of an activity; after the step is completed, the flow continues along the outgoing arrow.

Figure 2.4 illustrates a simple Activity Diagram (OMG Unified Modelling Language (UML), Version 2.5, 2015) including one initial node, two activity final nodes, 5 opaque action nodes, one merge node, one decision node and one fork node. Interconnections between the nodes are represented by the control flow edges (arcs). The diagram in Figure 2.4 shows a process in which a proposal can have two outcomes, either to be published or rejected. It is seen from the AD that once a

proposal is modified ('Modify Proposal') and reviewed ('Review Proposal') then a decision is made, using the decision node. There are three outgoing edges from the decision node dictating the three following paths for the proposal to: (i) be accepted and published ('Publish Proposal'); (ii) be rejected without considering publication ('Notify of Rejection'); and (iii) require further modification and hence a notification for this modification is identified using the 'Notify of Modification' opaque action node and concurrently the proposal returns through the merge node to the first opaque action node ('Modify Proposal') as seen in the AD for modification.

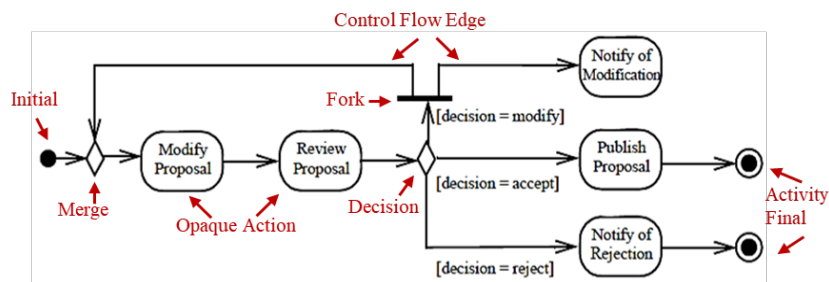


Figure 2.4 UML Activity Diagram Example (OMG Unified Modelling Language (UML), Version 2.5, 2015)

The diagram in Figure 2.4 shows a process in which a proposal can have two outcomes, either to be published or rejected. It is seen from the AD that once a proposal is modified ('Modify Proposal') and reviewed ('Review Proposal') then a decision is made, using the decision node. There are three outgoing edges from the decision node dictating the three following paths for the proposal to: (i) be accepted and published ('Publish Proposal'); (ii) be rejected without considering publication ('Notify of Rejection'); and (iii) require further modification and hence a notification for this modification is identified using the 'Notify of Modification' opaque action node and concurrently the proposal returns through the merge node to the first opaque action node ('Modify Proposal') as seen in the AD for modification. The first two paths can terminate the process illustrated in Figure 2.4. In this diagram a loop is identified, once the proposal token returns to the merge and 'Modify Proposal' nodes after it has passed from the decision and fork nodes, as seen in Figure 2.4.

2.3 Petri Net Modelling Review

According to the findings regarding reliability modelling in Chapter 1, section 1.2, the Petri Net model has been selected to be automatically generated due to: (i) its ability

to efficiently handle complex structures such as loops and dynamic characteristics, existing in complex industrial systems; and (ii) its mathematical and graphical perspectives. An overview of Petri Net model is carried out in this section.

Petri Nets (PNs), which have their origins in the thesis of C.A Petri in 1962 (Petri, 1962) and for which an international standard IEC 62551 has been published (Analysis techniques for dependability – Petri net techniques), are a visual tool that provide a rigorous and precise model analysis (Wang, 2006). PNs have been applied to a wide spectrum of cases in different sectors, such as computer networks (Marsan, 1986), communication systems (Wang, 2006), manufacturing plants (Venkatesh et al. 1994; Zhou & DiCesare, 1989), command and control systems (Andreadakis & Levis, 1988), real-time computing systems (Tsai et al. 1995; Mandrioli et al. 1996), logistic networks (Landeghem & Bobeanu, 2002) and workflows, for reliability, visualisation and verification reasons. This model can capture and describe different component combinations such as connected in series or parallel, repairable systems with warm spares, load sharing, multiphase missions, pooled repair, system on demand and damage tolerance (Volovoi, 2004). PNs are based on strong mathematical foundations and used as a visual communication aid to model the system/process behaviour (Girault & Valk, 2003).

The formal definition of a PN is taken from Schneeweiss (1999) and given in equation 2.1:

$$G_{PN} = (V_p, V_t, E; M(0), D, W) \quad (2.1)$$

Where: G_{PN} is the Petri Net graph; V_p is the set of places; V_t is the set of transitions; E is the set of edges (ordered pairs of nodes), where $E \subseteq (V_p \times V_t) \times (V_t \times V_p)$; $M(0)$ is the initial marking vector of the set V_p of places; D is the vector of switching delays (transition times); W is the vector of weights of edges.

For applications a Petri Net is a bipartite directed graph that includes two types of nodes: places, drawn as circles, and transitions, drawn as bars. There are two types of transitions, the immediate (drawn as solid rectangles), and timed (drawn as hollow rectangles). Directed edges (arcs) connect places to transitions and vice versa. Inhibitor arcs, an element that was added in order to increase the decision power of the Petri Nets, prevents the firing of a transition when the place it comes from is

marked. Inhibitor arcs are denoted as arcs terminated with a hollow circle. In Figure 2.5, t_1 is enabled if p_1 contains a token and will fire immediately removing the token from p_1 and placing a token in p_1' . The transition t_2 is enabled if p_2 contains a token and p_1 has no token.

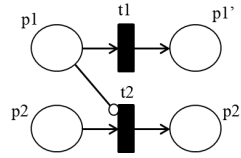


Figure 2.5 Inhibitor Arc

Figure 2.6 presents a timed Petri Net consisting of three transitions (T_1 , T_2 and T_3) and five places (p_1 , p_2 , p_3 , p_4 and p_5). Places p_1 and p_2 are marked, containing one token each. The firing delays are t_1 , t_2 ($t_1 < t_2$) and t_3 . Transitions T_1 and T_2 are initially enabled and after time t_1 one token is moved from p_1 and adds it to p_3 .

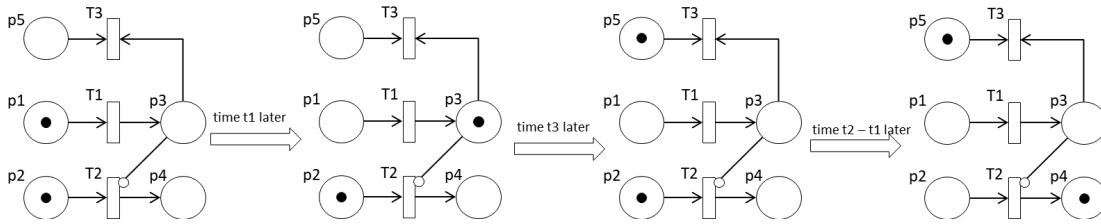


Figure 2.6 PN Firing Process with Inhibitor Arc

After time T_3 , the token from p_3 is moved to p_5 . The firing of transition T_2 is interrupted due to the inhibitor arc from p_3 to T_2 , so the remaining life of the token in p_2 has decreased to $t_2 - t_1$ and will stop dropping. p_3 is unmarked. Once p_3 is unmarked, T_2 is enabled again and the previous firing process is resumed. Hence, T_2 fires after $t_2 - t_1$, removing the token from p_2 to p_4 .

The movement of tokens through a Petri Net can be transformed into matrix form. Then the marking of the PN after the r^{th} transition, M_r , can be found by equation 2.2.

$$M_r = M_0 + A^T \cdot T_1 \quad (2.2)$$

Where: M_0 is a column matrix ($n, 1$), where n is the number of places, showing the initial marking of the net. T_1 is a column matrix ($m, 1$) where m is the number of transitions, showing the number of times each transition has fired in the r transitions, A is the incidence matrix (m, n) where each element a_{ij} corresponds to the effect that

transition i has on place j . Using equation 2.2 the marking of a net, the distribution of tokens within it, can be determined at any time.

Figure 2.7 shows a PN with three transitions (T_1 , T_2 and T_3) and five places (p_1 , p_2 , p_3 , p_4 and p_5). Places p_1 , p_2 and p_4 are marked, including one token each. It is also known that $T_1 < T_2 < T_3$. The initial marking, M_0 , seen in equation 2.3 represents the marking of the PN places in Figure 2.7, and shows that places p_1 , p_2 and p_4 hold one token each, whereas all the other places are empty. Similarly, the transpose of the incidence matrix, A^T , in equation 2.4, corresponds to the underlying transitions and places (the places and transitions are presented on the left and above the matrix, correspondingly), showing how a token moves from one place to another, once a transition fires. For instance, once transition T_1 fires, a token is removed from places p_1 and p_2 and reproduced to place p_3 .

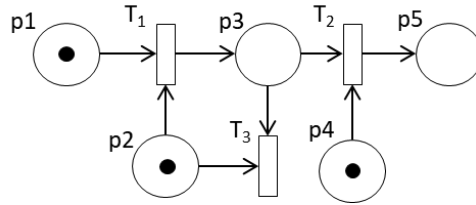


Figure 2.7 PN Firing Process

Hence, in the first column of equation 2.4, the value ‘-1’ is placed in the first and second rows that correspond to places p_1 and p_2 from which the tokens are removed, whereas the value ‘1’ is placed in the third row of the matrix that corresponds to place p_3 to which the token is added.

$$M_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad (2.3)$$

$$A^T = \begin{matrix} & \begin{matrix} T_1 & T_2 & T_3 \end{matrix} \\ \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{matrix} & \begin{bmatrix} -1 & 0 & 0 \\ -1 & 0 & 1 \\ 1 & -1 & -1 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{matrix} \quad (2.4)$$

Additionally, the matrices in equations 2.5 and 2.6 are created for each PN transition. For instance, equation 2.5, the transition matrix for T_1 , when it is applied in equation 2.2 M_1 is developed, showing that a token should be moved from p_1 and p_2 and added to place p_3 . Hence, places p_3 and p_4 are marked with one token each. Similarly, once equation 2.6, the transition matrix for T_2 , is applied in equation 2.2 matrix M_1 is used instead of M_0 .

$$T_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (2.5) \quad T_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad (2.6) \quad M_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (2.7) \quad M_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (2.8)$$

In this case, T2 fires and the tokens from p3 and p4 are added to place p5 as seen from matrix M₂ in equation 2.8.

Petri Net models can be analysed statically and dynamically. The static analysis can be performed by either developing the reachability graph or applying the invariants method (place/transition invariants). They can be analysed, simulated and modelled numerically and graphically, using various software tools, such as C++, MATLAB/Simulink, CPN toolset, PN Toolbox, SimHPN (GISED). The dynamic analysis of the PN is conducted performing model simulation in order to ensure that all the paths of the model have been executed properly and to detect any possible undesirable behaviour and incorrect or omitted logic. The static and dynamic analysis can also be used to validate the model, i.e. check its correctness and completeness, either in terms of syntax, semantics and structure through the structural and behavioural PN properties or in terms of logic through the simulation analysis detecting undesirable PN behaviour.

Over the years, the standard PN has been extended and hence, several models with advanced capabilities have been developed enabling the modelling of real systems used in industry.

Coloured Petri Nets (CPNs), an extension of PNs, introduced by K. Jensen 1990, is a high level PN form that uses tokens with colours, holding complex information. The tokens of this model can carry data values, i.e. system information, around the net and can hence be distinguished from each other. Each token can affect the firing of a particular transition in a different way, depending on its colour. The colour is a graphical way of distinguishing between different tokens, by giving them a label. The label/colour of the token can change once it has passed through a transition.

Stochastic Petri Nets with Aging Tokens (Volovoi, 2004) can be considered as an extension of CPNs providing more flexibility and agility to system modelling. The main idea of Stochastic Petri Nets (SPNs) with aging tokens is the introduction of tokens with memory. This extension is useful for modelling applications such as load

sharing, multiphase missions, repairable systems with spares, systems on demand and shared pools of identical imperfectly repaired components.

Another variation of traditional PNs is the *Abridged Petri Net (APN)*, introduced by Volovoi (2013), ensuring flexibility of token tracking through the transitions, which are depicted as directed arcs. The change of state in a system is modelled by moving a token from an input place to an output through the transition, i.e. the arc that connects the input and output places. Additionally, each transition can have at most one input and output place. This model includes two types of tokens, discrete ('colours') and continuous ('ages'). Discrete event and Monte-Carlo simulations can be used for the analysis and evaluation of system performance. The tokens are able to change during the simulation (be stationary or move during time) and either inhibitors or enablers are used for the modelling of component interactions. The basic characteristic of this model is its ability to link place nodes directly by arcs, similar to Markov models. APNs can be applied to large models taking advantage of the hierarchical model representation and also to complex systems with dynamic features.

2.4 Methods for Automation of Reliability Models

2.4.1 Introduction

The idea of automated reliability model generation is an important area of reliability engineering, which has been under development since the 1970s with the introduction of two major methods for the automatic construction of FTs namely, the decision table method (Salem et al., 1977) and the digraph method (Lapp & Powers, 1977). In the following years, several other methods have been developed considering commonly found structural complexities in engineering systems such as loops and electric circuits. Approaches have been presented for the automated construction of other reliability models such as Cause-Consequence Diagrams (Valaityte et al., 2010) Reliability Block Diagrams (Liu et al., 2013), Markov Chain Models (Katayama et al., 2014; Brameret et al., 2015) and Petri Nets (Aalst et al., 2004; Robidoux et al., 2009; Agarwal, 2013; Stockwell & Dunnett, 2013; André et al., 2014).

A review of various automated reliability methods has been undertaken in this section in order to identify advantageous properties for automation and to identify potential

gaps within the literature that this PN research study will attempt to address to support automation of this reliability modelling method.

2.4.2 Overview of Methods for Automated Reliability Modelling (not including PN)

2.4.2.1 Automated generation of the Fault Tree Model

The automated generation of Fault Trees has received the most attention, based on literature findings. Several methods have been proposed for the automated construction of this reliability model, as reviewed in this section.

Decision table method, introduced by Salem et al. (1977), models the system behaviour describing the relation between the inputs, outputs and the states of components. This method acts as a matrix in which the columns of decision tables correspond to the inputs, outputs and states of the components and the rows present all the possible combinations between the inputs and component states along with their respective outputs. Therefore, the tables can present different states of the component such as working or failed and how the component behaves as a result of different inputs from other components within the system.

Salem et al. introduced the decision table method by developing in 1979 a computer package, the Computer Automated Tree (CAT). The CAT code takes as an input the decision tables, created by the user, and the TOP event specification to generate the fault trees. The software has been designed to cater for multiple FTs simultaneously. The CAT code has been successfully applied in a Residual Heat Removal (RHR) system, generating the required FT in a short time. The main limitations of this methodology are the effort the user should make to recognise and define the TOP event and create the components decision tables. Also, this method does not provide any facilities for the detection and classification of control loops or circuits.

A relatively new method based on decision tables is that introduced by Majdara and Wakabayashi (2010). This method introduces the concept of two types of tables to model system components. The first table is the function table, which is the same of the decision table introduced by Salem et al. 1977 and the second type is the novel state-transition table that describes the operational states of a component. The state transition table is created for components with different states and shows the state

changes from initial state to final state. For example, the operational states of a switch are the open and close states. Both function and transition state tables are manually created by the user. Majdara and Wakabayashi also developed an algorithm to generate a FT based on an occurrence of an undesired event being defined. The code uses the input-output connections of components to trace the cause of the undesired event. The algorithm traces back from the occurrence and identifies the component states or outputs caused the event. Then the FT is generated based on the outputs. The FT is generated using the new tables following the same approach employed by Salem et al.

The *digraph method* is based on the construction of a directed graph, which consists of nodes that represent process variables. The process variables indicate properties of the flow such as mass flow rate, pressure, temperature and others. Any system or process can be described in terms of a flow such as flow of fluid, charge, data, information and signal. The nodes are connected by directed edges, where the direction is determined according to the relationship between the variables it joins. If a deviation in a variable A produces a deviation in variable B then the direction of the edge is from variable A to variable B. A directed edge can be characterised as: *normal* when the relationship is normally true; *conditional* when the relationship occurs only when a certain condition is satisfied; and *mutually exclusive* when several edges connect the same pair of nodes. Only one relationship is in operation at any one time.

A number is assigned to the edge depending on the rate of change of the second deviation relative to the first. The values that can be used are: ‘-10’, ‘-1’, ‘0’, ‘+1’ and ‘+10’. The magnitude of deviation is indicated by none (0), moderate (1) or very large (10), by implying the following:

- None (0): if a moderate deviation in one process variable causes none/negligible deviation in another; there is no edge drawn between the two nodes.
- Moderate (1): if a moderate deviation in one causes a moderate deviation in another, the directed edge between the two nodes is denoted by 1 preceded by a sign.

- Very large (10): if a moderate deviation in one causes a very large deviation in another, their relationship is denoted by 10 preceded by a sign.

The signs ‘+’ and ‘-’ depend on whether the deviations in the dependent variable increase, or decrease, when the independent variable increases. The number associated with directed edges is called gain and be considered as the partial derivative of the first variable with respect to the second variable.

Lapp and Powers 1977 were the first to integrate the digraph (directed graph) method into automated FT construction. This method starts with the development of the digraph by the user for a given system and then uses a programmed algorithm in order to transform the digraph into a fault tree. The digraph method constituted a remarkable achievement in the evolution of the automated generation of reliability models since it targeted modelling of complex systems including the identification and classification of control loops. In this approach, operators have been introduced which logically traverse fault propagation through Negative Feed-Back Loops (NFBL) and Negative Feed-Forward Loops (NFFL).

Andrews and Henry 1997 introduced the ***modified decision table method*** combining the decision table method due to its ability to identify the normal state of systems and the digraph method due to its ability to detect, classify and analyse control loops. The classification and analysis of control loops is accomplished by using two new circuit operators, one for tracing current and the other for tracing no current in circuits. These operators provide a more efficient FT development in terms of its logical consistency since the operators can significantly reduce the size of trees by eliminating repeated events. The modified decision tables include the inputs, outputs and states of the components as traditional decision tables.

AltaRica, a high-level formal description modelling language, first created at the Computer Science Laboratory of Bordeaux (LaBRI) by Point and Rauzy in 1999, is dedicated to safety analysis. A second version of this language, AltaRica Data-flow (ADF) (Rauzy, 2002; Boiteau et al., 2006), was developed to handle industrial scale models. This second version was improved in 2013 and AltaRica 3.0 was developed. In 2013, Prosvirnova and Rauzy proposed a method that compiles a mathematical model, the Guarded Transition System (GTS), which supports the representation of

components with bidirectional flows, into the description of a system model using the AltaRica syntax in order to automatically generate FTs for systems containing control loops. GTS is a state/transition formalism that uses concepts from various reliability modes such as Reliability Block Diagrams, Markov chains and PNs. This formalism provides higher efficiency to the system making it possible to design acausal components, i.e. components for which the input and output flows are decided at run time, and to handle control loops in the system. The GTS formalism enables the generation of FTs by transforming the states/transitions of the behavioural model into a set of Boolean formulae and also enhances the reusability of FTs and facilitates their maintenance since it is considered a high level structure.

Papadopoulos et al. (2001) developed the ***Hierarchical Performed Hazard Origin and Propagation Studies (HiP-HOPS)*** tool that performs an automated reliability analysis. The main idea of the HiP-HOPS is the automatic synthesis of FTs and Failure Modes and Effects Analyses (FMEAs) using fast linear-time algorithms. Adachi et al. (2011) integrated system modelling, automated dependability (reliability, availability, safety maintainability and security) and optimization techniques using HiP-HOPS, rendering this tool applicable to highly interactive and dynamic systems. This work overcomes limitations such as difficulties in conducting automatic analysis of complex systems with multiple failure modes, dealing with the assumption of the previous work that the system behaviour remains stable over time. HiP-HOPS, commercialised in 2012 by University of Hull, has been used in several automotive and engineering industrial cases. This tool is compatible with a range of modelling notations and offers scalability of the analysis and unique capabilities for fault modelling.

Joshi et al. (2007) proposed a method in which ***Static Fault Tree (SFT) models are developed, taking as input AADL models***. In this work, an *AADL model* that captures the architectural aspects of a system such as the properties of the components, features of the interactions between components, internal structure of components such as subcomponents' connections and properties, is used as an input and an *Error Model Annex* that captures the component faults and failure modes is manually developed by the analyst. Using these two models, a Directed Graph (DG), including topology system information faults and failure modes, is created that is then used to generate an

intermediate FT, by applying a recursive algorithm. Finally, Computer Aided Fault Tree Analysis (CAFTA), a commercial FTA tool, is applied to the intermediate FT and CAFTA FT is generated using software capabilities.

2.4.2.2 Automated generation of other Reliability Models

Valaityte et al. 2010 developed an algorithm for the automatic generation of *the Cause Consequence Diagram method*. The system information provided as inputs to this algorithm are: a topology diagram for a given system; failure modes and rates of system components; decision tables for each component; the initiating event/component and its function; and the stopping criteria, i.e. consequences from which the diagram path can be terminated. The proposed algorithm starts from the initiating component and then by determining the potential outputs of the component, creates a decision box with ‘YES’/’NO’ branches, according to the output. If the stopping criteria are satisfied, then a consequence box related to the output is added and the number of decision boxes is checked. Once all the boxes have been developed the CCD has been constructed.

Liu et al., 2013 proposed the automated generation of *Reliability Block Diagrams (RBDs) from a SysML Internal Block Diagram (IBD)* to describe the internal structure of the system. Once the IBD model is created, and expressed in XMI format, then the reachability matrix is generated. This matrix, A, that shows the physical relations between the system’s blocks, is based on graph theory and presented in the matrix in Figure 2.8 (a).

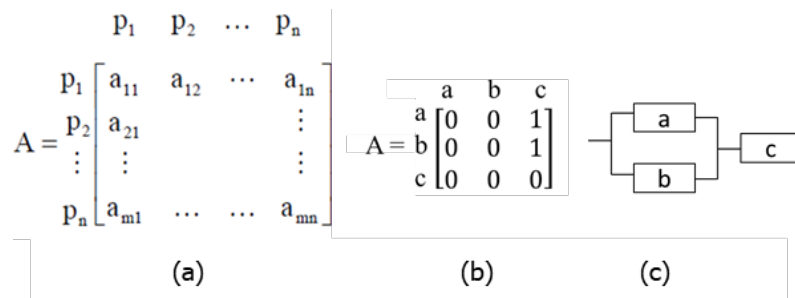


Figure 2.8 Reachability Matrix

The reachability matrix is translated into a RBD and each row of the reachability matrix is examined identifying the inputs to each block and the connectivity between blocks. The transformation rule is better explained with the help of a simple example. Hence, according to the transformation rule for matrix A in Figure 2.8 (b), block c has

two input blocks, a and b, denoted with value 1 in the first and second rows of the third column respectively. Additionally, the input blocks to block c, i.e. blocks a and b, constitute a parallel model since these blocks do not have any other inputs/outputs. The corresponding RBD for the matrix in Figure 2.8 (b) is illustrated in Figure 2.8 (c).

Katayama et al. (2014) proposed a methodology for the generation of ***Markov chain usage models from De-sequence diagram***. This diagram is generated from the combination of UML Sequence diagram and UML Deployment diagram. Therefore, once the De-sequence diagram is developed, the system is decomposed into cases to establish the time points based on the instances of the messages. For each case identified, initial and final states, pre-condition, post-condition and state name, and time violation and probability of time violation are defined. The *pre-conditions* that refer to the system or objects before execution of the operation and *post-conditions* that refer to the state of objects after completion of the operation, are examined to describe how the cases connect together, i.e. sequence of relationships of cases. Once individual Markov chain usage models have been developed for each case identified in system, the pre and post-conditions are examined for all the cases and are finally combined to generate the final Markov chain usage model of the system.

Brameret et al. (2015) proposed a heuristic algorithm for the automated generation of ***partial Markov chains from AltaRica***. This method generates a partial reachability graph for a given system avoiding the state-space explosion problem, keeping only the ‘best’ states, i.e. trying to keep states visited only once. Therefore, once the system is defined using the AltaRica syntax, applying the Guarded Transition System (GTS) variables, a partial reachability graph is automatically generated by discarding revisited states safely, keeping only the shortest paths from the source to the selected candidate, avoiding the state-explosion limitation from which the conventional Markov chains suffer.

2.4.2.3 Summary of Methods

The goal of the literature review conducted for the automated generation of the FT, CCD, RBD and Markov chain models is to identify merits, limitations and challenges regarding the automation procedure. The methods reviewed in this section for the automated reliability modelling have been summarised in Table 2.4, and advantages, disadvantages and automation challenges, describing the level of automation of each

Table 2.4 Table of Methods for Automated Reliability Modelling (not including PN)

Methods		Advantages	Disadvantages	Challenges
FT	Decision Table	Built into a computer package, the CAT, which is able to cater for multiple FTs simultaneously.	Requires the user's intervention to manually develop decision tables of system components.	Lacks full automation.
			Only limited system characteristics are catered for (no loops/circuits).	
	Digraph	Able to handle some complex system characteristics such as detection and classification of control loops and circuits.	Requires the user's intervention to construct the initial digraph structure from the system diagram/knowledge	Lacks full automation.
			The algorithm is not rigorous to be integrated into a computer package since it cannot be always known with certainty if FTs produce the correct minimal cut sets.	
	Modified Decision Table	Able to handle control loops and enables the identification of any circuits within systems.	Requires the user's intervention to manually develop decision tables of system components.	Lacks full automation.
	AltaRica	Able to handle control loops and enables the identification of any circuits within systems.	Requires the user to input the system's requirements (state, transitions, initial conditions, etc.) to describe the model by developing the corresponding code using AltaRica syntax.	Lacks full automation.
			AltaRica syntax can be complicated.	
	HiP-HOPS	Applicable to highly interactive and dynamic systems.	Unable to handle control loops.	Lacks full automation.
			Requires the user's intervention to define the failure modes of components and failure expressions that link these modes and inputs to the output of each component.	
	AADL models to SFTs (Joshi et al., 2007)	Built into a computer package named CAFTA.	An AADL Error Model Annex is developed by the user who requires advanced knowledge of the AADL modelling language.	Lacks full automation.
			Focus only on SFTs.	
			CAFTA used for model's construction is not able to handle large trees, since it cannot find all the minimal cut sets.	
CCD	Decision Tables to CCD (Valaityte et al., 2007)	Able to handle some complex system characteristics such as detection and classification of control loops and circuits.	Requires the user's intervention to manually develop decision tables of system components and input system requirements such as failure modes and rates, stopping criteria and others to the algorithm.	Lacks full automation.
RBD	SysML IBD to RBD (Liu et al., 2013)	Able to generate static RBD from SysML IBD that belongs to standard SysML diagrams.	Focus only on static RBDs. Does not provide a universal algorithm to support dynamic RBD modelling.	Full automated method.
			Unable to handle control loops.	
Markov Chain	De-sequence Diagram to Markov Chain (Katayama et al., 2014)	The concepts of standard UML diagrams (Sequence and Deployment) have used.	The algorithm input, De-sequence diagram, development requires manual effort and knowledge by the user to generate it, since this diagram does not belong to the standard UML/SysML diagrams.	Lacks full automation.
			Lacks evidence about the automation part of the reliability model generation since the technical information about the code development and software package is inadequate.	
			Applied only to simple systems.	
	AltaRica to Markov Chain (Brameret et al., 2015)	Able to avoid the state-space explosion problem.	Requires the user's intervention to manually define the AltaRica GTS variables for a given system.	Lacks full automation.

method, have been identified. Therefore, as can be seen from Table 2.4, the automated generation of the FT model has received considerable attention compared to the others, due to the extensive application of this model in industry. Additionally, following Table 2.4, the most common limitations and challenges met during the review of the methods proposed for the automated generation of these reliability models are that: (i) only some of the methods can model control loops; and (ii) the majority of the methods result in the semi-automated generation of reliability models, since only a limited number of them can automatically retrieve a given industrial system representation. Hence, during the automated construction of PN models, considerable attention should be given to these commonly met challenges, by developing a methodology that overcomes these limitations.

2.5 Automated Generation of the Petri Net Model

Given the Petri Net model has been selected in section 1.2 to be automatically generated, it is required to establish the current state of art regarding the automation procedure. Therefore, the methods identified for the PN automated generation and applied to industrial processes and systems are reviewed in the following sub-sections.

2.5.1 Review of Methods for Automated Petri Net Modelling (Process-based Approaches)

The *Alpha algorithm* put forward by van der Aalst, Weijters and Märušter in 2004 with the aim to reconstruct causality from event logs was first used in process mining. An event log consists of a multiset of traces, whereas a trace describes the order of activities following a particular path within the process. This algorithm is able to create automatically a Petri Net process model, taking as input an event log that contains all the possible traces of a model describing the control flow of different paths that exist in the model. The event logs are stored either in flat files, files without internal hierarchy, or database tables. This algorithm is also able to discover and handle concurrency, loops and choices if they exist in the models.

The following steps are pre-steps for application of the Alpha algorithm:

1. Obtain or create an Excel/XML file with the process traces (process data).

2. Convert the file from step 1 into eXtensible Event Stream (XES)/(Mining eXtensible Markup Language) MXML format using a tool such as Nitro or XESame.
3. Import the file created in step 2 into Process Mining (ProM) framework, an open source tool, developed at Eindhoven University of Technology, with more than 300 plug-ins that allow process mining algorithms.

Then, the Alpha algorithm is applied.

A simple example in terms of the Alpha Algorithm is discussed. The event log is assumed to be L1 which is presented in equation 2.9.

$$L1 = [< a, b, c, d >, < a, c, b, d >, < a, e, d >] \quad (2.9)$$

Where: a, b, c, d, e are different activities.

There are three traces (paths) in the event log as can be seen from equation 2.9. The first is 'abcd', the second 'acbd' and the third 'aed'. Once L1 event log is fed into ProM tool, the Alpha algorithm is applied and the corresponding PN is generated. The generated PN model has an initial place and a final place and all the activities included in L1 such a, b, c and d are transformed into PN transitions and between these transitions, places denoted as p1, p2, etc. are added. Arcs connect the places to transitions following the sequences of activities in the paths. The Alpha algorithm is used with the objective of automatically visualising business processes in terms of PN models and animated simulations. Additionally, this algorithm can be used to predict the completion time of a model or specific traces of a model, using data such as times and frequencies for the PN transitions provided by the user. According to the visualisation and animation of PN's behaviour and simulation results obtained from the PN model analysis, limiting factors such as unintended behaviour regarding the flow of information in the process or bottlenecks can be detected and, hence, recommendations can be made to enhance process' performance.

To conclude, this algorithm requires the order that the activities happen during the process and hence, the user needs to obtain or create the input file identifying the event logs, i.e. the sequence of activities, path id, i.e. a unique identification number, and pass and fail probability of each path. The main limitation is its weakness to

handle specific complex cases such as complex nested loops or multiple activities logged with the same footprint (does not allow duplicates). In such cases, the algorithm generates inaccurate PN models since it cannot graphically represent these features (Aalst, 2011). The Alpha algorithm cannot be taken forward due to the weakness to automatically retrieve the system description and to model the aforementioned advanced structural characteristics which are within the scope of this study.

Alhroob et al. 2010 presented a new methodology that transforms the *UML Sequence Diagram (SD) and Class Diagram* into *High Level Petri Net (HLPN) models*, known as Coloured Petri Nets. HLPN models use algebraic terms to explain PN elements and the places in these models are marked by tokens that hold complex structured data. The UML SD and Class diagram are used as the source of system behavioural specifications. The Object Constraint Language (OCL) is also used to provide structural specifications such as preconditions (must be true at the moment the operation is executed), postconditions (evaluated to true at the moment the operation ends) and guards (must be true before state transition can occur). The requirement of this methodology is the topology (events in the system and their ordering) of a given system. The steps of the methodology are described as follows:

1. A SD and a Class diagram are created or provided by industry.
2. Decomposition of the SD into fragments such as sequence (for events executed in series), parallel (for parallel processes that contain two or more sets of events executed concurrently), loop (for series of repeated events), alternative (divides fragment into two groups and defines conditions for each one) and option (for events executed only if a condition is true).
 - The SD of a system is manually decomposed into fragments by the user. If the system is complex, this step can entail much effort. Once the system is complex, this step can entail much effort. The fragments are used to provide flexibility in the next steps since the XML exported differentiates the fragments based on the labels mentioned earlier, i.e. sequence, loop, parallel etc. Once the fragment decomposition is completed, then, the software used divides each fragment into its events.

3. Transformation of the SD fragments edges to HLPN nodes.
 - For each SD edge, a HLPN node is created. A HLPN node consists of two classes (an input and an output) which are connected by an event as described in step 2. In a node, the event is represented by a rectangle, whereas the input and output with ovals. An algorithm is developed as follows:
 - a) Enumerate all the events.
 - b) The algorithm traces around the diagram and it detects the first event and checks if it was visited before. If it was not visited before, check if it has input and output arguments.
 - c) Determine the input and output arguments.
 - d) Once an event has input and output class, it becomes node.
 - e) Represent the event in rectangle shape, input and output with oval.
 - f) Connect the nodes with output arrow from input class to the event and second arrow from the event to the output class.
4. Join each node with its input derived from the class diagram and OCL.
 - HLPN models require more information and specifications and thus this additional information is derived from the class diagram (XML format) and OCL. The required information is stored in a table, named Information Table (InfT). The algorithm selects methods of Class diagram that are used in SD and stores its input and output to a table, the Information Table (InfT). Additionally, the event attributes from the Class diagram and the remaining information (preconditions, guards, etc.) are transferred from the OCL to InfT.
5. Identification of relationships between the HLPN nodes.
 - A Nodes Relationship Table (NRT) is proposed to set the relationships between the HLPN nodes, as developed in step 3, using the InfT. An algorithm has been developed to capture the relationship for every pair of nodes. The algorithm consists of various conditions that define the relationships between the nodes. This algorithm can identify if node X is ancestor/ parent of node Y. It can also identify if two nodes are connected in parallel or if two nodes are connected under a loop or if a

node is the last one before the final node. Each relationship is denoted by a different symbol in the NRT. A Node Relationships Table (NRT) is used to store all the possible conditions for the nodes.

6. Combination of the HLPN blocks developing a Combined Fragment Net (CFN).
 - According to the information stored in the NRT, a CFN is developed by connecting all nodes according to SD fragment ordering logic. This step is done manually.
7. The final step is the development of the PN model based on the NRT information and the CFN.

The proposed methodology, which is developed to formally verify the informal syntax and semantics of the UML diagrams, is demonstrated by its applicability to a process describing the operation of a car parking ticket machine.

To conclude, this approach focuses on the HLPN (CPN) generation using UML Class and Sequence diagrams to represent the behavioural and structural aspects of a given system respectively. A HLPN model is generated by the user based on the topology information stored in the NRT that is obtained from the UML SD. The proposed methodology, which is not fully automated, requires model pre-processing, since the system information cannot be obtained automatically from the UML diagrams. This technique indicates the semi-automated level of the method and hence, it has not been taken forward.

Agarwal (2013) proposes the ***UML 2 Activity Diagram (AD) transformation into PN models*** for verification purposes. This approach covers the transformation of fundamental elements of UML 2 AD, such as control edges (arcs in AD), opaque action nodes, fork and join nodes, decision and merge nodes, as well as initial and final activity nodes, into PNs. The mapping rules for the AD translation into a PN are explained in detail. Thus, the control edges, initial nodes, final activity nodes, decision and merge nodes of the UML 2 AD are mapped into the PN places, whereas the opaque action, fork and join nodes are mapped into the PN transitions. The tool developed is based on the Eclipse Java Platform and named AD2petri. AD2petri has as input the UML 2 AD in the form of an XML file and outputs PN files written in the Petri Net Markup Language (PNML).

The PNML is a standard defined by ISO/IEC 15909-2 that provides an intermediate representation, which enables the automated generation of various types of PNs such as Coloured PNs, Timed PNs and Stochastic PNs. The PNML interface is Extensible Markup Language-based meta-language, which supports the interoperability and exchangeability between various tools that used for PN models, such as Ina, CPN/Tools, PEP, TimeNet, etc. PNML can take as input XML files with specific structure that should comply with predefined rules.

Although some of the proposed transformation rules are validated by their application into a simple example for a management process, the documentation in this work regarding the information about the code development is inadequate, since the code proposed for the model transformation is not explained thoroughly. Additionally, two of the transformation rules, for the control loops and preceding transitions (where action x should precede action y), are not validated since these characteristics are not included in the UML 2 AD used for the proposed example. Finally, this approach does not provide a generic applicability, because firstly, it is only applied to a simple process without considering complex processes and secondly, the code does not cover the transformation of all the elements included in a UML 2 AD. For all the aforementioned limitations, this method has not been taken forward.

2.5.2 Review of Methods for Automated Petri Net Modelling (System-based Approaches)

Robidoux et al. (2009) presented an approach for formal modelling and verifying an extension to State-based RBD (SRBD), named ***Dynamic Reliability Block Diagram (DRBD)***, for computer-based systems. In this work, an algorithm that generates ***Coloured Petri Nets (CPNs)*** using as a starting point DRBD models is introduced. In order to identify design flaws and faulty states, the behavioural properties of the DRBD model are also verified, using existing CPN Tools. The novelty of DRBD is the ability to model the dynamic behaviour of systems, i.e. to model dependencies among components or subsystems. The capabilities of this model have been enhanced by introducing two new controller blocks, called State-based Dependency controller (SDEP) and Spare part controller (SPARE), that allow the modelling of dependency and redundancy relationships between components in a system, respectively.

In this work, a formal Reliability Markup Language (RML), an XML-based language, is introduced to formally describe the components, structure and dynamic behaviour of DRBD models. RML is based on the Backus-Naur Form (BNF) which is a formal notation used to describe the syntax of a given language. Therefore, DRBD is defined in RML including all the components and controllers that exist in the model. The properties of these components and controllers, such as the connection of components in series or parallel, the initial state of components (active/standby/failed), the trigger and target events as well as the state-based dependency between the components (activation/deactivation/failure), are presented as nested RML elements. This RML file, which is manually developed, is the input to the algorithm for the CPN generation.

Once the DRBD model has been manually defined according to RML, an algorithm is applied to the model transforming it into a CPN. The model transformation is conducted into two steps: in the first step the **SRBD** is converted into a CPN; in the second step the **controller blocks** of the model, such as **SPARE** and **SDEP**, are converted into controllers CPNs and then added into the CPN developed in the first step. A controller CPN has transitions and arcs that connect to the start places of the simple CPN components created in the first step.

Before presenting the algorithm steps for the SRBD conversion into a CPN model, the steps followed for a simple component conversion into a simple-component CPN are described as follows:

1. A simple-component CPN consists of two places, C_{1_start} and C_{1_up} and three transitions, in_C_1 , C_{1_fail} and $C_{1_destruct}$, as seen in Figure 2.9. C_{1_start} contains an active token, since the initial state of component is active, whereas the token state can have one of the following values: Active, Standby or Failed.
2. Once C_1 remains active and another input connection to the component has an 'active' token, then in_C_1 may fire.
3. This firing moves an 'active' token into place C_{1_up} , showing that component is active, as seen in Figure 2.9. This 'active' token can be moved into other nodes through the output connection (active) shown in Figure 2.9.
4. While C_1 is active and transition $C_{1_destruct}$ fires, then the active token in place C_{1_start} is replaced by a failed one.

5. Transition $C1_fail$ is enabled and a 'true' token is generated showing that $C1$ has failed. This 'true' token can be moved into other nodes through the output connections (failed) as illustrated in Figure 2.9.

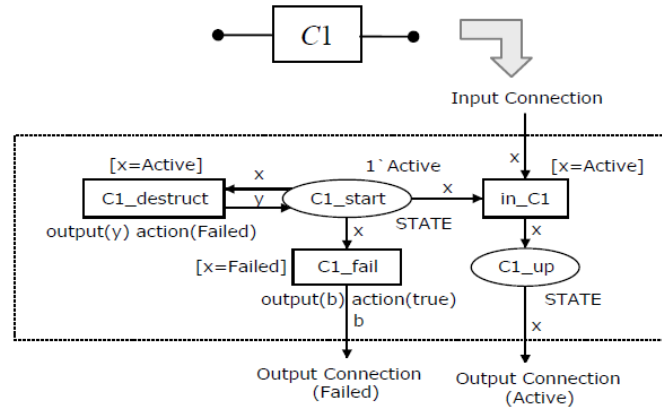


Figure 2. 9 Simple Component Conversion into a Simple-Component CPN (Robidoux et al., 2010)

The algorithm for the SRBD transformation into CPN performs the following steps:

1. SRBD model is considered as a serial component and input and output connections are created. A serial component may contain one or more simple/parallel components.
2. A *for-loop* is applied in order to convert each of the structural components into a CPN. If a contained compoennet is a:
 - a. simple (connected in series) or spare component, then it is directly converted into a CPN model.
 - b. parallel component, then input and output connections are created and then a *for-loop* is applied again to transform each of the contained components into a CPN. Step 2 is repeated for each contained structural component in the parallel component.
3. A parallel CPN component is generated by the connection of all simple and series CPN components only when all contained components in a parallel component have been transformed into CPNs.
4. A serial CPN component is generated by the connection of all simple and parallel CPN components only when all contained components in a serial component have been transformed into CPNs.

Similarly, the algorithms for the transformation of SPARE and SDEP controllers into controllers CPN is described as follows:

— Spare controller conversion into spare-controller DRBD is explained as follows:

1. A place P_{1_start} and a transition P_{1_fail} are created for the primary component.
2. For each spare component a place S_{i_start} and a transition S_{i_fail} .
3. SPC_P_1 transition is introduced connecting place P_{1_start} with place S_{1_start} such that when P_1 fails and S_1 is standby, S_1 to be activated.
4. For each spare component S_i ($i=1$ to $i=n-1$) a transition SPC_S_i is created and connecting S_{i_start} and $S_{(i+1)_start}$ such that when S_1 fails and $S_{(i+1)}$ is standby, $S_{(i+1)}$ to be activated.
5. Place SPC_sync_1 is created connecting transitions SPC_P_1 and P_{1_fail} .
6. For each spare component S_i ($i=1$ to $i=n-1$) a place $SPC_sync_{(i+1)}$ is created connecting transitions SPC_S_i and S_{i_fail} .

— State controller conversion into state-controller DRBD is explained as follows:

1. A place C_{1_start} is created for the trigger component.
2. For each target component a place T_{i_start} is created, for $i \geq 2$.
3. Transition $SDEP$ is introduced connecting all places T_{i_start} according to the trigger and target events defined in step 2.
4. If trigger event is activated create a transition in_C_i for trigger component and a place $SDEP_sync$ that connects $SDEP$ and in_C_i .
5. If trigger event failed create a transition C_{1_fail} for trigger component and a place $SDEP_sync$ that connects $SDEP$ and C_{1_fail} .

Finally, the controller CPNs are added into the CPN developed from the SRBD model. This is accomplished by merging the starting places such as P_{1_start} and status transitions such as S_{1_start} from the controller CPN to the corresponding places and transitions presented in the CPN for the SRBD.

The proposed methodology is demonstrated by its applicability to a redundant generator system. Thus, the algorithm creates a CPN model for the redundant generator system, taking as input static and dynamic RBDs with the system description in RML format. The output CPN model is successfully verified using CPN Tools in order to guarantee the correctness of the DRBD model, by checking the absence of deadlocks. Although this approach enables the automated CPN model

generation, the main limitation of this work is the development of the system representation i.e. development of SRBDs and DRBDs using RML, since it requires manual effort as well as programming language knowledge by the user.

Stockwell and Dunnett (2013) presented a novel approach for the automated generation of reliability models for *phased-mission systems, based on Petri Nets*. The system modelling consists of the: (i) component, system and mission descriptions; (ii) system failure conditions; and (iii) failure and repair data. For the component modelling two tables, the decision and operational mode tables, are manually developed for each component and added to the component library of the system. This library allows the reuse of components. Operational tables are only created for components that have more than one mode of operation. The system description corresponds to a topology diagram that shows how the components link together. The mission description corresponds to the development of phase models that describes the different phases the mission can enter with the conditions of the system needed to transition from one phase to another. The initial and starting conditions are also identified. Additional input information is the failure conditions and the system failure modes. Finally, the failure and repair (if available) data for each component is provided to determine a reliability estimate. It is noted that decision tables are required to include time dependencies in order to describe components' behaviour when the system undertakes different phases of the mission.

The main steps of the algorithm are reviewed as follows:

1. The circuit lists, only required when the system contains electrical components, are automatically generated by the code, written in C++, using the information stored in the component tables within the library and the system topology. These lists contain the unique identities of components, i.e. whether or not current flows, in an electrical system. Each of the circuit list is used to create a circuit PN (CiPN).
2. Each action of each component is presented using Petri Net models, and hence Component Petri Net (CPN), System Petri Net (SPN), Circuit Petri Net (CiPN) and Phase Petri Net (PPN) models are automatically generated.
3. The reliability model is the SPN, which is created from the combination of the CiPNs and CPNs, and the PPN connected together.

A simulator implemented to the proposed algorithm can perform a number of simulations to calculate the unreliability of the system for both a single mission and multiple successive missions. The shortcoming is that the user is required to generate as input to the software a system structure file including topology information, number of phases the system can reside in, failure modes and the repair data, etc., which is an error-prone and time-consuming process, particularly for larger systems, as the authors state.

Taibi et al. (2013) proposed an automatic transformation to model multi-agent systems (MAS.) The proposed automation procedure follows two steps: (i) the system is described using a language named Multi-Agent System Description Language (MASDL); and (ii) transformation rules are applied to the file obtained from the first step to generate formal *Coloured Petri Net (CPN) models* from the system specification with the view to analyse and verify multi-agent system. A Multi-Agent System is a computerised system or program that presents several complex characteristics interacting to achieve a common goal. A MAS consists of a set of agents, i.e. any entity that senses its environment and acts over it (Glavic, 2006), interacting to achieve a common goal. MAS are usually met in a dynamic large-scale environment providing properties such as autonomy, robustness, and flexibility. The steps followed for the CPN generation are:

1. An XML file, using Multi-Agent System Description Language (MASDL), is manually developed in order to describe the computer system. The information comes from the system topology, the state of the system, its initial conditions and failure modes. The Multi-Agent System specification contains a list of *agents* consisting of the agent name, a list of its attributes, the current state and a list of (*entry*) *actions*, a list of *resources* that specifies objects except for agents existing in the system, a set of *objects* including information about the system environment, a list of *states* that describes the agents' states and objects states and a list of *actions* that can be undertaken by agents.
2. Once the XML MASDL file is created, it is transformed into the corresponding CPN models, using an XML-based language with a specific syntax, the Petri Net Description Language (PNDL). The main rules of the transformation algorithm are outlined as follows:

- The *agents* and *resources* obtained from the first step can be transformed to the PN **colours**.
- The *states* and *actions* obtained from the first step correspond to the CPN **places** and **transitions** respectively.
- The *entry states* in combination with the *transitions* create the **arcs**.

The output of this second step is an XML file that includes all the necessary information for the generation of CPN models.

Both MASDL and PNDL are XML-based languages, providing many advantages such as *interoperability* due to its universal syntax since XML can be easily readable between systems and *universality* due to its ability to represent most of the models with its simple and powerful syntax.

To conclude, in this work, MASD language is introduced for the specification of the agents and their environment and transformation rules are proposed for the formalisation, i.e. verification, of multi-agent systems. The main drawbacks of this method are that: the user needs to write the MASDL code in order to import the system information into the algorithm; and the method cannot provide generic applicability since it targets computer systems.

André et al. (2014) proposed a method in which *UML State Machine Diagrams (SMDs)* are transformed automatically into *Coloured Petri Net (CPN) models* to formally guarantee the system safety by verifying it against properties. Although UML is widely used in industry its semantics are not formally expressed, preventing the application of model checking techniques that can guarantee the system safety. The objective of this approach, which is mainly applied to systems with different states, is to provide formal semantics for the UML SMD by translation to the CPN formalism. The automatically generated CPNs are used to test and check the model using powerful tools such as CPN Tools, which can formally prove/disprove the initial system safety.

This approach is based on the model-to-text (M2T) transformation technique and is carried out using the Acceleo, a user-friendly tool, which is integrated into the Eclipse environment and facilitates the CPN generation by generating templates. These templates are used to define the transformation rules between the SMD metamodel,

the SMD model and the final CPN that should be developed, as explained in the following methodology steps. Therefore, the methodology steps are as follows:

1. **OMG SMD metamodel development.** Metamodel is the abstract syntax of models in which the general modelling frame, rules and constraints can be defined (OMG Unified Modelling Language (UML), Version 2.5, 2015). The OMG provides a predefined metamodel for the SMD that has been used in this work conducting a few minor simplifications. The SMD metamodel consists of a global state machine (class StateMachine) and each state machine includes states (class State and FinalState), transitions (class Transition), behaviours (class Behaviour), pseudostates (class HistoryState) and arcs between the states and transitions (class InputArc and OutputArc). Each class is represented by a block and in each block there are several properties such as id, name, state, action etc. that provide information for each class.
2. **SMD development** (input model). Based on the SMD metamodel structure, the SMD for a given system is developed, using the classes referred to in step 1.
3. **Acceleo template generation.** Each rule developed in the template will map an element from the metamodel and model to the text that is generated and corresponds to the desired CPN model. In this method, three algorithms (templates) are developed, one for the SMD states, one for the transitions and one for the history pseudostates respectively.
4. **CPN model generation.** The templates in step 3 are applied to the models developed in steps 1 and 2 and the CPN model is automatically generated, using Acceleo. The CPN generated is presented in an XML format. The SMD states are transformed into CPN places, the SMD transitions into CPN transitions, whereas the history pseudostates are mapped into the CPN arcs.

Once the CPN model has been created in XML format, then the user can import the XML file into CPN Tools, a free software used for modelling and verifying CPN models.

A main advantage of this method is that the starting point is a well-structured OMG UML diagram, the SMD, which eases the user to import automatically the system data/information. Nevertheless, limitations of this work are the inefficiency of this method to deal with complex and large SMD models and model systems including

loops. Additional shortcomings are the weakness of the Acceleio tool to provide advanced features such as functions, global variables and data structures leading to the development of complicated codes that require additional time to be analysed. Finally, this method is highly dependable on the CPN Tools syntax, since the rules defined in the Acceleio templates for the model transformation have been defined according to the CPN Tool syntax. Therefore, if this syntax changes, the Acceleio templates will be incompatible with the CPN Tool, and hence the output CPN model either will be incomplete/faulty/missing.

Reza and Chatterjee (2014) developed a method in which an *AADL model is transformed into a PN model* for verification purposes, using a set of mappings and mapping rules between AADL and Petri Net Markup Language (PNML). In this approach, PN models are developed to specify and verify the logical behaviours of real time-embedded systems used in critical application systems such as nuclear and power plants, medical devices, etc.

The steps followed for the translation of a given system modelled in AADL into a PN are reviewed as follows:

1. **First transformation:** transformation of AADL text model (input model) to XML format using mapping rules.
2. **Second transformation:** mapping of XML model to PNML using XSLT templates. The first action of this step is the development of an XSLT template. The XSLT template acts as a path that defines how the AADL model is translated into a PNML model. The output of this mapping is an XML file. Some of the mapping rules between the AADL and PNML components as used in this work are presented as follows:
 - AADL in/out data port, event port, port group and data access are transformed into PNML Places.
 - AADL system, process, thread and memory are transformed into PNML transitions.
 - AADL connection and bus are transformed into PNML arcs.

Despite the PN model generation, this method is not fully automated since the user's intervention is still required to import the AADL-XML model, obtained from the 1st

transformation, into the PNML framework. Additional shortcomings of this approach are the weakness of this method to handle complex systems and the software dependency.

2.5.3 Summary of Methods for Automated Petri Net Modelling

Following the literature review conducted to identify the attempts targeting the PN model generation, it was found that in most cases, the industrial system representations have been manually developed. All these methods that require the user intervention to import system information to the algorithm result in the semi-automated generation of PN models. Some of the inputs used to describe system topology are: Excel/XML files; decision and operational mode tables; AADL models; and newly introduced diagrams such as the Class Diagram. However, two of the reviewed methods support the full automation PN model generation. These are the methods proposed by Agarwal (2013) and André et al. (2014), in which the algorithm can automatically retrieve the system topology from a UML 2 AD and a UML SMD, respectively. However, a shortcoming of André's methodology is its weakness to model large systems or control loops.

According to the review for the PN model generation, only a limited number of methods, such as the methods proposed by Robidoux et al. (2009) and Alhroob et al. (2010), can handle and efficiently model systems with many components or complex characteristics such as control loops. Unfortunately, these two methods (Robidoux et al. 2009; Alhroob et al., 2010) require the user's intervention to input to the algorithm the industrial system representation, lacking full automation. Additionally, some of the reviewed methods can only be applied to specific domains, such as the work proposed by Taibi et al. (2013) that focuses only on computer systems, or can only be applied to specific cases, such as the Agarwal's methodology which firstly does not provide transformation rules for all the elements included in a UML 2 AD and secondly has only been applied to a simple process. Hence, these methods cannot provide a generic applicability.

Finally, the majority of the reviewed methods has used XML-based languages, such as the RML (Robidoux et al., 2009), PNML (Agarwal, 2013; Reza & Chatterjee, 2014), PNDL (Taibi et al., 2013) or tools such as the Acceleo tool (André et al., 2014), to generate an XML file, with the view to import it in an industrial tool, such as

the CPN Tools, to enable the automated PN generation. Although these methods can automatically generate PN models, they lack efficiency and cannot provide a robust and rigorous methodology. This is explained with the help of an example. Therefore, taking as a starting point a UML/SysML diagram, an XMI file can be obtained and then following these methods the syntax of the XMI file should be transformed into a format that can be used as input to the selected industrial PN tool. This input is a well-formed XML file that conforms to a set of very strict rules defined for the corresponding PN tool. However, if the version of the selected PN tool is updated (versions of these tools are updated at least once every year), and hence its syntax changes, these methods cannot guarantee accurate PN model generation, since the XML file developed from the methodology may not comply with the syntax of the XML file used as input to the tool.

2.6 Review and Research Motivations

A common characteristic of all the **reviewed methods** identified was the requirement of a realistic representation/description of the system topology, which is performed using tables/graphs/high-level modelling languages/markup languages.

Additionally, it was concluded that there is a lot of room for improvement and development in the automated generation of PN model since the current attempts target the generation of this model, are limited regarding the:

- *Level of automation*, requiring the user intervention as described in the aforementioned models.
- *Systems/Processes structural characteristics*, as described in the aforementioned models.
- *Spectrum of their applicability*, targeting to specific domains without providing a generic applicability.
- *Software dependency*, developing an XML file, used as input to a PN tool, with tailored syntax that only complies with a specific version of the selected tool. This syntax may be incomplete/wrong after a version update and hence the PN model is considered inaccurate.

Therefore, the **research motivations**, drawn from the literature review for the automated generation of a Petri Net model, should cover existing literature gaps and extend current techniques that automatically generate reliability models by:

- Retrieving *fully automatically* without the user intervention the topology information from the graphical diagram, i.e. the UML/SysML AD, of the system description.
- The automated PN model generation methodology should be able to handle *large systems/processes with control loops*.
- The proposed method requires to provide a *generic applicability*, enabling the generation of PN models from any system/process being modelled using an AD. This means that all the elements included in an AD should be considered.
- The proposed methodology for the PN model generation should be *software independent*, by introducing a novel code without being based upon the syntax of any industrial software which can be easily modified after a version update, and hence to fail the desired model generation.

3 Methodology for the Automated Generation of a Petri Net Model

3.1 Introduction

This chapter describes the methodology followed for automated Petri Net model generation. The steps required to automate a PN model are identified at first and then the methodology to undertake these steps, **input-system modelling**; and **algorithm-Java database programming** using Structured Query Language (SQL), is explained in detail. Following the algorithm developed, the **mathematical representation of a PN model**, i.e. the transpose of the incidence matrix and initial marking, is automatically generated. Additionally, the **graphical representation of a PN model** is automatically generated, using data obtained during the Java database programming step. The proposed methodology is explained with the help of a generic example that applies both to systems and processes, but for simplification purposes, the term systems is used to cover processes as well. It is also noted that systems consist of components, whereas processes of activities. The layout of this chapter is illustrated in Figure 3.1.

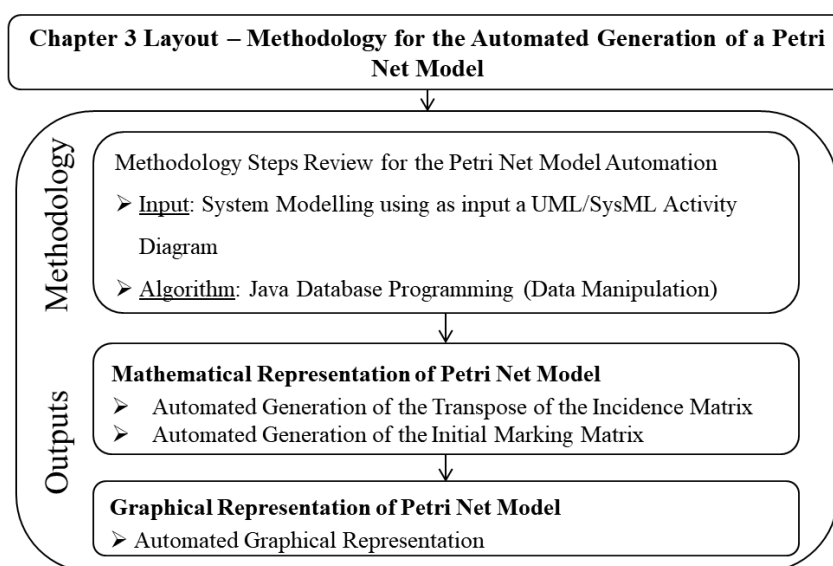


Figure 3.1 Illustration of the Structure of Chapter 3

3.2 Overview of Developed Methodology

This section describes the procedure followed for the automated PN model generation using as an input a commonly used description diagram of industrial systems, the UML/SysML Activity Diagram. Figure 3.2 illustrates a diagram outlining the methodology.

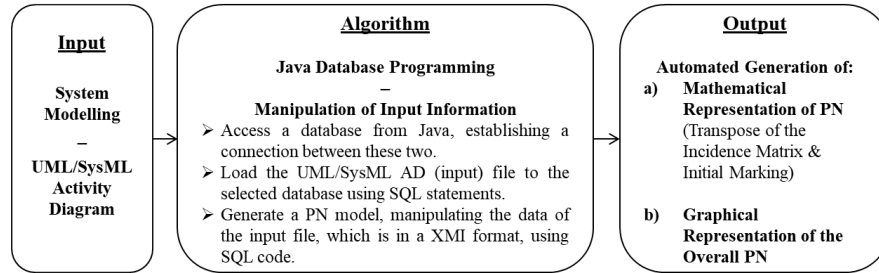


Figure 3.2 Methodology Steps for Automated PN Generation

Thus, the automation procedure takes as a starting point (input) a UML/SysML Activity Diagram and applying the algorithm developed for the manipulation of this input information, the mathematical and graphical representations of a PN model are generated (outputs). Each of these methodology steps is briefly explained:

- **Input (System Modelling):** The input to the proposed algorithm is the UML/SysML AD file created for a given system.
- **Algorithm (Java Database Programming):** The developed algorithm initially *establishes a connection to an SQL-based database with Java*, using Java Database Connectivity (JDBC). The SQL-based databases are able to capture and analyse data by organising it in an easy way to be accessed, managed and updated. Thus, once JDBC establishes connectivity, *the UML/SysML AD (input file) is loaded into the database* by executing SQL statements. This loaded file is in an XML Metadata Interchange (XMI) format. XMI, a specific application of XML, is an Object Management Group (OMG) standard for exchanging metadata information via Extensible Markup Language (XML). The *data within this XMI file*, obtained from the input AD, *is stored in tables in the selected database and an SQL code is developed* to manipulate and organise this data into a matrix form, similar to that used to describe the mathematical representation of PN models.

- **Output (PN model generation):** The output of the developed algorithm is the *transpose of the incidence matrix* and the *initial marking matrix*, described in equation 2.2. Additionally, the PN graphical representation is developed. The output PN model can be used for: (i) verification purposes, as it can prove the correctness of UML/SysML ADs, which have an informal syntax and semantics; and (ii) simulation purposes, as it can assess the performance of the initial system/process by predicting the average execution time of the various paths of which the system/process consists, the most common visited places in each path, as well as the paths resulting in the most failures and the nodes most involved in the route to failure; and make recommendations in order to enhance its efficiency. In addition to the PN mathematical form, the data required for the PN simulation is timed and probabilistic data, which correspond to the PN transitions. This data, provided by industry in an Excel file, is automatically retrieved in MATLAB where the simulation is carried out.

The methodology is explained in detail in the following sections.

3.3 Input – System Modelling

The system modelling tools focus on the graphical representation of a system and the retrieval of information for further analysis. This is the starting point of the automation process and it is assumed that this model is available, either created by software engineers or provided by industry. According to the modelling tools reviewed in section 2.2, it was concluded that the UML/SysML AD satisfies all the criteria, and hence is selected for this study.

The input used for the automated PN generation is described here using the Activity Diagram for a simple scenario, illustrated in Figure 3.3. This diagram consists of the most commonly used elements in system modelling, including an initial node, an activity final node, shown as ‘pin’ and ‘pout’ respectively, 4 opaque action nodes ‘Action_1’, ‘Action_2’, ‘Action_3’ and ‘Action_4’, a decision node named ‘Decision_1’, and a merge node, named ‘Merge_1’.

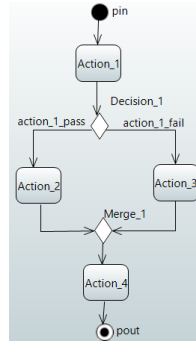


Figure 3.3 AD for a Simple Process

Once the validation of the diagram is successful, using the ‘Validate model’ option available in the Eclipse software, then its XMI format can be automatically loaded into a database system for further manipulation. The model validation considers possible errors found in an AD due to disconnected objects or errors regarding the connection between nodes and edges, for instance Input Pin nodes should be followed by object flow edges. The XMI file includes two necessary elements for the automated generation of a PN model, the nodes (<node .../>) and edges (<edge .../>). The XMI nodes are derived from the initial and activity final nodes, the opaque action nodes and the control nodes such as merge or decision nodes. Similarly, the XMI edges are derived from the control flow edges (shown as arcs in ADs).

The XMI file for the AD in Figure 3.3 can be found in Appendix A. This file consists of XMI nodes such as the ‘pin’, ‘Action_1’, ‘Decision_1’, ‘pout’, etc. and XMI edges such as the ‘action_1_pass’, ‘action_1_fail’, etc. A part of this XMI file, presented in Figure 3.4, has been used to describe the syntax of XMI nodes and edges.

<pre> <edge xmi:type="uml:ControlFlow" xmi:id="_hjulINaQEee330p70iFb5A" name="action_1_pass" target="_2ctboNaPEeeXUKMyPHN3Zw" source="_b330gNaQEee330p70iFb5A"/> <edge xmi:type="uml:ControlFlow" xmi:id="_iV_IgNaQEee330p70iFb5A" name="action_1_fail" target="_XoMwINaQEee330p70iFb5A" source="_b330gNaQEee330p70iFb5A"/> </pre>	} edges
<pre> <node xmi:type="uml:OpaqueAction" xmi:id="_2ctboNaPEeeXUKMyPHN3Zw" name="Action_2" incoming="_hjulINaQEee330p70iFb5A" outgoing="_jNf_UNaQEee330p70iFb5A"/> <node xmi:type="uml:DecisionNode" xmi:id="_b330gNaQEee330p70iFb5A" name="Decision_1" incoming="_gyJXMNaQEee330p70iFb5A" outgoing="_hjulINaQEee330p70iFb5A _iV_IgNaQEee330p70iFb5A"/> </pre>	} nodes

Figure 3.4 Part of XMI File retrieved from the AD for the Simple Scenario

Each XMI edge element consists of the following attributes:

- A “xmi:type” that corresponds to the edge used in the AD.
- A “xmi:id” that acts as a unique identifier for each element.

- A “name” as presented in the AD. If the edge does not have a name, then the “name” attribute is omitted.
- A “target” that corresponds to the node id attribute in which the edge ends up.
- A “source” that corresponds to the node id attribute from which the edge starts.

For example, the edge ‘action_1_pass’ in Figure 3.3 has: `xmi:type="uml:ControlFlow"`, `xmi:id="_hjullNaQEee33Op70iFb5A"`, `name="action_1_pass"`, `target="_2ctboNaPEee XUKMyPHN3Zw"` that corresponds to the xmi:id of ‘Action_2’ and `source="_b33OgNaQEee33Op70iFb5A"` that corresponds to the xmi:id of ‘Decision_1’.

Similarly, each XMI node element consists of the following attributes:

- A “xmi:type” that corresponds to the node used in the diagram.
- A “xmi:id” that acts as a unique identifier for each element.
- A “name” as presented in the AD.
- An “incoming” that corresponds to the edge id attribute that enters the node.
- An “outgoing” that corresponds to the edge id attribute that leaves the node.

For example, the node ‘Decision_1’ in Figure 3.4 has: `xmi:type="uml:DecisionNode"`, `xmi:id="_b33OgNaQEee33Op70iFb5A"`, `name="Decision_1"`, `incoming="_gyJXMNa QEee33Op70iFb5A"` which corresponds to the xmi:id of the edge that enters the node and `outgoing="_hjullNaQEee33Op70iFb5A_iV_IgNaQEee33Op70iFb5A"` which corresponds to the xmi:id of ‘action_1_pass’ and ‘action_1_fail’.

The XMI file obtained from the AD for the simple process in Figure 3.3 consists of 16 elements, 8 edges and 8 nodes. The 8 nodes found in the XMI document include, 1 initial node, 1 activity final node, 4 opaque action nodes, 1 decision node, and 1 merge node.

3.4 Algorithm – Java Database Programming

3.4.1 Transformation Rules

It is important to define the transformation rules used for the translation of elements used in UML/SysML Activity Diagrams to Petri Net models. Therefore, before explaining the algorithm (database modelling) followed for the automated PN model generation these rules are introduced. This section, defines the mapping rules for the most commonly used Activity Diagram elements such as opaque action, decision, merge, initial and activity final nodes and control flow edges. Therefore, for the AD, shown in Figure 3.3, a PN has been manually developed and presented in Figure 3.5, in order to help with the identification of these transformation rules.

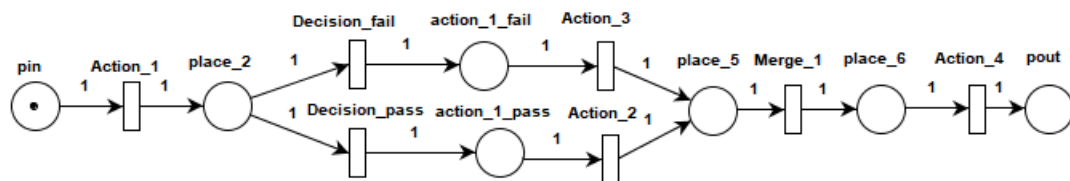


Figure 3.5 Model developed for the AD for the Simple Process

The PN model in Figure 3.5 consists of 7 transitions and 7 places. From the comparison of the manually developed PN and the given AD for the simple process, the relationships between the AD and PN notation and symbols are shown in Table 3.1.

Table 3.1 Relationships between the AD and PN Notation and Symbols

UML 2.0 Activity Diagram		Petri Net	
Name	Symbol	Name	Symbol
Control Flow Edge		Place	
Opaque Action Node		Transition	
Decision Node		PN Structure	
Merge Node		Place & Transition	
Initial Node		Place	
Activity Final Node		Place	
Final Flow Node		Place	

As can be seen, the edges of the AD are transformed into PN places, whereas the opaque action nodes are mapped into PN transitions. The decision and merge nodes have a certain sequence of PN places and transitions, as seen from Table 3.1. In the

PN structure, which is created by an AD decision node, the two transitions used correspond to the two outgoing edges from the AD decision node. Thus, in this transformation, the number of PN transitions should equal the number of the outgoing edges from the AD decision node. Hence, a PN structure developed by an AD decision node represents the branching of two or more PN edges, represented by two or more transitions, each followed by a PN place, as seen in Table 3.1. In the Place & Transition, which is created by an AD merge node, the PN developed represents the merging of two or more PN edges to a place which is followed by a transition. The initial node followed by an outgoing edge and the activity final/final flow node accepting an incoming edge are both transformed into PN places. Therefore, these are the mapping rules that were followed in the database modelling step of the methodology.

3.4.2 Database Introduction

A versatile development in the field of software engineering is the database concept that over the last 30 years has been used widely in industry (Connolly & Begg, 2005). The main idea of the database is to capture and analyse a collection of data by organising it in an easy way to be accessed, managed and updated. Databases can be categorised into relational and non-relational. The main difference between relational and non-relational databases is that the former stores the data in a tabular form, whereas the latter stores it as files.

A Database Management System (DBMS) is a software package that captures and analyses data, interacting with end-users and other applications, and the database itself. An extension of DBMS is the Relational Database Management System (RDBMS) that uses the relational model and hence allows the row-based table structure that connects related data elements to one another. Hence, the RDBMS supports a tabular structure for the data with enforced relations between the tables (Codd, 1970). Moreover, each row in a RDBMS table contains a unique value and each column lists values from the same domain, for instance, a column named address includes only addresses.

3.4.2.1 Relational Database Management Systems Products Review

Most Relational Database Management Systems (RDBMS) use the Structured Query Language (SQL), which is a computer language for storing, manipulating and

retrieving data stored in a RDBMS. SQL allows the user to link information from different tables using foreign keys/indexes, in order to identify uniquely stored data within the table. Other tables may refer to that foreign key/index, creating a link between the data. Hence, relational databases using SQL are good for applications involving connections between data in different tables. The research described in this work requires various transactions and hence, relational databases tools will be applied.

The most popular Relational Database Management Systems (RDBMS) products according to DB-Engines (2018) are:

- **IBM DB2** was released in 1983 by IBM.
- **Oracle** was introduced by Relational Software in 1977 was the first commercially available SQL-based RDBMS.
- **Microsoft SQL Server** is first released in 1998.
- **MySQL** (Michael Widenius Structured Query Language), owned by Oracle Corporation in 2009, was first released in 1996 by Ulf Michael Widenius and David Axmark.
- **PostgreSQL** was officially released in 1996 by PostgreSQL Global Development Group.

According to a review conducted for RDBMS, **IBM DB2** and **Oracle** are not open-source products, restricting their usage. Additionally, **Microsoft SQL Server**, although providing high flexibility, has a huge licensing cost and limited compatibility to run on non-Windows platforms (Microsoft, 2015). Hence, these three RDBMS have been singled out and are deemed not appropriate due to their aforementioned limitations.

MySQL and **PostgreSQL** are both open source relational systems. They are both row-oriented, general-purpose relational databases with many common characteristics. More specifically, **MySQL** is a powerful RDBMS with high-performance and scalability that uses the SQL data language and can work very efficiently with large data sets and with many languages, including C, C++, JAVA, PHP, etc. Its popularity has increased over the last few years and more specifically since 2010 when several Windows specific features and improvements were added enhancing its performance

and scalability (DB-Engines, 2018). Regarding **PostgreSQL**, this database offers various dynamic characteristics and functionalities (PostgreSQL, 2018), providing robustness, security and advanced features such as high reliability, data integrity, data analysis of complex systems, high speed and simple set ups (Riggs & Krosing, 2010). However, according to the RDBMS review, **MySQL** has been found to be more powerful than **PostgreSQL**, providing higher speed, dynamic characteristics and ability to create new projects quickly, and hence it has been selected for this work. The software that is used for the database development is **MySQL Workbench**, a visual database design tool suitable for SQL development, data modelling, server administration and data migration.

3.4.3 Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix

The data manipulation (representative of the AD) and organisation for the PN construction is carried out by generating an SQL code using the MySQL database. In order to achieve faster execution of complex SQL querying logic, i.e. SQL statements that return data, and avoid the user's intervention once SQL stored procedures are executed, the MySQL database has been accessed from Java via JDBC Application Programming Interface (API), used for database independent connectivity between Java programming language and databases. Using JDBC, SQL statements can be sent to any relational database. A stored procedure, mentioned earlier, is a set of SQL statements that has been created and stored in the dataset to perform a task. Although SQL stored procedures can be executed in MySQL, it is possible they require user intervention in order to input manually the output of a stored procedure to the SQL code. Therefore, for an SQL stored procedure in a Java application, a string is created outside the stored procedure and then this string is passed as one variable containing the complete SQL statement, without user's intervention.

The steps followed to connect the Java Programming language with the MySQL database in Java using JDBC and execute the SQL statements are:

1. **Register the MySQL JDBC driver.** This driver is a component that enables a Java application such as JDBC API to communicate with the MySQL database.

2. **Open a new connection.** In this step a new connection with the MySQL database is established, using the `getConnection()` method.
3. **Execute SQL queries.** In this step, SQL statements are created to build and submit SQL queries to the database in order to generate the mathematical representation of a PN model.
4. **Extract data from result-sets.** Having executed the SQL queries in step 3, the result-set objects are used in this step to return/present the results of these SQL queries in Java. This step is executed using the `ResultSet.get()` method.
5. **Close the connection.** In this final step, the connection with the MySQL database (set up in step 2), the statements (from step 3) and result-sets (from step 4) are terminated, using the corresponding `.close()` methods.

The SQL code developed in step 3 of the aforementioned list, has been used in this work to retrieve, manipulate and store the data included in the input UML/SysML AD file for the automated generation of the mathematical form of a PN model. The input AD file is loaded to the MySQL database using SQL statements. This file has an XMI format, and hence its data can be manipulated and appropriately stored in tables and, by extension, to generate the desired mathematical form of PNs. The purpose of each step followed in this SQL code is introduced in the flowchart illustrated in Figure 3.6.

In this flowchart, these 12 steps can be categorised as follows:

- **Retrieve data (steps 1.a and 1.b):** In this first step, the XMI attributes, such as “xmi:type”, “xmi:id”, “incoming”. “outgoing”, etc. of the AD nodes and edges are stored in two tables respectively.
- **Separate multiple edges (steps 2 - 5):** The values stored in the “incoming” and “outgoing” columns of the table created in the first step (step 1.a) for the AD nodes, named ‘node_xmi’, are scanned. If an AD node has multiple incoming/outgoing edges then multiple values are correspondingly stored in the “incoming”/“outgoing” columns of the ‘node_xmi’. Hence, the multiple values in these columns are separated to enable their further analysis and two tables are created: (i) the table in step 2 if multiple “outgoing” values exist; and (ii) the table in step 3 if multiple “incoming” values exist. Additionally, for the AD nodes that have only one incoming and one outgoing edge a third table in step 4 is created. Hence, the table (in step 4) is created for the nodes

that have both single “incoming” and “outgoing” records in the corresponding columns of the ‘node_xmi’. Finally, these three tables are unified, creating a new table in step 5, the ‘union_node’, in which each cell contains a unique value for any “incoming” and “outgoing” record. This last step creates a fundamental table from which the sequence of AD elements can be found in the following four steps.

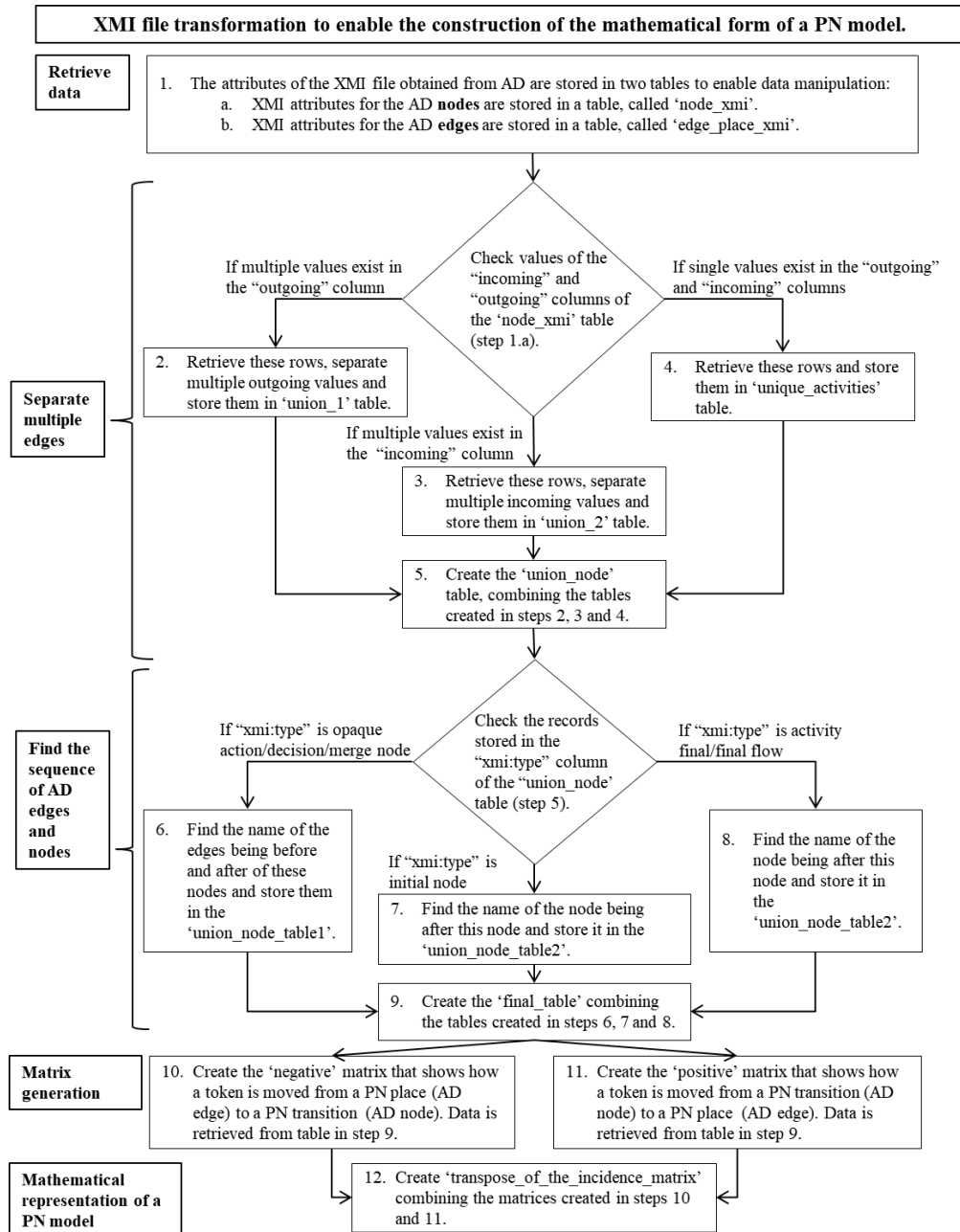


Figure 3.6 Flowchart for the steps followed for the Automated Generation of the Mathematical Representation of a PN Model

- **Find the sequence of AD edges and AD nodes (steps 6 - 9):** The values stored in the “xmi:type” column of the table created in the step 5 are scanned. The code identifies the type of each node and stores them in two tables: (i) one for the opaque action, decision and merge nodes (step 6); and (ii) one for the initial node (step 7), activity final and final flow nodes (step 8). The names of the incoming and outgoing edges to and from the nodes stored in these three tables are identified with the help of the table created in step 1.b and stored in these tables. Hence, each row of the tables created in steps 6, 7 and 8 includes the name of a node, the name of the edge exists before this node (if available) and the name of the edge exists after this node (if available). Finally, the tables obtained from these three steps are unified, creating a new table in step 9.
- **Matrix generation (steps 10 - 11):** Two tables in the form of matrices are created, retrieving the connectivity information from table developed in step 9. Therefore, the matrix in step 10 shows connections of AD edges/initial node/activity final nodes/final flow nodes (PN places) to AD opaque action/decision/merge nodes (PN transitions). Similarly, the matrix generated in step 11 shows connections of AD opaque action/decision/merge nodes (PN transition) to AD edges/initial nodes/activity final nodes/final flow nodes (PN places).
- **Mathematical representation of a PN model (step 12):** In this final step, the PN is generated by combining the two matrices developed in steps 11 and 12 respectively.

The SQL code followed for the automated PN generation, presented in Figure 3.6, can be better explained with the help of the SysML /UML AD, presented in Figure 3.3 for a simple scenario. The mathematical representation of a PN model is generated by applying the following steps:

1. The XMI file obtained from the AD, provided as input of the methodology, is loaded into the MySQL database. Two tables, named ‘node_xmi’ and ‘edge_place_xmi’, are created as follows:

Retrieve data for AD nodes. The ‘node_xmi’ table, part of which is shown in Table 3.2 for three elements from Figure 3.3, is created from the XMI file. The corresponding part of the XMI file used for the

development of Table 3.2 is illustrated in Figure 3.7. It can be seen from Table 3.2 that a column is initially created for each XMI attribute, such as “xmi:type”, “xmi:id”, “name”, etc. and then the values of these attributes are stored in the corresponding columns.

Figure 3.7 Part of XMI File retrieved from the AD

```

<node xmi:type="uml:OpaqueAction" xmi:id="_YsEn8NaQEee33Op70iFb5A"
name="Action_4" incoming="_kLTfwNaQEee33Op70iFb5A"
outgoing="_LRWGYNaQEee33Op70iFb5A"/>
<node xmi:type="uml:MergeNode" xmi:id="_aBN18NaQEee33Op70iFb5A"
name="Merge_1" incoming="_jNf_UNaQEee33Op70iFb5A _j433kNaQEee33Op70iFb5A"
outgoing="_kLTfwNaQEee33Op70iFb5A"/>
<node xmi:type="uml:DecisionNode" xmi:id="_b330gNaQEee33Op70iFb5A"
name="Decision_1" incoming="_gyJXMNaQEee33Op70iFb5A"
outgoing="_hJulINaQEee33Op70iFb5A _iV_IgNaQEee33Op70iFb5A"/>

```

Table 3.2 MySQL ‘node_xmi’ Table Extract

id	xmi:type	xmi:id	name	incoming	outgoing
6	uml:OpaqueAction	_YsEn8NaQEee33Op70iFb5A	Action_4	_kLTfwNaQEee33Op70iFb5A	_LRWGYNaQEee33Op70iFb5A
7	uml:MergeNode	_aBN18NaQEee33Op70iFb5A	Merge_1	_jNf_UNaQEee33Op70iFb5A _j433kNaQEee33Op70iFb5A	_kLTfwNaQEee33Op70iFb5A
8	uml:DecisionNode	_b330gNaQEee33Op70iFb5A	Decision_1	_gyJXMNaQEee33Op70iFb5A	_hJulINaQEee33Op70iFb5A _iV_IgNaQEee33Op70iFb5A

Retrieve data for AD edges. As for the ‘node_xmi’ table, the ‘edge_place_xmi’ table, part of which is presented in Table 3.3 for two elements from Figure 3.3, is created from the XMI file. The corresponding part of the XMI file used for the development of Table 3.3 is illustrated in Figure 3.8. It can be seen from Table 3.3 that a column is initially created for each XMI attribute, such as “xmi:type”, “target”, etc. and then the values of these attributes are stored in the corresponding columns.

Figure 3.8 Part of XMI File retrieved from the AD

```

<edge xmi:type="uml:ControlFlow" xmi:id="_hJulINaQEee33Op70iFb5A"
name="action_1_pass" target="_2ctboNaPEeeXUKMyPHN3Zw"
source="_b330gNaQEee33Op70iFb5A"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_iV_IgNaQEee33Op70iFb5A"
name="action_1_fail" target="_XoMwINaQEee33Op70iFb5A"
source="_b330gNaQEee33Op70iFb5A"/>

```

Table 3.3 MySQL ‘edge_place_xmi’ Table Extract

id	xmi:type	xmi:id	name	source	target
3	uml:ControlFlow	_hJulINaQEee33Op70iFb5A	action_1_pass	_b330gNaQEee33Op70iFb5A	_2ctboNaPEeeXUKMyPHN3Zw
4	uml:ControlFlow	_iV_IgNaQEee33Op70iFb5A	action_1_fail	_b330gNaQEee33Op70iFb5A	_XoMwINaQEee33Op70iFb5A

The “incoming” and “outgoing” columns of ‘node_xmi’ table created in step 1.a, may contain multiple text values, separated by a space if a node has more than one incoming/outgoing edge. For instance, this is observed in the 2nd and 3rd rows of Table 3.2 in the “incoming” and “outgoing” columns respectively. These multiple text data should be separated, since all the cells of tables have to store single text values in

order to allow the identification of these values in other tables and, by extension, the accurate generation of the mathematical representation of PN. It can be seen from Table 3.2 in the “outgoing” column in the 3rd row that the 1st part of the value, “_hjuliNaQEee33Op70iFb5A”, also exists in Table 3.3 in the “xmi:id” column in the 2nd row. If double value, placed in Table 3.2, is not separated, then SQL code is not able to identify that these two cells refer to the same edge. Therefore, outgoing and incoming values are separated and steps 2 and 3 are carried out to achieve this. Step 4 creates a table for the nodes that have single incoming and outgoing edges and then step 5 combines the tables created in steps 2, 3 and 4. Hence, the table created in step 5 stores information about all the nodes included in AD, holding single values in each cell.

Steps 2, 3, 4 and 5 are explained in detail, as follows:

2. **Separate multiple “outgoing” values.** In this step, the values stored in the ‘node_xmi’ table are scanned to determine if there are any multiple values in the “outgoing” column. If so, these are separated so there is one row for each value and these are stored in a table called ‘union_1’. Therefore, for the simple example illustrated in Figure 3.3, Table 3.4 is created, for the decision node, “Decision_1”, whose outgoing value is more than one, as can be seen from the “outgoing” column of Table 3.2 (3rd row). Two rows are created for the decision node in Table 3.4, showing that this node has two outgoing arcs.

Table 3.4 MySQL ‘union_1’ Table Extract

id	outgoing	xmi:type	xmi:id	name	incoming
1	_iV_IgNaQEee33Op70iFb5A	uml:DecisionNode	_b33OgNaQEee33Op70iFb5A	Decision_pass	_gyJXMNaQEee33Op70iFb5A
8	_hjuliNaQEee33Op70iFb5A	uml:DecisionNode	_b33OgNaQEee33Op70iFb5A	Decision_fail	_gyJXMNaQEee33Op70iFb5A

3. **Separate multiple “incoming” values.** In this step, similar code to step 2 was developed for the values in the “incoming” column of ‘node_xmi’ table. Therefore, if there are multiple values in this column, they are separated and stored in different rows in a table called ‘union_2’.
4. **Store single “incoming” and “outgoing” values.** This step creates a ‘unique_activities’ table retrieving the rows from the ‘node_xmi’ table from step 1.a that only have one incoming and one outgoing record in the corresponding columns. For the AD presented in Figure 3.3, the ‘unique_activities’ table is presented in Table 3.5.

Table 3.5 MySQL 'unique_activities' Table

id	outgoing	xmi:type	xmi:id	name	incoming
1	_f-3gINaQEee33Op70IFb5A	uml:InitialNode	_xBDrQNaPEeeXUKMyPHN3Zw	pin	NULL
2	NULL	uml:ActivityFinalNode	_yMFT8NaPEeeXUKMyPHN3Zw	pout	_JRWGYNaQEee33Op70IFb5A
3	_gyJXMNaQEee33Op70IFb5A	uml:OpaqueAction	_0KksNaPEeeXUKMyPHN3Zw	Action_1	_f-3gINaQEee33Op70IFb5A
4	_jNf_UNaQEee33Op70IFb5A	uml:OpaqueAction	_2ctboNaPEeeXUKMyPHN3Zw	Action_2	_hJulINaQEee33Op70IFb5A
5	_j433kNaQEee33Op70IFb5A	uml:OpaqueAction	_XoMwINaQEee33Op70IFb5A	Action_3	_jV_IgNaQEee33Op70IFb5A
6	_JRWGYNaQEee33Op70IFb5A	uml:OpaqueAction	_YsEn8NaQEee33Op70IFb5A	Action_4	_kTfwNaQEee33Op70IFb5A

5. **Create a table in which single values are stored in each cell.** This step creates a 'union_node' table, unifying the tables created in steps 3, 4 and 5, i.e. the 'union_1', 'union2' and 'unique-activities' tables. For the AD presented in Figure 3.3, the 'union_node' table, presented in Table 3.6, is created unifying tables created in steps 2,3 and 4. At this point Table 3.6 includes all the information of nodes, included in the AD, and additionally each node has unique incoming and outgoing arcs.

Table 3.6 MySQL 'union_node' Table

id	outgoing	xmi:type	xmi:id	name	incoming
1	_jV_IgNaQEee33Op70IFb5A	uml:DecisionNode	_b33OgNaQEee33Op70IFb5A	Decision_pass	_gyJXMNaQEee33Op70IFb5A
2	_hJulINaQEee33Op70IFb5A	uml:DecisionNode	_b33OgNaQEee33Op70IFb5A	Decision_fail	_gyJXMNaQEee33Op70IFb5A
3	_kTfwNaQEee33Op70IFb5A	uml:MergeNode	_aBN18NaQEee33Op70IFb5A	Merge_1	_jNf_UNaQEee33Op70IFb5A
4	_kTfwNaQEee33Op70IFb5A	uml:MergeNode	_aBN18NaQEee33Op70IFb5A	Merge_1	_j433kNaQEee33Op70IFb5A
5	_f-3gINaQEee33Op70IFb5A	uml:InitialNode	_xBDrQNaPEeeXUKMyPHN3Zw	pin	NULL
6	NULL	uml:ActivityFinalNode	_yMFT8NaPEeeXUKMyPHN3Zw	pout	_JRWGYNaQEee33Op70IFb5A
7	_gyJXMNaQEee33Op70IFb5A	uml:OpaqueAction	_0KksNaPEeeXUKMyPHN3Zw	Action_1	_f-3gINaQEee33Op70IFb5A
8	_jNf_UNaQEee33Op70IFb5A	uml:OpaqueAction	_2ctboNaPEeeXUKMyPHN3Zw	Action_2	_hJulINaQEee33Op70IFb5A
9	_j433kNaQEee33Op70IFb5A	uml:OpaqueAction	_XoMwINaQEee33Op70IFb5A	Action_3	_jV_IgNaQEee33Op70IFb5A
10	_JRWGYNaQEee33Op70IFb5A	uml:OpaqueAction	_YsEn8NaQEee33Op70IFb5A	Action_4	_kTfwNaQEee33Op70IFb5A

The next four steps generate a table that shows the sequence of AD edges and AD nodes. Thus, step 6 creates a table storing all the opaque action, decision and merge nodes that exist in the AD, including their pre-edges, i.e. incoming edge(s), and post-edges, i.e. outgoing edge(s). Steps 7 and 8 generate similar tables for the initial and final nodes of the AD, respectively. Finally, the tables created in steps 6, 7 and 8 are unified in step 9. Steps 6 - 9 are explained in detail, as follows:

6. **Identify the sequence between the AD opaque/decision/merge nodes and their preceding and following edges.** This step initially creates a table, named 'union_node_table1', retrieving the rows from the 'union_node' table (from step 5) where their "xmi:type" is equal to 'uml:OpaqueAction' or 'uml:DecisionNode' or 'uml:MergeNode'. Two null (empty) new columns are added in this table, called "place_before_node" and "place_after_node". The 'union_node_table1' is then updated inserting to the "place_after_node"

column the records from the “name” column of the ‘edge_place_xmi’ table where the “outgoing” values (from the ‘union_node_table1’) are equal to the “xmi:id” values from the ‘edge_place_xmi’ table. The ‘union_node_table1’ is updated again inserting in the “place_before_node” column the records from the “name” column of the ‘edge_place_xmi’ table where the “incoming” values (from the ‘union_node_table1’) are equal to the “xmi:id” values from the ‘edge_place_xmi’ table. Therefore, for the AD presented in Figure 3.3, the ‘union-node_table1’ is created retrieving the data from Table 3.6 and part of it for the decision node is illustrated in Table 3.7.

Table 3.7 MySQL ‘union_node_table1’ Table Extract

place_before_node	place_after_node	id	xmi:type	xmi:id_primary	name_primary	incoming	outgoing
place_2	action_1_fail	1	uml:DecisionNode	_b330gNaQEee33Op70Fb5A	Decision_pass	_gyJX4NaQEee33Op70Fb5A	_jV_IgNaQEee33Op70Fb5A
place_2	action_1_pass	2	uml:DecisionNode	_b330gNaQEee33Op70Fb5A	Decision_fail	_gyJX4NaQEee33Op70Fb5A	_hJulINaQEee33Op70Fb5A

7. **Identify the sequence between the AD initial nodes and their following nodes.** This step creates a table called ‘union_node_table2’, consisting of three columns, the “place_before_node”, “name_primary” and “place_after_node”. This table initially stores to the “place_before_node” column the records from the “name” column of the ‘union_node’ table (from step 5) where their “xmi:type” is equal to ‘uml:InitialNode’. The table is then updated storing to the “name_primary” column the record from the “name” column of the ‘union_node’ table where the “outgoing” value of the examined initial node is equal to the value stored in the “incoming” column of the same table (step 5). For the initial node of the example in Figure 3.3, the ‘union_node_table2’, shown in Table 3.8, is created retrieving the “name_primary” and “place_after_node” values, which will be used in the next steps, from Table 3.6.

Table 3.8 MySQL ‘union_node_table2’ Table

place_before_node	name_primary	place_after_node
pin	Action_1	NULL

8. **Identify the sequence between the AD final nodes and their preceding nodes.** This step initially stores to the “place_after_node” column of the ‘union_node_table2’, created in step 7, the records from the “name” column of the ‘union_node’ table (from step 5) where their “xmi:type” is equal to ‘uml:ActivityFinalNode’ or ‘uml:FlowFinalNode’. The table is then updated storing to the “name_primary” column the record from the “name” column of

the ‘union_node’ table where the “incoming” value of the examined final node is equal to the value stored in the “outgoing” column of the same table (step 5). For the final node of the example in Figure 3.3, the ‘union_node_table2’ shown in Table 3.9, is created retrieving the “name_primary” and “place_after_node” values, which will be used in the next steps, from Table 3.6.

Table 3.9 MySQL ‘union_node_table2’ Table Extract

place_before_node	name_primary	place_after_node
NULL	Action_4	pout

9. **Identify the sequence of nodes and edges in an AD.** This step creates a table named ‘final_table’ selecting the “place_before_node”, “name_primary” and “place_after_node” columns from the table developed in steps 6, 7 and 8. For the simple example in Figure 3.3, examined in this section, Table 3.10 is created, retrieving data from Tables 3.7 and 3.8 and 3.9.

Table 3.10 MySQL ‘final_table’ Table

place_before_node	name_primary	place_after_node
pin	Action_1	place_2
place_2	Decision_pass	action_1_pass
place_2	Decision_fail	action_1_fail
action_1_pass	Action_2	place_5
action_1_fail	Action_3	place_5
place_5	Merge_1	place_6
place_6	Action_4	pout

The ‘final_table’ shows the sequence of nodes and edges. The “name_primary” column includes the values for the opaque action, decision and merge nodes of the AD in Figure 3.3, whereas the “place_before_node” and “place_after_node” columns include all the edges placed before and after each node. The 1st and 3rd columns of the table created in step 9 correspond to PN places, whereas the 2nd column contains all the PN transitions.

The following three steps, 10, 11 and 12, describe the generation of the mathematical representation of PN model, retrieving the information from the table created in step 9.

10. **Create a matrix that shows how a token is removed from each of its pre-places, when an enabled transition fires (shows the connection from PN places to PN transitions).** This step creates a matrix, named ‘negative’, with the columns defined by the records retrieved from the “name_primary” column of the ‘final_table’ table and the rows defined by the records retrieved

from the “place_before_node” column of the same table. If a “name_primary” record and a “place_before_node” record are in the same row in the ‘final_table’ table, then the value ‘-1’ should be put in the corresponding matrix cell otherwise a ‘0’ is inserted. Table 3.11 shows the ‘negative’ matrix created for the AD in Figure 3.3, using Table 3.10.

Table 3.11 MySQL ‘negative’ Table

place_before_node	Decision_fail	Decision_pass	Action_2	Action_1	Action_3	Merge_1	Action_4
action_1_fail	0	0	0	0	-1	0	0
action_1_pass	0	0	-1	0	0	0	0
pin	0	0	0	-1	0	0	0
place_2	-1	-1	0	0	0	0	0
place_5	0	0	0	0	0	-1	0
place_6	0	0	0	0	0	0	-1

11. **Create a matrix that shows how a token is inserted to each of its PN post-places, when an enabled transition fires (shows the connection from PN transitions to PN places).** This step creates a second matrix, named ‘positive’, with the columns defined by the records stored in the “name_primary” column of the ‘final_table’ table and the rows defined by the entries in the “place_after_node” column of the same table. If a “name_primary” record and a “place_after_node” record are in the same row in the ‘final_table’ table, then the value ‘1’ should be put in the corresponding matrix cell otherwise ‘0’ is inserted. Table 3.12 shows the ‘positive’ matrix created for the AD in Figure 3.3, using Table 3.10.

Table 3.12 MySQL ‘positive’ Table

place_after_node	Decision_fail	Decision_pass	Action_2	Action_1	Action_3	Merge_1	Action_4
action_1_fail	1	0	0	0	0	0	0
action_1_pass	0	1	0	0	0	0	0
palce_5	0	0	1	0	1	0	0
place_2	0	0	0	1	0	0	0
pout	0	0	0	0	0	0	1
place_6	0	0	0	0	0	1	0

Generate the mathematical form of PN model. This step creates the transpose of the incidence matrix of the PN model by adding the ‘negative’ and ‘positive’ matrices created in steps 10 and 11. This matrix expresses the connectivity between the places and the transitions of PN models defining the movement of tokens through the net. Table 3.13 shows the combination of the matrices shown in Tables 3.11 and 3.12. This is the mathematical

representation of the Petri net for the AD in Figure 3.3 in the form of the transpose of the incidence matrix

Table 3.13 MySQL 'transpose_of_the_incidence_matrix' Table

place_before_node	Decision_fail	Decision_pass	Action_2	Action_1	Action_3	Merge_1	Action_4
action_1_fail	1	0	0	0	-1	0	0
action_1_pass	0	1	-1	0	0	0	0
pin	0	0	0	-1	0	0	0
place_2	-1	-1	0	1	0	0	0
place_5	0	0	1	0	1	-1	0
place_6	0	0	0	0	0	1	-1
pout	0	0	0	0	0	0	1

The code, developed for the automated generation of the transpose of the incidence matrix of a PN model is found in Appendix B.

In order to check the correctness of the matrix in Table 3.13, the transpose of the incidence matrix for the PN model illustrated in Figure 3.5 has been manually constructed in equation 3.1, showing the sequence of PN places and transitions in Figure 3.5.

$$A^T = \begin{matrix} & \text{Dec ..._pass} & \text{Dec ..._fail} & \text{Merge_1} & \text{Action_1} & \text{Action_2} & \text{Action_3} & \text{Action_4} \\ \begin{matrix} \text{place_2} \\ \text{place_5} \\ \text{place_6} \\ \text{pout} \\ \text{pin} \\ \text{action_1_fail} \\ \text{action_1_pass} \end{matrix} & \begin{bmatrix} -1 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 \end{bmatrix} \end{matrix} \quad (3.1)$$

The transpose of the incidence matrix consists of 7 columns (PN transitions) and 7 rows (PN places). The transpose of the PN incidence matrix in Table 3.13 is identical with the matrix developed manually for the simple process, shown in equation 3.1, showing the models correlation.

Therefore, since this representation is created from the system information, the automated generation of the PN has been successful. Using this matrix, a graphical PN representation developed and also the system can be simulated if desired.

3.4.4 Algorithm – Java Database Programming – Petri Net Initial Marking Matrix

The mathematical form of a PN model is considered to be completed with the generation of its initial marking matrix, after obtaining the transpose of the incidence

matrix of this model. Therefore, as for the transpose of the incidence matrix in section 3.4.3, a code is also developed to execute SQL queries for the generation of the PN initial marking. The initial marking matrix shows the number of token(s) included in each PN place before any transition is enabled.

The procedure followed for the PN initial marking has been applied to the simple process shown in Figure 3.3 and the following step, for which an SQL code has been developed in step 3 of the algorithm introduced in section 3.4.3 to execute SQL queries, is completed:

1. **Generate the initial marking of a PN model.** In this step a table named ‘initial_marking’ is created consisting of three columns, the “primary_id” column, the “activity” column in which the places of PN models are listed and the “process_number_of_devices” column, in which the number of tokens, i.e. marking, is placed. Once the table is created, the ‘pin’ value is found in the “activity” column and the corresponding value of the “process_number_of_devices” column replaced by ‘1’, whereas in all the other rows of this column value ‘0’ is inserted. For the AD in Figure 3.3, the ‘initial_marking’ table is obtained and presented in Table 3.14. The records in the “activity” column correspond to the PN places, whereas the records in the “process_number_of_devices” column show that the ‘pin’ place holds one token, i.e. one device, and all the other places hold zero tokens.

Table 3.14 MySQL Database ‘initial_marking Table

primary_id	activity	process_number_of_devices
1	action_1_fail	0
2	action_1_pass	0
3	pin	1
4	place_2	0
5	place_5	0
6	place_6	0
7	pout	0

The initial marking for the PN model shown in Figure 3.5 has been manually developed and illustrated in equation 3.2, showing that place ‘pin’ contains one token, i.e. one device. The matrix of the initial marking in Table 3.14 is identical with the manually developed matrix in equation 3.2 for the simple process, showing the models correlation. At this stage, the ‘pin’ value is always considered as the first place of the PN, which always includes 1 token.

$$M_0 = \begin{matrix} \text{action_1_fail} \\ \text{action_1_fail} \\ \text{pin} \\ \text{place_2} \\ \text{place_5} \\ \text{place_6} \\ \text{pout} \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.2)$$

Later in this thesis, cases that are more complicated have been investigated for the initial marking generation. The code developed for the automated generation of the PN initial marking matrix can be found in Appendix C.

3.5 Automated Graphical Representation of a Petri Net Model

In this section, the graphical representation of the PN model is generated automatically. Since the PN mathematical representation has been automatically generated, the graphical representation of the model is not necessary for its definition. However, the PN graphical representation can be used for the verification of PN model and help users easier understand the underlying model. The PN is automatically generated from the table created in step 9 in section 3.3.3.4 which consists of three columns, the “place_before_node”, “name_primary” and “place_after_node”. The 1st and 3rd columns correspond to the PN places, whereas the 2nd column holds data for the transitions of a PN model. Each row of this table shows the sequence of two places and the transition that is found in-between them. Hence, the data stored in the “primary_name” column follows the data stored in the “place_before_node” column and is followed by the data stored in the “place_after_node” column.

The graphical tool selected to represent the structural information of PN models is the Graph Visualisation (Graphviz), which is an open source graph drawing package developed at American Telephone & Telegraph (AT&T) Labs and firstly released under a Massachusetts Institute of Technology (MIT) license in 1991. This software can take descriptions of graphs in a simple text language, named DOT, and generate undirected or directed graphs. DOT, a description language used for graphs construction, is composed of nodes and edges and allows the hierarchical representation of complex networks and systems. DOT language provides a wide variety of attributes for the *nodes* such as node shape, colour, size, etc., the *edges* such as several styles of the arrowheads at the head/tail of an edge, edge stroke colour and others, and the *graphs* such as graph font family, font size, etc. It also provides

various *node shapes* including boxes, circles, points, diamonds, etc. and *arrowhead types* such as normal (an edge with an arc at the end), dot (an edge with a solid circle at the end), odot (an edge with a hollow circle at the end) and others. Therefore, all these features provided by the DOT are able to graphically represent the PN elements including places, transitions, arcs (normal and inhibitors) and tokens. For this reason, the Graphviz software using the DOT language was found suitable for the PN model graphical representation, and hence it has been taken forward.

For the automated graphical representation of the PN model, the following steps have been applied:

1. The MySQL database has been accessed in Java using SQL statements and all the data stored in the 'final_table', created in step 9 in section 3.4.3, is selected using the corresponding SQL query.
2. The System.out.println() Java statement is used to display messages to the console window in Eclipse software. The steps taken for this part of the code are shown in the flowchart in Figure 3.9. Following the code introduced in step 2.i, the word '**strict**' is used before the word '**digraph**' to avoid the construction of multiple edges between the same pairs of elements.

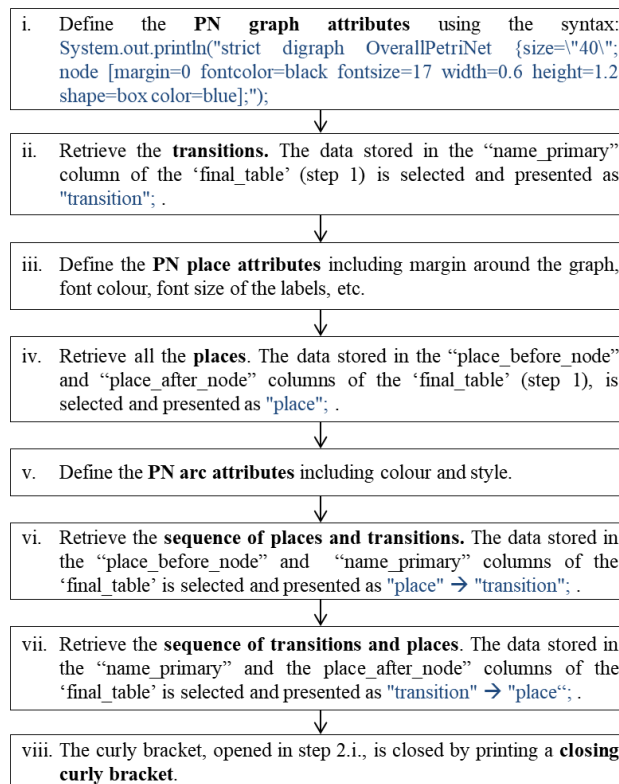


Figure 3.9 Flowchart for Step 2 for the Automated Graphical Representation of a PN Model

Similarly, the **size** of the drawing is defined at 40 mm. Additionally, in this step, the PN transitions attributes such as the **font colour** and **font size** of the labels used for the transitions, as well as the size (**width** and **height**), **shape** and **colour** of the transitions are introduced. The ‘**margin=0**’ used in this statement shows that no space left around the graph.

3. Once the code, found in Appendix D, part A, runs in Eclipse, an output is obtained in the Console window in Eclipse. This output has the form of a DOT file, which can then be imported in the Graphviz software and the corresponding PN model visualised.

The methodology steps explained in this section for the automated graphical representation of a PN model have been applied to the simple example introduced in this chapter. The output file, written in DOT, which has been obtained in the Console window in Eclipse after running the code explained above, is imported in the Graphviz software and the PN model in Figure 3.10 is generated. This PN is identical with the PN model created manually in Figure 3.5, demonstrating the models correlation.

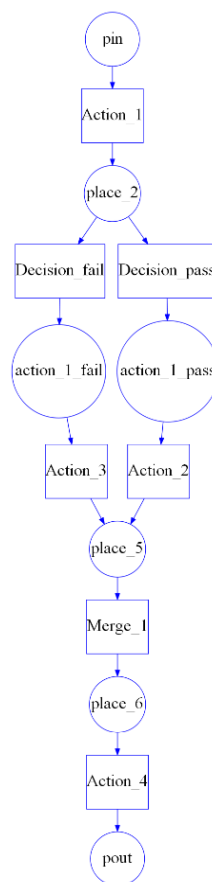


Figure 3.10 Automated Layout of the Petri Net Model for the Simple Example

3.6 Summary

The novel methodology, proposed in this chapter for the automated PN model generation, applying a Java database (MySQL) algorithm, contributes to knowledge through the combination of the following:

- **Java database algorithm:** the method applied integrates the UML/SysML AD (used as input) from system modeling tools and Java database programming where an algorithm is generated for the PN model automation.
- **Fully automated PN model generation capability:** the proposed algorithm retrieves without user intervention the topology information from the graphical diagram, i.e. the UML/SysML AD, of the system description and generates the mathematical and graphical representations of the corresponding PN model.
- **Generic domain applicability:** the proposed methodology does not target specific domains; hence it provides a wide applicability spectrum. This will be further demonstrated in subsequent chapters.
- **Software independence:** The outputs of the proposed methodology, i.e. matrices and PN model, are readily understandable by the user without being based upon the syntax of any industrial software. (Software dependent applications are considered these that generate outputs in XML format that have to be imported to tools to produce either a matrix or a net, which can be meaningful to users.)

In the following chapter, the proposed methodology discussed in this chapter is demonstrated by its applicability to an end of life manufacturing process.

4 Application of the Automated Petri Net Model Generation Methodology to a Recycling IT Asset Process

4.1 Introduction

In this chapter, the methodology introduced in Chapter 3 for the automated generation of a PN model is applied to a recycling IT asset process to demonstrate its applicability and functionality. A description of the process, with the help of the AD developed for the recycling IT asset process, is initially introduced. The AD provided for this process includes all the basic AD elements, and hence it can be used to check the correctness of the developed algorithm introduced in Chapter 3. The mathematical and graphical representations of the PN model for the IT asset recycling process have been manually developed in order to be later compared with those generated automatically to demonstrate their correlation.

4.2 Process Description

An end of life manufacturing process (EOL) is used as an example to illustrate the study. The EOL manufacturing process considered is a recycling IT asset process that focuses on the repair of electronic devices, primarily mobiles phones. Once a mobile device enters the process line, it can end up in one of two states, either refurbished or scrapped. Decisions and actions along the potential paths in the process include seven different possible activities as described below:

- Asset Track (AT): Asset information is introduced into the traceability system. The characteristics of each product such as model device, battery and memory capacity, screen size, etc., are recorded.
- Visual Inspection (VI): The physical condition of each asset is assessed. If the repair or refurbishment of the device is economically viable, it is forwarded to

the Functional Test activity. Otherwise, the device is forwarded to Strip and Scrap.

- Functional Test (FT): The functionality of each product is investigated by conducting the following tests/activities: charger check, battery test, LCD screen check, and ringing test, vibration, microphone and speaker test.
- Data Erasure (DE): Data is erased securely by using specific licensed software.
- Cleaning and De-Labeling (CD): Refurbished products are cleaned properly. Labels are removed and replaced only if considered necessary.
- Repair (R): A product is repaired if its repair is economically viable.
- Strip and Scrap (SS): Failed assets are checked for any useful parts that can be salvaged and recycled to be used in other devices and are then sent for secure destruction.

In the case of a scrapped device, there are two options for it. It can be used either at a unit level, meaning price sought per tonne for scrap, or at a component level, meaning components are extracted from the device and used within this process for future repairs or sold for spares. All activities listed can handle only one device at a time except for Data Erasure that can accept 100 devices simultaneously. Each activity has a time to completion associated with it, which can vary for different devices and product types.

Additionally, activities can have pass and fail probabilities. In practise, most of the activities are carried out at the same physical location, i.e. on the computer. The repair activity (R) however, takes place away from the main refurbishment process but in the same factory, and is not performed until there is a batch, requiring repair. For that reason, there is a delay between preceding activity ending and Repair (R) starting.

An AD of the process that includes all the possible paths of the recycling IT process is illustrated in Figure 4.1. This diagram, that includes all the AD basic elements, has been developed using an open source and most used Java Integrated Development Environment (IDE), Eclipse software, version 4.5 Mars. Eclipse also provides syntax checking, helping users out with writing correct code as well as several extensions, plugins and tools, and hence it has been considered appropriate. The AD in Figure 4.1 consists of nodes and control flow edges. There is an initial node ('pin') which corresponds to the start of the process i.e. where a mobile device enters the system and

an activity final node ('pout') when the process is completed for a device. There are seven opaque action nodes ('Asset_Track', 'Visual_Inspection', etc.), that correspond to the activities carried out through the process, a merge node ('M') used when the output of two activities have a common source node and four decision nodes ('D_VI', 'D_FT', etc.), used when one activity has two target nodes.

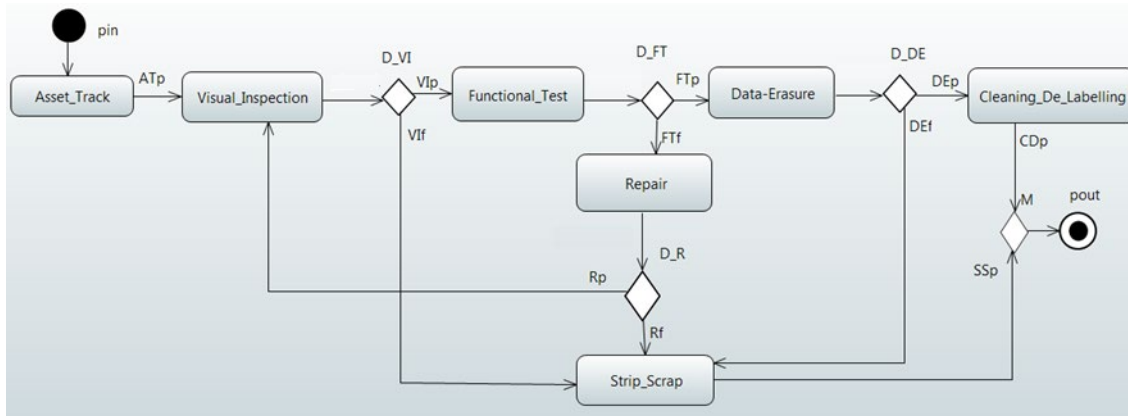


Figure 4.1 UML AD of the Recycling IT Asset Process

It can be seen from the diagram that the control flow edges, which have unique names (i.e. 'ATp', 'VIp', 'VIf', etc.), are used to create links between the nodes showing the main routes through the AD. Table 4.1 explains the abbreviations used in Figure 4.1.

Table 4.1 Abbreviations and Full Names of Nodes and Edges from UML AD

Abbreviation	Full Name
pin	place_in
ATp	Asset_Track_pass
VIp	Visual_Inspection_pass
VIf	Visual_Inspection_fail
FTp	Functional_Test_pass
FTf	Functional_Test_fail
DEp	Data_Erasure_pass
DEf	Data_Erasure_fail
Rp	Repair_pass
Rf	Repair_fail
CDp	Cleaning_De_Labelling_pass
SSp	Strip_Scrap_pass
pout	place_out
D_VI	Decision_Visual_Inspection
D_FT	Decision_Functional_Test
D_DE	Decision_Data_Erasure
D_R	Decision_Repair
M	Merge

4.3 Manual Development of the Petri Net Model for the Recycling IT Asset Process

From the diagram, shown in Figure 4.1 a PN has been developed manually in order to provide information on what the desired outcome of the automation process will be.

Using the transformation rules defined in Chapter 3, in Table 3.1, the PN model is presented in Figure 4.2. The PN model consists of 12 transitions and 17 places. The edges that appear without a name in the AD in Figure 4.1 as for the edge that connects ‘Visual_Inspection’ to ‘D_VI’, they have been transformed into PN places, named place_1 – place_4, as seen in the PN model in Figure 4.2. The transformation rules used for the PN development in Figure 4.2, from the AD in Figure 4.1 are: (i) the AD edges, initial nodes with the outgoing edges and final activity nodes with the incoming edges are transformed into PN places; and (ii) the AD opaque action nodes, decision nodes and merge nodes are transformed into PN transitions.

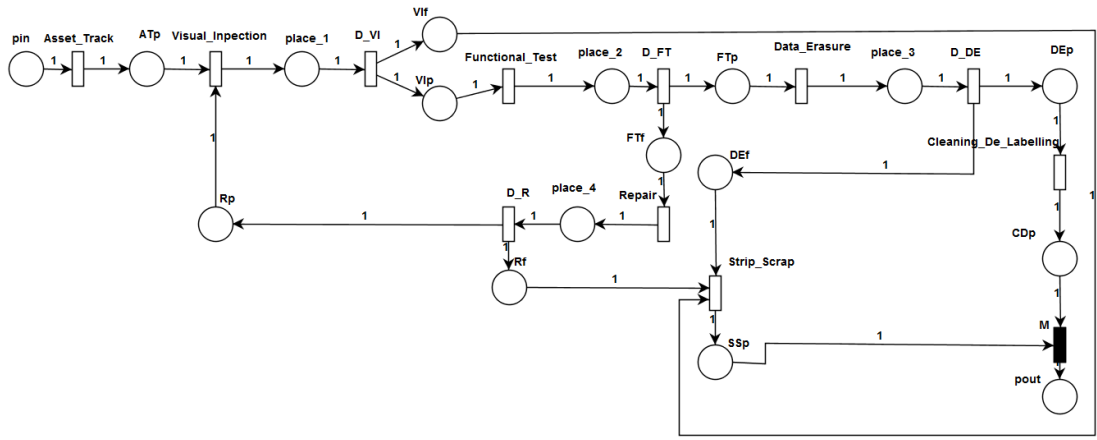


Figure 4.2 PN Model developed for the UML AD for the Recycling IT Asset Process

Additionally, the mathematical representation (transpose of the incidence matrix and initial marking matrix) of the PN in Figure 4.2 is manually developed in order to be compared to the outputs of the algorithm, which is later applied for the automated PN model generation taking as input the AD shown in Figure 4.1. Therefore, the transpose of the incidence matrix of the PN in Figure 4.2 is developed as shown in matrix 4.1.

$$A^T = \begin{matrix} & \begin{matrix} VI & D_VI & D_FT & D_DE & D_R & SS & M & AT & FT & DE & R & CD \end{matrix} \\ \begin{matrix} ATP \\ CDp \\ DEf \\ DEp \\ FTf \\ FTp \\ pin \\ place_1 \\ place_2 \\ place_3 \\ place_4 \\ pout \\ Rf \\ Rp \\ SSp \\ VIf \\ VIp \end{matrix} & \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad (4.1)$$

This matrix defines the movement of tokens through the net, showing the connectivity between the places and the transitions of the PN in Figure 4.2 and hence the sequence of edges and nodes in Figure 4.1. For example, the first row of the matrix describes the movement of one token from place ‘pin’ to place ‘ATp’ through transition ‘Asset_Track’. The transpose of the incidence matrix consists of 12 columns (PN transitions) and 17 rows (PN places). Finally, the matrix representing the initial marking of the net shown in Figure 4.2 is developed, as illustrated in matrix 4.2. The number of tokens has been assumed to be one, which means one phone in process at one time, and hence matrix 4.2 shows that place ‘pin’ contains one token, i.e. one device. Later in this thesis, the existence of multiple tokens/products in the net has been discussed for the initial marking generation.

$$M_0 = \begin{matrix} & \begin{matrix} ATp \\ CDp \\ DEf \\ DEp \\ FTf \\ FTp \\ pin \\ place_1 \\ place_2 \\ place_3 \\ place_4 \\ pout \\ Rf \\ Rp \\ SSP \\ VIf \\ VIp \end{matrix} \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4.2)$$

4.4 Automated Mathematical Representation of the Petri Net Model for the Recycling IT Asset Process

4.4.1 Input – System Modelling

The AD in Figure 4.1 is validated, using the ‘Validate model’ option available in the Eclipse software and then the XMI format of the diagram can be automatically loaded into the MySQL database for further manipulation. The XMI file for the AD in Figure 4.1 can be found in Appendix E. This file consists of XMI nodes such as the ‘pin’, ‘Asset_Track’, ‘D_VI’, ‘Visual_Inspection’, etc. and XMI edges such as the ‘ATp’, ‘VIp’, ‘VIf’, etc. The XMI file obtained from the AD for the recycling IT asset process in Figure 4.1 consists of 16 elements, 17 are edges, and 14 are nodes. The 14

nodes found in the XMI document include, an initial node, an activity final node, seven opaque action nodes, four decision nodes, and a merge node.

4.4.2 Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix

The code, introduced in Chapter 3, section 3.4.3 (found in Appendix B) for the automated generation of the mathematical representation of a PN model, has been applied to generate the transpose of the PN incidence matrix for the recycling IT asset process. This code initially creates a connection between Java programming language and MySQL database via JDBC API and SQL statements are created to build and submit SQL queries. After executing these SQL queries, the mathematical PN model representation of the recycling IT asset process is retrieved from the MySQL database and presented in Java. The steps that the SQL code applies for the automated PN model generation of the recycling IT asset process are as follows:

1. The XMI file, named Activity_Diagram_S2S.uml, provided as input from the first step of the methodology (system modelling), is loaded into the MySQL database. Two tables, named 'node_xmi' and 'edge_place_xmi', are created as follows:
 - a. **Retrieve data for AD nodes:** A part of the 'node_xmi' table for three elements from the AD shown in Figure 4.1 is presented in Table 4.2. The corresponding part of the XMI file from which the values, stored in this table, are retrieved, is illustrated in Figure 4.3.

Figure 4.3 Part of XMI File retrieved from the AD for Table 4.2 Generation

<pre><node xmi:type="uml:InitialNode" xmi:id="_Ftmh0LJTEeaTirLhAX5dxQ" name="pin" outgoing="_pWiwMLJTEeaTirLhAX5dxQ"/></pre>	}	pin
<pre><node xmi:type="uml:OpaqueAction" xmi:id="_InKv8LJTEeaTirLhAX5dxQ" name="Asset_Track" incoming="_pWiwMLJTEeaTirLhAX5dxQ" outgoing="_0ZeSILJTEeaTirLhAX5dxQ"/></pre>		
<pre><node xmi:type="uml:OpaqueAction" xmi:id="_MAZVkJTEeaTirLhAX5dxQ" name="Visual_Inspection" incoming="_0ZeSILJTEeaTirLhAX5dxQ" outgoing="_LLpAkLJUEeaTirLhAX5dxQ" outgoing="_Ao6DcLJUEeaTirLhAX5dxQ" _XOBqQLJtEevDqMum9V7CA"/></pre>	}	Visual_Inspection

Table 4.2 MySQL 'node_xmi' Table Extract

id	xmi:type	xmi:id	name	incoming	outgoing
1	uml:InitialNode	_Ftmh0LJTEeaTirLhAX5dxQ	pin		_pWiwMLJTEeaTirLhAX5dxQ
2	uml:OpaqueAction	_InKv8LJTEeaTirLhAX5dxQ	Asset_Track	_pWiwMLJTEeaTirLhAX5dxQ	_0ZeSILJTEeaTirLhAX5dxQ
3	uml:OpaqueAction	_MAZVkJTEeaTirLhAX5dxQ	Visual_Inspection	_0ZeSILJTEeaTirLhAX5dxQ _LLpAkLJUEeaTirLhAX5dxQ _Ao6DcLJUEeaTirLhAX5dxQ _XOBqQLJtEevDqMum9V7CA	

- b. **Retrieve data for AD edges:** A part of the 'edge_place_xmi' table for three elements from the AD shown in Figure 4.1 is presented in Table

4.3. The corresponding part of the XMI file from which the values, stored in this table, are retrieved, is illustrated in Figure 4.4.

Figure 4.4 Part of XMI File retrieved from the AD for Table 4.3 Generation

```

<edge xmi:type="uml:ControlFlow" xmi:id="_jOjo0LJUEeaTirLhAX5dxQ"
name="Rf" target="_OIQZ8LJTEeaTirLhAX5dxQ"
source="_LtzH4LJTEeaTirLhAX5dxQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_LLpAkLJUEeaTirLhAX5dxQ"
name="Rp" target="_MAZVklJTEeaTirLhAX5dxQ"
source="_LtzH4LJTEeaTirLhAX5dxQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_pJgkULJUEeaTirLhAX5dxQ"
name="SSp" target="_9vls8LJTEeaTirLhAX5dxQ"
source="_OIQZ8LJTEeaTirLhAX5dxQ"/>

```

Table 4.3 MySQL 'edge_place_xmi' Table Extract

id	xmi:type	xmi:id	name	source	target
13	uml:ControlFlow	_jOjo0LJUEeaTirLhAX5dxQ	Rf	_LtzH4LJTEeaTirLhAX5dxQ	_OIQZ8LJTEeaTirLhAX5dxQ
14	uml:ControlFlow	_LLpAkLJUEeaTirLhAX5dxQ	Rp	_LtzH4LJTEeaTirLhAX5dxQ	_MAZVklJTEeaTirLhAX5dxQ
15	uml:ControlFlow	_pJgkULJUEeaTirLhAX5dxQ	SSp	_OIQZ8LJTEeaTirLhAX5dxQ	_9vls8LJTEeaTirLhAX5dxQ

Steps 2 – 5 target the separation of multiple values that exist in the “incoming” and “outgoing” columns of the ‘node_xmi’ table (created in step 1.a) in order to enable data manipulation for the automated generation of the mathematical representation of PN model. Steps 2 – 5 are applied as follows:

2. **Separate multiple “outgoing” values:** The SQL code retrieves and separates the outgoing values from “outgoing” column found in the ‘node_xmi’ table. This new data obtained after the outgoing values separation is stored in a new table, named ‘union_1’, part of which is presented in Table 4.4 for two elements from the AD in Figure 4.1. Therefore, for instance the first two rows of Table 4.4 correspond to the third row of the ‘node_xmi’ table, shown in Table 4.2, which has two outgoing values, i.e. _0ZeSILJTEeaTirLhAX5dxQ _ILpAkLJUEeaTirLhAX5dxQ.

Table 4.4 MySQL ‘union_1’ Table Extract

id	outgoing	xmi:type	xmi:id	name	incoming
3	_Ao6DclJUEeaTirLhAX5dxQ	uml:OpaqueAction	_MAZVklJTEeaTirLhAX5dxQ	Visual_Inspection	_0ZeSILJTEeaTirLhAX5dxQ _ILpAkLJUEeaTirLhAX5dxQ
3	_XOBqQLJTEeaTirLhAX5dxQ	uml:OpaqueAction	_MAZVklJTEeaTirLhAX5dxQ	Visual_Inspection	_0ZeSILJTEeaTirLhAX5dxQ _ILpAkLJUEeaTirLhAX5dxQ
10	_FrRKELJUEeaTirLhAX5dxQ	uml:DecisionNode	_jwxWELJTEeaTirLhAX5dxQ	D_VI	_Ao6DclJUEeaTirLhAX5dxQ _XOBqQLJTEeaTirLhAX5dxQ
10	_Jxp8QLJUEeaTirLhAX5dxQ	uml:DecisionNode	_jwxWELJTEeaTirLhAX5dxQ	D_VI	_Ao6DclJUEeaTirLhAX5dxQ _XOBqQLJTEeaTirLhAX5dxQ

3. **Separate multiple “incoming” values:** Similarly, the code retrieves from the “incoming” column of the ‘node_xmi’ table (step 1.a) the rows that contain multiple values, such as the merge node, named ‘M’ that has two incoming edges, the ‘CDp’ and ‘SSp’, as seen from the AD in Figure 4.1. These values are then separated and stored in different rows in a new table, named ‘union_2’.

4. **Store single “incoming” and “outgoing” values:** The ‘unique-activities’ table is generated storing the records from the ‘node_xmi’ table from step 1.a that have only one incoming and one outgoing record in their corresponding columns. According to the AD in Figure 4.1, the values of the ‘Asset_Track’, ‘Data_Erasure’, ‘Functional_Test’, ‘Repair’ and ‘Cleaning_De_Labelling’ nodes are stored in this table.
5. **Create a table in which single values are stored in each cell:** The tables created in steps 2, 3 and 4 are unified in a new table, named ‘union_node’, part of which for the ‘D_FT’ (Decision Functional Test) node of the AD (Figure 4.1) is presented in Table 4.5

Table 4.5 MySQL ‘union_node’ Table Extract

id	xmi:type	xmi:id_primary	name_primary	incoming	outgoing
5	uml:DecisionNode	_jxsAglJTEeaTirhAX5dxQ	D_FT	_NbtAglJUEeaTirhAX5dxQ	_N9kXclJUEeaTirhAX5dxQ
6	uml:DecisionNode	_jxsAglJTEeaTirhAX5dxQ	D_FT	_NbtAglJUEeaTirhAX5dxQ	_QEscALJUEeaTirhAX5dxQ

Steps 6 – 9 result in the development of a table that shows the sequence of the nodes and edges contained in the AD shown in Figure 4.1 for the recycling IT asset process. The sequence of these AD elements can then be used to identify the sequence of PN places and transitions, i.e. the connectivity between PN elements, which is the mathematical representation of the PN for the IT process. Steps 6 – 9 are applied as follows:

6. **Identify the sequence between the AD opaque/decision/merge nodes and their preceding and following edges:** The ‘union_node_table1’, part of which is presented in Table 4.6 for the ‘D_FT’ of the AD in Figure 4.1, is created retrieving the opaque action/decision/merge nodes from the ‘union_node’ table created in step 5. The names of the AD edges placed before and after of the ‘D_FT’ node are correspondingly stored in the “place_before_node” and “place_after_node” columns. As can be seen from Table 4.6, the ‘D_FT’ node follows edge ‘place_2’ (“place_before_node” column) whereas it is followed by edges ‘FTp’ and ‘FTf’ (“place_after_node” column).

Table 4.6 MySQL ‘union_node_table1’ Table Extract

place_before_node	place_after_node	id	xmi:type	xmi:id_primary	name_primary	incoming	outgoing
place_2	FTp	5	uml:DecisionNode	_jxsAglJTEeaTirhAX5dxQ	D_FT	_NbtAglJUEeaTirhAX5dxQ	_N9kXclJUEeaTirhAX5dxQ
place_2	FTf	6	uml:DecisionNode	_jxsAglJTEeaTirhAX5dxQ	D_FT	_NbtAglJUEeaTirhAX5dxQ	_QEscALJUEeaTirhAX5dxQ

7. **Identify the sequence between the AD initial nodes and their following edges:** A table named 'union_node_table2' is created retrieving the initial node named 'pin' from the 'union_node' table created in step 5 for the AD in Figure 4.1. The name of the node placed after the initial node 'pin' in the AD is then retrieved and stored in the "name_primary" column of 'union_node_table2' as presented in the 1st row of Table 4.7.
8. **Identify the sequence between the AD final nodes and their preceding edges:** Similarly, as for the initial node, the final node of the AD in Figure 4.2, named 'pout', is stored in the 'union_node_table2' (created in step 7). The name of the node placed before the final node 'pout' is then retrieved and stored in the "name_primary" column of 'union_node_table2' as presented in the 2nd row of Table 4.7.

Table 4.7 MySQL 'union_node_table2' Table

place_before_node	name_primary	place_after_node
pin	Asset_Track	NULL
NULL	M	pout

9. **Identify the sequence of nodes and edges in an AD:** The 'final_table', shown in Table 4.8, is created unifying the data from the "place_before_node", "name_primary" and "place_after_node" columns from the tables created in steps 6, 7 and 8.

Table 4.8 MySQL 'final_table' Table

place_before_node	name_primary	place_after_node
ATp	Visual_Inspection	place_1
Rp	Visual_Inspection	place_1
place_1	D_VI	VIp
place_1	D_VI	VIf
place_2	D_FT	FTp
place_2	D_FT	FTf
place_3	D_DE	DEp
place_3	D_DE	DEf
place_4	D_R	Rf
place_4	D_R	Rp
VIf	Strip_Scrap	SSp
DEf	Strip_Scrap	SSp
Rf	Strip_Scrap	SSp
SSp	M	pout
CDp	M	pout
pin	Asset_Track	ATp
VIp	Functional_Test	place_2
FTp	Data_Erasure	place_3
FTf	Repair	place_4
DEp	Cleaning_De_Labeling	CDp

Steps 10 – 12 result in the development of the mathematical representation of PN the model of the recycling IT asset process, retrieving the information from the table created in step 9.

10. **Create a matrix that shows how a token is removed from each of its pre-places, when an enabled transition fires (shows the connection from PN**

places to PN transitions): A ‘negative’ matrix with the ‘-1’ and ‘0’ values is created, using the 1st and 2nd columns from Table 4.8. Part of this matrix is presented in Table 4.9 for some of the elements included in the AD in Figure 4.1. For example, if the ‘Asset Track’ is in the same row as the ‘pin’ in the ‘final’ table (Table 4.8), then the SQL code adds in the corresponding cell of the matrix (Table 4.9) the value ‘-1’, otherwise ‘0’ is inserted, as seen in the 1st row of Table 4.9.

Table 4.9 MySQL ‘Negative’ Table Extract

place_before_node	Visual_Inspection	D_VI	D_FT	D_DE	D_R	Strip_Scrap	M	Asset_Track	Functional_Test	Data_Erasure	Repair	Cleaning_De_Labelling
ATp	-1	0	0	0	0	0	0	0	0	0	0	0
Rp	-1	0	0	0	0	0	0	0	0	0	0	0
place_1	0	-1	0	0	0	0	0	0	0	0	0	0
place_2	0	0	-1	0	0	0	0	0	0	0	0	0
place_3	0	0	0	-1	0	0	0	0	0	0	0	0
place_4	0	0	0	0	-1	0	0	0	0	0	0	0
VIf	0	0	0	0	0	-1	0	0	0	0	0	0
DEf	0	0	0	0	0	-1	0	0	0	0	0	0

11. **Create a matrix that shows how a token is inserted to each of its PN post-places, when an enabled transition fires (shows the connection from PN transitions to PN places):** Similar to step 10, a second matrix with the ‘1’ and ‘0’ values is generated using the 2nd and 3rd columns from the ‘final_table’ created in step 9. For example, if the ‘Visual Inspection’ is in the same row with the ‘VIp’/ ‘VIf’ in the ‘final’ table (Table 4.8), then the SQL code adds in the corresponding cell of the matrix the value ‘1’, otherwise ‘0’ is inserted.
12. **Mathematical form of PN model:** In this final step, the transpose of the incidence matrix of the PN model developed for the recycling IT asset process is created as shown in Figure 4.10.

Table 4.10 Transpose of the PN Incidence Matrix

place_after_node	Visual_Inspection	D_VI	D_FT	D_DE	D_R	Strip_Scrap	M	Asset_Track	Functional_Test	Data_Erasure	Repair	Cleaning_De_Labelling
ATp	-1	0	0	0	0	0	0	1	0	0	0	0
CDp	0	0	0	0	0	0	-1	0	0	0	0	1
DEf	0	0	0	1	0	-1	0	0	0	0	0	0
DEp	0	0	0	1	0	0	0	0	0	0	0	-1
FTf	0	0	1	0	0	0	0	0	0	0	-1	0
FTp	0	0	1	0	0	0	0	0	0	-1	0	0
pin	0	0	0	0	0	0	0	-1	0	0	0	0
place_1	1	-1	0	0	0	0	0	0	0	0	0	0
place_2	0	0	-1	0	0	0	0	0	1	0	0	0
place_3	0	0	0	-1	0	0	0	0	0	1	0	0
place_4	0	0	0	0	-1	0	0	0	0	0	1	0
pout	0	0	0	0	0	0	1	0	0	0	0	0
Rf	0	0	0	0	1	-1	0	0	0	0	0	0
Rp	-1	0	0	0	1	0	0	0	0	0	0	0
SSp	0	0	0	0	0	1	-1	0	0	0	0	0
VIf	0	1	0	0	0	-1	0	0	0	0	0	0
VIp	0	1	0	0	0	0	0	0	-1	0	0	0

This matrix is obtained by combining the matrices developed in steps 11 and 12. Thus, the negative and positive matrices are unified and the sum of each

row is found. The first row of Table 4.10, presents the transitions of the matrix, whereas the first column consists of the places. The transpose of the incidence matrix in Table 4.10 consists of 17 rows (PN places) and 12 columns (PN transitions).

The two matrices, the one developed manually in equation 4.1 and the other generated automatically in Table 4.10 for the recycling IT asset process, are examined. The number and contents of the 1st column and the 1st row between these two matrices are the same, whereas the values ('-1', '0', '1') placed in these two matrices are found in the same cells. Thus, the two matrices are found identical, proving the PN model validation. Therefore, since this representation is created from the system information, the automated generation of the PN for the recycling IT asset process has been successful.

4.4.3 Algorithm – Java Database Programming – Petri Net Initial Marking Matrix

After obtaining the transpose of the incidence matrix of the PN model for the recycling IT asset process, in section 4.4.2, the initial marking matrix of this net is developed, using the SQL code introduced in Chapter 3, section 3.4.4, completing the mathematical form of the PN for this process. Thus, the step undertaken for the generation of the initial marking matrix for the recycling IT asset process is:

Initial marking of a PN model: The 'initial_marking' table, presented in Table 4.11, is created. The records in the "activity" column retrieved from the first column of the matrix in Table 4.10 corresponding to the PN places, whereas the value '1' in the "process_number_of_devices" column it is shown that the 'pin' place holds one token, i.e. one device.

Table 4.11 MySQL 'initial_maecking' Table

primary_id	activity	process_number_of_devices
1	ATo	0
2	CDo	0
3	DEf	0
4	DEo	0
5	FTf	0
6	FTo	0
7	pin	1
8	place 1	0
9	place 2	0
10	place 3	0
11	place 4	0
12	pout	0
13	Rf	0
14	Ro	0
15	SSo	0
16	VIf	0
17	VIo	0

The matrix of the initial marking in Table 4.11 is identical with the matrix of the initial marking in matrix 4.2 developed manually for the recycling IT asset process, proving the model validation.

4.5 Automated Graphical Representation of the Petri Net

Model for the Recycling IT Asset Process

For the graphical representation of the PN model for the recycling IT asset process, the steps introduced in Chapter 3, section 3.4.1 have been followed. The ‘final_table’ created in step 9 (Table 4.8) during the automated generation of the mathematical representation of the PN for the IT process is used as input for this graphical representation. Hence, all the data from the ‘final_table’, illustrated in Table 4.8, is selected and the code introduced in section 3.4.1 (Appendix D, part A) is executed. The output obtained in the Console window in Eclipse is a DOT file found in Appendix D, Part B. This output DOT file is imported into the Graphviz software and the PN model for the recycling IT asset process is obtained as presented in Figure 4.5.

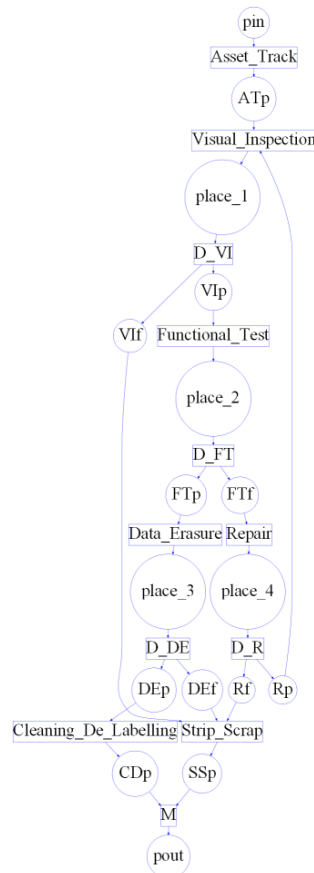


Figure 4.5 PN Model for the Recycling IT Asset Process

This PN model consists of 12 transitions and 17 places. The 12 transitions exist in the PN are equal to the sum of the seven opaque action nodes, the four decision node and the one merge node that was presented in the AD in Figure 4.1. The PN model in Figure 4.5, generated automatically for the IT asset process, is identical with the PN model created manually in Figure 4.2, proving the model validation.

4.6 Summary

This chapter has demonstrated the automated PN model capability to a real-life industrial process. The examined process is an extension in complexity of the simple process examined in Chapter 3, which now includes a greater number of activities and, by extension, more paths and all the fundamental elements of the AD. The mathematical and graphical representations of the PN model for the recycling IT asset process have been obtained both manually and automatically and it has been shown after comparison that they correlate. The correctness of the automation end model is further explored by a formal verification and validation of the method, viewed in the next chapter.

Additionally, the PN for the recycling process discussed in this chapter will be simulated in Chapter 5 to: (i) assess the process' performance by predicting the average execution time of the various paths of which the recycling process consists; and (ii) identify possible deficiencies that exist in the process by predicting the average time for each PN transition, the most common visited places in each path and the paths resulted most in failure. The timed and probabilistic data, needed for the simulation and correspond to the timed and probabilistic PN transitions, will be retrieved from an Excel file where the information is stored, whereas the transpose of the incidence matrix and the initial marking developed for the recycling IT asset process will be retrieved from the MySQL Workbench.

5 Verification & Validation of Petri Net Model

5.1 Introduction

In this chapter, the verification and validation of the Petri Net model are examined. Verification, related to building the model correctly, confirms that a model works properly, whereas validation, related to building the correct model, confirms that a model is the accurate representation of the real system (Banks et al., 1987; Balci, 1998; Balci 2004; Law, 2005). Therefore, the correctness of the algorithm developed for the PN automation procedure is checked by: (i) verifying that the PN model obtained performs the correct function; and (ii) validating the PN model obtained accurately represents the system architecture. If these two requirements are satisfied, the developed algorithm for the PN automation procedure is verified and validated. A review and evaluation of the most commonly used PN verification and validation methods is conducted in order to select the most appropriate methods. This is then demonstrated via evaluation of the recycling IT asset process considered in Chapter 4. Three main verification methods, i.e. methods related to PN behavioural and structural properties (static verification), methods related to PN dynamic behaviour (dynamic verification) and those in which two or more PN models are compared together and equivalent relations are identified between them (bi-simulation), have been identified. Similarly, for model validation, three main methods, i.e. expert intuition, real system measurements and theoretical analysis, have been identified. The layout of this chapter is illustrated in Figure 5.1.

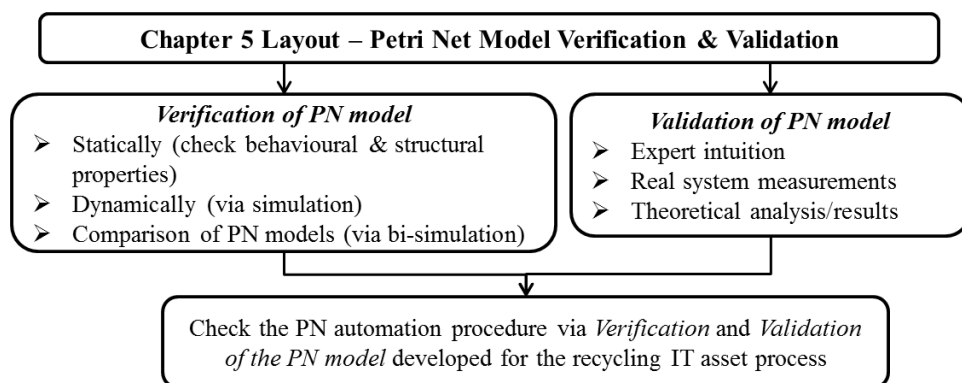


Figure 5.1 Illustration of the Structure of Chapter 5

5.2 Petri Net Model Verification Methods

The objectives of model verification are to check correctness, completeness and consistency of the developed model ensuring that the physical system, i.e. PN model, works properly, performing the required functions as specified. These characteristics can be checked through:

- *Static* verification which checks behavioural and structural properties of PNs via:
 - **Reachability graph analysis** that relies on the initial state of PN models, examining their behavioural properties; and
 - **Place/Transition invariants method** that relies on the topology of PN models, examining their structural properties.

For a complete static verification, both behavioural and structural properties should be checked.

- *Dynamic* verification which checks the correct execution of model paths via **simulation** analysis examining the PN model logic.
- *Comparing PN models* via **bi-simulation methods**.

In this section, an overview of static, dynamic and bi-simulation PN model verification methods is carried out.

5.2.1 Static Verification Methods

Petri Net models can be analysed *statically* checking their behavioural and structural properties by either applying the **reachability graph** and the **place/transition invariants method** respectively.

The behavioural properties of PNs are dependent on the initial marking M_0 (Li & Zhou, 2009) and listed below:

- *Reachability or deadlock-free* indicates that each reachable marking enables a transition. A marking M' is reachable from a marking M in a Petri Net N , if there is a firing sequence σ from M' to M .
- *Behavioural liveness* defines that each transition is enabled by at least one reachable marking and the transition can fire at least once.

- *Behavioural boundedness* shows that the number of tokens in each place does not exceed a finite number n from any marking reachable from the initial marking (M_0).
- *Safeness* is related to the bounded memory capacity. A PN is safe if it is 1-bounded, i.e. if the places always contain at most one token.
- *Reversibility* (home marking) shows that the initial marking is reachable from all possible reachable markings.
- *Persistence* defines that for any two enabled transitions, the firing of one transition will not disable the other.

The types of behavioural properties, which should be checked for the verification of a PN model, are defined according to the sub-class to which the examined PN belongs (Murata, 1989) hence not all properties need to be satisfied to be behaviourally verified.

The **reachability graph** is used to check the PN behavioural properties that serve as measures of effectiveness of the PN (Aalst, 1998). A reachability graph, an acyclic graph, indicates all possible future markings at some point in a PN model. It consists of nodes, which represent the possible system states, and arcs, which represent the possible state change. The graph starts from the initial marking and each possible reachable marking is listed and then connected with directed arcs, which are labelled with the corresponding transition needed to reach the marking.

A simple example of a reachability graph and its behavioural properties is presented in Figure 5.2 (Aalst, 2011). The PN, presented on the left side of the figure, consists of seven places ($p_1 - p_7$) and 6 transitions ($t_1 - t_6$). Places p_1 , p_4 and p_7 are marked with one token each. The corresponding reachability graph has been created and presented on the right side of Figure 5.2. The graph has five reachable states, defined in the bracketed terms in Figure 5.2. It starts with the initial marking of the PN, placed in the centre of the graph, and then according to the enabled transition the marking changes respectively. For instance, the initial marking $(1, 0, 0, 1, 0, 0, 1)$ shows that places p_1 , p_4 and p_7 have one token each, whereas places p_3 , p_4 and p_5 are empty. Once t_1 fires, the marking changes from $(1, 0, 0, 1, 0, 0, 1)$ to $(0, 1, 0, 1, 0, 0, 0)$, as can be seen in Figure 5.2, indicating that places p_1 and p_7 move one token each to place p_2 , whereas no further token movement is observed through the places.

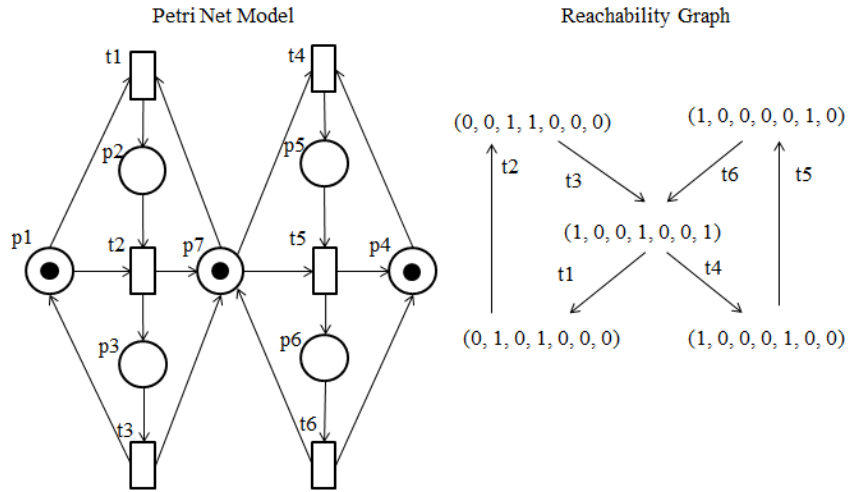


Figure 5.2 Reachability Graph Example (Aalst, 2011)

The remaining markings have been created following the same concept. Once the reachability graph has been developed, the behavioural PN properties are checked for model verification. Hence, according to Figure 5.2, the reachability graph is *deadlock-free*, and, by extension, *reachable*, because each reachable marking enables at least one transition to fire. For instance, the initial marking, presented as $(1, 0, 0, 1, 0, 0, 1)$ in Figure 5.2, enables the transitions $t1$ and $t4$. The graph is also *live* since it is possible to fire any transition, by progressing through a firing sequence. For instance, the initial marking $(1, 0, 0, 1, 0, 0, 1)$ enables a firing sequence containing all the transitions. Additionally, the PN is *1-bounded*, because the number of tokens included in each place does not exceed the finite number one for any marking reachable from the initial marking. Since the PN is 1-bounded, it is also *safe*. Finally, the graph is *reversible* since the initial/home marking can be reached from any reachable marking following the arcs presented in the reachability graph.

Although the reachability graph is the most common used method for the verification of PN behavioural properties, it lacks applicability due to its state-space explosion problem once applied to large and complex PN systems.

The structural PN model properties that depend on the incidence matrix (Proth & Xie., 1996; Cassandras & Lafortune, 2008) are listed below:

- A PN is characterised *structurally bounded* if it is (behaviourally) bounded for any initial marking M_0 . A PN, which is structurally bounded, is also behaviourally bounded, but the reciprocal is not true.

- A PN N is *structurally live* if there exists an initial marking M_0 such that the net is live. A PN, which is behaviourally live, is also structurally live, but the reciprocal is not true.
- A PN is *conservative* if there is at least one set of places with all the places equal to zero.
- A PN is *repetitive* if there exists an initial marking M_0 and a firing sequence σ from M_0 back to M_0 such that every transition fires infinitely often in σ .
- A PN is *consistent* if there exists an initial marking M_0 and a firing sequence σ from M_0 back to M_0 such that every transition fires at least once in σ .

The **place/transition invariants method** can be used to check the PN structural properties, which depend on the topological structure of PN models, applying linear algebraic techniques (Colom & Silva, 1991; Desel & Reisig, 1998; Recalde et al., 1998). There are two kinds of invariants: the *P-invariants*, related to places, which are the sets of places for which the sum of tokens remains unchanged for every marking; and the *T-invariants*, related to transitions, which are the sets of transitions for which the PN marking remains unchanged after firing each transition. Similar to the behavioural properties, the types of structural properties that should be verified each time are selected according to the sub-class to which the corresponding PN belongs. Large PNs can lead to infinite invariants, rendering it impossible to solve the equations by hand. Therefore, software that applies linear algebraic techniques can be used to obtain all the possible solutions of the equations (Colom & Silva, 1991; Desel & Reisig, 1998; Recalde et al., 1998).

5.2.2 Dynamic Verification Method

Petri Net models can also be verified *dynamically*, analysing the logic and behaviour of systems. Dynamic verification is performed via model **simulation** in order to check that system paths have been executed properly, detecting any possible undesirable behaviour and incorrect or omitted logic. However, this method lacks the ability to check if PNs satisfy a desired set of properties, as static verification does, and cannot guarantee that all possible simulation paths of the system have been covered (Mhairi, 2009). Therefore, although model simulation can verify the logic of Petri Net elements, it is not an exhaustive means of proving model correctness (Obaidat & Boudriga, 2010).

5.2.3 Comparison of PN Models (Bi-simulation) for Verification

Another method, identified for PN model verification, is *PN model comparison* via **bi-simulation** equivalence, which can be used to verify whether two models have equivalent behaviours. Hence, two PNs are characterised bi-similar if one can simulate the other and vice-versa (Jančar et al., 1999). According to Girault and Valk (2003), two nets are considered bi-similar if and only if a correspondence between their markings can be identified such that in corresponding markings every firing transition in one net can be matched by a similar firing transition in the other net, leading to corresponding markings.

5.3 Verification of Automated Petri Net Development

The PN automation procedure introduced in Chapter 3 is verified via evaluation of the PN model generated for the recycling IT asset process introduced in Chapter 4. According to the review conducted in this chapter for the verification methods of a PN model, static analysis has been found to be the most suitable method, with the others having deficiencies. The dynamic verification via simulation was determined to be unsuitable, since it cannot ensure the execution of all paths of the model. Although the bisimulation method can be used for verifying the PN automation procedure, comparing the two PN models developed in Chapter 4, i.e. the one developed manually for the process in section 4.3 and the other generated automatically in section 4.5, it is not applied in this work, since it can be an error-prone and time consuming method, especially when it is applied to complex and large PNs, since it is manually conducted.

The PN model, developed for the recycling IT asset process, belongs to a special class of PNs, workflow-nets. A PN is called a workflow-net (WF-net) if and only if it satisfies the criteria defined by Aalst (1998):

1. The PN model has two places, a source place, denoted as '*i*', and a sink place, denoted as '*o*', that correspond to the beginning and the termination of the net respectively.
2. The PN is strongly connected, this means that if a transition '*t**' is added to the PN, then this transition connects the source place '*i*' with the sink '*o*'.

If a third criterion is met, in addition to the two aforementioned requirements, then the WF-net can be *structurally* verified, i.e. based on the nets structure (Aalst, 1998). The third criterion is:

3. Once WF-net execution is terminated, there is at least one token in place ‘*o*’ (sink place) and all the other places are empty.

According to Aalst (1998), besides structural analysis, WF-nets can be verified based on their *behavioural* properties, if they satisfy the *soundness* property for which three requirements are identified as follows:

- For every state ‘*M*’ (marking) reachable from place ‘*i*’ (initial marking, M_0), there exists a firing sequence leading from state ‘*M*’ to place ‘*o*’ (final marking).
- Place ‘*o*’ is the only state reachable from place ‘*i*’ with at least one token in sink place ‘*o*’.
- Absence of dead transitions, i.e. transitions that can never become enabled, from the initial state ‘*i*’. Therefore, it should be possible to execute an arbitrary transition by following the appropriate path through the net.

Aalst has proven that a WF-net $PN = (P, T, F)$ is *sound* if and only if the model is *behaviourally live and bounded*. Where: P is a set of places; T is a set of transition ($P \cap T = \emptyset$); F is a set of arcs ($F \subseteq (P \times T) \cup (T \times P)$).

Therefore in this work, Hierarchical Petri net Simulator (HiPS), a tool able to design and analyse hierarchical and timed PN models, has been used for checking the correctness of the PN automation procedure by statically verifying the PN developed for the WF-net developed for the recycling IT asset process. This verification is carried out by checking its *structural* and *behavioural* properties, using HiPS. This open source tool has been selected because it provides a simple user-interface, an interactive and user-friendly environment and a clear function of each icon including those used both for PN model construction (places, transitions, arcs and token) and for the analysis of the developed PN (structural and behavioural properties and simulation).

The PN model for the recycling IT asset process, generated automatically in Chapter 4, has initially been transformed to a HiPS representation, as illustrated in Figure 5.3. The places of this net are numbered in red, whereas the transitions are labelled in green, as seen in Figure 5.3. Table 5.1 explains the abbreviations of places and transitions included in the PN model illustrated in Figure 5.3. The **structural** and **behavioural properties** of this WF-net are then verified.

Table 5.1 Abbreviations and Full Names of Places and Transitions included in the PN in Figure 5.3

Abbreviation	Full Name
pin	place_in
ATp	Asset_Track_pass
VIp	Visual_Inspection_pass
VI _f	Visual_Inspection_fail
FTp	Functional_Test_pass
FT _f	Functional_Test_fail
DEp	Data_Erasure_pass
DE _f	Data_Erasure_fail
Rp	Repair_pass
R _f	Repair_fail
CDp	Cleaning_De_Labelling_pass
SSp	Strip_Scrap_pass
pout	place_out
D_VI	Decision_Visual_Inspection
D_FT	Decision_Functional_Test
D_DE	Decision_Data_Erasure
D_R	Decision_Repair
M	Merge

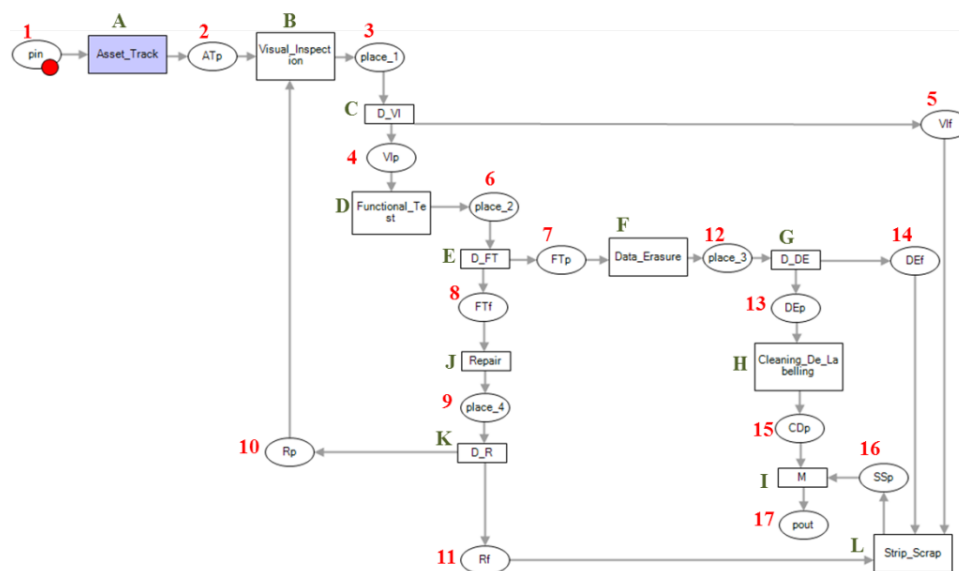


Figure 5.3 Petri Net developed in HiPS for the Recycling IT Asset Process

The user visually verifies the **structural properties** of the WF-net, illustrated in Figure 5.3, as follows:

1. The net has an initial source place, named '*pin*', and a final sink place, named '*pout*'.
2. Each transition has a unique name and all places are on a path from '*pin*' (source) place to '*pout*' (sink) place.
3. The process execution initiates with the '*pin*' (source) place, being marked with at least one token and no other place are marked, whereas the process terminates with the '*pout*' (sink) place, being marked with at least one token, while all the other places are empty.

Hence, regarding the **structural properties** of the WF-net developed for the recycling IT asset process, the net is successfully verified since it has been demonstrated that it satisfies all three of the criteria defined from Aalst (1998).

In addition to the aforementioned verification of the structural properties, the WF-net developed for the recycling IT asset process is also checked for its structural boundedness. Therefore, using the 'Reachability/Coverability Analyze (Auto)' HiPS option, the WF-net is verified as *structurally bounded*, as seen on the left side in Figure 5.4. This property is satisfied since for every initial marking, M_0 , there exists a finite number, n , such that all markings have at most this n number of tokens in all places.

The **behavioural properties** of the WF-net created for the recycling IT asset process are checked by the HiPS tool using the software options: 1) 'Bounded', which checks if the PN is behaviourally bounded; and 2) 'FC/AC Liveness/Safeness Checker', which checks if the PN is live, safe and marked. The **behavioural properties** of the WF-net, illustrated in Figure 5.3, are verified as follows:

1. HiPS, as seen on the right side in Figure 5.4, which shows that the model is 1-bounded, i.e. the number of tokens in each place does not exceed the one in number for any marking reachable from the initial marking, M_0 , check the model's boundedness property.
2. The model's *liveness* and *safeness* properties are checked in HiPS, as seen in Figure 5.5, which shows that the net satisfies both behavioural properties. The WF-net in Figure 5.3 is *live* since it is possible to fire any transition by progressing through some further firing sequence. Additionally, the PN is *safe*,

because all its places are 1-bounded, as shown on the right side of Figure 5.4. Two additional characteristics are seen in Figure 5.5, the net is characterised as a *free-choice net* because it keeps places that have more than one output transition apart from transitions with more than one input place and it is a marked graph since there exists at least one token. The final characteristic identified is that since the liveness property is satisfied, the WF-net can also guarantee *deadlock-free* operation.

Hence, regarding the **behavioural properties** of the WF-net developed for the recycling IT asset process, the net is successfully verified since it has been demonstrated to be *behaviourally live and bounded*, and hence *sound*, which guarantees the net's verification according to Aalst (1998).

Therefore, according to the analysis, conducted in HiPS, the correctness, completeness and consistency of the WF-net constructed for the recycling IT asset process and, by extension, of the automation PN procedure, have been proven, by successfully verifying the structural and behavioural properties of the model.

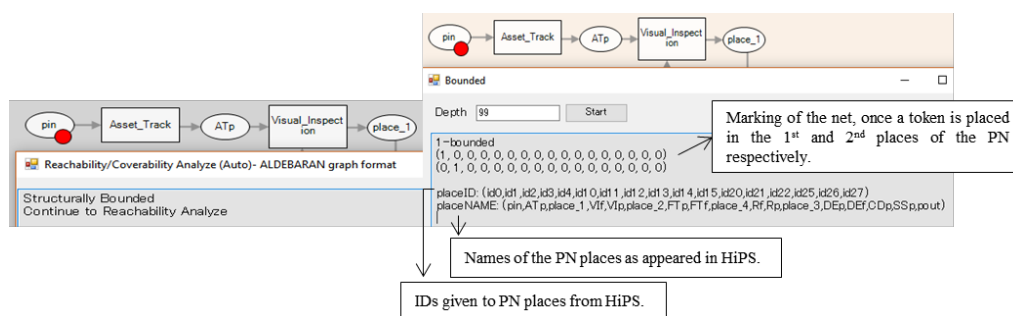


Figure 5.4 Structurally and Behaviourally Bounded Check in HiPS for the Recycling IT asset Process

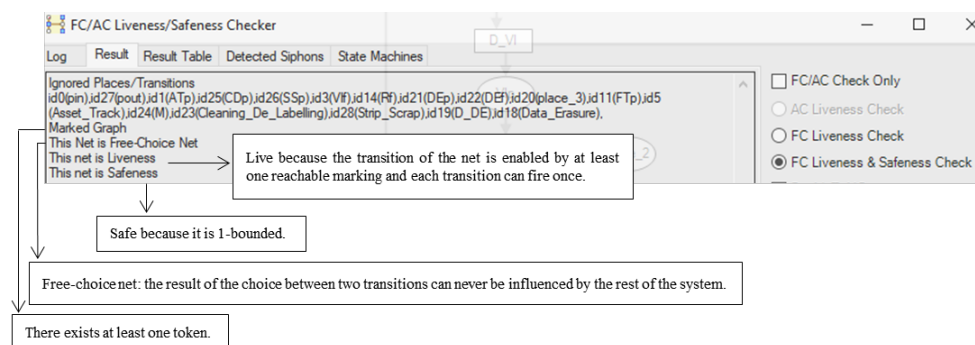


Figure 5.5 Behavioural Liveness and Safeness Properties Check in HiPS for the Recycling IT asset Process

5.4 Petri Net Model Validation Methods

The objective of validation is twofold: (i) to check the system's behaviour, using a realistic representation of the actual system, which would be able to reproduce the system's behaviour and satisfy the analysis objectives; and (ii) to analyse the system's performance, detect the system's limitations and draw conclusions that enable decision-making and potentially system optimisation. In this section, an overview of model validation methods is carried out. The following model validation methods can be applied either individually or in combinations.

5.4.1 Expert Intuition Validation Method

People, such as system designers or service engineers, who have a detailed knowledge of the real system for which the PN is created, rather than knowledge of PN modelling, apply this method (Al-Aomar et al., 2015). The expert intuition method lacks formal methods for model validation (Robinson, 1997). Experts carefully check the graphical representation and behaviour of the PN to extract system information, ensuring that the PN model represents accurately the underlying real-world system. The method is mainly applied to simple models since there is high risk of model misinterpretation and it can also become time consuming when applied to complex systems (Hillston, 2017).

5.4.2 Real System Measurements Validation Method

This method, applied in cases when the internal structure of the system including data and behaviour of elements and paths is known, is used to ensure that the parts of the computer model, which simulate the behaviour of conceptual models, correspond accurately to the elements and logic of the underlying real-world system (Sargent, 1992). In this method, the conceptual model is the Petri Net model. Real system measurements, also named white-box validation, can check a wide spectrum of model aspects such as timings (activity or repair times), control logic or control of flows (routing), distributions, etc. (Robinson, 1997). In this method, the PN's correctness can be proved by conducting visual checks and inspections of the output results. For the visual checks, the graphical representation of the PN model as well as its behaviour can be traced and animated, playing the 'token game' in which the user can observe the marking changes once transitions fire and corresponding tokens move through the net. In addition, for the inspection of the output results, computer code, developed

following the paths of the PN model, can simulate the various scenarios of the model, extracting output results for these paths. Therefore, these output values can be compared to the corresponding values of the real system and the percentage error between these values can be found. Although the real system measurements method is the most reliable model validation approach since it checks the model correctness using a high level of detail comparing it to real data, it is not always applicable, either due to the lack of output data for the real system or due to high costs of measurements (Hillston, 2017).

5.4.3 Theoretical Results/Analysis Method

Theoretical results/analysis or black box methods are mainly applied in cases when the internal structure of the system is unknown. This method examines a more abstract representation of the system reproducing the systems behaviour with a low level of detail and via simulation analysis can check the correctness of model output results compared either to historical/expected values of the system or to other models (Robinson, 1997; Hillston, 2017). Comparing outputs from simulation models with those historical/expected values from the real system, the consistency between the PN model simulation and the theoretical (historical/expected) data is checked. The confidence in the results can be improved once multiple model replications, i.e. simulation runs, are executed (Robinson, 1994). Additionally, comparison of the real-world system with other models can be carried out using mathematical models, such as spreadsheet analysis and queuing networks, providing an approximation of output results of the real system that can be compared with the results gained from the PN model simulation. This method is fraught with considerable risk, since both the mathematical models and PN model may be invalid, leading to inaccurate results (Hillston, 2017).

5.5 Validation of Automated Petri Net Development

Following the review, conducted for the PN validation methods, the real system measurements approach has been selected due to its ability to:

1. **Check visually** the system's behaviour playing the token game.

2. Check the PN model's quality by obtaining numerical results and comparing these numerical results with those observed in the real world process (**numerical simulation**).
3. Conduct system **performance analysis** to detect limitations or incorrect/omitted logic existing in the model.

The last two points require the introduction of data to the timed and probabilistic transitions included in the PN model, in order for numerical results to be obtained. The data used is explained in detail in section 5.5.2.

The expert intuition and theoretical analysis validation methods have been considered inappropriate for this study due to the absence of system design experts and due to the possession of adequate data and information on the internal structure of the recycling IT asset process, enabling a more thorough investigation.

5.5.1 Petri Net Model Simulation Algorithm

In this section, a summary of the algorithmic steps followed for the simulation of the PN model generated for the recycling IT asset process, is described. The simulation algorithmic steps are presented in Figure 5.6.

The inputs required for the PN model simulation are: (i) the mathematical representation of the PN model obtained from the automated generation of PN model in Chapter 4; and (ii) data for the timed and pass/fail probabilistic transitions including in the PN. Thus, the algorithm developed in MATLAB, initially retrieves the transpose of the incidence matrix and the initial marking developed for the recycling IT asset process from the MySQL Workbench and secondly reads the timed and probabilistic data stored in Excel.

For the **visual check** of the system behaviour, both the PN mathematical representation and the pass/fail probabilistic data are required. The PN mathematical representation is necessary to show the movement of tokens/devices in the net using equation 2.2, which indicates the sequence of places and transitions, whereas the pass/fail probabilistic data is required to show the probability of completing an activity. An algorithm is developed which: (i) traces the movement of the token through the net, applying the token game, and (ii) validates the PN behaviour with the

expected movement of the device through the process. The algorithm developed, found in Appendix F, is discussed in detail in section 5.5.3.

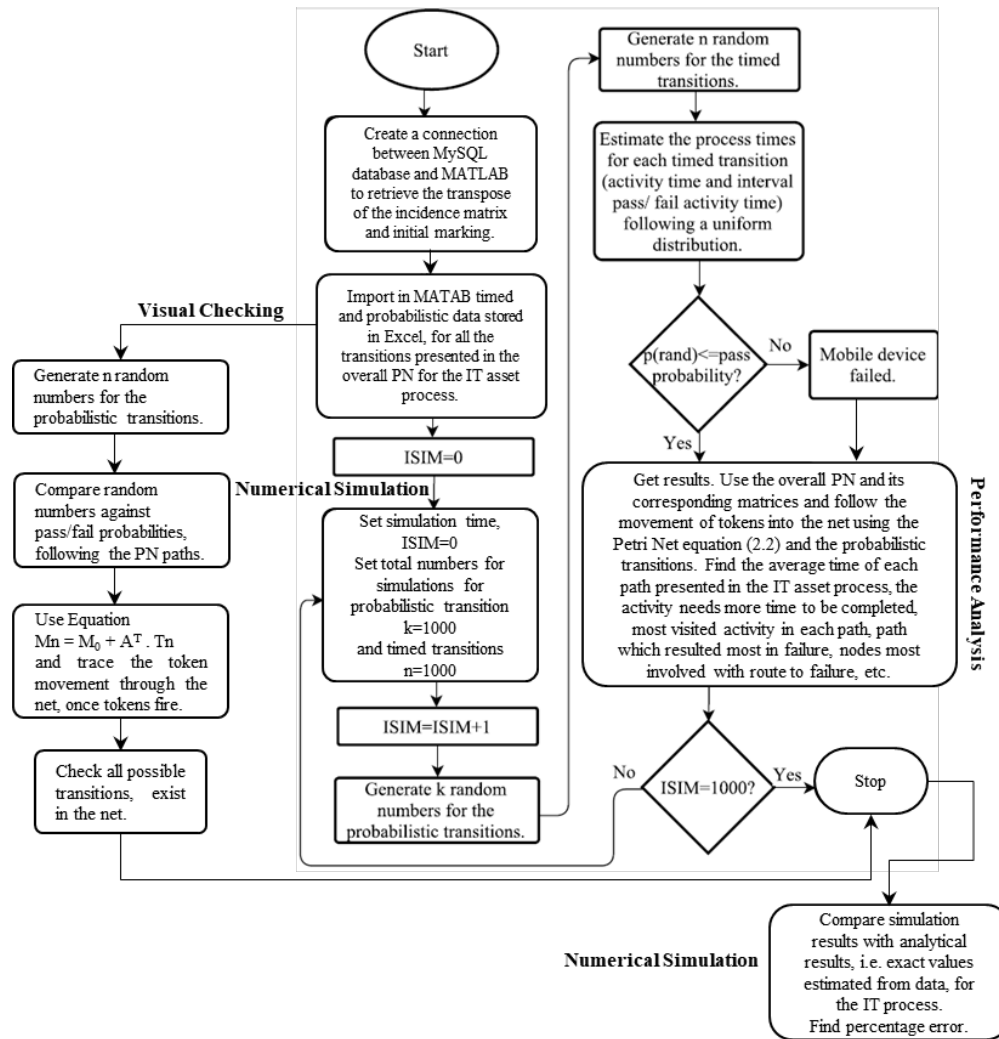


Figure 5.6 Flowchart for the Simulation Steps followed for the Recycling IT Asset Process

A second algorithm is developed to obtain numerical results in order to conduct: (i) **numerical simulation** (compare the obtained results with values provided by industry); and (ii) **analysis of the system's performance**. For the simulation, the timed transitions are divided into two categories, the activity and interval transitions, for which data is provided by industry. The activity transitions correspond to the actual time needed for an activity, such as Asset Track or Visual Inspection, to be completed and the interval transitions correspond to delays between activities. For the probabilistic transitions, random numbers are generated and compared with the given pass/fail probabilities. If the random number is lower than or equal to the pass probability of an activity then the device is assumed to pass, otherwise the device

fails. Activity and interval times, following a continuous uniform distribution, are then estimated. The algorithm developed for the numerical simulation of the PN model is found in Appendix F, part B. The PN model evaluation is completed by comparing calculated results obtained using the input data, such as average time of each path, with the corresponding simulation results.

Finally, for system's **performance analysis**, the second algorithm has been used to obtain more results regarding system's performance, such as the average time each path requires to be completed, the most common visited places in each path, as well as the paths resulting in the most failures and the nodes most involved in the route to failure. The additional part of the second algorithm developed for the performance analysis of the PN model, is found in Appendix F, part C. The simulation results can also enable the identification of possible limitations and the provision of recommendations to improve system's performance.

The application of the simulation algorithmic steps described in this section to the recycling IT asset process is described in detail in sections 5.5.3 and 5.5.4.

5.5.2 Process Input Data

The data used for the simulation comes from 2113 mobile phones processed over 323 hours. The timed and probabilistic data is shown in Tables 5.2, 5.3 and 5.4. Table 5.2 shows the pass (second column) and fail (third column) probabilities for each PN probabilistic transition included in the PN. The probabilistic transitions correspond to the PN transitions labelled as 'D_x', such as 'D_VI' (Decision_Visual_Inspection), 'D_R' (Decision_Repair), etc., as seen in Table 5.2.

Table 5.2 Probabilities for the PN developed for the Recycling IT Asset Process

Transition Name	Pass Probability	Fail Probability
D_VI	0.688	0.312
D_FT	0.733	0.267
D_DE	0.88	0.12
D_R	0.294	0.706

Tables 5.3 and 5.4 show the timed data for activity and interval times respectively. Table 5.3 presents the minimum and maximum times needed to complete each activity.

Table 5.3 Activity Times for the PN developed for the Recycling IT Asset Process PN

Transition Name	Activity Time (seconds)	
	min time	max time
Asset_Track	107	148
Visual_Inspection	5	10
Functional_Test	60	180
Data_Erasure	30	40
Repair	240	900
Strip_Scrap/Cleaning_De_Labelling	30	60

Table 5.4 shows the minimum and maximum interval times required for a device to move from one activity to another. The interval pass and fail times shown Table 5.4 are used according to pass and fail paths that a device can take following the probabilistic transitions explained in Table 5.2.

Table 5.4 Interval Times for PN developed for the Recycling IT Asset Process

Transition Name	Interval Time (seconds)			
	Pass Time		Fail Time	
	min time	max time	min time	max time
Asset_Track	30	120	0	0
Visual_Inspection	300	1800	300	3600
Functional_Test	1800	7200	7200	8640
Data_Erasure	1800	10800	1800	10800
Repair	1800	28800	1800	28800

The data in these tables is explained with the help of an example for the Visual Inspection activity. The part of the PN, which corresponds to this activity including the ‘D_VI’ (Decision_Visual_Inspection) and ‘Visual_Inspection’ transitions, is presented in Figure 5.7. According to Figure 5.7, the first transition of this net, presented as ‘Visual_Inspection’, is found in the second row of Table 5.3 and shows the time needed for the Visual Inspection to be completed. The same transition is also appeared in the second row of Table 5.4 showing the time required for a device to be transferred to the next activity, once the Visual Inspection has been completed. The ‘pass’ and ‘fail’ terms found in this table correspond to the successful and unsuccessful completion of the Visual Inspection respectively. Hence, the minimum and maximum pass times (second and third columns of Table 5.4) are used if the Visual Inspection is successfully completed, whereas once the Visual Inspection fails, the minimum and maximum fail times (fourth and fifth columns) are considered.

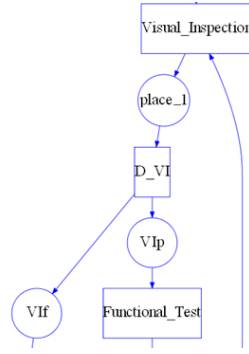


Figure 5.7 PN Model Extract from the Recycling IT Asset Process

Therefore, the ‘Visual_Inspection’ transition contains two times, one for the activity to be completed and one for the device to pass to the next activity. Finally, the probabilistic transition of the PN in Figure 5.7, presented as ‘D_VI’ can be found in the first row of Table 5.2 for the pass and fail probabilities respectively.

5.5.3 Petri Net Model Visual Check

The algorithm, developed for the PN visual check is based on equation 5.1:

$$M_r = M_0 + A^T \cdot T_r \quad (5.1)$$

The algorithm uses the following matrices:

- M_0 : the initial marking for the IT asset process, which is retrieved from the MySQL database.
- A^T : the transpose of the incidence matrix for the IT asset process, which is also retrieved from the MySQL database.
- T_r : transition matrices are created in Excel for each transition included in the net and then retrieved from the algorithm.

The algorithm, found in Appendix F, part A, applies the following steps:

1. Generate random numbers from zero to one.
2. Load into Matlab the Excel tables with the probabilistic data shown in Table 5.2.
3. Load into Matlab the initial marking (M_0) and the transpose of the incidence matrix (A^T) developed in the MySQL database.
4. Develop the necessary transition matrices (T_r in equation 5.1) for each PN transition.

5. Follow the PN paths and compare the random numbers, developed in step 1, against the pass/fail probabilities given data in Table 5.2, using ‘if’ conditions.
6. Apply equation 5.1 and examine the marking generated.

The methodology followed for visually checking the PN model developed for the recycling IT asset process is explained for the first three transitions presented in Figure 5.8, i.e. for the ‘Asset_Track’, ‘Visual_Inspection’ and ‘D_VI’. The path used for this example corresponds to the route where the device passes successfully the visual inspection, functional test and data erasure activities.

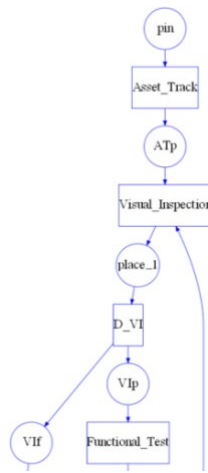


Figure 5.8 PN Model Extract from the Recycling IT Asset Process

Therefore, the initial marking, M_0 , in equation 5.2 represents the marking of the first five places i.e. ‘pin’, ‘ATp’, ‘place_1’, ‘VIp’ and ‘VIf’, of the PN created for the recycling IT asset process, as illustrated in Figure 5.8. This matrix shows that only the ‘pin’ place holds one token, whereas all the other places are empty.

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \end{bmatrix} \quad (5.2)$$

Similarly, the transpose of the incidence matrix, A^T , in equation 5.3 corresponds to the underlying transitions and places (the places and transitions are presented on the left and above the matrix, correspondingly), showing how a token moves from one place to another, once a transition fires.

$$A^T = \begin{matrix} & \textit{pin} \\ & \textit{ATp} \\ \textit{place_1} & \\ \textit{VIp} & \\ \textit{VIf} & \\ \dots & \end{matrix} \begin{bmatrix} -1 & 0 & 0 & \dots \\ 1 & -1 & 0 & \dots \\ 0 & 1 & -1 & \dots \\ 0 & 0 & 1 & \dots \\ 0 & 0 & 0 & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix} \quad (5.3)$$

Additionally, for each transition the matrices in equations 5.4, 5.5 and 5.6 are created. For instance, T_{D_VI} in equation 5.6 is the transition matrix for transition ‘D_VI’ (Decision Visual Inspection).

$$T_{\textit{Asset_Track}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \dots \end{bmatrix} \quad (5.4) \quad T_{\textit{Visual_Inspection}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \dots \end{bmatrix} \quad (5.5) \quad T_{D_VI} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \dots \end{bmatrix} \quad (5.6)$$

When it is applied in equation 5.1, the transition named ‘D_VI’ fires and a token should be moved from ‘place_1’ and added to place ‘VIp’ (Visual Inspection pass), indicating that the device has passed the visual inspection.

Besides the initial marking, the transpose of the incidence matrix and the transition matrices, the data in Table 5.2 for the probabilistic transitions, i.e. ‘D_VI’, ‘D_FT’, ‘D_DE’, ‘D_R’, has been used as inputs for the execution of the algorithm. Hence, for the probabilistic transitions, random probabilities uniformly distributed in the interval (0, 1), are generated. In the next stage, the algorithm runs for the six paths identified in the recycling IT asset process applying steps 5 and 6 in the algorithm. The marking matrices, generated after the firing of transitions ‘Asset_Track’, ‘Visual_Inspection’ and ‘D_VI’, are shown in equations 5.7, 5.8 and 5.9, respectively.

$$M_{\textit{Asset_Track}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ \dots \end{bmatrix} \quad (5.7) \quad M_{\textit{Visual_Inspection}} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \dots \end{bmatrix} \quad (5.8) \quad M_{D_VI} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \dots \end{bmatrix} \quad (5.9)$$

Therefore, for instance, the ‘Visual_Inspection’ marking, in equation 5.8, shows that a token is held in ‘place_1’, shown in Figure 5.8, which agrees with the expected matrix, since once ‘Visual_Inspection’ fires, it moves one token from place ‘ATp’ to ‘place_1’. Subsequently, in the ‘D_VI’ marking, in equation 5.9, the token is moved to the ‘VIp’ place, since the device passes. The flow traced in the PN model also

agreed with the UML/SysML Activity Diagram for the IT asset process when the device successfully completes the Asset Track activity.

Executing the algorithm, developed for the visual check of all the paths of the PN model, and using the necessary matrices in conjunction with the probability paths, it has been found that the net follows the expected behaviour, according to the flow of the Activity Diagram for the recycling IT asset process. Hence, this method has validated the algorithm developed for the PN model generation by visually checking the graphical representation of PN model in Figure 5.8, referring to the Activity Diagram from which the model has been developed. The correctness of the algorithm through the token game, which enables the graphical visualisation of the behaviour of the PN, has been proven since no errors such as unintended behaviour regarding the flow of items/information in the system have been identified.

5.5.4 Petri Net Model Numerical Simulation and Performance Analysis

In this section, the validation of the algorithm developed for the automated PN model generation is carried out via simulation of the sub-PNs developed for the recycling IT asset process, described in Chapter 4. For the model's execution, timing and probabilities have been introduced in the timed and probabilistic transitions of the models respectively and the six paths, identified in the recycling IT asset process using the transpose of the incidence matrix, obtained automatically, and the PN, generated automatically, have been simulated in order to obtain numerical results. These simulation results can be compared with the analytical results, obtained from the given data. This comparison of the results can judge the quality of the PN model, leading to a better understanding of the recycling IT asset process and additionally providing deeper insights into the behaviour of the real system.

Therefore, as it can be seen from the flowchart in Figure 5.6, the algorithm for the numerical simulation generates random probabilities, for the probabilistic transitions. Similarly, random numbers for activity transitions, shown in Table 5.3, using equation 5.10 and for interval activity transitions, listed in Table 5.4, using equation 5.11, are estimated. The times obtained from equations 5.10 and 5.11 are the times in the current activity and interval activity respectively, whereas the time obtained from equation 5.12 is the sum of activity (service) time and interval (waiting) time.

$$t_{activity} = \min_time + (\max_time - \min_time).x \quad (5.10)$$

$$t_{interval_activity} = \min_time + (\max_time_{pass/fail} - \min_time_{pass/fail}).x \quad (5.11)$$

$$t_{sojourn} = t_{activity} + t_{interval_activity} \quad (5.12)$$

Where: \min_time and \max_time are obtained from Table 5.3 for each activity, $\min_time_{pass/fail}$ and $\max_time_{pass/fail}$ are obtained from Table 5.4 for each interval time for pass and fail respectively; and x returns a uniformly distributed random number in the interval (0, 1).

The PN model is transformed into a Stochastic PN (SPN) since time t obtained applying equations 5.10 and 5.11 is a random variable. The sojourn time in the current state, found by applying equation 5.12 can be computationally modelled using any cumulated distribution function, such as exponential, related to the time of occurrence of the corresponding event. This indicates that the developed model follows the SPN concept where the delays are randomly chosen by sampling distributions associated with transitions. Hence, the SPN that adds flexibility and a wider range of applicability is considered.

The algorithm for the numerical simulation was run for random numbers of simulations, between 0 and 2500, in order to find the optimum simulation number. The results obtained for these simulations during the calculation of the average time needed for path 1 (see Table 5.5) of the recycling IT asset process to be completed, have been plotted in Figure 5.9.

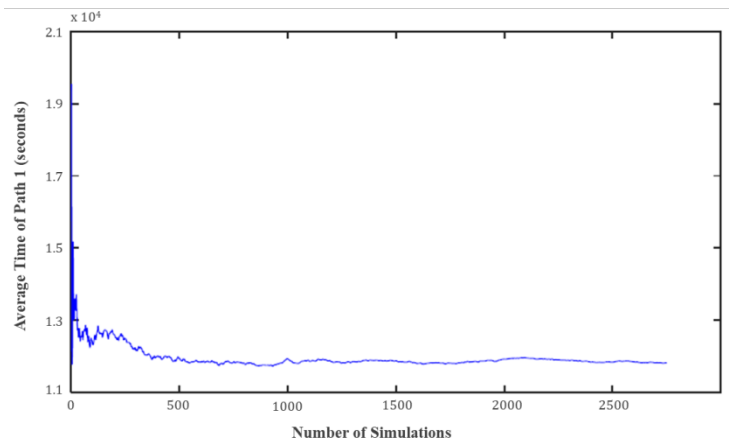


Figure 5.9 Simulation Results for the 1st Path of the Recycling IT Asset Process for 2500 Simulations

According to this figure, very small variations were observed in the simulation results after 1000 runs and hence, in order to save time, the number of simulations has been defined at 1000. Additionally, the simulation results obtained during the calculation of the completion times of the other five paths of the recycling IT asset process are in a good agreement with the results obtained for the first path. Therefore, the algorithm runs 1000 simulations for the six paths identified in the recycling IT asset process.

Following the paths of the PN model and using the pass/fail probabilities and the random (activity and interval) times, the average time for each path has been estimated and the simulation results are presented in Table 5.5. The analytical results for the average time of each path, identified in the model, have also been manually calculated, using the given timed data from Tables 5.3 and 5.4, and presented in Table 5.5.

Table 5.5 Average Completion Time for Each Path of the Recycling IT Asset Process

Path ID	Path Activity	Analytical Results Average Path Time (secs)	Simulation Results Average Path Time(secs)	% error
1	A-B-C-D-E-F-G-H-I	12260	12239	0.17129
2	A-B-C-D-E-F-G-L-I	12260	12234	0.21207
3	A-B-C-D-E-J-K-B-C-D-E-F-G-H-I	37167.5	36589	1.55647
4	A-B-C-D-E-J-K-B-C-D-E-F-G-L-I	37167.5	36590	1.55378
5	A-B-C-D-E-J-K-L-I	25208.25	25447	0.94711
6	A-B-C-L-I	2205	2239.1	1.54649

The letters in the path activity column are retrieved from Figure 5.3. Subsequently, the simulation and analytical results are compared and the error is estimated, as shown in Table 5.5. The error in all the paths is found to be low, i.e. less than 1.6%, confirming that the simulation results agree with those obtained from the analytical results that correspond to the real recycling IT asset process. According to the low error found, the validity of the PN model as being a realistic graphical representation of the IT asset process can be confirmed.

5.5.5 Performance Analysis Results and Discussion

Some additional results in order to examine the IT process's performance have been obtained. The algorithm, found in Appendix F, part C, uses the data given for the recycling IT asset process in section 5.5.2.

As well as the estimation of average time needed for each path to be executed, the average activity and pass/fail interval times have been simulated and the results are presented in Table 5.6 and 5.7 respectively. From the results in Tables 5.6 and 5.7, it can be seen that, in general, the activity times are a lot less than the pass/fail interval times between activities, and hence these times between activities cause long delays in the recycling IT asset process. From these two tables, it is seen that the transitions related to the Repair, i.e. 'Repair' in Table 5.6 and 'Repair' in Table 5.8, take the longest time to be completed. The average interval time has been calculated to be 4 hours and 15 minutes (15310.043 seconds), as seen from Table 5.7. This happens because the repair activity takes place in a different location from the rest of the activities and extra time is required for the transportation of the devices.

Table 5.6 Average Time for each Activity Timed Transition of the Recycling IT Asset Process PN

Transition Name	Activity Time (seconds)
	Average Time
Asset_Track	127.381
Visual_Inspection	7.485
Functional_Test	119.682
Data_Erasure	35.037
Repair	568.589
Strip_Scrap	44.842
Cleaning_De_Labelling	44.908

Table 5.7 Average Time for each the Interval Timed Transition of the Recycling IT Asset Process PN

Transition Name	Interval Time (seconds)	
	Average Pass Time	Average Fail Time
Asset_Track	75.335	0
Visual_Inspection	1050.836	1952.051
Functional_Test	4504.242	7927.789
Data_Erasure	6229.501	6229.501
Repair	15310.043	15310.043

The numbers of visits to the places of the PN for each path have been found and are presented in Table 5.8. Numbers, retrieved from Figure 5.3, presents the places involved in each path in the second column of Table 5.8 and in the third column each number corresponds to the number of visits to the places as they were described in the second column. It can be seen that some of the most common visited places apart from the 'pin', 'ATp', 'place_1' and 'pout' which are always visited, are the 'VIp', 'place_2', 'FTp' and 'FTf' and 'place_4'. The number of times each path is taken can also be found in the fourth column of Table 5.8. Therefore, the most visited path is the first, whereas the least visited path is the fourth that was visited only three times in a 1000 simulation. From the table it can be seen that the sixth path where the Visual Inspection fails, results most often in failure, i.e. Strip and Scrap. The second path fails because the Data Erasure activity fails, whereas the fifth path fails due to the Repair activity failure. According to Table 5.5, it can be seen that the fifth path is less efficient than the second, since it takes a lot longer to be completed. However, according to Table 5.8, the fifth path has been taken 129 times during the simulation, whereas the second path only 60.

Table 5.8 Number of Visits to Places of the Recycling IT asset Process PN

Path ID	Path Places	Number of Visits to Path Places	No. of path executions
1	1-2-3-4-6-7-12-13-15-17	1000 - 1000 - 1000 - 688 - 688 - 504 - 504 - 443 - 443 - 443	443
2	1-2-3-4-6-7-12-14-16-17	1000 - 1000 - 1000 - 688 - 688 - 504 - 504 - 60 - 60 - 60	60
3	1-2-3-4-6-8-9-10-3-4-6-7-12-13-15-17	1000 - 1000 - 1000 - 688 - 688 - 183 - 183 - 54 - 54 - 37 - 37 - 27 - 27 - 23 - 23 - 23	23
4	1-2-3-4-6-8-9-10-3-4-6-7-12-14-16-17	1000 - 1000 - 1000 - 688 - 688 - 183 - 183 - 54 - 54 - 37 - 37 - 27 - 27 - 3 - 3 - 3	3
5	1-2-3-4-6-8-9-11-16-17	1000 - 1000 - 1000 - 688 - 688 - 183 - 183 - 129 - 129 - 129	129
6	1-2-3-5-16-17	1000 - 1000 - 1000 - 312 - 312 - 312	312

To conclude the simulation findings, the results have shown that:

- The third and fourth paths are the longest due to the Repair stage, which is the most time consuming stage in both paths. According to this finding, a limiting factor of the process is the long time that the Repair action needs to be

completed due to the device transportation to a different location. The recycling IT asset process performance could be improved if all the activities are located at the same place in order to decrease: (i) the interval time between the ‘Functional_Test’ and ‘Repair’ activities; and (ii) the manpower required.

- The pass/fail average interval times, shown in Table 5.7, last longer than the average activity times illustrated in Table 5.6, causing long delays in the process paths. The process performance could be improved by: (i) increasing the ability of the activities to accept multiple devices simultaneously as the Data Erasure does; and (ii) locating the activities at the same place.
- According to Table 5.8, the most commonly resulting action is the Strip and Scrap, since the sum of the executed paths that result in Strip and Scrap, i.e. paths with ID 2, 4, 5 and 6, is larger than the sum of the executed paths that result in Cleaning and De_Labelling, i.e. paths with ID 1 and 3. Hence, it is most possible for a device to end up in the Strip and Scrap stage due to the product’s parts failure, than to be refurbished in Cleaning and De_Labelling action.

5.6 Summary

The *verification and validation* of the PN automation procedure via evaluation of the recycling IT asset process, reviewed in Chapter 4, has been discussed in this chapter. The PN obtained by the procedure for the recycling IT asset process has been successfully *verified*, by checking its structural and behavioural properties such as boundedness, liveness. Additionally, once timing and probabilistic data was introduced into the corresponding PN transitions, the automated PN generation procedure was *validated* via the PN’s simulation. Initially, the simulation algorithm visually checked the movement of tokens through the PN paths, *validating* that the paths followed the same route as the paths existing in the UML/SysML Activity Diagram provided for the recycling IT asset process. The algorithm has also been *validated* by comparing data from the IT process, with simulation results, which were estimated by following the various PN paths, proving that the PN is a realistic representation of the recycling IT asset process. Finally, an algorithm executed to investigate the process *performance* identified possible deficiencies that exist in the IT process. Therefore, the algorithm used for the automated PN model generation is

correct, complete and develops PN models with accuracy satisfying its intended purpose.

At present, the applicability to one process has been shown, however further generic capability of the method is to be explored. The developed methodology cannot provide valid matrices for any UML/SysML Activity Diagram, since only certain AD attributes such as opaque action, decision, merge, initial, final activity, etc., have been considered. Thus, in the next chapters, this algorithm is extended to a generic methodology that provides transformation rules for mapping all the AD elements into PNs, for any potential AD provided by industry.

6 Advanced Generic Methodology for the Automated Generation of a Petri Net Model

6.1 Introduction

In this chapter, the methodology steps undertaken for the automation of a PN model in Chapter 3 are extended to cover the transformation of any possible AD element into the corresponding PN structure. The non-reviewed AD elements in Chapter 2 are initially introduced and then the methodology steps, i.e. **input-system modelling** and **algorithm-Java database programming**, are investigated. During this investigation, it was observed from the XMI files obtained from ADs containing the newly introduced elements that some of these elements are presented as nested within other elements. It was also observed that once XMI files with nested elements are loaded to the MySQL database, they result in the creation of tables with inadequate data, which, consequently, leads to the generation of inaccurate PNs. Therefore, to overcome this shortcoming and enable the complete data retrieval, some straightforward *transformation of XMI files*, using Extensible Stylesheet Language Transformation (XSLT) documents, is considered.

Thus, the automated modelling is extended with the **system modelling** step taking as input the outputs of the *XMI model transformation* and the **algorithm-Java database programming** step is modified, providing transformations rules for any AD element. Following this advanced generic methodology, the mathematical representation of a PN model is automatically generated and demonstrated with the help of a simple example. The layout of this chapter is illustrated in Figure 6.1.

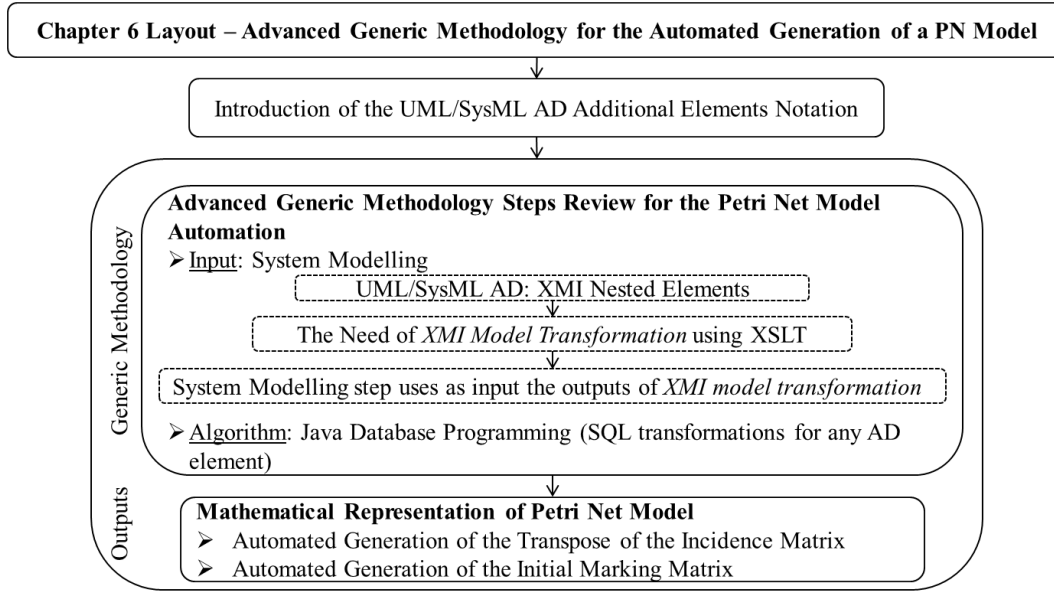


Figure 6.1 Illustration of the Structure of Chapter 6

6.2 Introduction of UML/SysML AD Additional Elements Notation

The purpose of this section is to present the UML/SysML AD elements that have not been examined in Chapter 2 in order to ensure that the methodology developed here works for all AD's developed. The less commonly used AD elements are investigated, explaining their notations with the help of tables, (Tables 6.1 – 6.5).

Table 6.1 includes the **object nodes** used in Activity Diagrams to describe the information flowing between activities, i.e. object flows. An **object node** represents something that stores one or more values that pass from one action to another.

An *expansion region node* represents a nested area that includes actions. This object node has an *input expansion node* that holds the input values received from other actions and an *output expansion node* that holds the output values, which an action produces, shown in the second and third rows of Table 6.1 correspondingly. The *activity parameter node* is used to accept input to/provide output from an activity respectively. The *central buffer node* combines data received from various sources, whereas the *data store node* is used to update existing data by permanently storing the information tokens that enter this node. The last three object nodes in Table 6.1, *input pin*, *output pin* and *value pin*, are used to define input and output parameters that are introduced and produced to and from an activity.

Table 6.1 Notation and Description of Activity Diagram Object Nodes

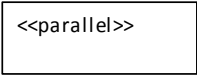


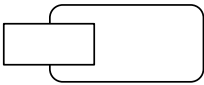

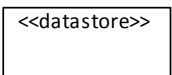




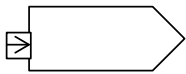
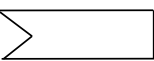
Nodes		
Notation		Description
Name	Symbol	
Expansion Region		There are three modes, i.e. parallel (independent interactions), iterative (interactions occur in order of the elements) and stream (a stream of values flows into a single execution) in which this node can be executed.
Input Expansion Node		Represents a collection of input values.
Output Expansion Node		Represents a collection of output values.
Activity Parameter Node		Accepts inputs to an activity or provides outputs from an activity. This object node is displayed on the border of an activity node.
Central Buffer Node		Accepts tokens from upstream object nodes and passes them along to downstream object nodes. This object node is not connected directly to actions (pin nodes are needed).
Data Store Node		Represents a central buffer node that includes non-transient information. This node is notated as object node using the keyword <<datastore>>.
Input Pin		Receives values from other actions through object flow edges.
Output Pin		Delivers values to other actions through object flows edges.
Value Pin		An input pin node that provides a value to the action to which it is attached.

Table 6.2 lists the **actions** used in Activity Diagrams to describe a single step in an activity.

Table 6.2 Notation and Description of Activity Diagram Actions (Nodes)

Nodes		
Notation		Description
Name	Symbol	
Call Behaviour Action		Represents a behaviour directly, rather than an operation that invokes the behaviour. This can avoid redundant definitions of activities.
Send Signal Action		Represents that a signal is sent to a receiving activity.
Accept Event Action		Represents that an event is received and hence accepted.

The *call behaviour action node* helps to define behaviour from different ADs, referencing (calling) an activity. Once a *send signal action node* is executed, a signal is created and sent to the target node, whereas an *accept event action node* is executed when a specified condition, defined by the user, is met and hence an event occurs.

Table 6.3 lists another type of AD elements, named **structured activity node**, used to create logical groups of activity nodes and edges. These nodes and edges belong only to the structured activity node, meaning they are not shared with other structured activities. This node may contain a group of subordinate nodes, i.e. a set of nodes that create an independent Activity Group. Four structured activity node types, including *simple*, *conditional*, *loop* and *sequence*, have been identified in ADs. The description of each of these four structured activity nodes is included in Table 6.3.

Table 6.3 Notation and Description of Activity Diagram Structured Activity Elements (Nodes)

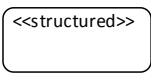
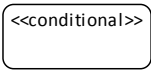
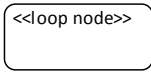
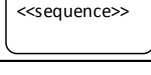
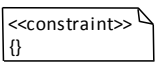
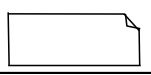
Nodes		
Notation		Description
Name	Symbol	
Simple Structured Activity Node		Represents an ordered arrangement of the activity nodes.
Conditional Node		Represents an arrangement of actions and activities where choice determines which activities are performed.
Loop Node		Represents a repetitive sequence of actions and activities that are included in the object.
Sequence Node		Represents a sequential arrangement of the activity nodes.

Table 6.4 describes the nodes presented as notes in an AD.


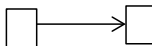
Table 6.4 Notation and Description of Activity Diagram Notes (Nodes)

Nodes		
Notation		Description
Name	Symbol	
Constraint		Is a condition/restriction in natural language text/machine readable language that declares some of the semantics of a node. Precondition constraints specify what must be fulfilled when the behaviour is invoked. Postcondition constraints specify what is fulfilled after the execution of the behaviour is complete, once its precondition was fulfilled before its invocation. Constraints are attached to action nodes.
Comment		Enables the attachment of remarks to nodes.

These nodes, including *constraints* and *comments*, do not carry any semantic force, i.e. do not contain information relevant to PN model transformation, but may contain information that is useful to a modeller. More details about these nodes can be found in the description column of Table 6.4.

Finally, Table 6.5 shows some of the AD **edges**, including exception handlers, object flows and links. An *exception handler edge* is used to show the exception node, i.e. the node in which the edge ends up, may occur during the execution of a process, interrupting its normal execution. Another edge used in ADs is the *object flow* that represents the flow of objects/data from one node to another. Finally, a *link edge* can be used in ADs to make a comment. However, links do not carry any semantic force, but may contain information useful to a modeller.

Table 6.5 Notation and Description of Activity Diagram Edges

Edges		
Notation		Description
Name	Symbol	
Exception Handler		Connects either two nodes or an Interruptible Activity Region and an activity. If the outgoing edge of a node is an exception handler, then this node is executed only if the handler satisfies an uncaught exception. An Interruptible Activity Region is an activity group of nodes in which once a token leaves this region, all tokens and behaviours in this region are terminated.
Object Flow		Connects two elements, with object/data passing through it.
Link	-----	Connects action nodes with comment nodes/constraint nodes.

Therefore, the elements available to be used in an AD have been reviewed and their XMI structure is examined in the following sections, in order to be appropriately transformed into corresponding PN elements.

6.3 Input – System Modelling

6.3.1 Introduction

In this section, for the input-system modelling step, the **format** of XMI files obtained from ADs that include the newly introduced additional AD elements reviewed in

section 6.2, as well as their **behaviour** once they are loaded to the MySQL database, are extensively investigated. Thus, it is noticed that some of these AD additional elements, once they are expressed in XMI format, are expressed as nested within other elements and also when these XMI files are loaded to the MySQL database for further data manipulation, they result in creating tables with missing data, inhibiting the complete PN model generation. Therefore, in the following subsections, it is shown, with the help of simple examples, that XMI files with nested elements cannot directly be loaded to the database, as followed in the methodology in Chapter 3.

To overcome this limitation and generate complete PNs, the *XMI model transformation* concept using XSLT documents is introduced in the following sections. For this model transformation, two XSLT documents are developed and applied to the XMI file (source), obtained from a given AD, transforming this file into two target files, an XMI and an XML. These two target files are then ready to be loaded to the MySQL database for further data manipulation. The target XMI file is used to retrieve nested AD elements such as input/output value nodes, expansion regions, exception handlers, etc. The root AD elements such as the opaque action/decision/accept event action nodes and the control/object flow edges are retrieved from the target XML file. The XMI model transformation, carried out using XSLT documents, is computationally performed in Java and demonstrated with the help of an example.

6.3.2 UML/SysML AD: Review of XMI Nested Elements

In this section, the AD elements identified presenting as nested within others in XMI format are initially discussed and then the structures of these XMI nested elements are investigated using simple examples. During the investigation of these nested structures, the source XMI files are loaded to the MySQL database, checking if the tables created contain all the necessary data from which the PN model is to be generated. An element is called nested if another element exists within this element and each element's start tag (<) has a corresponding end tag (/>) before another element's start tag begins.

The AD elements found presented as nested in XMI format, are listed as follows:

— **Nodes:**

- Expansion region; input expansion; output expansion; and input/output (value) pin, as shown in Table 6.1.
- Send signal action; and accept event action, as shown in Table 6.2.

— **Edge:**

- Exception handler, as shown in Table 6.5.

Three simple examples from previous work of ADs using nested elements, presented in Figures 6.2, 6.4 and 6.6, are introduced in this section. Each example uses different elements of the above list. Hence, in the first example the expansion region, input and output expansion nodes and input and output pin nodes are investigated. In the second example the behaviour of the send signal and accept event action nodes are studied, whereas in the third example, the exception handler edge is considered. The names of the AD symbols, used in these examples, are written in red. Additionally, the elements of each example are numbered in black. The structure of each XMI document, obtained from these examples, is initially examined and then loaded into the MySQL database. The data stored in the MySQL database is reviewed and compared to that should have been stored in the database according to the information included in the initial XMI.

The **first example** of an AD considered, taken from Fowler, 2004, is illustrated in Figure 6.2 and models the different stages taken for publishing a newsletter.

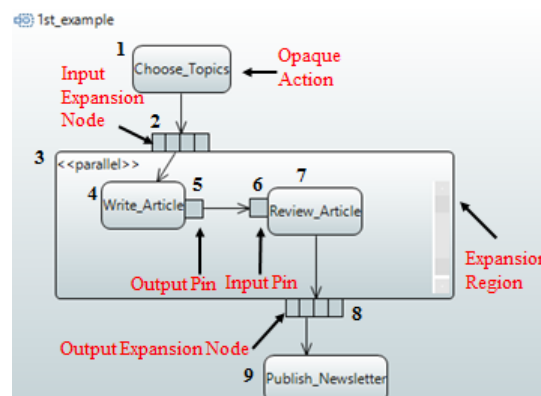


Figure 6.2 AD Example (Fowler, 2004)

The diagram in Figure 6.2 consists of four opaque action nodes ('Choose_Topics' (1), 'Write_Article' (4), 'Review_Article' (7) and 'Publish_Newspaper' (9)), an expansion region (3) with an input expansion node (2) and an output expansion node (8), as well as two pin nodes: an input (6) and an output (5). The expansion region (3) is used to show that actions included in this area occur once for each item in a collection, i.e.

input expansion node (2). All the connectors used in this diagram are control flow edges.

In this first example, a list of topics, presented by the input expansion node (2) in the AD, is created by the ‘Choose_Topics’ node. Each topic of this list corresponds to an input token to the ‘Write_Article’ node. Once each article is written and reviewed, it is added to the output list, presented by the output expansion node (8) in the AD. When the number of output tokens equals the number of input tokens, then the expansion region creates a single token, which is sent to the ‘Publish_Newsletter’ node (9). In this example, the articles are written and reviewed simultaneously, since the keyword used in the expansion region is <<parallel>>, as shown in Figure 6.2.

The XMI document corresponding to the diagram in Figure 6.2 is found in Appendix G, part A. A part of this XMI file for the elements contained in the expansion region (3) is shown in Figure 6.3. From this figure, it can be seen that the two opaque action nodes (4 and 7) related to the expansion region structured node are presented as XMI nested elements, shown by the <structuredNode.../> element, starting in line 1. Additionally, the input pin node (6) which is attached to the ‘Review_Article’ node and the output pin node (5) attached to the ‘Write_Article’ node are also shown as XMI nested elements in their corresponding opaque action nodes.

```

1  <structuredNode xmi:type="uml:ExpansionRegion"
2  xmi:id=" _jM0MsF2YEeC05B8er0jow" name="ExpansionRegion2" mode="parallel"
3  outputElement=" _qNgOQF2YEeC05B8er0jow"
4  inputElement=" _pH_KoF2YEeC05B8er0jow">
5  <node xmi:type="uml:OpaqueAction" xmi:id=" _gbxocF2YEeC05B8er0jow"
6  name="Write_Article" incoming=" _VcMF2gEeC05B8er0jow">
7  <outputValue xmi:type="uml:OutputPin" xmi:id=" _HWo2YF2gEeC05B8er0jow"
8  outgoing=" _MI-zSF2gEeC05B8er0jow" isControlType="true">
9  <upperBound xmi:type="uml:LiteralInteger" xmi:id=" _HWpdcF2gEeC05B8er0jow"
10 value="1"/>
11 </outputValue>
12 </node>
13 <node xmi:type="uml:OpaqueAction" xmi:id=" _oW66EF2YEeC05B8er0jow"
14 name="Review_Article" outgoing=" _fU6K4OBZEeIaM4coD475g">
15 <inputValue xmi:type="uml:InputPin" xmi:id=" _KB7XSF2gEeC05B8er0jow"
16 name="" incoming=" _MI-zSF2gEeC05B8er0jow" isControlType="true">
17 <upperBound xmi:type="uml:LiteralInteger" xmi:id=" _KB7_AF2gEeC05B8er0jow"
18 value="1"/>
19 </inputValue>
20 </node>
21 </structuredNode>

```

Diagram illustrating the XMI Extract for the AD in Figure 6.2. The extract shows two nested opaque action nodes (4 and 7) within a structured node (3). Node 4 is 'Write_Article' and Node 7 is 'Review_Article'. Both nodes have associated input and output pins (5 and 6 respectively) and are connected by control flow edges. The diagram also shows the expansion region (2) which is parallel and contains the two opaque action nodes.

Figure 6.3 XMI Extract for the AD in Figure 6.2

The AD example in Figure 6.2, is loaded into the MySQL database and following step 1.a from the methodology, proposed in Chapter 3, section 3.4.3, the ‘node_xmi’ table is created retrieving values from the XMI file. The part of the ‘node_xmi’ table

corresponding to the section of XMI file shown in Figure 6.3, is presented in Table 6.6. According to the data stored in Table 6.6, the “xmi:type” value shown as *uml:LiteralInteger* in the first row of the table is retrieved from the upperBound element (line 9 in Figure 6.3) which is nested in the ‘Write_Article’ node (4). Similarly, the “xmi:id” value in the first row of Table 6.6 is retrieved, i.e. *xmi:id=“_HWpdcF2gEeeC05B8er0jow”*, from the upperBound element. As is the “outgoing” value, i.e. *“_M1-z8F2gEeeC05B8er0jow”* in the first row of Table 6.6.

Table 6.6 MySQL ‘node_xmi’ Table for XMI in Figure 6.3

id	xmi:type	xmi:id	name	incoming	outgoing
1	uml:LiteralInteger	_HWpdcF2gEeeC05B8er0jow	Write_Article	_VdMF2gEeeC05B8er0jow	_M1-z8F2gEeeC05B8er0jow
2	uml:LiteralInteger	_KB7_AF2gEeeC05B8er0jow	Review_Article	_M1-z8F2gEeeC05B8er0jow	_fU6K40BZEeeIaM4coD475g

Hence, the values of the “xmi:type” and “xmi:id” attributes of the ‘Write_Article’ (4) node, i.e. *xmi:type=“uml:OpaqueAction”* and *xmi:id=“_gbxocF2YEeeC05B8er0jow”*, given in line 5 in Figure 6.3 are missing. These two values should be in line 2 in Table 6.6. This is also true for the values of the “xmi:type” and “xmi:id” attributes of the outputValue (5) nested element, i.e. *xmi:type=“uml:OutputPin”* and *xmi:id=“_HWO2YF2gEeeC05B8er0jow”*, given in line 7 in Figure 6.3. This missing data results in the algorithm not identifying the sequence of nodes and edges that exist in the AD, leading to the generation of inaccurate PN models.

The **second example** of an AD, taken from the literature (MSDN Microsoft, 2017) and presented in Figure 6.4, describes the steps required to complete an order. This diagram consists of two opaque action nodes (‘Create_Order’ (1) and ‘Close_Order’ (11)) and one send signal action (‘Send_Invoice’ (5)) followed by an accept event signal (‘Receive_Payment’ (7)). Additionally, two input pin nodes (4 and 10) and two output pin nodes (2 and 8) are used in this AD, as shown in Figure 6.4. One control flow edge, numbered by 6, connects the send and event signal action nodes, whereas the other two connectors, i.e. (3) and (9), used in the diagram are object flow edges. Guard and weight conditions have been defined for the object flow edges (3 and 9) as illustrated in Figure 6.4. A guard condition, shown in square brackets, evaluates to true for every token that is offered to pass along the edge. A weight condition, shown in curly brackets, defines the number of tokens that can flow along that connector. In this example, the number of tokens has been defined to be one. In the AD in Figure 6.4, once the order is ready a signal/message is sent to the send signal action node and

then the invoice for the order is sent to the accept event action that waits for the signal/message in order to receive the payment. Once the process of the payment is complete, the order closes.

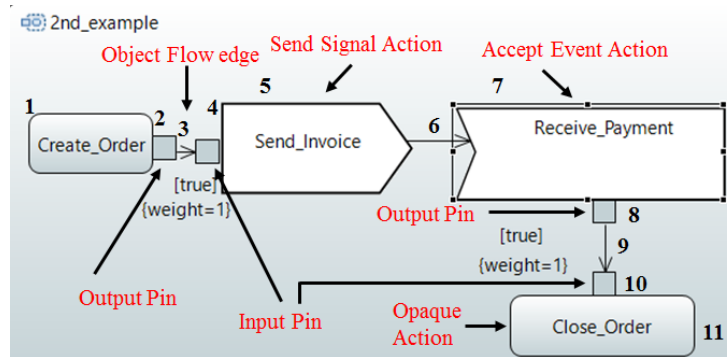


Figure 6.4 AD Example (MSDN Microsoft, 2017)

The XMI document corresponds to the diagram in Figure 6.4 is found in Appendix G, part A. A part of this XMI document for the ‘Create_Order’ (1) opaque action node and the ‘Send_Invoice’ (5) send signal action node and their nested elements is shown in Figure 6.5. From the XMI in Figure 6.5, it is seen that the element created for the ‘Create_Order’ (1) node, given in the first line in Figure 6.5, has a nested output value element for the ‘OutputPin’ (2) node, presented in the third line in Figure 6.5.

```

1  <node xmi:type="uml:OpaqueAction" xmi:id="_NBPV0F2tEeeC05B8er0jow"
2  name="Create_Order">
3    <outputValue xmi:type="uml:OutputPin" xmi:id="_r6HugF2tEeeC05B8er0jow"
4    name="" outgoing="_4BTivF2tEeeC05B8er0jow"/>
5  </node>
6  <node xmi:type="uml:SendSignalAction" xmi:id="_XXc7sF2tEeeC05B8er0jow"
7  name="Send_Invoice" outgoing="_mSSjUF2tEeeC05B8er0jow"
8  signal="_auo58F2tEeeC05B8er0jow">
9    <target xmi:type="uml:InputPin" xmi:id="_c2xrIF2tEeeC05B8er0jow"
10   incoming="_4BTivF2tEeeC05B8er0jow" type="_auo58F2tEeeC05B8er0jow">
11     <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_c2xrIV2tEeeC05B8er0jow"
12     value="1"/>
13     <upperValue xmi:type="uml:LiteralUnlimitedNatural"
14     xmi:id="_c2xrJl2tEeeC05B8er0jow" value="1"/>
15   </target>
16 </node>

```

Figure 6.5 XMI Extract for the AD in Figure 6.4

Additionally, the XMI element created for the ‘Send_Invoice’ (5) node, given in line 6 in Figure 6.5, has a nested target element, in line 9, which corresponds to the input pin node, numbered by 4, in the AD. This input pin node appears as being attached to the ‘Send_Invoice’ (5) node in Figure 6.4. It is also seen that the input pin node (4) has two nested lower and upper value XMI elements in lines 11 and 13 respectively.

Using the methodology developed in Chapter 3, the XMI file for the second AD example in Figure 6.4 is loaded into the MySQL database. For the part of the XMI file presented in Figure 6.5, the ‘node_xmi’ table is created and shown in Table 6.7. From this table, it is seen that in the case of nested XMI elements in the example considered, the SQL code retrieves and stores in the ‘node_xmi’ table the values of the attributes that belong to the nested XMI elements, instead of retrieving the values of the attributes belonging to the node XMI elements.

Table 6.7 MySQL ‘node_xmi’ Table for the XMI in Figure 6.4

id	xmi:type	xmi:id	name	incoming	outgoing
1	uml:OutputPin	_r6HugF2tEeeC05B8er0jow	Create_Order	NULL	_4BTtwF2tEeeC05B8er0jow
4	uml:LiteralUnlimitedNatural	_c2xrTl2tEeeC05B8er0jow	Send_Invoice	_4BTtwF2tEeeC05B8er0jow	_mSSjUF2tEeeC05B8er0jow

This is the same as was seen in the first example presented in this section. For example, in the first row of Table 6.7, it is seen the value stored in the “xmi:id” column, i.e. [_r6HugF2tEeeC05B8er0jow](#), corresponds to the “xmi:id” value of the nested outputValue element (2), found in line 3 in Figure 6.5, instead of the “xmi:id” value of the node element (1), i.e. [_NBPV0F2tEeeC05B8er0jow](#), found in line 1 in Figure 6.5. Therefore, the “xmi:id” value of the node element is missing and the identification of the sequence of nodes and edges for an accurate PN model to be generated is not possible.

The **third example** of an AD (Sparx Systems, 2018) in Figure 6.6 models the procedure once a user cancels their account. This diagram consists of two opaque action nodes (‘User_Cancels’ (1) and ‘Account_Cancelled’ (4)) and one input (value) pin node (‘InputPin1’ node (3)). An exception handler edge (2) is used to connect the ‘User_Cancels’ node (1) with the ‘InputPin1’ node (3) which is attached to the ‘Account_Cancelled’ node (4), as shown in Figure 6.6. In Figure 6.6, a part of a process is presented, showing that if the user cancels the process the account will be cancelled and potentially the process could terminate.

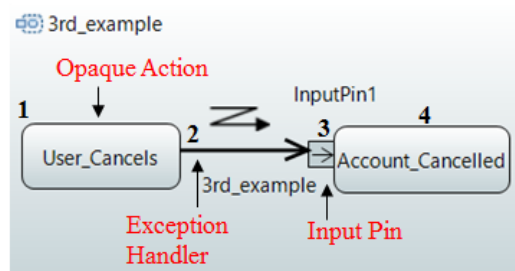


Figure 6.6 AD Example (Sparx Systems, 2018)

The XMI file corresponds to the diagram presented in Figure 6.6 is found in Appendix G, part A. A part of this XMI file for the two opaque action nodes (1 and 4) is shown in Figure 6.7. From the XMI document in Figure 6.7, it can be seen that the XMI element created for the ‘User_Cancels’ (1) node illustrated in line 1, has a nested handler element, shown in line 3 that corresponds to the exception handler edge (2) in Figure 6.6. Although the exception handler symbol belongs to the edges of ADs, it is shown as a nested element to the node from which it starts, i.e. ‘User_Cancels’ (1) node in Figure 6.6.



Figure 6.7 XMI Extract for the AD in Figure 6.6

Besides the “xmi:type” and “xmi:id”, the exception handler edge introduces the following attributes:

- **“exceptionInput”** The value of this attribute is the “xmi:id” value of the node in which the exception handler edge ends up. For the example in Figure 6.6, the “exceptionInput” value is taken from the “xmi:id” attribute of the ‘InputPin1’ (3) element.
- **“exceptionType”** The value of this attribute is a unique value given from the software package.
- **“handlerBody”** The value of this attribute is the “xmi:id” value of the node in which the inputValue element, mentioned in the “exceptionInput” attribute is nested to. For the example in Figure 6.6, the “handlerBody” value is taken from the “xmi:id” attribute of the ‘Account_Cancelled’ (4) node.

Additionally, according to Figure 6.7, the second XMI node in line 8 created for the ‘Account_Cancelled’ (8) node, as shown in Figure 6.6, has a nested inputValue element given in line 10 in Figure 6.7. This element corresponds to the ‘InputPin1’ (3) node, shown attached to the ‘Account_Cancelled’ (4) node in Figure 6.6.

The XMI file, obtained for the AD in Figure 6.6, is loaded into the MySQL database and for the part of the XMI file presented in Figure 6.7 the ‘node_xmi’ table is created as shown in Table 6.8.

Table 6.8 MySQL ‘node_xmi’ Table for XMI in Figure 6.7

id	xmi:type	xmi:id	name	incoming	outgoing
1	uml:ExceptionHandler	_wcswkF2nEeeC05B8er0jow	User_Cancels	NULL	NULL
2	uml:LiteralInteger	_uQxrkF2nEeeC05B8er0jow	InputPin1	NULL	NULL

From this table, it is seen that, as for the previous two examples, in case of nested XMI elements the SQL code retrieves and stores in the ‘node_xmi’ table the values of the attributes that belong to the nested elements, instead of retrieving the values of the attributes belong to the node elements. Hence, the “xmi:id” values of the nodes are not retrieved, preventing the accurate PN model generation.

According to the XMI documents review, it can be concluded that the XMI root and nested elements include necessary information for the sequence of elements included in ADs and, by extension, for the generation of PN models. From the examples reviewed, it is also concluded that XMI node elements cannot be properly loaded into MySQL once nested elements also exist in the XMI documents. Consequently, the algorithm developed earlier creates tables with missing information and generates incomplete models since vital information for the sequence of elements is missing. Therefore, the manipulation of the structure of XMI documents is necessary in order to allow the retrieval of the XMI attributes values for each AD element. This manipulation is performed by transforming the XMI document obtained from an AD into two documents, an XMI with different structure from the initial and an XML. These two files enable the complete retrieval of all the values of the AD elements, once loaded into the MySQL database

In the following two sections, the XMI model transformation using XSLT documents is explained in detail using examples to illustrate the concept.

6.3.3 The Need of XMI Model Transformation using XSLT

The XMI model transformation that follows specific rules defined in XSLT files is necessary for the complete retrieval of data included in ADs (input-system modelling step). For the proposed methodology, the XMI model transformation concept, carried out in Java, refers to the process where an XMI file (source model), obtained from the Activity Diagram, is transformed into two XMI/XML files (target models). Two

XSLT files have been developed in order to obtain one XMI with the elements that are nested to the nodes of the initial XMI file and one XML with the nodes and edges included in the initial XMI file. The two XSLT documents provide the rules so that all the necessary information that exists in the initial XMI file is properly structured into two output files (XMI and XML). The structure of these two output files is formed so that all the necessary information for the sequence of AD elements is properly loaded into the MySQL database enabling the automated generation of PN models, without missing any data.

It is noted that XMI is a specific application of XML, meaning that XMI documents can only be used for XMI purposes, whereas XML documents can be used for all XML applications, including XMI. In other words, all XMI files are XML, but not all XML files are XMI. Additionally, the XSLT that belongs to the XML family is used to perform XML transformations allowing the user to specify the desired structure and content of the output file. XSLT documents can reorder XML elements, add new elements and decide which elements should be displayed or omitted. The XSL transformation process is based on specific template rules defined by the user.

In the two following subsections, the model transformation of the XMI file obtained from an AD to two models, in XMI and XML format respectively, using XSLT files, is described.

6.3.3.1 First XMI Model Transformation using XSLT

For the first XMI model transformation, the rules of the developed XSLT file are applied to the XMI root nodes and edges and the elements presented as nested within the XMI nodes (XMI file obtained from the input AD). The structure of the XSLT document used for the first XMI model transformation starts with a template that contains processing instructions and commands for the XMI nodes and edges that match the specified XPath, a query language for selecting nodes from an XMI/XML document, expression. In this model transformation, the template defined in the XSLT file matches any child element of the XMI root packaged Element and then selects all the attributes of these elements as well as any existing immediate children of these elements. The packaged Element is used in XMI to group all the elements included in the AD, providing a hierarchical organisation of elements such as nodes, edges, etc.

The first XSLT document including explanatory comment is found in Appendix G, part B.

The XSLT, discussed in this section, generates an XMI file (with different structure from the initial XMI file) from which the elements found as nested within XMI nodes can be properly loaded to the MySQL database. This new developed XMI file consists of the root edges (<edge.../>) and nodes (<node.../>) from the initial XMI file, as well as the elements appeared as nested within XMI nodes (in the initial XMI file) such as <inputValue.../>, omitting any further nested elements, such as <upperBound.../>. Although the developed XMI file contains the data for edges and nodes, these elements cannot always ensure their proper loading to the MySQL database since in the case of a node holds a nested element, only the values of the nested element are loaded to database tables, omitting the values of the root element (node).

The attributes of the XMI elements, such as “xmi:type”, “xmi:id”, etc., and their corresponding values which are required to be retrieved, during the first XMI model transformation, to enable the automated PN model generation, are identified. For each element presented as nested within an XMI node, the attributes that need to be loaded to the MySQL database from the developed XMI file, are:

- For the *inputValue* nested elements in the XMI file created from an input pin node in an AD and for the *target* nested elements, attached to accept event action nodes, the retrieved attributes required are: “xmi:type”; “xmi:id”; “name” (if available); and “incoming”.
- For the *outputValue* nested elements created from an output pin node in an AD and for the *result* nested elements, attached to send signal action nodes, the retrieved attributes required are as above including the “outgoing”.
- For the *handler* nested elements, created from the handler exception edges in an AD, the retrieved attributes required are: “xmi:type”; “xmi:id”; “exceptionInput”; and “handlerBody”.

The first XMI transformation was carried out in Java using the javax.xml.transform package. This package defines the generic APIs for processing transformation instructions and performs a transformation from source to result. For this first XMI model transformation, the target XML file (new_xmi_file.xml) is generated by taking

as source file the XMI file (Activity_Diagram.uml), obtained from AD, and applying the first XSLT file (xmi2xmi.xsl), presented in this section. The Java code, used for this transformation, including explanatory comments is presented in Appendix G, part C.

The developed XMI document, generated from the application of the XSLT discussed in this section, includes all the XMI nodes and edges, except for the upperBound elements as well as the “name” values from the inputValue and outputValue elements, as they are presented in an AD. None of the attributes of upperBound element is necessary for the generation of the mathematical representation of PN and hence, they are not retrieved to the XMI file generated from the model transformation. Similarly, the text values of the “name” attributes from the XMI inputValue and outputValue elements are also omitted to be retrieved into the developed XMI file. This is done so that later SQL code to be able to load to the MySQL database the “name” values from the root nodes (rather than the “name” values of the inputValue and outputValue elements) that are necessary to achieve the sequence of nodes and edges. Therefore, this transformation generates an XMI file from which the elements presented as nested within the XMI nodes such as inputValue, outputValue, handler, etc. will be retrieved in the MySQL database for the automated generation of PN.

6.3.3.2 Second XMI Model Transformation using XSLT

For the second XMI model transformation, the rules of the second XSLT file are only applied to the XMI root nodes and edges (XMI file obtained from the input AD). Applying this second XSLT, an XML file is generated that contains the attribute values of all the XMI root nodes and edges. These attribute values can then be properly loaded into the MySQL database.

The structure of this second XSLT file starts with a template that matches any child element of the XMI root element such as packagedElement, nodes, edges, etc. and then selects all the attributes and the corresponding values of these child elements. Then two templates are introduced for the edges and nodes respectively. Each template selects the attributes and the corresponding values of the edge/node element and transforms the attributes to elements. The attributes for each root element that need to be loaded to the MySQL database from this developed XML file, are:

- For the *nodes* that exist in an AD, the retrieved attributes required are: “xmi:type”; “xmi:id”; “name” (if available); “incoming” (if available); and “outgoing” (if available).
- For the *edges* that exist in an AD, the retrieved attributes required are: “xmi:type”; “xmi:id”; “name” (if available); “target” (if available); and “source” (if available).

The node/edge attributes in the XML file are followed by the corresponding values according to the initial XMI document. The second XSLT document including explanatory comment is found in Appendix G, part B

As the first XMI model transformation, the second transformation, discussed in this section, is performed in Java using again the javax.xml.transform package, takes as input the XMI file of the AD (Activity_Diagram.uml) and applying the XSLT file discussed in this section (xmi2xml.xsl), it develops the target XML document (new_xml_file.xml). The Java code, used for this transformation found in Appendix G, part C, applies the same steps as followed by the Java code described in section 6.3.3.1 for the first XML transformation.

The XML file developed from the second XMI model transformation includes all the data for the nodes and edges as presented in an AD that would be used in the next step of the methodology for the automated generation of PN.

6.3.4 Application of the XMI Model Transformations to a Simple AD

Example

In this section, the XMI model transformation concept, reviewed in section 6.3.3, is explained explicitly with the help of the AD presented in Figure 6.4 for a simple process. In the first XMI model transformation, the first XSLT document discussed in section 6.3.3.1 is applied to the initial XMI file obtained from the AD in Figure 6.4 and the XMI file found in Appendix G, part D is generated. This new XMI file includes the root XMI nodes and edges, as well as the elements appeared as nested within nodes in the initial XMI file. Part of the XMI file, developed for the AD in Figure 6.4, for the input pin (10) node that is nested to the ‘Close_Order’ node is illustrated in Figure 6.8. The text next to Figure 6.8 shows how the text values of the attributes of the inputValue node are related to the AD (Figure 6.4). The first XMI file

developed in this section for the AD shown in Figure 6.4 consists of four elements (including outputValue, inputValue, result and target) presented as nested within other nodes in the initial XMI file.

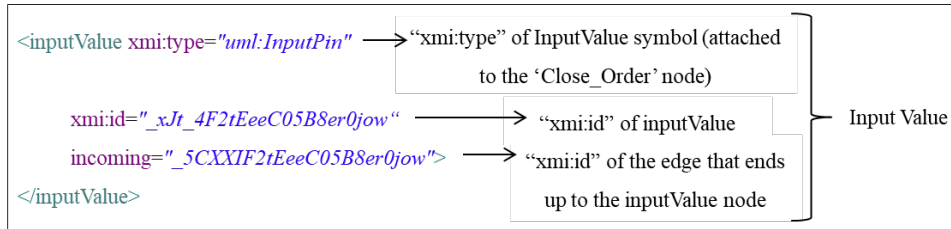


Figure 6.8 Part of the XMI File developed form the 1st XMI Model Transformation

The number of nested elements of this XMI file is the same with those found in the XMI document obtained from the AD example in Figure 6.4.

Additionally, in the second XMI model transformation, the second XSLT document discussed in section 6.3.3.2 is applied to the initial XMI file obtained from the AD in Figure 6.4 and the XML file found in Appendix G, part D is generated. Applying the second XSLT document, each child node/edge element from the source XMI file is transformed into a root node/edge element. The attributes of a node/edge child element (“incoming”, “name”, etc.) are then transformed into sub-elements (incoming, name, etc.) of the root node/edge element. Then, the two templates explained in section 6.3.3.2 are applied to the XMI nodes and edges. The first template is applied to the XMI nodes asking for the “xmi:type”, “xmi:id”, “incoming”, “name” and “outgoing” attributes and their corresponding values, whereas the second template is applied to the XMI edges asking for the “xmi:type”, “xmi:id”, “id”, “name”, “target” and “source” attributes and their corresponding values. The developed XML file contains the values of the edges and nodes attributes as presented in the initial XMI file. These XML values will later be loaded to the MySQL database for the mathematical PN generation.

Part of the XML file, developed for the AD in Figure 6.4, for the ‘Receive_Payment’ node and the first edge, labelled by 3 in the AD, included in the XML file, is presented Figure 6.9. The text next to Figure 6.9 shows how the XML elements are related to the AD (Figure 6.4). The XML file developed in this section for the AD in Figure 6.4 consists of seven elements from which three are edges and four are nodes.

The size of this XML file is the same as the initial XMI file obtained from the AD in Figure 6.4.

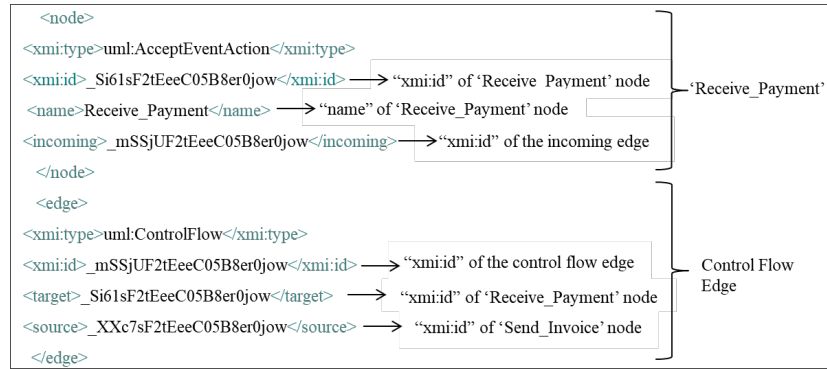


Figure 6.9 Part of the XML File developed from the 2nd XMI Model Transformation

Having completed the two XMI model transformations, the two output files, in XMI and XML formats respectively, used as inputs in the system modelling step, are then ready to be loaded to the MySQL database to be further manipulated and organised in such a way that the mathematical form of PN models can be generated (algorithm-Java database programming step).

6.4 Generic Algorithm – Java Database Programming

6.4.1 Transformation Rules

In this section, the transformation rules for the AD additional elements introduced in section 6.2 are identified. In order to help with the identification of these transformation rules, the PNs, for the ADs shown in Figures 6.2, 6.4 and 6.6, have been manually developed and presented in Figures 6.10, 6.11 and 6.12 correspondingly. The PNs in Figures 6.10 and 6.11 consist of four transitions and three places, whereas the model shown in Figure 6.12 contains two transitions and one place.



Figure 6.10 PN Model developed for the AD in Figure 6.2



Figure 6.11 PN Model developed for the AD in Figure 6.4

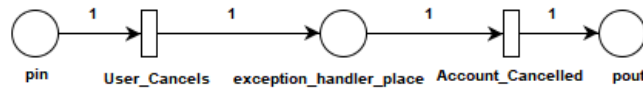


Figure 6.12 PN Model developed for the AD in Figure 6.6

Five additional AD examples and their corresponding manually developed PNs have also been studied in this section, as seen in Figures 6.13 – 6.17. The ADs illustrated in these figures contain elements that have been introduced in section 6.2, but have not been considered in the three AD examples reviewed in Figures 6.2, 6.4 and 6.6. Hence, in order to define a complete set of relationships between the AD and PN elements, the following examples are investigated.

The AD in Figure 6.13 consists of three opaque action nodes, presented as ‘Assemble_Car’, ‘Sell_Cars’ and ‘Refresh_Company_Cars’ respectively, and a central buffer node, shown as ‘Car’ in the diagram. The AD also contains three pin nodes (two inputs and one output) attached to the opaque action nodes, indicated as ‘Car’. In this AD the central buffer node, ‘Car’, has been placed between the manufacturing plants (‘Assemble_Cars’) and the dealers (‘Sell_Cars’ and ‘Refresh_Company_Cars’) in order to arrange deliveries, or sorting of the manufactured cars. The new elements in this example are the central buffer node and the pin nodes. The corresponding PN model developed for this AD, illustrated on the right side of Figure 6.13, consists of three transitions and a place.

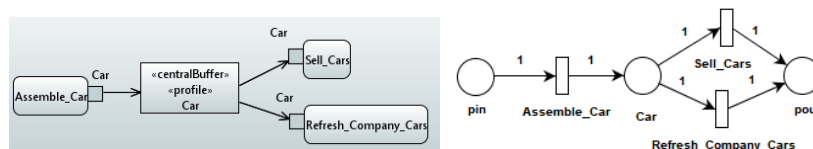


Figure 6.13 AD (Central Buffer Node) and Corresponding PN Model (Pilone & Pitman, 2005)

The AD in Figure 6.14 contains two opaque action nodes, presented as ‘Make_Sale’ and ‘Ship_Item’, and a data store node, shown as ‘Customer_Database’. In this AD, once a sale is made, information regarding this sale is stored in the customer database, before the sold item is shipped. The corresponding PN model developed for this AD, shown on the right side of Figure 6.14, consists of two transitions and a place.

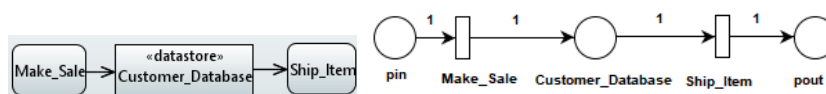


Figure 6.14 AD (Data Store Node) and Corresponding PN Model (Pilone & Pitman, 2005)

The AD in Figure 6.15 consists of an initial node, two opaque action nodes ('Visit_Page' and 'Login'), a decision node ('Decision') and a call behaviour action node ('Register'). This AD shows that once a customer visits the page, they can login if they have an account (registered), or they can be registered creating a new account. The call behaviour action node, which belongs to the additional AD elements, shows that an external activity that includes a sequence of actions is called. Thus, the 'Register' call behaviour action node shown in the AD in Figure 6.15 is externalising into another AD. The PN model developed for this AD, viewed on the right side of Figure 6.15, consists of four transitions and four places.

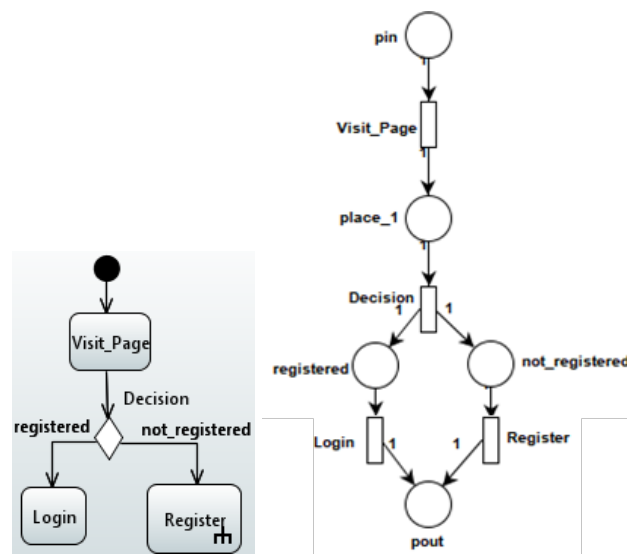


Figure 6.15 AD (Call Behaviour Action) and Corresponding PN Model (Söding, 2009)

The AD example in Figure 6.16 has been considered to show the use of incoming ('Wood') and outgoing ('Ream') activity parameter nodes, which belong to the additional AD elements. Besides the parameter nodes, three opaque action nodes ('Create_Pulp', 'Press_Paper' and 'Package_Reams') are contained in this diagram. The incoming parameter node ('Wood') shows that wood is fed into a paper production procedure, whereas the outgoing parameter ('Ream') shows that paper is finally produced. The corresponding PN model, manually developed for this AD as shown on the right side of Figure 6.16, consists of three transitions and four places.

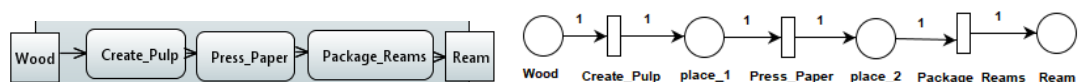


Figure 6.16 AD (Activity Parameter Node) and Corresponding PN Model (Pilone & Pitman, 2005)

Finally, the AD in Figure 6.17 is considered to show the application of a structured activity node. The structured node, included in the additional AD elements, can be simple, as in Figure 6.17, or conditional/looped/sequential. According to each case, the nodes included within the structured activity node express different behaviours. Considering Figure 6.17, the AD consists of three opaque action nodes ('Check_Order', 'Fill_Order' and 'Close_Order') which are contained into the structured activity node, an exception handler edge ('NoFillReason'), included in the additional AD elements, and an input pin followed by an opaque action node ('Notify_Buyer'). This diagram examines a process order in which, according to the sequence of the three actions, presented within the structured activity node, the order closes only if it is checked and filled. If the second action of the structured node, i.e. 'Fill_Order', is not performed, then the 'NoFillReason' exception is trapped by the structured node and the buyer is notified that something is wrong with the order. The manually developed PN model for this AD is illustrated on the right side of Figure 6.17, consisting of four transitions and three places.

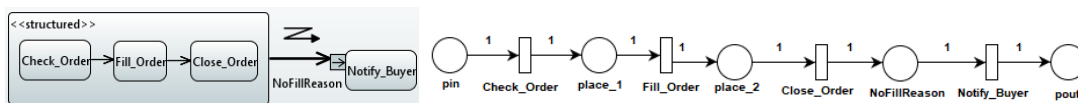
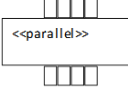
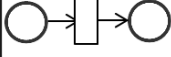
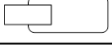











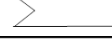

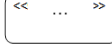







Figure 6.17 AD (Structured Activity Node) and Corresponding PN Model (Bock, 2005)

From the comparison of the manually developed PNs and the given ADs for the examples shown in this section, the relationships between the AD and PN notation and symbols are shown in Table 6.9. According to the first row of Table 6.9, the input and output expansion nodes are mapped into PN places, whereas the expansion region into a PN transition. Once the expansion region contains other nodes, these are transformed into PN transitions and the PN transformation of the expansion region is omitted. As can be seen, the activity parameter, central buffer and data store nodes as well as the AD edges, including exception handlers and object flows, are transformed into PN places. Additionally, the structure of an opaque action node with two pin nodes, an input and an output, as well the call behaviour nodes are mapped into PN transitions. Finally, the send signal action with one input pin node attached, the accept event action and the structured activity nodes are also transformed into PN transitions. Therefore, these are the mapping rules followed in the Java database programming step in the next section.

Table 6.9 Relationships between the AD and PN Notation and Symbols

UML 2.0 Activity Diagram		Petri Net	
Name	Symbol	Name	Symbol
Expansion Region with Input and Output Expansion Nodes		Two places and one transition	
Activity Parameter Node		Place	
Central Buffer Node		Place	
Data Store Node		Place	
Opaque Action Node with Input and Output Pin Nodes or Value Nodes		Transition	
Call Behaviour Action		Transition	
Send Signal Action with Input Pin Node/Value Node		Transition	
Accept Event Action		Transition	
Structured Activity Node (Simple/Conditional/Loop/Sequence)		Transition	
Exception Handler Edge with Input Pin Node/Value Node		Place	
Object Flow Edge with Input and Output Pin Nodes/Value Nodes		Place	

6.4.2 Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix

The SQL code proposed in Chapter 3 for the automated generation of the mathematical form of a PN model, needs to be extended to allow the transformation of the additional AD elements, discussed in section 6.2, and hence to provide a generic applicability. The extended SQL code, explained in this section, is used to retrieve, manipulate and store the data of the two input files (XMI and XML), obtained from the XMI model transformation using as source file the XMI from a UML/SysML AD. These two input files are loaded to the MySQL database using SQL statements. The purpose of each step, used in this extended (generic) SQL code, is introduced in the flowchart illustrated in Figure 6.18. Comparing the flowcharts in Figures 3.6 and 6.18, it can be seen, this new generic code follows the same concept as the code proposed in Chapter 3, with some amendments and newly introduced steps for the transformation of the AD additional elements. The numbering of the steps that have been modified or these that are newly introduced is presented in the flowchart in red.

In this flowchart, these 15 steps can be categorised as follows:

- **Retrieve data (steps 1.a and 1.b):** In this first step, two tables are created, one for the XMI attributes of the AD exception handlers, results, targets and the XML elements of the AD nodes (step 1.a) and one for the XML elements of the AD (step 1.b). For the first table, the data for the exception handlers, results and targets is retrieved from the XMI file obtained from the first XMI model transformation. For the same table, the data for the nodes, including opaque action, decision, merge, fork, join, initial, final activity, final flow, central buffer, data store, activity parameter and call behaviour nodes, is retrieved from the XML file obtained from the second XMI model transformation. Finally, for the second table, the data for the edges including control and object flows is also retrieved from the XML file.
- **Separate multiple edges (steps 2 – 5):** These four steps are identical with the corresponding steps proposed in the code in Chapter 3.
- **Find the sequence between AD elements (steps 6 – 12):** The values stored in the “xmi:type” column of the table created in step 5 are scanned and for each different type identified a table is created. The code creates seven tables: (i) one for the opaque action, decision, merge, fork, join, accept event, send signal and call behaviour action nodes (step 6); (ii) one for the initial nodes (step 7); (iii) one for the activity final and final flow nodes (step 8); and (iv) four additional for the exception handlers, activity parameters, central buffers and data stores (step 9). The names of the incoming and outgoing edges to and from the nodes stored in the table created in step 6 are identified with the help of the table created in step 1.b and they are stored in this table (created in step 6). Additionally, the names of the nodes placed before and after the AD elements stored in the tables created in steps 7 – 9 are also identified with the help of the tables created in steps 1.b and 5 and stored in these tables (developed in steps 7 – 9). Steps 7 and 8 are identical to the corresponding steps proposed in the code in Chapter 3 for the AD initial and final nodes. In steps 10 and 11, two new tables are created, one for the *object nodes* such as input/output/value pin nodes, and one for the *expansion elements*, including expansion regions and input/output expansion nodes. These two newly introduced tables retrieve the value attributes from the XMI file (step 1.a),

used as input in this code. The nodes/edges found in the AD before and after these examined elements are tracked using the data stored in the tables created in steps 1.b and 5, and replaced in the tables created in steps 10 and 11. Hence, each row of the tables created in these steps 6 – 11 includes the name of a node, the name of the element that exists before this node (if available) and the name of the element that exists after this node (if available). Finally, the tables obtained from these six steps are unified, creating a new table in step 12.

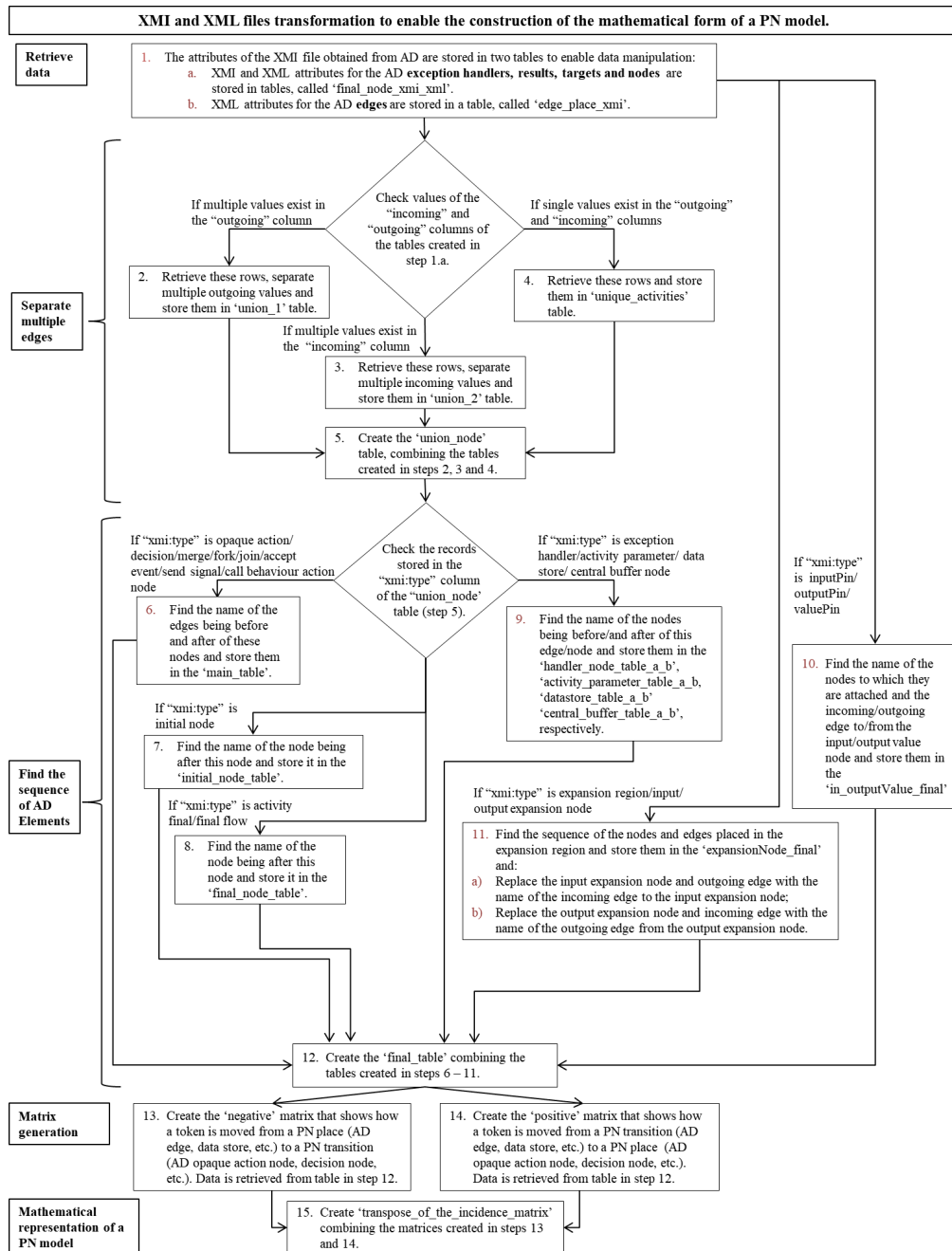


Figure 6.18 Flowchart for the steps followed for the Generic Automated Generation of the Mathematical Representation of a PN Model

- **Matrix generation (steps 13 - 14):** Two tables in the form of matrices are created, retrieving the connectivity information from the table developed in step 12, as proposed for the similar set of steps in Chapter 3. The matrix created in step 13 shows connections of AD edges/initial/final (activity and flow)/exception handler/central buffer/data store/activity parameter nodes (PN places) to AD opaque action/decision/merge/join/fork/send signal/accept event/call behaviour action nodes (PN transitions). Similarly, the matrix generated in step 14 shows connections of AD opaque action/decision/merge/join/fork/send signal/accept event/call behaviour action nodes (PN transitions) to AD edges/initial /activity final/final flow/exception handler/central buffer/data store/activity parameter nodes (PN places).
- **Mathematical representation of a PN model (step 15):** In this final step, the PN is generated by combining the two matrices developed in steps 13 and 14 respectively, as also proposed in the code in Chapter 3.

The SQL code followed for the automated PN generation, found in Appendix H, is thoroughly explained and discussed in the next chapter with the help of two real-life scenarios.

6.4.3 Algorithm – Java Database Programming – Petri Net Initial Marking Matrix

For the complete generation of the PN's mathematical form, the initial marking matrix is also required. The automated generation of the initial marking has already been discussed in Chapter 3, assuming that only one item exists in a system/process. However, in this section, the algorithm for the initial marking generation is extended allowing any number of items to be considered. The extended SQL code for the initial marking generation takes as input an Excel file with two columns. In the first column, the names of the AD nodes are stored and in the second column the number of items each AD node contains before the system/process execution, are stored.

The procedure followed for the PN initial marking generation, which is an extension of the step outlined in Chapter 3 for the initial marking generation, for generic cases, is outlined below:

1. **Generate the initial marking of a PN model.** In this step, a table named ‘initial_marking_final’ is created consisting of the places of PN models (stored in the “activity” column) and the number of tokens (stored in the “process_number_of_devices” column), as explained in Chapter 3. Once the structure of the table is created, this extended SQL code identifies in the “initial_marking” column of a given Excel file, the rows that contain numbers equal to/greater than one. For these rows, the data stored in the “activity” column of this file is retrieved, and then the AD elements found before this retrieved data are tracked, using the first column of the matrix, created in step 15. Finally, for these tracked elements, the corresponding values found in the ‘initial_marking’ column of the Excel table are stored in the ‘process_number_of_devices’ in the ‘initial_marking’, whereas in all the other rows of this column value ‘0’ is inserted.

The code developed for the automated generation of the PN initial marking matrix can be found in Appendix I.

6.5 Summary

The generic novel methodology, proposed in this chapter for the automated PN model generation, applying a Java database (MySQL) algorithm, extends the methodology proposed in Chapter 3 and contributes to knowledge through the combination of the following:

- **Java database algorithm**, as explained in Chapter 3.
- **Fully automated PN model generation capability**: as explained in Chapter 3.
- **Generic domain applicability**: as explained in Chapter 3.
- **Software independence**: as explained in Chapter 3.
- **Generic applicability and scalability**: The extended methodology for the automated PN model generation provides rules for the transformation of any AD element to the corresponding PN structure, broadening the range of applications.

In the following chapter, to further explore the automated PN model capability, the proposed generic methodology is demonstrated by its applicability to two industrial

cases, a system and a process. Additionally, according to the methodology followed in Chapter 5 for the verification and validation of PN models, the correctness of the generic advanced algorithm for the PN automation procedure discussed in this chapter is also checked.

7 Application of the Generic Automated Petri Net Model Generation Methodology to Real-Life Scenarios

7.1 Introduction

In this chapter, the generic methodology discussed in Chapter 6 for the automated generation of a PN model is applied to two real-life scenarios, a production system and an online shopping process, to demonstrate its applicability and functionality to both systems and processes. A description of each scenario, with the help of the corresponding AD, retrieved from the literature, is initially introduced. The AD provided for the production system includes all the basic AD elements discussed in Chapter 3, whereas the online shopping process includes, in addition to the basic elements, the additional AD elements, introduced in Chapters 6. Besides the AD elements, these two scenarios consider features widely associated with complexity, such as control loops, a large number of components/activities and dependent (concurrent/parallel) events, and hence they have been selected to check the correctness and the functionality of the algorithm introduced in Chapter 6. The mathematical and graphical representations of the PN models for these two scenarios are automatically generated and then the correctness of the automation end models is explored by verifying and validating the method.

7.2 Production System

7.2.1 Process Description

A production system, taken from the literature (Villani et al., 2007), is used as an example to illustrate the applicability of research to systems. The production system shown in Figure 7.1 produces two products (P1 and P2). These two products are created from the mixing of a common base (B) with one of the two colouring

additives (C1 and C2). Thus, product P1 is produced by mixing B with C1, whereas, product P2 is produced by mixing B with C2, as seen in Figure 7.1.

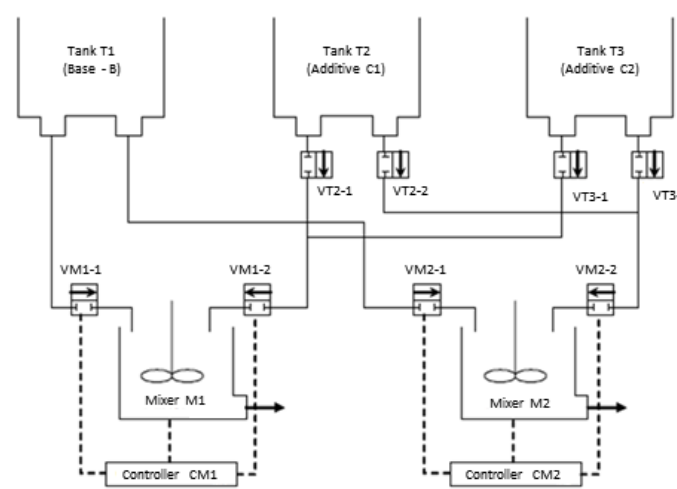


Figure 7.1 Production System (Villani et al., 2007)

The production system, examined in this section, consists of the **system components** and the **controlled objects** (controlled by the supervisory team). The **system components** are:

- Three tanks, T1 (Base - B), T2 (Additive C1) and T3 (Additive C2), which provide the ingredients into the mixers.
- Eight valves, VT2-1 and VT2-2 for T2, VT3-1 and VT3-2 for T3, VM1-1 and VM1-2 for mixer M1 and VM2-1 and VM2-2 for mixer M2, which can open, close and inform for a flow interruption.
- Two mixers, M1 and M2, in which the ingredients are mixing. M1 and M2 can start mixing, stop mixing, start emptying, and stop mixing and start emptying.
- Two local controllers, CM1 and CM2, which can be used to begin a batch, inform the end of batch or inform the end of additive loading, by opening/closing the system valves.

Controllers and mixers interact during the filling and emptying activities. The valve that controls the admission of the base from the tank 1 (T1) to a mixer as well as the output valve that controls the admission of the additive tank (T2 or T3) to a mixer are open and closed by the supervisory system.

In addition to the system components, the **controlled objects** in the production system are:

- Four valve interfaces, VT2-1, VT2-2, VT3-1 and VT3-2, which are controlled (opened/closed) by the supervisory team.
- Two additional interfaces, CM1 and CM2, which are used to begin a batch, inform the end of batch or inform the end of additive loading.
- Receipt P1, associated with the production receipt of products, is used to either begin or inform the end of production P1.
- Production Order_1 asks receipt P1 to make a batch of P1.

The UML AD for the production system that describes how to make a batch of P1 is taken from Villani et al., 2007 and illustrated in Figure 7.2.

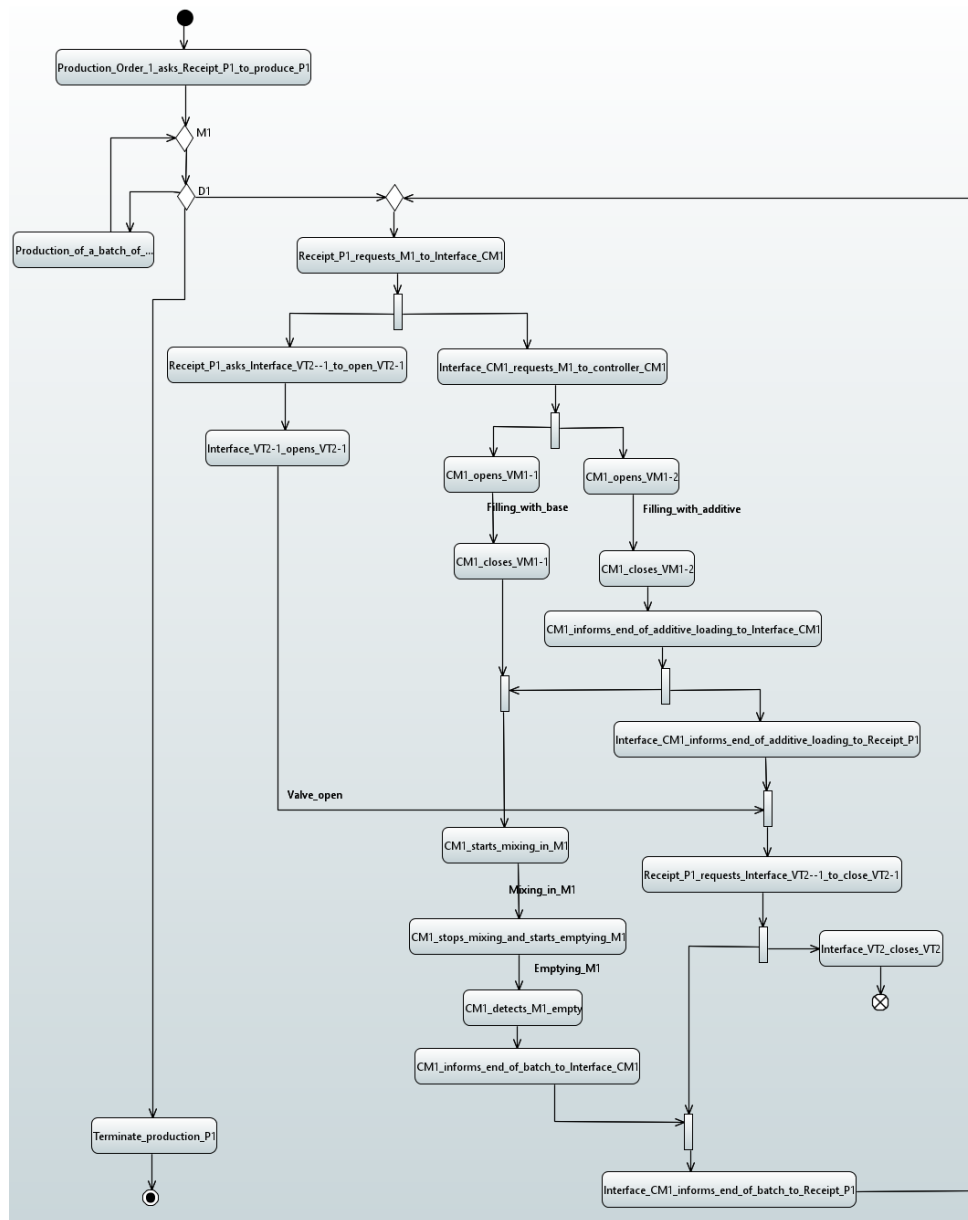


Figure 7.2 UML AD for the Production System (Villani et al., 2007)

In order to produce a batch of P1, the mixer, M1 or M2, is filled with base and additive by opening the corresponding valves. Once the mixer is filled, the base and additive are mixed for a certain time and then the mixer is emptied.

The UML AD for the production system has been selected due to the large number of components it is consisted of, including twenty opaque action nodes, one decision, two merge, three join and four fork nodes, as well as an initial, a final activity and a final flow node. Additionally, in this diagram, there exist two control loops as well as dependent events showing either parallel division, indicated by AD fork nodes that split an incoming flow into multiple concurrent activities, or synchronisation, indicated by AD join nodes in which the flow can proceed only after all incoming flows have reached the join point. Finally, the multiple control nodes presented in series in the AD, such as at the beginning of the diagram where the decision node, 'D1', is placed in between two merge nodes, as well as the three outgoing arcs produced from 'D1', can add complexity in the automation PN process, since structures like these may result in omitting information regarding the sequence of AD elements.

7.2.2 Automated Mathematical Representation of the Petri Net Model for the Production System

7.2.2.1 Input – System Modelling

The AD in Figure 7.2 is initially validated, using the 'Validate model' option available in the Eclipse software and then the two model transformations, discussed in Chapter 6, are applied, to the XMI file obtained from the examined AD. Having completed the two XMI model transformations, two output files, in XMI and XML formats respectively, are generated. These two files are used as inputs to the next step (algorithm-Java database programming step) where they are loaded into the MySQL database to be further manipulated and organised in the mathematical form needed for the PN models. The XMI file obtained from the AD in Figure 7.2 can be found in Appendix J, part A. This initial XMI file consists of XMI nodes such as 'pin', 'Termination_production_P1', 'CM1_opens_VM1-1', 'CMI_detects_M1_empty', etc. and XMI edges such as 'Emptying_M1', 'Filling_with_additive', 'Filling_with_base', etc. The XMI file obtained consists of 70 elements, 37 are edges, and 33 are nodes.

7.2.2.2 Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix

The generic code, introduced in Chapter 6, section 6.4.2 (found in Appendix H) for the automated generation of the mathematical representation of a PN model, has been applied to generate the transpose of the PN incidence matrix for the production system.

The steps that the SQL code applies for the automated PN model generation of the production system are as follows:

1. As discussed in section 7.2.2.1, the XMI and XML files generated from the model transformation of XMI, obtained from the AD in Figure 7.2, are provided as inputs to the methodology and loaded into the MySQL database. Two tables, named ‘final_node_xmi_xml’ and ‘edge_place_xmi’, are created as follows:
 - a. **Retrieve data for AD exception handlers, results, targets and nodes:** The ‘final_node_xmi_xml’ table is created consisting of 33 rows that correspond to the number of nodes found in the AD.
 - b. **Retrieve data for AD edges:** As for the ‘node_xmi_xml’ table, the ‘edge_place_xmi’ table is created consisting of 37 rows that correspond to the number of edges found in the AD.

In order to enable data manipulation, the multiple values that may exist in the “incoming” and “outgoing” columns of the ‘final_node_xmi_xml’ table (created in step 1.a) are separated in the following four steps (steps 2 – 5) as follows:

2. **Separate multiple “outgoing” values:** A table named ‘union_1’ is created separating any multiple values stored in the “outgoing” column of ‘final_node_xmi_xml’ table, such as for the decision nodes exist.
3. **Separate multiple “incoming” values:** A table named ‘union_2’ is created separating any multiple values stored in the “incoming” column of ‘final_node_xmi_xml’ table, such as for the merge nodes.
4. **Store single “incoming” and “outgoing” values:** A table named ‘unique_activities’ is created retrieving from the ‘final_node_xmi_xml’ the

rows that store in the “incoming” and “outgoing” columns single values, such as for the opaque action nodes.

5. **Create a table in which single values are stored in each cell:** A table named ‘union-node’ is created by unifying the results obtained from steps 2 – 4 for the AD shown in Figure 7.2. A part of this table is presented in Table 7.1 for six elements. Each row of this table includes the “xmi:type” and “name” of each node as well as the “xmi:id” values of the incoming and outgoing edges related to the examined node.

Table 7.1 MySQL ‘union_node’ Table Extract

id	outgoing	xmi:type	name	incoming
1	_uzeU0CH7EeilppsneFPfg	uml:InitialNode	place_1	NULL
2	_wBAe8CH7EeilppsneFPfg	uml:OpaqueAction	Production_Order_1_asks_Receipt_P1_to_produce_P1	_uzeU0CH7EeilppsneFPfg
3	_ymKjUCH7EeilppsneFPfg	uml:MergeNode	M1	_wBAe8CH7EeilppsneFPfg
4	_zYyTECH7EeilppsneFPfg	uml:DecisionNode	D1	_ymKjUCH7EeilppsneFPfg
5	_0guLgCH7EeilppsneFPfg	uml:DecisionNode	D1	_ymKjUCH7EeilppsneFPfg
6	_JfkCtIAEeilppsneFPfg	uml:DecisionNode	D1	_ymKjUCH7EeilppsneFPfg

The following seven steps (steps 6 – 12) generate a table that shows the sequence of the AD elements. The sequence of these AD elements can then be used to identify the sequence of PN places and transitions resulting in mathematical and graphical representations of the PN for the production system. Steps 6 – 12 are applied as follows:

6. **Identify the sequence between the AD opaque/decision/merge/fork/join/accept event/send signal/call behaviour action nodes and their preceding and following edges:** In this step, a table, named ‘main_table’, is created retrieving the rows from the ‘union_node’ table (from step 5) that store values for the opaque action, decision, merge, fork and join nodes. The names of the edges found before and after of each node are also identified.
7. **Identify the sequence between the AD initial nodes and their following nodes:** In this step, a table, named ‘initial_node_table’, is created for the initial node. The name of the node placed after the initial node is then retrieved and stored in the “name_primary” column of ‘initial_node_table’ as presented in Table 7.2.

Table 7.2 MySQL ‘initial_node_table’

place_before_node	name_primary	place_after_node
place_1	Production_Order_1_asks_Receipt_P1_to_produce_P1	place_2

8. **Identify the sequence between the AD final nodes and their preceding nodes:** Similarly, as for the initial node, the final nodes are stored in a new table name ‘final_node_table’. The names of the nodes placed before the final activity/final flow nodes are then retrieved and stored in the “name_primary” column of ‘final_node_table’ as presented in Table 7.3.

Table 7.3 MySQL ‘final_node_table’

place_before_node	name_primary	place_after_node
place_30	Interface_VT2_closes_VT2	final_flow
place_6	Terminate_production_P1	final_node_x

For the following three steps (steps 9 – 11), no tables are created, since the program automatically recognises that elements such as data stores, input values, expansion regions, etc. are not included in the AD. Hence, omitting these three steps, the program generates a table in step 12.

9. **Identify the sequence between the AD exception handler edges/activity parameters/data stores/central buffers, and their preceding and following nodes.**
10. **Identify the sequence between the AD nodes to which input/output/value pin nodes are attached, and their preceding and following edges.**
11. **Identify the sequence between the nodes and edges that are included in expansion regions in the AD.**
12. **Identify the sequence of AD elements:** The ‘final_table’, part of which is viewed in Table 7.4, is created unifying the data from the “place_before_node”, “name_primary” and “place_after_node” columns from the tables created in steps 6, 7 and 8.

Table 7.4 MySQL ‘final_table’ Extract

place_before_node	name_primary	place_after_node
place_4	D1	place_5
place_4	D1	place_6
place_4	D1	place_34
place_5	Production_of_a_batch_of_P1_in_M2(similar_to_M1)	place_3
place_37	Receipt_P1_requests_M1_to_Interface_CM1	place_7
place_7	fork_8	place_8
place_7	fork_8	place_9
place_8	Receipt_P1_asks_Interface_VT2--1_to_open_VT2-1	place_10

Steps 13 – 15 result in the development of the mathematical representation of the PN for the model of the production system, retrieving the information from the table created in step 12.

13. **Create a matrix that shows how a token is removed from each of its pre-places, when an enabled transition fires (shows the connection from PN places to PN transitions):** A ‘negative’ matrix with the ‘-1’ and ‘0’ values is created, using the 1st and 2nd columns from Table 7.4.
14. **Create a matrix that shows how a token is inserted to each of its PN post-places, when an enabled transition fires (shows the connection from PN transitions to PN places):** A ‘positive’ matrix with the ‘1’ and ‘0’ values is created, using the 2nd and 3rd columns from Table 7.4.
15. **Generate the mathematical form of the PN model:** In this step, the matrix that shows the mathematical representation of the Petri net for the AD in Figure 7.2 in the form of the transpose of the incidence matrix is created. A part of this matrix is shown in Table 7.5. This matrix, found in Appendix K, part A, is generated by the combination of the matrices obtained in steps 13 and 14. The matrix for the AD in Figure 7.2 for the production system consists of 37 rows, i.e. places, and 30 columns, i.e. transitions.

Table 7.5 MySQL ‘Transpose_of_the_PN_Incidence Matrix’ Extract

place_after_node	D1	Production_of_a_be	Receipt_P1_req	fork_8	Receipt_P1_ask	Interface_CM1_r	Interface_VT2-	fork_12	CM1_opens_VM1-1	CM1_opens_VM1-2	CM1_closes_VM1-1	CM1_closes_VM1-2
Emptying_M1	0	0	0	0	0	0	0	0	0	0	0	0
Filling_with_additive	0	0	0	0	0	0	0	0	1	0	-1	0
Filling_with_base	0	0	0	0	0	0	0	0	1	0	-1	0
Mixing_in_M1	0	0	0	0	0	0	0	0	0	0	0	0
place_1	0	0	0	0	0	0	0	0	0	0	0	0
place_10	0	0	0	0	1	0	-1	0	0	0	0	0
place_11	0	0	0	0	0	1	0	-1	0	0	0	0
place_12	0	0	0	0	0	0	0	1	-1	0	0	0
place_13	0	0	0	0	0	0	0	1	0	-1	0	0
place_16	0	0	0	0	0	0	0	0	0	0	0	1
place_17	0	0	0	0	0	0	0	0	0	0	0	0
place_18	0	0	0	0	0	0	0	0	0	0	0	0
place_19	0	0	0	0	0	0	0	0	0	0	1	0
place_2	0	0	0	0	0	0	0	0	0	0	0	0
place_20	0	0	0	0	0	0	0	0	0	0	0	0
place_21	0	0	0	0	0	0	0	0	0	0	0	0
place_24	0	0	0	0	0	0	0	0	0	0	0	0
place_25	0	0	0	0	0	0	0	0	0	0	0	0
place_26	0	0	0	0	0	0	0	0	0	0	0	0
place_27	0	0	0	0	0	0	0	0	0	0	0	0
place_28	0	0	0	0	0	0	0	0	0	0	0	0
place_29	0	0	0	0	0	0	0	0	0	0	0	0
place_3	0	1	0	0	0	0	0	0	0	0	0	0

7.2.2.3 Algorithm – Java Database Programming – Petri Net Initial Marking Matrix

After obtaining the transpose of the incidence matrix of the PN model for the production system, the initial marking matrix of this net is developed, using the SQL code introduced in Chapter 6, completing the mathematical form of the PN for this system. Thus, the step undertaken for the generation of the initial marking matrix for the production system is:

1. **Initial marking of a PN model:** The ‘initial_marking’ table created for the production system, part of which is shown in Table 7.6, can be found in Appendix K, part B, Table K.1. The records in the “activity” column retrieved from the first column of the transpose of the incidence matrix corresponding to the PN places, whereas the value ‘1’ in the “process_number_of_devices” column it is shown that ‘place_1’ holds one token, i.e. a batch.

Table 7.6 MySQL ‘initial_marking’ Extract

activity	process_number_of_devices
place_1	1
Emptying_M1	0
Filling_with_additive	0
Filling_with_base	0
Mixing_in_M1	0
place_10	0

The matrix of the initial marking automatically developed for the production system examined in this section consists of 37 rows (37 PN places).

7.2.3 Automated Graphical Representation of the Petri Net Model for the Production System

For the graphical representation of the PN model for the production system, the steps introduced in Chapter 3 have been applied. The ‘final_table’ created in step 12 during the automated generation of the mathematical representation of the PN for the system is used as input for this graphical representation. Hence, all the data from the ‘final_table’, part of which is illustrated in Table 7.4, is selected and the code introduced in section 3.4.1 (Appendix D, part A) is executed. The output obtained in the Console window in Eclipse is a DOT file, which is imported into the Graphviz software and the PN model for the production system examined in this section is obtained. The PN model generated for the production system can be viewed in Figure 7.3. The PN model obtained for the production systems consists of 30 transitions and 37 places. The 30 transitions that exist in the PN are equal to the sum of the twenty opaque action nodes, the one decision, two merge, three join and four fork nodes that was presented in the production system AD.

The process has successfully yielded a PN from the AD automatically, and hence shown applicability to systems and also increased scalability with more components than in previous ADs. Further scalability will be tested with an online shopping process, viewed in the next section. Formal verification and validation of the method

to check the correctness of the automation end model are also investigated in the following sections.

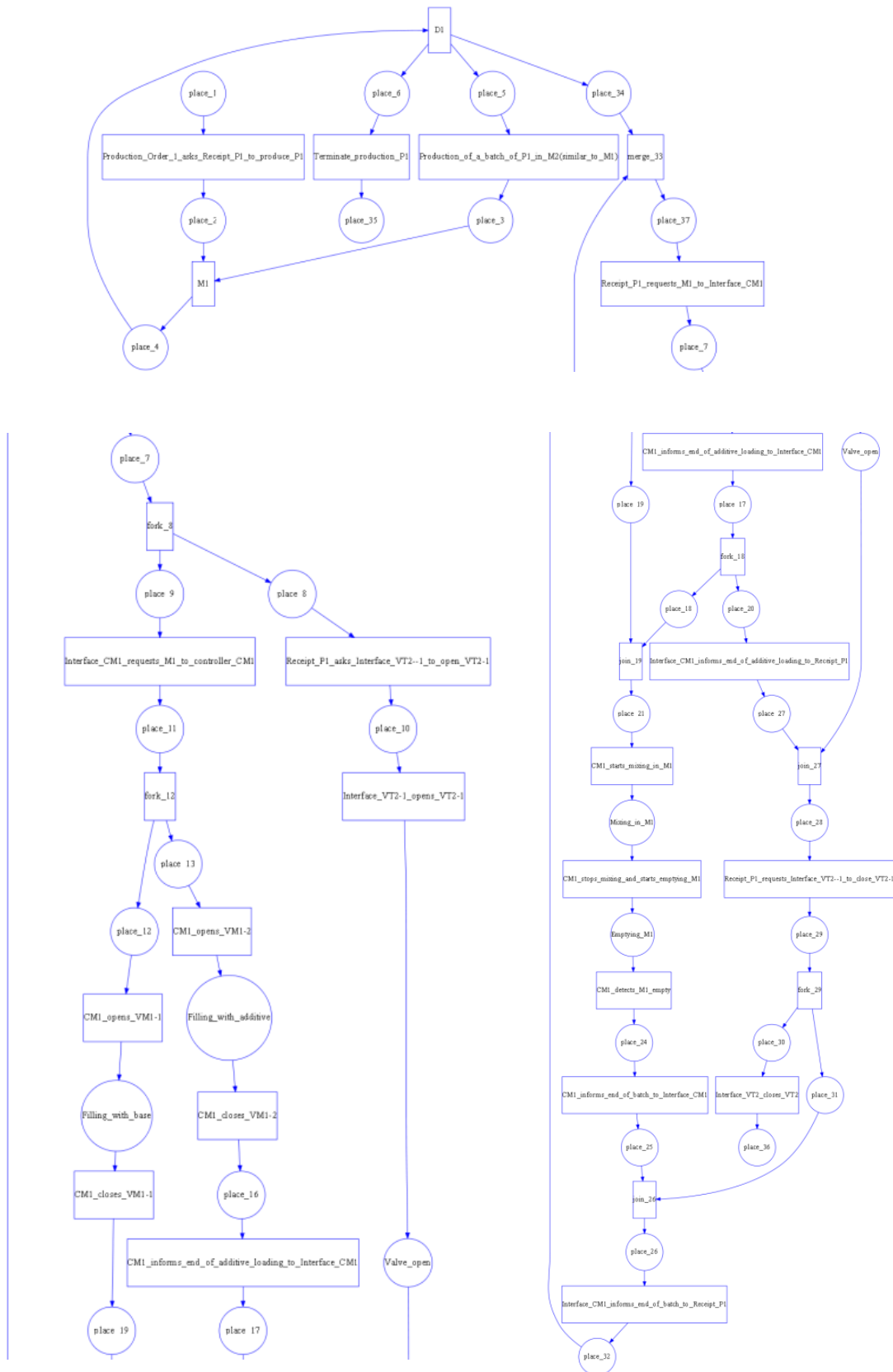


Figure 7.3 PN Model Automatically developed for the Production System

7.3 Online Shopping Process

7.3.1 Process Description

An online shopping process taken from the literature (Banas, 2012) is used as an example in this section to illustrate the applicability of this developed automated modelling capability study to complex processes. Advancements with this example are a larger number of activities and more AD elements. The process investigates online purchases of monitors and computers and the corresponding UML AD is illustrated in Figure 7.4.

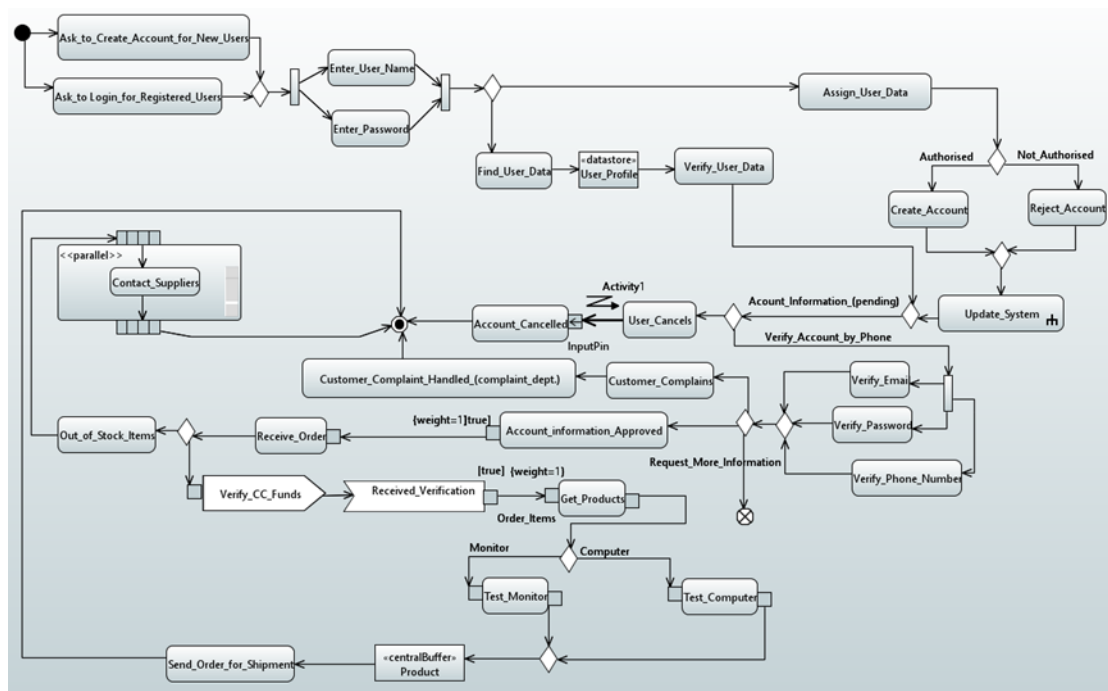


Figure 7.4 UML AD for the Online Shopping Process (Banas, 2012)

In the process, once customers complete product selection, the personal details checking begins. New customers are asked to create a new account, whereas registered customers are asked to log in to their existing accounts. The customers then enter their user name and password and the process can proceed to either the user data verification for old customers or the assignment of user data for new customers. In the case of existing customers, the company's system is looking for the user's profile in its database (datastore node) and the customer's data is verified. In the case of new customers, the username and password are assigned, and (i) if user's data is authorized, the account is created; (ii) if not the account is rejected. In both cases the system is updated (using the call behaviour node) creating and saving the customer's

profile in the system. If more account information is required and the customer decides not to provide it, the account is cancelled and the process is terminated as seen by the final activity node in the AD in Figure 7.4. If the customer is able to provide further information the account verification is completed over the phone. The customers verify their email, password, and phone number and three possible outcomes have been identified for the online shopping process:

1. Customers cannot proceed due to inadequate information and hence the process is terminated with the flow final node as seen in Figure 7.4. The execution of this node indicates that a use case is finished but the whole process might be continued.
2. Customers complain to the corresponding department. The process in this case is also terminated with the activity final node seen in the AD which indicates that all flows of the process terminate, once this node is executed.
3. The account information is approved and the order is received. The product availability is checked and in case: (i) the product is out of stock the company contacts the suppliers and the process ends; (ii) there is product availability the customer credit card details are verified and once the verification is received the quality test of each product is conducted. An AD decision node is used to direct the products to the right test action since different tests are required for monitors and computers. Once the quality test is completed, the products are collected and sent for shipment, terminating the process. The central buffer node, used at this point in the AD in Figure 7.4, manages object flows of various incoming and outgoing edges for monitors and computers.

The online shopping process consists of 24 opaque action nodes, a call behaviour action node, an accept event and a send signal action node, six decision, five merge, one join and two fork nodes. Additionally, it contains a datastore, a central buffer, an initial, a final flow and a final activity node. An exception handler edge and an expansion region with one input and one output expansion node are also used in the AD. Finally, in this diagram, input and output value/pin nodes can be found. The reason for the selection of this diagram is the variety of the AD elements that it includes.

7.3.2 Automated Mathematical Representation of the Petri Net Model for the Online Shopping Process

7.3.2.1 Input – System Modelling

Using the ‘Validate model’ option available in the Eclipse software the AD in Figure 7.4 is validated and then by applying the two model transformations, discussed in Chapter 6, to the XMI file obtained from the examined AD, two output files, in XMI and XML formats respectively, are generated. These two files are used as inputs to the algorithm-Java database programming step where they are loaded to the MySQL database for further manipulation to result in the development of the mathematical form of PN model. The XMI file obtained can be found in Appendix J, part B. This initial XMI file consists of XMI nodes such as the ‘Assign_User_Data’, ‘Create_Account’, ‘Update_System’, etc. and XMI edges such as the ‘Authorised’, ‘Not_Authorised’, ‘Order_Items’, etc. The XMI file obtained from the AD for the production process consists of 105 elements, 56 are edges, and 49 are nodes. The 49 nodes found in the XMI document include, 24 opaque action nodes, a call behaviour action node, an accept event and a send signal action node, six decision, five merge, one join and two fork nodes. Additionally, it contains a datastore, a central buffer, an initial, a final flow and a final activity node, as well as an expansion region with two expansion nodes, an input and an output.

7.3.2.2 Algorithm – Java Database Programming – Transpose of the Petri Net Incidence Matrix

The extended advanced code, discussed in Chapter 6 (found in Appendix H) for the automated generation of the mathematical representation of a PN model, has been followed to generate the transpose of the PN incidence matrix for the online shopping process. The mathematical representation of a PN model is generated by applying the following steps:

1. The XMI and XML files acquired from the model transformations of the XMI file obtained from the AD in Figure 7.4 are provided as inputs to the methodology. These two files, discussed in section 7.3.2.1, and loaded into the MySQL database and the ‘final_node_xmi_xml’ and ‘edge_place_xmi’ tables, are created as follows:

- a. **Retrieve data for AD exception handlers, results, targets and nodes:** The ‘final_node_xmi_xml’ table is created consisting of 49 rows that correspond to the number of nodes found in the AD.
- b. **Retrieve data for AD edges:** As for the ‘final_node_xmi_xml’ table, the ‘edge_place_xmi’ table is also created consisting of 56 rows that correspond to the number of edges included in the AD.

Steps 2 – 5 target the separation of multiple values that exist in the “incoming” and “outgoing” columns of the ‘final_node_xmi_xml’ table (created in step 1.a) in order to enable data manipulation for the automated generation of the mathematical representation of PN model as follows:

2. **Separate multiple “outgoing” values:** Table named ‘union_1’ is created for the AD separating the multiple values stored in the “outgoing” column of the ‘final_node_xmi_xml’ table.
3. **Separate multiple “incoming” values:** Table named ‘union_2’ is created for the AD separating the multiple values stored in the “incoming” column of the ‘final_node_xmi_xml’ table.
4. **Store single “incoming” and “outgoing” values:** A table named ‘unique_activities’ is created from the rows of the ‘final_node_xmi_xml’ table that present single incoming and outgoing values.
5. **Create a table in which single values are stored in each cell:** In this step, a table named ‘union_node’ is created for the AD, part of which is viewed in Table 7.7 for seven elements. This table is created by unifying the tables in steps 2, 3 and 4. According to the two first rows of this table, it can be seen that node ‘fork_7’ should be represented in the AD with one incoming and two outgoing edges, which can be verified by the diagram.

Table 7.7 MySQL ‘union_node’ Table Extract

id	outgoing	xmi:type	name	incoming
7	_xoAP8JR3EebXObshy6vIA	uml:ForkNode	fork_7	_7aWKAJkoEeeOIORO-VyZIA
8	_y2k74JR3EebXObshy6vIA	uml:ForkNode	fork_7	_7aWKAJkoEeeOIORO-VyZIA
9	_zrTkcJR3EebXObshy6vIA	uml:OpaqueAction	Enter_User_Name	_xoAP8JR3EebXObshy6vIA
10	_0qqvsJR3EebXObshy6vIA	uml:OpaqueAction	Enter_Password	_y2k74JR3EebXObshy6vIA
12	_aoYIMJ7uEeel-svp6Z91RQ	uml:OpaqueAction	Verify_User_Data	_L8ShMJ7uEeel-svp6Z91RQ
13	_LId3kJ7uEeel-svp6Z91RQ	uml:OpaqueAction	Find_User_Data	_1s9TlJR3EebXObshy6vIA
14	_L8ShMJ7uEeel-svp6Z91RQ	uml:DataStoreNode	User_Profile	_LId3kJ7uEeel-svp6Z91RQ

The next seven steps (steps 6 – 12) result in the generation of a table that shows the sequence of AD elements. This set of seven steps enhances the capability of the SQL code to track the sequence of AD elements, both fundamental as discussed in Chapter 3 and additional as introduced in Chapter 6. The final table obtained in step 12 is used as the basis for the identification of the sequence of PN places and transitions enabling the development of the mathematical and graphical representations of the PN for the online shopping process. Steps 6 - 12 are explained in detail, as follows:

6. **Identify the sequence between the AD opaque/decision/merge/fork/join/accept event/send signal/call behaviour action nodes and their preceding and following edges:** A table named ‘main_table’ is generated retrieving the rows from the ‘union_node’ table (from step 5) where their “xmi:type” is equal to ‘uml:OpaqueAction’ or ‘uml:DecisionNode’ or ‘uml:MergeNode’ or ‘uml:ForkNode’ or ‘uml:JoinNode’ or ‘uml:AcceptEventAction’ or ‘uml:SendSignalAction’ or ‘uml:CallBehaviorAction’. Therefore, for the online shopping example the ‘main_table’ is created storing data for 24 opaque action nodes, one call behaviour action node, one accept event and one send signal action node, six decision, five merge, one join and two fork nodes as viewed in the AD. The names of the edges found before and after of each of these nodes are also retrieved from the ‘edge_place_xmi’ table (step 1.b).
7. **Identify the sequence between the AD initial nodes and their following nodes:** For the initial AD node, a table, named ‘initial_node_table’, is created. The name of the node that follows the initial node in the diagram, i.e. ‘Ask_to_Create_for_New_Users’, is retrieved and stored in the “name_primary” column of ‘initial_node_table’ as presented in Table 7.8.

Table 7.8 MySQL ‘initial_node_table’

place_before_node	name_primary	place_after_node
pin_5	Ask_to_Create_Account_for_New_Users	NULL

8. **Identify the sequence between the AD final nodes and their preceding nodes:** For the final activity and the final flow nodes of the AD, a table, named ‘final_node_table’, is created. The names of the nodes shown before the final activity and the final flow nodes are retrieved and stored in the “name_primary” column, as viewed in Table 7.9.

Table 7.9 MySQL 'final_node_table'

place_before_node	name_primary	place_after_node
NULL	Account_Cancelled	pout_28
NULL	Customer_Complaint_Handled_(complaint_dept.)	pout_28
NULL	Send_Order_for_Shipment	pout_28
NULL	decision_29	flow_pout_30

9. **Identify the sequence between the AD exception handler edges/activity parameters/data stores/central buffers, and their preceding and following nodes:** In this step, three tables called 'exception_handler', 'datastore_table_a_b' and 'central_buffer_table_a_b' are created for the AD exception handler edge, data store node and central buffer node respectively as seen in Tables 7.10, 7.11 and 7.12. For each case, the table created stores to the "place_before_node" and "place_after_node" columns records from the "name" column of the 'union_node' table (from step 5) where the "xmi:type" is equal to 'uml:ExceptionHandler'/'uml:DataStoreNode'/'uml:CentralBufferNode'. Each table is then updated storing to the "name_primary" column the records from the "name" column of the 'union_node' table where the "outgoing"/"incoming" value of the examined node is equal to the value stored in the "incoming"/"outgoing" column of the same table (step 5).

Table 7.10 MySQL 'exception_handler' Table

place_before_node	name_primary	place_after_node
place_15	User_Cancels	User_Cancels_handler
User_Cancels_handler	Account_Cancelled	NULL

Table 7.11 MySQL 'datastore_table_a_b' Table

place_before_node	name_primary	place_after_node
NULL	Find_User_Data	User_Profile
User_Profile	Verify_User_Data	NULL

Table 7.12 MySQL 'central_buffer_table-a_b' Table

place_before_node	name_primary	place_after_node
NULL	merge_48	Product
Product	Send_Order_for_Shipment	NULL

10. **Identify the sequence between the AD nodes to which input/output/value pin nodes are attached, and their preceding and following edges:** In this step, the code initially identifies the rows in the table created in step 1.a where the "xmi:type_nested" is equal to 'uml:InputPin'/'uml:OutputPin'/'uml:ValuePin' and creating a new table, named 'in_outputValue_final', stores in the "name_primary" column the name values found in the table created in step 1.a which store

input/output/value pin nodes. The code then stores in the “place_before_node” and “place_after_node” columns the names of the edges represented as incoming to the input/value pin nodes and outgoing from the output/value pin nodes of the examined node. The values for the first and third columns from the table created in step 1.b. For the online shopping AD, part of the ‘in_outputValue_final’ table, shown in Table 7.13, is created.

Table 7.13 MySQL ‘in_outputValue_final’ Table

place_before_node	name_primary	place_after_node
place_43	Receive_Order	place_27
Order_Items	Get_Products	place_33
Computer	Test_Computer	place_36
Monitor	Test_Monitor	place_45

11. **Identify the sequence between the nodes and edges that are included in expansion regions in the AD:** In this step, the ‘expansionNode_final’ table is generated tracking the rows in the table created in step 1.a where the “xmi:type_nested” is equal to ‘uml:ExpansionRegion’. Then the “name” values of these rows are stored in the “name_primary” column of table ‘in_outputValue_final’. Finally, the code retrieves from the table created in step 1.b the names of the edges placed before and after of the examined nodes and stores these names in the “place_before_node” and “place_after_node” columns respectively. The ‘expansionNode_final’ table shown in Table 7.14 for the AD in Figure 7.4 is created. The values found in the first and third columns of this table correspond to the incoming edge to the input expansion node and to the outgoing edge from the output expansion node respectively.

Table 7.14 MySQL ‘expansionNode_final’ Extract Table

place_before_node	name_primary	place_after_node
place_56	Contact_Suppliers	place_55

12. **Identify the sequence of AD elements:** In this step, the tables obtained from steps 6 – 11 are unified and the ‘final_table’, part of which is presented in Table 7.15, is created. The ‘pin_5’ value found in the “place_before_node” column in the second row of this table corresponds to the AD initial node.

Table 7.15 MySQL ‘final_table’ Extract

place_before_node	name_primary	place_after_node
place_46	Account_information_Approved	place_43
pin_5	Ask_to_Create_Account_for_New_Users	place_2
place_49	Assign_User_Data	place_8
Authorised	Create_Account	place_11
place_26	Customer_Complains	place_31
place_8	decision_15	Authorised

The next three steps (steps 13 – 15) describe the generation of the mathematical representation of the PN for the model of the online shopping process, acquiring the data from the ‘final_table’, created in step 12.

13. **Create a matrix that shows how a token is removed from each of its pre-places, when an enabled transition fires (shows the connection from PN places to PN transitions):** A matrix with rows the values from the 1st column of the table developed in step 12 and columns the values from the 2nd column of the same table is created. Once records from the 1st and 2nd columns of the ‘final_table’ are in the same row, then the value ‘-1’ should be put in the corresponding matrix cell otherwise a ‘0’ is inserted.
14. **Create a matrix that shows how a token is inserted to each of its PN post-places, when an enabled transition fires (shows the connection from PN transitions to PN places):** A similar matrix with rows the values from the 3rd column of the table developed in step 12 and columns the values from the 2nd column of the same table is created. Once records from the 2nd and 3rd columns of the ‘final_table’ are in the same row, then the value ‘1’ should be put in the corresponding matrix cell otherwise a ‘0’ is inserted.
15. **Generate the mathematical form of the PN model:** The mathematical representation of the Petri net for the online shopping AD in the form of the transpose of the incidence matrix is created by unifying the two matrices generated in steps 13 and 14. This matrix, and part of which is shown in Table 7.16 can be found in Appendix K, part C. The matrix created in this step consists of 49 rows, i.e. places and 41 columns, i.e. transitions.

Table 7.16 MySQL ‘Transpose_of_the_PN_Incidence Matrix’ Extract

place_after_node	decision_29	Customer_Complair	merge_48	Ask_to_Create_A	Send_Order_foi	Verify_User_J	fork_7	Enter_User_	Enter_Password	Find_User_Data	Assign_User_Data	decision_15
Verify_Account_by_Phone	0	0	0	0	0	0	0	0	0	0	0	0
User_Profile	0	0	0	0	0	-1	0	0	0	1	0	0
Product	0	0	1	0	-1	0	0	0	0	0	0	0
pout	0	1	0	0	1	0	0	0	0	0	0	0
place_8	0	0	0	0	0	0	0	0	0	0	1	-1
place_7	0	0	0	0	0	0	0	0	0	-1	0	0
place_6	0	0	0	0	0	0	0	0	1	0	0	0
place_56	0	0	0	0	0	0	0	0	0	0	0	0
place_55	0	0	0	0	0	0	0	0	0	0	0	0
place_54	0	0	0	0	0	0	0	0	0	0	0	0
place_53	0	0	0	0	0	0	0	0	0	0	0	0
place_52	0	0	0	0	0	1	0	0	0	0	0	0
place_5	0	0	0	0	0	0	0	1	0	0	0	0
place_49	0	0	0	0	0	0	0	0	0	0	-1	0
place_48	0	0	0	0	0	0	0	0	0	0	0	0
place_46	1	0	0	0	0	0	0	0	0	0	0	0
place_45	0	0	-1	0	0	0	0	0	0	0	0	0
place_44	0	0	0	0	0	0	0	0	0	0	0	0
place_43	0	0	0	0	0	0	0	0	0	0	0	0
place_42	0	0	0	0	0	0	-1	0	0	0	0	0

7.3.2.3 Algorithm – Java Database Programming – Petri Net Initial Marking Matrix

The initial marking matrix for the online shopping process is obtained applying the SQL code discussed in Chapter 6 for the automated generation of the initial marking matrix of the PN as follows:

1. **Initial marking of a PN model:** The ‘initial_marking’ table for the online shopping process, part of which is viewed in Table 7.17, is created. The value ‘1’ in the “process_number_of_devices” column it is shown that ‘pin_5’ holds one token, i.e. a customer. The matrix of the initial marking consisting of 49 rows, i.e. 49 places, can be found in Appendix K, part D, Table K.2.

Table 7.17 MySQL ‘initial_marking’ Extract

activity	process_number_of_devices
Account_Information_(pending)	0
Authorised	0
Computer	0
flow_pout_30	0
Monitor	0
Not_Authorised	0
Order_Items	0
pin_5	1
place_11	0
place_12	0

7.3.3 Automated Graphical Representation of the Petri Net Model for the Online Shopping Process

For the graphical representation of the PN model for the online shopping process, the steps introduced in Chapter 3 for the automated PN graphical representation have been applied, retrieving the data form the ‘final_table’ created in step 12 (Table 7.15) during the automated generation of the mathematical representation of the PN for the process. A DOT file is generated executing the code introduced in section 3.4.1 (Appendix D, part A) and then importing this DOT file into the Graphviz software the PN model for the online shopping process is obtained as presented in Figure 7.5.

The PN model obtained consists of 41 transitions and 49 places. The 41 transitions exist in the PN are equal to the sum of the 24 opaque action nodes, the one call behaviour action node, the one accept event action node, the one send signal action node, and the six decision, five merge, one join and two fork control nodes that was presented in the AD in Figure 7.4.

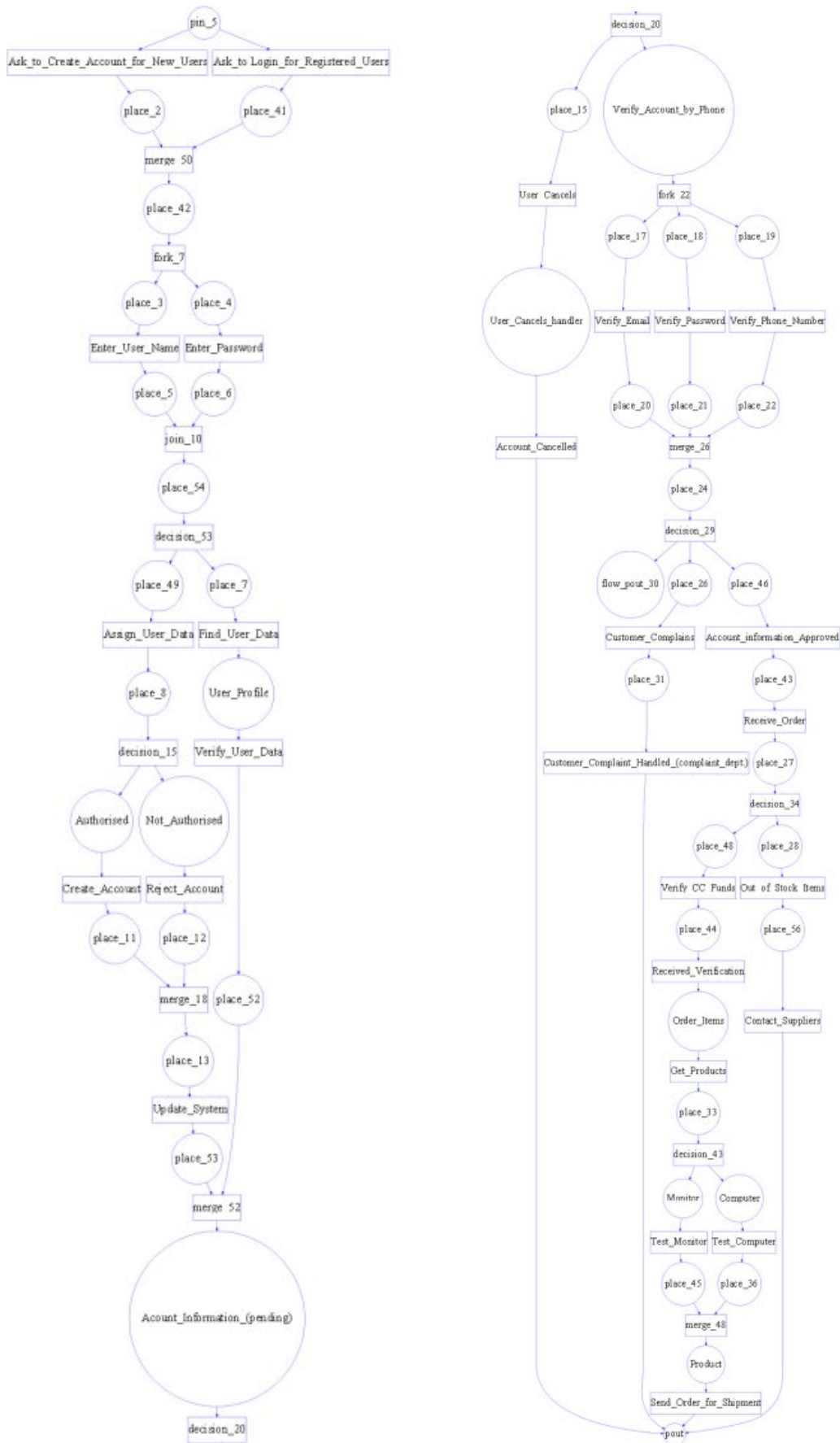


Figure 7.5 PN Model Automatically developed for the Online Shopping Process

A PN model has been successfully produced from the AD automatically, and thus increased process scalability has been proven with more AD elements than in the previous ADs considered, since the transformation of the additional AD elements to corresponding PN structures has been achieved. The correctness of the PN model is further explored by a formal verification and validation of the method, seen in the following section.

7.4 Verification and Validation of Real-Life Scenarios

The correctness of the extended (generic) algorithm proposed in Chapter 6 for the PN automation procedure is checked, as discussed in Chapter 5, by: (i) verifying that the PN model obtained performs the correct function; and (ii) validating the PN model obtained accurately represents the system/process architecture. This is demonstrated via evaluation of the two PN models automatically generated for the two real-life scenarios considered in this chapter.

The verification of the two PN models, obtained for the production system and the online shopping process respectively, is carried out by checking both the structural and behavioural properties of these models, using HiPS, as demonstrated in Chapter 5. Both examined models belong to a special class of PNs, workflow-nets, satisfying the two criteria defined by Aalst (1998), as explained in Chapter 5, section 5.3. Besides these two criteria, a third criterion, also investigated in the same section regarding the nets structure is met, and hence the two WF-nets are structurally verified. In addition to the structural analysis, the two WF-nets are behaviourally verified since they satisfy the soundness property, i.e. the two models are behaviourally live and bounded, as also discussed in Chapter 5. Figures L.1 – L.4, in Appendix L, show the verification analysis conducted using HIPS for the two PN models.

The validation of the two PN models, obtained for the production system and the online shopping process, is carried out by applying the real system measurements approach, by visually checking the behaviour of the given system and process, playing the token game, as demonstrated in Chapter 5, section 5.4.2. For each example, random pass/fail probabilities are considered for each PN transition and random numbers from zero to one are generated. The PN paths are then followed comparing the random numbers, against the pass/fail probabilities using ‘if’ conditions and

equation 5.1 is applied. In this equation, the initial marking and transpose of the incidence matrix have been automatically generated in the previous sections in this chapter, whereas the transition matrices required are manually developed. Therefore, a simulation algorithm was developed that visually checks the removal/addition of tokens from/to places through the PN paths, validating that the paths followed the same route as the paths existing in the Activity Diagrams provided for the production system and online shopping process, respectively.

7.5 Summary

This chapter demonstrated the automated PN model capability to two real-life scenarios, a production system and an online shopping process. The examined scenarios are extensions in complexity of the simple process examined in Chapter 6, and include a greater number of components/activities and, by extension, more paths, as well as control loops and all the fundamental and additional elements of the AD. The mathematical and graphical representations of the PN models for both real-life scenarios were obtained automatically. Finally, a formal verification and validation of the method was also investigated proving the correctness of the automation end models.

8 Conclusions and Future Work

8.1 Introduction

The research presented in this thesis provides a powerful methodology for the automated generation of PNs for large systems and processes including those with control loops and dependent events. The proposed methodology accepts as input a UML/SysML AD, as used in industry, with the topology system/process representation. This chapter outlines the conclusions of this research, proving that all the research objectives established in Chapter 1 have been addressed. Contributions to knowledge are then presented. The thesis concludes with recommendations for future work.

8.2 Conclusions

Following the research conducted in this thesis, the key conclusions drawn related to the research objectives are:

1. *Identify the most suitable UML or SysML diagram:* The UML/SysML Activity Diagram was identified as the most suitable diagram from which the topology information of a system or process can be retrieved and used as a starting point for the automated PN model generation. This diagram can (i) capture the behavioural aspects of system components and process activities which are required for reliability modelling; (ii) be applicable to a wide spectrum of disciplines such as aerospace, engineering, telecommunications, etc.; (iii) model complex industrial systems and processes, maximising future applicability; and (iv) ease the communication between different business stakeholders by providing a notation understandable by all business users.
2. *Perform a detailed literature review of Petri Net model:* Petri Nets are a versatile tool suitable for complex system and process modelling and analysis. The graphic perspective of this model can be used both as an aided design tool and with the help of marking to obtain information for the behaviour of a

given system/process. A PN can also be expressed by means of mathematical equations and used to describe the system/process behaviour.

3. *Review the automated model generation methods:* According to the literature findings, FTs and PNs have received the most attention in the automated construction of reliability models. The main limitations identified during the literature review (literature gaps) carried out for the automated generation of CCDs, RBDs, Markov Chains, FTs and PNs are the: (i) degree of automation (limited capability for full automation); (ii) non-generic domain applicability; (iii) difficulty in handling and efficiently modelling systems with many components or complex characteristics such as control loops and dependent events; and (iv) software dependency.
4. *Develop a methodology for the automated Petri Net model generation:* A novel methodology that automatically generates PN models is introduced. The proposed methodology can handle most of the limitations identified in the current methods reviewed for the automated construction of reliability models, except for the generic domain applicability of the method to systems and processes (ii from point 3). The methodology is applicable only to processes that consist of the most commonly used AD elements and include loops and dependencies. The developed methodology uses as a starting point the UML/SysML AD and applying a Java Database (MySQL) algorithm, it generates the mathematical and graphical representations of a PN model.
5. *Validate and verify the Petri Net model:* Verification and validation of the PN model to check that it performs the correct function and accurate representation of the system architecture respectively have been conducted, proving the correctness and completeness of the Java Database algorithm developed for the PN automation procedure.
6. *Extend the proposed methodology providing scalability:* Additional transformation rules to cover the full range of UML/SysML AD nodes have been introduced to the Java Database algorithm. The extended methodology has proven scalability through its application to real-life industrial systems and processes with loops and dependencies that consist of any possible AD element. Hence, all the limitations discussed in point 3 have been addressed.

To summarise, according to the conclusions, the research objectives have been achieved, and hence the aim of this thesis has been successfully accomplished.

8.3 Contributions to Knowledge

This thesis has contributed to knowledge in the area of automated reliability model generation in the following ways:

- *Full automated PN model generation*: direct transformation of an industrial description diagram, i.e. the UML/SysML AD, of a system/process to the corresponding Petri Net model.
- *Generic applicability*: the proposed methodology can handle and efficiently model both systems and processes without targeting specific domains, providing potential applicability to a wide spectrum of cases.
- *Modelling of advanced structural characteristics of systems and processes*: capability of the proposed methodology to model real-life industrial scenarios with control loops and dependent events, overcoming the weakness of reviewed methods that cannot.
- *Software independence*: the proposed methodology is considered as software independent since the output mathematical and graphical representations of the PN model, are in a usable format, without further manipulation to different software being required.

8.4 Recommendations for Future Work

Following the research outcome in this thesis, potential future work (or research challenges) that can be carried out in various areas is outlined in this section.

8.4.1 Automated sub-PNs Construction followed by Simulation

Analysis

As discussed in Chapter 5 for the recycling IT asset process, the overall PN model, automatically generated from the application of the Java Database algorithm in Chapter 4, can be expanded with the help of sub-PNs, by generating one net for each transition included in the overall PN. These sub-PNs provide a more comprehensive understanding of the IT asset process by showing how the data provided can be

represented using a PN model. This expansion of the model that provides an extra layer of model granularity can be carried out in various ways according to the data available. For the recycling IT asset process, for which probabilistic and timed data was given, the Java Database algorithm has been extended and the corresponding sub-PNs have been obtained automatically. The code added produces a final PN that consists of the sub-nets developed for the transitions of the overall PN, considering both probabilistic and timed data. In order to provide a generic applicability, additional cases should be investigated. Thus, further work can be carried out to extend the methodology to automatically identify the type of given data, i.e. (i) only probabilistic data; (ii) only timed data; or (iii) both probabilistic and timed data, to produce the corresponding final PN.

Further to this extension, the proposed algorithm can be enhanced to support the automated simulation analysis of the final PN models by recognising different PN formats. Three cases need to be considered based on the data available, as discussed earlier. Hence, the software added for the simulation should be able to recognise the type of the final PN model automatically generated and conduct the corresponding simulation each time. The automated simulation will aid the methodology to validate the PN models without user's intervention and also to acquire results that can help decision-making with the view to the increase system's or process' reliability and performance.

8.4.2 Automated Reliability Analysis

The software could be expanded to automatically return the overall system/process unreliability, which can be useful to identify system limitations and, by extension, to enhance system's performance. This could be accomplished by developing an algorithm that would accept as inputs the failure data of system's components/process' activities as well as the relationships between them in order to calculate the system/process unreliability. For the identification of system's components/process' activities relationships, the Java Database algorithm, introduced in this research study, could be enhanced with rules that would allow automatic identification of the connectivity between components/activities such as connection in series or parallel. For example, the AD nodes placed after a fork node could be

considered as parallel events, whereas two or more AD nodes connected end-to-end, forming a single path could be considered as serial events.

8.4.3 Additional PN Model Features

The strength of this method could be enhanced by considering additional PN features such as inhibitor arcs to restrict the process to only one token, i.e. item/device, at the same time. This addition would increase the applicability range of the proposed method to real-life industrial queuing systems/processes. This could be accomplished by extending the algorithm developed for the graphical representation of the PNs to automatically add inhibitors where needed according to the timed data provided. Additionally, the code could be extended to automatically generate and simulate PNs with multiple initial markings, i.e. multiple tokens residing in one/or more places. Finally, alternative types of PNs such as Coloured PNs could be considered to be automatically generated, by introducing new construction rules such as allowing tokens to have a data value attached to them. All these features that add complexity to the PN model behaviour could increase the flexibility and popularity of the proposed method.

8.4.4 Investigation of Inputs

Another line of future investigation could include the adjustment of the proposed methodology and expansion of the software to accept as input multiple systems/processes that interact with each other. This extension can be beneficial since these types of multiple interacting systems/processes are commonly met in industry. This could be accomplished by allowing the user to enter multiple topology diagrams, i.e. UML/SysML ADs, into the algorithm, and then generate a sub-PN model for each UML/SysML AD and then link them together by generating an overall PN. Additional extension of this work could be if the software accepts alternative UML/SysML diagrams such as the BDD and IBD. This could be accomplished by: (i) examining the structure of the XMI files obtained from the new diagrams to decide if XMI model transformations are necessary to be conducted; (ii) introducing new transformation rules for the elements employed in these new diagrams; and (iii) defining rules to determine how the data retrieved from each diagram can be used for the PN construction. This last extension of the software to accept alternative diagrams would be beneficial in increasing the methodology's applicability.

8.4.5 Representation of PN results into the UML/SysML AD

An animated Graphical User Interface (GUI), readily understandable by users, could be built to represent the results of calculations made on the low level model (PN) into the high level model (UML/SysML). This extension would facilitate the users to understand how the results obtained from the simulation analysis are related to the UML/SysML AD and, by extension, to identify limitations and make possible recommendations to enhance the reliability and performance of examined systems and processes. Additionally, the design and implementation of a GUI to the proposed methodology for the automated PN model construction could potentially lead to an increase in popularity of the methodology since it would not require the users to have knowledge of PN formalisms.

Bibliography

AADL (2004). Society of Automotive Engineers (SAE) Avionics Systems Division (ASD) AS-2c Subcommittee. (2004). *Avionics Architecture Description Language Standards*. Draft v0.99.

Adachi, M., Papadopoulos, Y., Sharvia, S., Parker, D. and Tohdo, T. (2011). An approach to optimisation of fault tolerant architectures using HIP-HOPS. *Software Practice and Experience*, 41(11): 1303-1327.

Agarwal, B. (2013). Transformation of UML Activity Diagrams into Petri Nets for Verification Purposes. *International Journal of Engineering and Computer Science*, 2(3): 798-805.

Al-Aomar, R., Ülgen, O.M. and Williams, E.J. (2015). Process simulation using witness. Wiley, Hoboken.

Alhroob, A., Dahal, K. and Hossain, A. (2010). Transforming UML sequence diagram to High Level Petri Net. *2nd International Conference on Software Technology and Engineering (ICSTE)*, USA, (1): 260-264.

André, É., Benmoussa, M.M., and Choppy, C. (2014). Translating UML state machines to coloured Petri Nets using Acceleo: A report. *ESSS. EPTCS*.

Andreadakis, S.K. and Levis, A.H. (1988). Synthesis of distributed command and control for the outer air battle. *Proceedings of the 1988 Symposium on C² Research*, SAIC, McLean, VA.

Andrews, J. D. and Ridley, I. M. (2001). Reliability of Sequential Systems Using the Cause-Consequence Method. *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering*, 215(3): 207-220.

Andrews, J.D. and Henry, J.J. (1997). A computerized fault tree construction methodology. *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering*, 211(3): 171-183.

Andrews, J.D. and Moss, B. (2002). *Reliability and Risk Assessment*. (2nd edition). Wiley-Blackwell.

Balci, O. (1998). *Verification, Validation, and Testing*. Banks, J. (ed.) The Handbook of Simulation, ch. 10, John Wiley & Sons, Chichester.

Balci, O. (2004). Quality Assessment, Verification, and Validation of Modelling and Simulation Applications. *Proceedings of the 2004 Winter Simulation Conference*, 122-129.

Banas, D. (2012). UML 2.0 ACTIVITY DIAGRAMS. Available via: <http://www.newthinktank.com>

Banks, J., Gerstein, S. and Searles, S.P. (1987). Validation, and Verification of Complex Simulations: A Survey. *Proceedings of the Conference on Methodology and Validation*, 13-18.

Bao, N.Q. (2010). A proposal for a method to translate BPMN model into UML activity diagram. Vietnamese-German University – BIS.

Bock, C. (2005). UML 2 Activity and Action Models Part 6: Structures Activities. *Journal of Object Technology*, 4(4): 43-66.

Boiteau, M., Dutuit, Y., Rauzy, A. and Signoret, J.-P. (2006). The data-flow language in use: modelling of production availability of a multi-state system. *Reliability Engineering and System Safety*, 91(7): 747-755.

Bouissou, M. (2006). A Generalised of Dynamic Fault Trees through Boolean logic Driven Markov Processes (BDMP). *ESREL, IEEE Computer Society*, 2(1): 708-717.

Brameret P.-A., Rauzy, A. and Roussel, J.-M. (2015). Automated generation of partial Markov chain from high level descriptions. *Reliability Engineering and System Safety*, 139: 179-187.

Cardoso, A.J.S (2002). Quality of Service and Semantic Composition of Workflows. (Doctoral dissertation, University of Georgia).

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. (2nd edition). New York: Springer.

Chew, S.P. (2010). *Systems Reliability Modelling for Phased Missions with Maintenance Free Operating Periods*. Ph.D. thesis, Loughborough University.

Chew, S.P., Dunnett, S.J. and Andrews, J.D. (2008). Phased mission modelling of systems with maintenance-free operating periods using simulated petri nets. *Reliability Engineering and System Safety*, 91, 980-994.

Codd, E.F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 15(3): 162-166.

Colom, J.M. and Silva, M. (1991). Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal p-semiflows. *Lecture Notes in Computer Science. Advances in Petri Nets 1990*, 483: 79-112. Springer-Verlag, Berlin.

Connolly, T.M. and Begg, C.E. (2005). *Database systems: A practical approach to design, implementation, and management*. (4. [rev.] ed.), Harlow: Addison-Wesley.

DB-Engines (2018). <https://www.db-engines.com>

de Niz, D. (2007). *Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL*. *Software Engineering Institute*, Carnegie Mellon University.

De Saqui-Sannes, P. and Hugues, P. (2012). Combining SysML and AADL for the design, validation and implementation of critical systems. *ERTSS 2012 (Embedded Real Time Software and Systems)*, France.

Delligatti, L. (2013). *SysML Distilled: A Brief Guide to the Systems Modelling Language*. Addison-Wesley.

Desel, J. and Reisig, W. (1998). *Place/Transition Petri Nets*. Reisig, W. and Rozenberg, G. (eds.) APN 1998. LNCS, 1491: 122-173. Berlin: Springer.

Dugan, J.B., Sullivan, K.J. and Coppit, D. (2000). Developing a Low-Cost, High-Quality Software for Dynamic Fault-Tree Analysis. *IEEE Transactions on Reliability*, 49(1):49-59.

Eclipse. (2015). Available via <http://www.eclipse.org/>

- Evensen, K.D. and Weiss, K.A. (2010). A comparison and evaluation of real-time software systems modelling languages. *Aerospace Conference*, Georgia, Atlanta.
- Feiler, P.H, Gluch, D., Hudak, J. and Lewis, B. (2004). Embedded Systems Architecture Analysis Using SAE AADL. *Carnegie Mellon Software Engineering Institute. Carnegie Mellon Institute CMU/SEI-2004-TN-005*.
- Feiler, P.H. and Lewis, B. (2004). The SAE AADL Standard: An Architecture Analysis & Design Language for Embedded Real-Time Systems. *IFIP World Computer Congress 2004 – Proceedings of the Workshop on Architecture Description Languages*, Toulouse, France.
- Feiler, P.H., Glunch, D.P. and Hudak, J.J. (2006). The Architecture Analysis of Design Language (AADL): An Introduction. *Carnegie Mellon Software Engineering Institute. Carnegie Mellon Institute CMU/SEI-2006-TN-011*.
- Fleming, K.N. and Kalinowski, A.M. (1983). *An Extension of the Beta Factor Method for Systems with High Levels of Redundancy*. PLG-0289. Newport Beach, CA: PLG.
- Fowler, M. (2004). *UML Distilled Third edition: A Brief Guide to the Standard Object Modelling Language*. Addison-Wesley Pearson Education.
- Friedenthal, S., Moore, A. and Steiner R. (2011). *A Practical Guide to SysML: The Systems Modelling Language*. 2nd ed. Morgan Kaufmann, Burlington.
- Fruchterman, T.M.J. and Reingold, E.M. (1991). Graph Drawing by Force-directed Placement. *Software-Practice and Experience*, 21(11):1129-1164.
- Girault, C. and Valk, R. (2003). *Petri Nets for Systems Engineering: A Guide to Modelling, Verification, and Applications*. Springer-Verlag, Berlin.
- Glavic, M. (2006). *Agents and Multi-Agent Systems: A Short Introduction for Power Engineers*. Technical report, University of Liege Electrical Engineering and Computer Science Department.
- Gulati, R. (1996). *A modular approach to static and dynamic fault tree analysis. Master's Thesis*. University of Virginia, Department of Electrical Engineering.

Gulati, R. and Dugan, J. B. (1997). A modular approach for analysing static and dynamic fault trees. *Proceedings of the Annual Reliability and Maintainability Symposium*, USA, (1):57-63.

Hause, M. C. (2006). The Systems Modeling Language – SysML. *INCOSE EuSEC Symposium*, Edinburgh.

Heller, S. (1997). *Introduction to C++*. Academic Press. USA.

Hillston, J. (2017). *Performance Modelling – Lecture 16: Model Validation and Verification*. School of Informatics, The University of Scotland. Available via: <https://pdfs.semanticscholar.org/presentation/5482/9f2231bff0c2ad6b2c8dbce4bee151469839.pdf>

HiPS – Hierarchical petri net simulator. (2017). Shinshu University. Available via: <http://sourceforge.net/projects/hops-tools/>.

Jančar, P., Esparza, J. and Moller, F (1999). Petri Nets and Regular Processes. *Journal of Computer and System Sciences*, 59(3): 476-503.

Jensen, K. (1990). A High-level Language for System Design and Analysis. G. Rozenberg (ed.).

Jensen, K. (1991). Coloured Petri Nets: a High-Level Language for System Design and Analysis. *Advances in Petri Nets 1990*, Rozenberg, G. (ed.), Lecture Notes in Computer Science, (483):342-416. Springer-Verlag, Berlin.

Johansson, L., Cronquist, B. and Kjellin, H. (2007). Visualisation as a tool in action case research. *6th European Conference on Research Methods in Business and Management*, Lisbon, Portugal.

Joshi, A., Vestal, S. and Binns, P. (2007). Automatic Generation of Static Fault Trees form AADL Models. *Proceedings of the IEEE/IFIP Conference on Dependable Systems and Networks' Workshop on Dependable Systems*, Edinburgh, UK.

Kapos, G.D., Dalakas, V, Nikolaidou, M. and Anagnostopoulos, D. (2014). An integrated framework for automated simulation of SysML using DEVS. *Transactions of the Society for Modeling and Simulation International*, 90(6): 717-744.

- Karban, R., Weilkiens, T., Hauber, R., Zamparelli, M., Diekmann, R. and Hein, A., M. (2011). *Cookbook for MBSE with SysML*. MBSE initiative – SE2 challenge team.
- Katayama, T., Zhao, Z., Kita, Y., Yamaba, H. and Okazaki, N. (2014). Proposal of a Method to Build Markov Chain Models from UML Diagrams for Communication Delay Testing in Distributed Systems. *Journal of Robotics, Networking and Artificial Life*, 1(2): 120-124.
- Khabbazi, M.R., Hasan, M.K., Sulaiman, R. and Shapi'i A. (2013). Business Process Modelling in Production Logistics: Complementary Use of BPMN and UML. *Middle-East Journal of Scientific Research*, 15(4):516-529.
- Landeghem R.V. and Bobeanu C.-V. (2002). Formal modelling of supply chain: an incremental approach using Petri Nets. *Proceedings of 14th SCS Europe BVBA*. Dresden, Germany.
- Lanus, M., Yin. L. and Trivedi, K.S. (2003). Hierarchical composition and aggregation of state-based availability and performability models. *IEEE Transactions on Reliability*, 52(1): 44-52.
- Lapp, S. A. and Powers, G. J. (1977). Computer-aided Synthesis of Fault Trees. *IEEE Transactions on Reliability*, 26(1):2-13.
- Law, A. (2005). How to Build Valid and Credible Simulation Models. *Proceeding of the 2005 Winter Simulation Conference*.
- Li, Z.W. and Zhou, M.C. (2009). *Deadlock resolution in automated manufacturing systems: A novel Petri net approach*. Springer-Verlag, London.
- Liu, X., Ren, Y., Wang, Z., and Liu, L. (2013). Modelling method of SysML-based reliability block diagram. *Proceeding 2013 International Conference on Mechatronic Sciences, Electrical Engineering and Computer (MEC)*: 206-209.
- Majdara, A. and Wakabayashi, T. (2010). Automated fault tree construction for a sample chemical plant. *Journal of Risk and Reliability*, 224(3): 207-216.
- Mandrioli, D., Morzenti, A., Pezze, M., San Pietro, P. and Silva, S. (1996). A Petri Net and logic approach to the specification and verification of real time systems.

Formal Methods for Real Time Computing (C. Heitmeyer and D. Mandrioli, eds.). New York: Wiley.

Marsan, M.A. (1990). Stochastic Petri Nets: An Elementary Introduction. Rozenberg, G. (de.) *Advances in Petri Nets 1989*, LNCS, 424, 1-29. Springer-Verlag, Berlin.

MathWorks. (2018). Available via <http://www.mathworks.com/>

Mhairi, S.K. (2009) *The Impact of Petri Nets in System-of-Systems Engineering*. Durham theses, Durham University. Available via Durham E-Theses Online <http://etheses.dur.ac.uk/212>

Mheni F., Nguyen, N. and Choley, J.Y. (2014). Automatic fault tree generation from SysML system models. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 715-720.

Microsoft (2015). <http://www.microsoft.com>

Mosleh, A., Fleming, K.N., Parry, G.W., Paula, H.M., Worledge, D.H. and Rasmuson, D.M. (1988). *Procedures for Treating Common Cause Failures in Safety and Reliability Studies*. NUREG/CR-4780 (EPRI NP-5613), PLG-0547, 1.

MSDN Microsoft. (2017). <http://msdn.microsoft.com>

Mura, I. and Bondavalli, A. (2001). Markov Regenerative Stochastic Petri Nets to Model and Evaluate the Dependability of Phased Missions. *IEEE Transactions on Computers*, 50(1):1337-1351.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4): 541-580.

MySQL Workbench. (2018). Available via <http://www.mysql.com>

Obaidat, M.S. and Boudriga, N.A. (2010). *Fundamentals of performance of computer and telecommunications systems*. John Wiley & Sons, Hoboken, New Jersey.

Object Management Group (OMG). (2005). *OMG Unified Modeling Language (OMG UML): Infrastructure, Version 2.0*. Available via www.omg.org

- Object Management Group (OMG). (2015). *OMG Unified Modeling Language (OMG UML), Version 2.5*. <https://www.omg.org/spec/UML/2.5/About-UML/>
- Papadopoulos, Y. and Maruhn, M. (2001). Model-based Synthesis of Fault Trees from Matlab-Simulink models. *International Conference on Dependable Systems and Networks (DSN 2001)*, (1): 77-82.
- Papadopoulos, Y., McDermid, J.A. and Heiner, G. (2001). Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering and System Safety*, 71(3): 229-247.
- Papadopoulos, Y. and Grante, C. (2005). Evolving car designs using model-based automated safety analysis and optimisation technique. *The Journal of Systems and Software*, 76(1): 77-89.
- Petri, C.A. (1962). *Kommunikation with Automaten*. English Translation, 1966: *Communication with Automata*, Technical Report RADC-TR-65-377, Rome Air Dev. Centre, New York.
- Pilone, D. and Pitman, N. (2005). *UML2.0 in a Nutshell*. (In a Nutshell (O'Reilly)). O'Reilly Media.
- Point, G. and Rauzy, A. (1999). AltaRica: Constraint automata as a description language. *Journal Européen des Systèmes Automatisés*, 33(8-9):1033-1052.
- PostgreSQL. (2018). <http://www.postgresql.org>
- Prosvirnova, T. and Rauzy, A. (2013). AltaRica 3.0 project: compile Guarded Transition Systems into Fault Trees. *European Safety, Reliability and Reliability Conference, ESREL 2013*, 1121- 1128.
- Proth, J.-M. and Xie, X. (1996). *Petri nets: a tool for design and management of manufacturing systems*. John Wiley & Sons.
- Rauzy, A. (2002). Mode automata and their compilation into fault trees. *Reliability Engineering and System Safety*, 78: 1-12.
- Raychaudhuri, S. (2008). Introduction to Monte Carlo Simulation. Proceedings of the 40th Conference on Winter Simulation. Miami, New York, 91-100.

Recalde, L., Teruel, E. and Silva, M. (1998). On linear algebraic techniques for liveness analysis of P/T systems. *Journal of Circuits, Systems and Computers*, 8(1): 223-265.

Reza, H. and Chatterjee, A. (2014). Mapping AADL to Petri Net Tool-Sets Using PNML Framework. *Journal of Software Engineering and Application*, 7: 920-933.

Richardeau, F. and Pham, T.T.L. (2013). ‘Reliability Calculation of Multilevel Converters: Theory and Applications.’ *IEEE Transactions on Industrial Electronics*, 60(10): 4225-4233.

Riggs, S. and Krosing, H. (2010). *PostgreSQL 9 Administration Cookbook: Solve real-world PostgreSQL problems with over 100 simple, yet incredibly effective recipes*. Birmingham: Packt Publishing Ltd. ISBN 978-1-849510-28-8.

Robidoux, R., Xu, H., Xing, L., & Zhou, M. (2009). Automated Modelling of Dynamic Reliability Block Diagrams Using Coloured Petri Nets. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 40(2): 337-351.

Robinson, S. (1994). *Successful Simulation: A Practical Approach to Simulation Projects*. Maidenhead, UK: McGraw-Hill.

Robinson, S. (1997). Simulation model verification and validation: Increasing the user’s confidence. *Proceeding of the Winter Simulation Conference*, 53-59.

Salem, S.L., Apostolakis, G.E. and Okrent, D. (1977). A new methodology for the computer-aided construction of fault trees. *Annals of Nuclear Energy*, 4(9-10): 417-433.

Salem, S.L., Wu, J.S. and Apostolakis, G.E. (1979). Decision Table Development and Application to the Construction of Fault Trees, *Nuclear Technology*, 42: 51-64.

Sargent, R.G. (1992). Validation and Verification of Simulation Models. *Proceedings of 1992 Winter Simulation Conference*, Arlington, Virginia, USA, 104-114.

Schneeweiss, W.G. (1999). *Petri nets for reliability modelling: in the fields of engineering safety and dependability*. LiLoLe Verlag GmbH, Hagen, Germany.

Söding, R. (2009). A Brief Introduction into UML 2. Available via: <http://www.metagear.de>

SPARX Systems. (2018). Available via: <http://www.sparxsystems.com>

Stockwell, K.S. and Dunnett, S.J. (2013). Automatic construction of a reliability model for a phased mission system. *Proceedings of the 20th Advances in Risk and Reliability Technology Symposium*, 192-204.

Taibi, M., Ioualalen M. and Abdmeziem, R. (2013). An Automatic Petri-net Generator for Modelling Multi-agent Systems. *Proceedings of the 8th International Conference on Software Engineering Advances*, 128-133.

Tangkawarow I.R.H.T. and Waworuntu, J. (2016). A Comparative of business process modelling techniques. *IOP Conference Series: Materials Science and Engineering*, 128(1): 1-16.

Tsai, L.S. and Chang, Y. (1995). Timing Constraint Petri Nets and Their Application to Schedulability Analysis of Real-Time System Specifications. *IEEE Transactions on Software Engineering*, 21(1): 32-49.

Valaityte, A., Dunnett, S.J. and Andrews, J.D. (2010). Development of an algorithm for automated cause-consequence diagram construction. *International Journal of Reliability and Safety*, 4(1): 46-68.

van der Aalst, W.M.P. (1998). The Application of Petri nets to Workflow Managements. *Journal of Circuits Systems and Computers*, 8(1): 21-66.

van der Aalst, W.M.P. (1999). Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10): 639-650.

van der Aalst, W.M.P. (2011). Alpha Algorithm: Limitations. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin.

van der Aalst, W.M.P. (2011). *Analysis of Process Models: Introduction, state space analysis and simulation in CPN Tools*. [Power Point Presentation] Available via: cpntools.org/wp-content/uploads/2018/01/analysis.pdf (Accessed: 07 June 2017).

van der Aalst, W.M.P., Weijters, A.J.M.M. and Mărușter, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9): 1128-1142.

van Landeghem, R. and Bobeanu, C.-V. (2002). Formal modelling of supply chain: An incremental approach using Petri nets. *14th European Simulations Symposium and Exhibition*.

Venkatesh, K., Zhou, M. and Claudill, R., J. (1994). Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design Through a Discrete Manufacturing System. *IEEE Transactions on Industrial Electronics*, 41(6): 611-619.

Villani, E., Miyagi, P.E. and Valette, R. (2007). *Modelling and Analysis of Hybrid Supervisory Systems: A Petri Net Approach*. Advances in Industrial Control, Springer-Verlag, London.

Villemeur, A. (1992). *Reliability, Availability, Maintainability and Safety Assessment, Vol.1: Methods and Techniques*. Wiley, New York.

Volovoi, V. V. (2004). Modeling of system reliability Petri Nets with aging tokens. *Reliability Engineering and System Safety*, 84(2): 149-161.

Volovoi, V.V. (2013). Abridged Petri Nets. *ArXiv*, arXiv: 1312.2865.

Vyzaite, G., Dunnett, S.J. and Andrews, J.D. (2005). Cause-consequence analysis of non-repairable phased missions. *Reliability Engineering & System Safety*, 91(4):398-406.

Walker, M. and Papadopoulos, Y. (2009). Qualitative temporal analysis: towards a full implementation of the fault tree handbook. *Control Engineering Practise*, 71(10): 1115-1125.

Wang, J. (2006). *Petri nets dynamic event-driven system modelling*. Paul Fishwick (eds), Handbook of Dynamic System Modeling, 1-17. CRC Press.

Wang, P. (2017). *Civil Aircraft Electrical Power System Safety Assessment: Issues and Practices*. Civil Aviation University of China, Tianjin, China, 270-276.

Xie, G, Xue, D. and Xi, S. (1993). Tree-Expert: A tree based expert system for fault tree construction. *Reliability Engineering and System Safety*, 40(1): 295-309.

Zhao, C., Bhushan, M. and Venkatasubramanian, V. (2005). PHASUITE: An automated HAZOP analysis tool for chemical processes: Part I. Knowledge Engineering Framework. *Process Safety and Environmental Protection*, 83(B6): 509-532.

Zhou, M.C. and DiCesare, F. (1989). Adaptive design of Petri Net controllers for error recovery in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5): 963-973.

Zille, V., Bérenguer, C., Grall, A. and Despujols, A. (2010). Simulation of maintained multicomponent systems for dependability assessment. In Faulin, Javier and Juan, Angel A. and Martorell, Sebastian and Ramirez-Marquez, J.E. (eds), *Simulation Methods for Reliability and Availability of Complex Systems*, Springer Series in Reliability Engineering, 12(1): 253-272. London: Springer London.

Appendix A - Simple Process Example (XMI File)

Appendix A includes the XMI file for the simple process shown in Chapter 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:uml="http://www.eclipse.org/UML/2/5.0.0/UML"
xmi:id="_vGtecNaPEeeXUKMyPHN3Zw" name="RootElement">
  <packagedElement xmi:type="uml:Activity" xmi:id="_vIoKANAPEeeXUKMyPHN3Zw"
name="Activity1" node="_xBDrQNaPEeeXUKMyPHN3Zw _yMFT8NaPEeeXUKMyPHN3Zw
_0KiksNaPEeeXUKMyPHN3Zw _2ctboNaPEeeXUKMyPHN3Zw _XoMwINaQEee330p70iFb5A
_YsEn8NaQEee330p70iFb5A _aBN18NaQEee330p70iFb5A _b330gNaQEee330p70iFb5A">
    <edge xmi:type="uml:ControlFlow" xmi:id="_f-3gINaQEee330p70iFb5A"
target="_0KiksNaPEeeXUKMyPHN3Zw" source="_xBDrQNaPEeeXUKMyPHN3Zw"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_gyJXMNaQEee330p70iFb5A"
target="_b330gNaQEee330p70iFb5A" source="_0KiksNaPEeeXUKMyPHN3Zw"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_hjulINaQEee330p70iFb5A"
name="action_1_pass" target="_2ctboNaPEeeXUKMyPHN3Zw"
source="_b330gNaQEee330p70iFb5A"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_iV_IgNaQEee330p70iFb5A"
name="action_1_fail" target="_XoMwINaQEee330p70iFb5A"
source="_b330gNaQEee330p70iFb5A"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_jNf_UNaQEee330p70iFb5A"
target="_aBN18NaQEee330p70iFb5A" source="_2ctboNaPEeeXUKMyPHN3Zw"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_j433kNaQEee330p70iFb5A"
target="_aBN18NaQEee330p70iFb5A" source="_XoMwINaQEee330p70iFb5A"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_kLTfwNaQEee330p70iFb5A"
target="_YsEn8NaQEee330p70iFb5A" source="_aBN18NaQEee330p70iFb5A"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_LRWGYNaQEee330p70iFb5A"
target="_yMFT8NaPEeeXUKMyPHN3Zw" source="_YsEn8NaQEee330p70iFb5A"/>
    <node xmi:type="uml:InitialNode" xmi:id="_xBDrQNaPEeeXUKMyPHN3Zw"
name="pin" outgoing="_f-3gINaQEee330p70iFb5A"/>
    <node xmi:type="uml:ActivityFinalNode" xmi:id="_yMFT8NaPEeeXUKMyPHN3Zw"
name="pout" incoming="_LRWGYNaQEee330p70iFb5A"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_0KiksNaPEeeXUKMyPHN3Zw"
name="Action_1" incoming="_f-3gINaQEee330p70iFb5A"
outgoing="_gyJXMNaQEee330p70iFb5A"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_2ctboNaPEeeXUKMyPHN3Zw"
name="Action_2" incoming="_hjulINaQEee330p70iFb5A"
```

```

outgoing="_jNf_UNaQEee330p70iFb5A"/>
  <node xmi:type="uml:OpaqueAction" xmi:id="_XoMwINaQEee330p70iFb5A"
name="Action_3" incoming="_iV_IgNaQEee330p70iFb5A"
outgoing="_j433kNaQEee330p70iFb5A"/>
  <node xmi:type="uml:OpaqueAction" xmi:id="_YsEn8NaQEee330p70iFb5A"
name="Action_4" incoming="_kLTfwNaQEee330p70iFb5A"
outgoing="_LRWGYNaQEee330p70iFb5A"/>
  <node xmi:type="uml:MergeNode" xmi:id="_aBN18NaQEee330p70iFb5A"
name="Merge_1" incoming="_jNf_UNaQEee330p70iFb5A _j433kNaQEee330p70iFb5A"
outgoing="_kLTfwNaQEee330p70iFb5A"/>
  <node xmi:type="uml:DecisionNode" xmi:id="_b330gNaQEee330p70iFb5A"
name="Decision_1" incoming="_gyJXMNaQEee330p70iFb5A"
outgoing="_hjuLlNaQEee330p70iFb5A _iV_IgNaQEee330p70iFb5A"/>
  </packagedElement>
</uml:Model>

```

Appendix B – SQL Code [A^T]

Appendix B shows the SQL code developed for the automated generation of the transpose of the PN incidence matrix [A^T], discussed in Chapter 3.

```
package data;
import java.sql.*;

public class incidence_matrix_s2s{
    static String t1;
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://127.0.0.1:3306/sql";
    // Database credentials
    static final String USER = "root";
    static final String PASS = "Xristina23";

    public static void main(String[] args) throws SQLException {
        Connection con = null;
        Statement stmt = null;
        PreparedStatement pst = null;
        ResultSet rs = null;

        try {
            con = DriverManager.getConnection(DB_URL, USER, PASS);
String code1 = "drop table if exists edge_xmi;"
            + "CREATE TABLE edge_xmi (id int NOT NULL AUTO_INCREMENT
PRIMARY KEY, `xmi:type` VARCHAR(200) NULL, `xmi:id` VARCHAR(200) NULL,
`name` VARCHAR(200) NULL, `source` varchar(200)null, `target` VARCHAR(200)
NULL) ";
String code2 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/Activity_Diagram_S2S.uml' INTO TABLE
edge_xmi ROWS IDENTIFIED BY '<edge>' ; "
            + "update edge_xmi as t1 inner join edge_xmi as t2 on
(t1.`name`=t2.`name`) and t1.`xmi:id` <> t2.`xmi:id` set t1.name='place_'
;"
            + "drop table if exists place_name ;"
            + "CREATE TABLE place_name as SELECT id, `xmi:type`, `xmi:id`,
CONCAT(name,'', id) AS name, source, target FROM edge_xmi where
edge_xmi.name='place_' ;"
            + "drop table if exists edge_place_xmi ;"
            + "CREATE TABLE edge_place_xmi SELECT * FROM place_name UNION
SELECT * FROM edge_xmi ;"
            + "ALTER IGNORE TABLE `edge_place_xmi` ADD UNIQUE (id,
`xmi:id`) ;"
            + "drop table if exists node_xmi ;"
            + "CREATE TABLE node_xmi (id int NOT NULL AUTO_INCREMENT
PRIMARY KEY, `xmi:type` VARCHAR(200) NULL, `xmi:id` VARCHAR(200) NULL,
name VARCHAR(200) NULL, incoming VARCHAR(200) NULL, outgoing
varchar(200)null);";

String code3 = " LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/Activity_Diagram_S2S.uml' INTO TABLE
node_xmi ROWS IDENTIFIED BY '<node>'; ";
String code4 = "drop table if exists double_nodes_outgoing ;"
            + "create table double_nodes_outgoing as select * from node_xmi
where outgoing like '%_%' ;"
```

```

+ "drop table if exists numbers ;"
+ "create table numbers (n int not null) ;"
+ "insert into numbers (n) values (1), (2), (3), (4), (5), (6),
(7), (8), (9),(10), (11), (12), (13), (14), (15) ;"
+ "drop table if exists double_separate_nodes ;"
+ "create table double_separate_nodes as select
double_nodes_outgoing.id, `xmi:type`, `xmi:id`, `name`, incoming,
SUBSTRING_INDEX(SUBSTRING_INDEX(double_nodes_outgoing.outgoing, '_',
numbers.n), '_', -1) outgoing from numbers inner join double_nodes_outgoing
on CHAR_LENGTH(double_nodes_outgoing.outgoing)-
CHAR_LENGTH(REPLACE(double_nodes_outgoing.outgoing, '_', ''))>=numbers.n-1
order by id, n, `xmi:type`, `xmi:id`, `name`, incoming, outgoing ;"
+ "DELETE FROM double_separate_nodes WHERE outgoing = ' ' ;"
+ "update double_separate_nodes set outgoing = concat('_',
outgoing) ;"
+ "alter table `double_separate_nodes` change column outgoing
outgoing varchar(255) after `id` ;"
+ "drop table if exists lessthan17 ;"
+ "create table lessthan17 (id int NOT NULL AUTO_INCREMENT
PRIMARY KEY) as SELECT `outgoing`, `xmi:type`, `xmi:id`, `name`, incoming
FROM double_separate_nodes WHERE LENGTH(outgoing) < 22 ;"
+ "drop table if exists merge_shorter_than22 ;"
+ "create table merge_shorter_than22 (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY) as SELECT GROUP_CONCAT(outgoing SEPARATOR ''),
`xmi:type`, `xmi:id`, `name`, incoming FROM lessthan17 GROUP BY name ;"
+ "ALTER TABLE `merge_shorter_than22` CHANGE COLUMN
`GROUP_CONCAT(outgoing SEPARATOR '')` `outgoing` VARCHAR(255) NOT NULL ;"
+ "drop table if exists union_1 ;"
+ "create table union_1 select * from merge_shorter_than22
union all select * from double_separate_nodes ;"
+ "DELETE FROM union_1 where LENGTH(outgoing) < 21 ;"
+ "drop table if exists double_nodes_incoming ;"
+ "create table double_nodes_incoming as select * from node_xmi
where incoming like '%_ %' ;"
+ "drop table if exists double_separate_nodes ;"
+ "create table double_separate_nodes as select
double_nodes_incoming.id, `xmi:type`, `xmi:id`, `name`, outgoing,
SUBSTRING_INDEX(SUBSTRING_INDEX(double_nodes_incoming.incoming, '_',
numbers.n), '_', -1) incoming from numbers inner join double_nodes_incoming
on CHAR_LENGTH(double_nodes_incoming.incoming)-
CHAR_LENGTH(REPLACE(double_nodes_incoming.incoming, '_', ''))>=numbers.n-1
order by id, n, `xmi:type`, `xmi:id`, `name`, incoming, outgoing ;"
+ "DELETE FROM double_separate_nodes WHERE incoming = ' ' ;"
+ "update double_separate_nodes set incoming = concat('_',
incoming) ;"
+ "alter table `double_separate_nodes` change column incoming
incoming varchar(255) after `id` ;"
+ "drop table if exists lessthan17 ;"
+ "create table lessthan17 (id int NOT NULL AUTO_INCREMENT
PRIMARY KEY) SELECT `outgoing`, `xmi:type`, `xmi:id`, `name`, incoming FROM
double_separate_nodes WHERE LENGTH(incoming) < 22; "
+ "drop table if exists merge_shorter_than22 ;"
+ "create table merge_shorter_than22 (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY) as SELECT GROUP_CONCAT(incoming SEPARATOR ''),
`xmi:type`, `xmi:id`, `name`, outgoing FROM lessthan17 GROUP BY name ;"
+ "ALTER TABLE `merge_shorter_than22` CHANGE COLUMN
`GROUP_CONCAT(incoming SEPARATOR '')` `incoming` VARCHAR(255) NOT NULL ;"
+ "drop table if exists union_2a ;"
+ "create table union_2a select * from merge_shorter_than22

```

```

union all select * from double_separate_nodes ;"
+ "drop table if exists union_2 ;"
+ "create table union_2 as select id, `outgoing`, `xmi:type`,
`xmi:id`, `name`, incoming from union_2a ;"
+ "DELETE FROM union_2 where LENGTH(incoming) < 21 ;"
+ "drop table if exists unique_activities ;"
+ "create table unique_activities as select id, outgoing,
`xmi:type`, `xmi:id`, `name`, incoming from node_xmi ;"
+ "DELETE FROM unique_activities where incoming like '%%_'"
;"
+ "DELETE FROM unique_activities where outgoing like '%%_'"
;"
+ "drop table if exists union_node ;"
+ "create table union_node select * from union_1 union select *
from union_2 union select * from unique_activities ;"
+ "alter table union_node drop column id ;"
+ "alter table union_node add column id int NOT NULL
AUTO_INCREMENT PRIMARY KEY FIRST ;"
+ "drop table if exists union_node_table1 ;"
+ "CREATE TABLE union_node_table1 (`place_before_node`
VARCHAR(200) NULL, `place_after_node` VARCHAR(200) NULL) as SELECT id,
`xmi:type`, `xmi:id`, `name`, incoming, outgoing FROM union_node where
(union_node.`xmi:type`='uml:OpaqueAction') or
(union_node.`xmi:type`='uml:MergeNode') or
(union_node.`xmi:type`='uml:DecisionNode');"
+ "ALTER TABLE union_node_table1 CHANGE COLUMN `xmi:id`
`xmi:id_primary` VARCHAR(255) NULL ;"
+ "ALTER TABLE union_node_table1 CHANGE COLUMN `name`
`name_primary` VARCHAR(255) NULL ;"
+ "drop table if exists union_node_node ;"
+ "CREATE TABLE union_node_node AS SELECT m.*, u2.`xmi:id`,
u2.`name` FROM union_node_table1 m INNER JOIN edge_place_xmi u2 ON
(m.`incoming`= u2.`xmi:id`) or (m.`outgoing`= u2.`xmi:id`) ;"
+ "UPDATE union_node_table1 t1 INNER JOIN union_node_node t2 ON
t1.outgoing = t2.`xmi:id` SET t1.place_after_node = t2.name ;"
+ "UPDATE union_node_table1 t1 INNER JOIN union_node_node t2 ON
t1.incoming = t2.`xmi:id` SET t1.place_before_node = t2.name ;"
+ "delete from union_node_table1 WHERE (place_before_node is
null) and (place_after_node is null) ;"
+ "drop table if exists final_table ;"
+ "CREATE TABLE final_table SELECT `place_before_node`,
`name_primary`, `place_after_node` FROM union_node_table1;"
+ "drop table if exists initial_final_table ;"
+ "CREATE TABLE initial_final_table (`transition_before_node`
VARCHAR(200) NULL, `transition_after_node` VARCHAR(200) NULL) as SELECT id,
`xmi:type`, `xmi:id`, `name` FROM union_node where
(union_node.`xmi:type`='uml:ActivityFinalNode') or
(union_node.`xmi:type`='uml:FlowFinalNode') or
(union_node.`xmi:type`='uml:InitialNode');"
+ "ALTER TABLE initial_final_table CHANGE COLUMN `xmi:id`
`xmi:id_primary` VARCHAR(255) NULL ;"
+ "drop table if exists final_node;"
+ "CREATE TABLE final_node AS SELECT m.*, u1.target, u1.source
FROM initial_final_table m INNER JOIN edge_place_xmi u1 ON
(m.`xmi:id_primary`= u1.target);"
+ "ALTER TABLE `final_node` ADD COLUMN `xmi:id` VARCHAR(255)
NOT NULL ;"

```

```

+ "ALTER TABLE `final_node` DROP COLUMN `transition_after_node`
;"
+ "UPDATE final_node t1 INNER JOIN edge_place_xmi t2 ON
t1.target = t2.target SET t1.`xmi:id` = t2.`xmi:id` ;"
+ "UPDATE final_node INNER JOIN union_node ON final_node.source
= union_node.`xmi:id` SET final_node.transition_before_node =
union_node.name ;"
+ "drop table if exists final_node_table ;"
+ "create table final_node_table as select
`transition_before_node`,`name` from final_node ;"
+ "ALTER TABLE final_node_table CHANGE COLUMN `name`
`place_after_node` VARCHAR(255) NULL ;"
+ "ALTER TABLE final_node_table CHANGE COLUMN
`transition_before_node` `name_primary` VARCHAR(255) NULL ;"
+ "ALTER TABLE final_node_table ADD COLUMN `place_before_node`
VARCHAR(255) NULL FIRST;"
+ "drop table if exists initial_node ;"
+ "CREATE TABLE initial_node AS SELECT m.*, u2.target,
u2.source FROM initial_final_table m INNER JOIN edge_place_xmi u2 ON
(m.`xmi:id_primary`= u2.source) ;"
+ "ALTER TABLE `initial_node` ADD COLUMN `xmi:id` VARCHAR(255)
NOT NULL ;"
+ "UPDATE initial_node t1 INNER JOIN edge_place_xmi t2 ON
t1.source = t2.source SET t1.`xmi:id` = t2.`xmi:id` ;"
+ "ALTER TABLE `initial_node` DROP COLUMN
`transition_before_node` ;"
+ "UPDATE initial_node INNER JOIN union_node ON
initial_node.target = union_node.`xmi:id` SET
initial_node.transition_after_node = union_node.name ;"
+ "drop table if exists initial_node_table; "
+ "create table initial_node_table as select `name`,
`transition_after_node` from initial_node ;"
+ "ALTER TABLE initial_node_table CHANGE COLUMN `name`
`place_before_node` VARCHAR(255) NULL ;"
+ "ALTER TABLE initial_node_table CHANGE COLUMN
`transition_after_node` `name_primary` VARCHAR(255) NULL ;"
+ "ALTER TABLE initial_node_table ADD COLUMN `place_after_node`
VARCHAR(255) NULL ;"
+ "update final_table as t1 inner join final_node_table as t2
on (t1.`name_primary`=t2.`name_primary`) set
t1.place_after_node=t2.place_after_node;"
+ "update final_table as t1 inner join initial_node_table as t2
on (t1.`name_primary`=t2.`name_primary`) set
t1.place_before_node=t2.place_before_node;"
+ "drop table if exists null_after ;"
+ "CREATE TABLE null_after as SELECT * FROM final_table where
place_after_node is null and name_primary in (select name_primary from
final_table GROUP BY name_primary HAVING COUNT(*)>1) ;"
+ "UPDATE null_after na, final_table mt SET na.place_after_node
= mt.place_after_node WHERE na.name_primary = mt.name_primary and
mt.place_after_node is not null and mt.place_before_node is null ;"
+ "drop table if exists null_before ;"
+ "CREATE TABLE null_before as SELECT * FROM final_table where
place_before_node is null and name_primary in (select name_primary from
final_table GROUP BY name_primary HAVING COUNT(*)>1) ;"
+ "UPDATE null_before na, final_table mt SET
na.place_before_node = mt.place_before_node WHERE na.name_primary =
mt.name_primary and mt.place_before_node is not null and
mt.place_after_node is null ;"

```

```

+ "drop table if exists final_table ;"

+ "CREATE TABLE final_table select * from null_after union
select * from null_before union select * from final_table ; "
+ "DELETE n1 FROM final_table n1 JOIN final_table n2 ON
n1.name_primary = n2.name_primary AND n1.place_before_node =
n2.place_before_node and n1.place_after_node is null ;"
+ "DELETE n1 FROM final_table n1 JOIN final_table n2 ON
n1.place_after_node = n2.place_after_node AND n1.place_before_node is null
AND n1.name_primary = n2.name_primary ;"
+ "alter table final_table add column id int AUTO_INCREMENT
primary key;"

+ "drop table negative_records;"
+ "create table negative_records ( id int not null
AUTO_INCREMENT PRIMARY KEY) as select distinct place_before_node,
name_primary from final_table;"
+ "drop table positive_records;"
+ "create table positive_records ( id int not null
AUTO_INCREMENT PRIMARY KEY) as select distinct name_primary,
place_after_node from final_table;";

pst = con.prepareStatement(code1);
boolean isResult1 = pst.execute();
pst = con.prepareStatement(code2);
boolean isResult2 = pst.execute();
pst = con.prepareStatement(code3);
boolean isResult3 = pst.execute();
pst = con.prepareStatement(code4);
boolean isResult4 = pst.execute();

String code111 = "drop table if exists negative;"
+ "SET group_concat_max_len=15000;"
+ "SELECT CONCAT('create table
negative as SELECT place_before_node,', GROUP_CONCAT(sums), 'FROM
negative_records GROUP BY id') FROM (SELECT distinct CONCAT('(case when
negative_records.name_primary = '', name_primary, '' then -1 else 0
end) as `', name_primary, ``')sums FROM negative_records GROUP BY id) s
INTO @sql;"

+ "PREPARE stmt FROM @sql;"

"

+ "EXECUTE stmt; "
+ "DEALLOCATE PREPARE

stmt";
pst = con.prepareStatement(code111);
isResult111 = pst.execute();

String code222 = "drop
table if exists positive;"

+ "SELECT CONCAT('create
table positive as SELECT place_after_node,', GROUP_CONCAT(sums), 'FROM
positive_records GROUP BY id') FROM (SELECT distinct CONCAT('(case when
positive_records.name_primary = '', name_primary, '' then 1 else 0 end)
as `', name_primary, ``')sums FROM positive_records GROUP BY id) s INTO
@sql;"

+ "PREPARE

stmt FROM @sql;"

+ "EXECUTE

stmt;"

```



```

+ "DEALLOCATE
PREPARE stmt;"
+ "drop table if exists overall;"
+ "create
table overall SELECT * FROM positive UNION SELECT * FROM negative;"
+ "drop table
if exists schema_table; "
+ "create
table schema_table as select * from overall;"
+ "drop table
if exists column_table; "
+ "create
table column_table (primary_id int NOT NULL AUTO_INCREMENT PRIMARY KEY) as
select column_name from information_schema.columns where
table_name='overall';"
+ "DELETE
FROM column_table where primary_id=1;"
+ "drop table
if exists matrix_pass_fail;"
+ "drop table
if exists incidence_matrix_single_device;"
+ "drop table
if exists table_union1";
pst =
con.prepareStatement(code222);          boolean isResult222 =
pst.execute();

String code334 = "set session
sql_mode = 'NO_ENGINE_SUBSTITUTION';"
+ "DROP PROCEDURE IF
EXISTS `Te`;"
+ "SET group_concat_max_len= 150000;";
code334 += "CREATE PROCEDURE `Te`()";
code334 += "BEGIN ";
code334 += "create table
matrix_pass_fail (column_name varchar(150000)) ";
code334 += " SELECT
@query7:=GROUP_CONCAT(CONCAT('sum(`',column_name,`)`,`',column_name,`')) "
+ "AS column_name from
column_table order by CHAR_LENGTH(column_name); ";
code334 += "PREPARE stmt FROM
@query7; ";
code334 += "EXECUTE stmt;";
code334 += "DEALLOCATE PREPARE
stmt;";
code334 += "END ";
pst.execute(code334); boolean
isResult334 = pst.execute();

String query1
= "Call Te()";

pst = con.prepareStatement(query1);
boolean isResultA = pst.execute();
}
catch(SQLException e){}
try{ if (rs != null) rs.close();

```

```

        if (pst != null) pst.close();
        if (con != null) con.close();}
    catch(Exception e){}

    try{
        con

=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/sql",
"root","Xristina23");

        pst =con.prepareStatement("select * from
matrix_pass_fail");
        rs= pst.executeQuery();

        while (rs.next())
            t1=(("create table table_union1 as select place_after_node,")
+ rs.getString("column_name")+ (" ") +("from overall group by
place_after_node;"));
        System.out.println(t1);
    }
    catch(SQLException e){} try{ if (rs != null) rs.close(); if
(pst != null) pst.close();
    if (con != null) con.close();}catch(Exception e){}
    try{
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");
        //STEP 3: Open a connection
        System.out.println("Connecting to a selected
database...");
        con = DriverManager.getConnection(DB_URL, USER, PASS);
        System.out.println("Connected database
successfully...");
        //STEP 4: Execute a query
        System.out.println("Creating table in given
database...");

        stmt = con.createStatement();
        String sql1 = t1;
        stmt.executeUpdate(sql1);

        System.out.println("Created table in given
database...");
    }catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
    }catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
    }finally{
        //finally block used to close resources
        try{
            if(stmt!=null)
                con.close();
        }catch(SQLException se){
        }// do nothing
        try{
            if(con!=null)
                con.close();
        }catch(SQLException se){
            se.printStackTrace();

```

```

        } //end finally try
        } //end try
    try {
        con = DriverManager.getConnection(DB_URL, USER, PASS);
        String t2=("create table
incidence_matrix_single_device as select * from table_union1 GROUP by
place_after_node asc;");

        String query = "select * from
incidence_matrix_single_device;";
        pst = con.prepareStatement(t2);
        isResultt2 = pst.execute();
        pst = con.prepareStatement(query);
        isResult = pst.execute();
        do {
            rs = pst.getResultSet();
            ResultSetMetaData rsmd = rs.getMetaData();
            int columnsNumber = rsmd.getColumnCount();
            int col = rsmd.getColumnCount();
            for (int i = 1; i <= col; i++){
                String col_name = rsmd.getColumnName(i);
                System.out.print(col_name + " ");
            }
            System.out.println("
");
            // Iterate through the data in the result set and
            display it.

            while (rs.next()) {
                //Print one row
                for(int i = 1 ; i <= columnsNumber; i++){
                    System.out.print(rs.getString(i) + "
"); //Print one element of a row
                }
                System.out.println();
            }
            isResult = pst.getMoreResults();
        }
        while (isResult);
    }
    finally {
        if (rs != null) {
            rs.close(); }
        if (pst != null) {
            pst.close();}
        if (con != null) {
            con.close();
        }
    }
    System.out.println("Goodbye!");
}
} //end main
} //end JDBCExample

```

Appendix C – SQL Code [M₀]

Appendix C presents the SQL code developed for the automated generation of the PN initial marking matrix [M₀], discussed in Chapter 3.

```
package step1_initial_marking;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
public class initial_marking {
    public static void main(String[] args) throws SQLException {
        Connection con = null;
        PreparedStatement pst = null;
        ResultSet rs=null;
        String cs =
"jdbc:mysql://localhost:3306/sql?allowMultiQueries=true";
        String user = "root";
        String password = "Xristina23";
        try {
            con = DriverManager.getConnection(cs, user, password);
            String code1 = "SET SQL_SAFE_UPDATES=0;"
                + "drop table if exists initial_marking;"
                + "create table initial_marking (primary_id int not null
auto_increment primary key, activity varchar(250),
process_number_of_devices int);"
                + "insert into initial_marking (activity) select
place_after_node from incidence_matrix_single_device;"
                + "drop table if exists m0_marking;"
                + "create table m0_marking (primary_id int not null
auto_increment primary key) SELECT IFNULL(process_number_of_devices, 0)
FROM initial_marking;"
                + "ALTER TABLE `m0_marking` CHANGE COLUMN
`IFNULL(process_number_of_devices, 0)` `process_number_of_devices` int;"
                + "alter table initial_marking drop column
process_number_of_devices;"
                + "drop table if exists initial_marking_final;"
                + "CREATE TABLE initial_marking_final AS (SELECT
initial_marking.*, m0_marking.process_number_of_devices FROM
initial_marking INNER JOIN m0_marking ON initial_marking.primary_id =
m0_marking.primary_id);"
                + "UPDATE initial_marking_final SET
initial_marking_final.process_number_of_devices = "
                + "REPLACE(initial_marking_final.process_number_of_devices,
'0', '1') "
                + " where initial_marking_final.activity like 'pin%';" ;

            pst = con.prepareStatement(code1);
            boolean isResult1 = pst.execute();

            String query = "select * from initial_marking_final;";
            pst = con.prepareStatement(query);
            boolean isResult = pst.execute();
            do {
                rs = pst.getResultSet();
```

```

        ResultSetMetaData rsmd = rs.getMetaData();
        int columnsNumber = rsmd.getColumnCount();
        int col = rsmd.getColumnCount();
        for (int i = 1; i <= col; i++){
            String col_name = rsmd.getColumnName(i);
            System.out.print(col_name + " ");
        }
        System.out.println(" ");
        // Iterate through the data in the result set and display
it.
        while (rs.next()) {
            //Print one row
            for(int i = 1 ; i <= columnsNumber; i++){
                System.out.print(rs.getString(i) + " ");
            //Print one element of a row
            }
            System.out.println();
        }
        isResult = pst.getMoreResults();
    }
    while (isResult);
    } finally {
        if (rs != null) {
            rs.close();
        }
        if (pst != null) {
            pst.close();
        }
        if (con != null) {
            con.close();
        }
    }
}
}

```

Appendix D – Graphical Representation of PN Model

Part A – SQL Code for the Automated PN Generation

Appendix D, part A includes the SQL code created for the automated generation of the graphical representation of PNs, discussed in Chapter 3.

```
package step1_overall_visualisation_PN;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.sql.PreparedStatement;
public class Visualisation_PN_Overall {
    public static void main (String[] args) {
        Connection conn = null;
        PreparedStatement statement = null;
        ResultSet rs = null;
        try{

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/sql?verifySer
verCertificate=false&useSSL=true", "root", "Xristina23");
        //PreparedStatement statement
        =conn.prepareStatement("select Input,Output from input_output");
        statement =conn.prepareStatement("select * from
final_table");

        rs= statement.executeQuery();

        System.out.println( "strict digraph OverallPetriNet{ size=\"40\"
;node [margin=0 fontcolor=black fontsize=17 width=0.6 height=1.2 shape=box
color=blue];");
        while (rs.next())
            System.out.println(('') + rs.getString("name_primary") + ('')
+(";"));
        }
        catch(SQLException e){}
        try{
            if (rs != null)
                rs.close();
            if (statement != null)
                statement.close();
            if (conn != null)
                conn.close();
        }catch(Exception e){}

        try{

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/sql?verifySer
verCertificate=false&useSSL=true", "root", "Xristina23");

        statement =conn.prepareStatement("select *
from final_table");

        rs= statement.executeQuery();
```

```

        System.out.println( "node [margin=0
fontcolor=black fontsize=17 width=0.3 shape=circle color=blue];");

        while (rs.next())
            System.out.println(('') +
rs.getString("place_before_node") + ('') + (";" + ('') +
rs.getString("place_after_node")+ ('') + (";"));
        }
        catch(SQLException e){}
        try{
            if (rs != null)
                rs.close();
            if (statement != null)
                statement.close();
            if (conn != null)
                conn.close();
        }catch(Exception e){}

        try{

            conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/sql?ver
ifyServerCertificate=false&useSSL=true", "root", "Xristina23");
            statement
=conn.prepareStatement("select * from final_table");
            rs= statement.executeQuery();
            System.out.println( "edge [color=Blue,
style=normal] ");

            while (rs.next())
                System.out.println(('')
+rs.getString("name_primary") + ('') +(" -> ") + ('') +
rs.getString("place_after_node")+ ('') + (";"));
            }
            catch(SQLException e){}
            try{
                if (rs != null)
                    rs.close();
                if (statement != null)
                    statement.close();
                if (conn != null)
                    conn.close();
            }catch(Exception e){}

            try{

            conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/sql?verifySer
verCertificate=false&useSSL=true", "root", "Xristina23");
            statement
=conn.prepareStatement("select * from final_table");
            rs= statement.executeQuery();

            while (rs.next())
                System.out.println(('') +
rs.getString("place_before_node") + ('') + (" -> ") + ('') +
rs.getString("name_primary")+ ('') + (";"));
            }

            catch(SQLException e){}
            try{

```

```

        if (rs != null)
            rs.close();
        if (statement != null)
            statement.close();
        if (conn != null)
            conn.close();
    }catch(Exception e){}
    System.out.println(
"overlap=false label=\"Automatic Layout of the Overall Petri Net Model for
the handler_case\" fontsize=13; } ");

    }
}

```

Part B – DOT File for the PN Model Generation (GraphViz Input)

Appendix D, part B covers the DOT file obtained from the execution of the SQL code shown in Appendix D, part A, for the recycling IT asset process, discussed in Chapter 4.

```

strict digraph OverallPetriNet{ size="20"; node [margin=0 fontcolor=black
fontsize=27 width=0.6 height=1.2 shape=box color=blue];
"Visual_Inspection"; "Visual_Inspection"; "D_VI"; "D_VI"; "D_FT"; "D_FT";
"D_DE"; "D_DE"; "D_R"; "D_R"; "Strip_Scrap"; "Strip_Scrap"; "Strip_Scrap"; "M";
"M"; "Asset_Track"; "Functional_Test"; "Data_Erasure"; "Repair";
"Cleaning_De_Labelling";
node [margin=0 fontcolor=black fontsize=27 width=0.3 shape=circle color=blue];
"ATp";"place_1"; "Rp";"place_1"; "place_1";"VIp"; "place_1";"VIf";
"place_2";"FTp"; "place_2";"FTf"; "place_3";"DEp"; "place_3";"DEf";
"place_4";"Rf"; "place_4";"Rp"; "VIf"; "SSp"; "DEf"; "SSp"; "Rf"; "SSp"; "SSp";
"pout"; "CDp"; "pout"; "pin"; "ATp"; "VIp";"place_2"; "FTp";"place_3";
"FTf";"place_4"; "DEp"; "CDp";
edge [color=Blue, style=normal]
"ATp" -> "Visual_Inspection"; "Rp" -> "Visual_Inspection"; "place_1" ->
"D_VI";"place_1" -> "D_VI"; "place_2" -> "D_FT";"place_2" -> "D_FT"; "place_3" -
> "D_DE";"place_3" -> "D_DE"; "place_4" -> "D_R";"place_4" -> "D_R"; "VIf" ->
"Strip_Scrap"; "DEf" -> "Strip_Scrap"; "Rf" -> "Strip_Scrap"; "SSp" -> "M"; "CDp" -
> "M"; "pin" -> "Asset_Track"; "VIp" -> "Functional_Test"; "FTp" ->
"Data_Erasure"; "FTf" -> "Repair"; "DEp" -> "Cleaning_De_Labelling";
"Visual_Inspection" -> "place_1"; "Visual_Inspection" -> "place_1"; "D_VI" ->
"VIp"; "D_VI" -> "VIf"; "D_FT" -> "FTp"; "D_FT" -> "FTf"; "D_DE" -> "DEp";
"D_DE" -> "DEf"; "D_R" -> "Rf"; "D_R" -> "Rp"; "Strip_Scrap" -> "SSp";
"Strip_Scrap" -> "SSp"; "Strip_Scrap" -> "SSp"; "M" -> "pout"; "M" -> "pout";
"Asset_Track" -> "ATp"; "Functional_Test" -> "place_2"; "Data_Erasure" ->
"place_3"; "Repair" -> "place_4"; "Cleaning_De_Labelling" -> "CDp"; }

```


Appendix E – Recycling IT Asset Process Example (XMI File)

Appendix E includes the XMI file for the recycling IT asset process shown in Chapter 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:uml="http://www.eclipse.org/UML/2/5.0.0/UML"
xmi:id="_BUZs8LJTEeaTirLhAX5dxQ" name="RootElement">
  <packagedElement xmi:type="uml:Activity" xmi:id="_BxHeILJTEeaTirLhAX5dxQ"
name="Activity1" node="_Ftmh0LJTEeaTirLhAX5dxQ _InKv8LJTEeaTirLhAX5dxQ
_MAZVkJTEeaTirLhAX5dxQ _M-adALJTEeaTirLhAX5dxQ _NahxQLJTEeaTirLhAX5dxQ
_OIQZ8LJTEeaTirLhAX5dxQ _PsLTcLJTEeaTirLhAX5dxQ _QDcUQLJTEeaTirLhAX5dxQ
_ex_zYLJTEeaTirLhAX5dxQ _iwxWELJTEeaTirLhAX5dxQ _jxsAgLJTEeaTirLhAX5dxQ
_ksMg8LJTEeaTirLhAX5dxQ _ltzH4LJTEeaTirLhAX5dxQ _9vls8LJTEeaTirLhAX5dxQ">
    <edge xmi:type="uml:ControlFlow" xmi:id="_pWiwMLJTEeaTirLhAX5dxQ"
target="_InKv8LJTEeaTirLhAX5dxQ" source="_Ftmh0LJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_0ZeSILJTEeaTirLhAX5dxQ"
name="AtP" target="_MAZVkJTEeaTirLhAX5dxQ"
source="_InKv8LJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_Ao6DcLJUEeaTirLhAX5dxQ"
name="place_1" target="_iwxWELJTEeaTirLhAX5dxQ"
source="_MAZVkJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_FrRKELJUEeaTirLhAX5dxQ"
name="Vip" target="_M-adALJTEeaTirLhAX5dxQ"
source="_iwxWELJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_JXp8QLJUEeaTirLhAX5dxQ"
name="Vif" target="_OIQZ8LJTEeaTirLhAX5dxQ"
source="_iwxWELJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_NbtAgLJUEeaTirLhAX5dxQ"
name="place_2" target="_jxsAgLJTEeaTirLhAX5dxQ" source="_M-
adALJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_N9kXcLJUEeaTirLhAX5dxQ"
name="Ftp" target="_NahxQLJTEeaTirLhAX5dxQ"
source="_jxsAgLJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_QEscALJUEeaTirLhAX5dxQ"
name="Ftf" target="_PsLTcLJTEeaTirLhAX5dxQ"
source="_jxsAgLJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_XK0xMLJUEeaTirLhAX5dxQ"
name="place_3" target="_ksMg8LJTEeaTirLhAX5dxQ"
source="_NahxQLJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_Ysk30LJUEeaTirLhAX5dxQ"
name="DEp" target="_QDcUQLJTEeaTirLhAX5dxQ"
source="_ksMg8LJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_d8He0LJUEeaTirLhAX5dxQ"
name="DEf" target="_OIQZ8LJTEeaTirLhAX5dxQ"
source="_ksMg8LJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_h5c3ELJUEeaTirLhAX5dxQ"
name="place_4" target="_ltzH4LJTEeaTirLhAX5dxQ"
source="_PsLTcLJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_j0jo0LJUEeaTirLhAX5dxQ"
name="Rf" target="_OIQZ8LJTEeaTirLhAX5dxQ"
source="_ltzH4LJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_LLpAkLJUEeaTirLhAX5dxQ"
name="Rp" target="_MAZVkJTEeaTirLhAX5dxQ"
source="_ltzH4LJTEeaTirLhAX5dxQ"/>
  </packagedElement>
</uml:Model>
```

```

    <edge xmi:type="uml:ControlFlow" xmi:id="_pJgkULJUEeaTirLhAX5dxQ"
name="SSp" target="_9vls8LJTEeaTirLhAX5dxQ"
source="_OIQZ8LJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_vA0V8LJUEeaTirLhAX5dxQ"
name="CDp" target="_9vls8LJTEeaTirLhAX5dxQ"
source="_QDcUQLJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_1tIc8LJUEeaTirLhAX5dxQ"
target="_ex_zYLJTEeaTirLhAX5dxQ" source="_9vls8LJTEeaTirLhAX5dxQ"/>
    <edge xmi:type="uml:ObjectFlow" xmi:id="_XOBqQLJtEeevDqMum9V7CA"
target="_iwxWELJTEeaTirLhAX5dxQ" source="_MAZVkJTEeaTirLhAX5dxQ">
    <guard xmi:type="uml:LiteralBoolean" xmi:id="_XP7HsLJtEeevDqMum9V7CA"
value="true"/>
    <weight xmi:type="uml:LiteralInteger"
xmi:id="_XP7uwLJtEeevDqMum9V7CA" value="1"/>
</edge>
    <node xmi:type="uml:InitialNode" xmi:id="_Ftmh0LJTEeaTirLhAX5dxQ"
name="pin" outgoing="_pWiwMLJTEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_InKv8LJTEeaTirLhAX5dxQ"
name="Asset_Track" incoming="_pWiwMLJTEeaTirLhAX5dxQ"
outgoing="_0ZeSILJTEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_MAZVkJTEeaTirLhAX5dxQ"
name="Visual_Inspection" incoming="_0ZeSILJTEeaTirLhAX5dxQ
_LlpAkLJUEeaTirLhAX5dxQ" outgoing="_Ao6DcLJUEeaTirLhAX5dxQ
_XOBqQLJtEeevDqMum9V7CA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_M-adALJTEeaTirLhAX5dxQ"
name="Functional_Test" incoming="_FrRKELJUEeaTirLhAX5dxQ"
outgoing="_NbtAgLJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_NahxQLJTEeaTirLhAX5dxQ"
name="Data-Erasure" incoming="_N9kXcLJUEeaTirLhAX5dxQ"
outgoing="_XK0xMLJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_OIQZ8LJTEeaTirLhAX5dxQ"
name="Strip_Scrap" incoming="_JXp8QLJUEeaTirLhAX5dxQ
_d8He0LJUEeaTirLhAX5dxQ _j0jo0LJUEeaTirLhAX5dxQ"
outgoing="_pJgkULJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_PsLTcLJTEeaTirLhAX5dxQ"
name="Repair" incoming="_QEscALJUEeaTirLhAX5dxQ"
outgoing="_h5c3ELJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_QDcUQLJTEeaTirLhAX5dxQ"
name="Cleaning_De_Labelling" incoming="_Ysk30LJUEeaTirLhAX5dxQ"
outgoing="_vA0V8LJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:ActivityFinalNode" xmi:id="_ex_zYLJTEeaTirLhAX5dxQ"
name="pout" incoming="_1tIc8LJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:DecisionNode" xmi:id="_iwxWELJTEeaTirLhAX5dxQ"
name="D_VI" incoming="_Ao6DcLJUEeaTirLhAX5dxQ _XOBqQLJtEeevDqMum9V7CA"
outgoing="_FrRKELJUEeaTirLhAX5dxQ _JXp8QLJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:DecisionNode" xmi:id="_jxsAgLJTEeaTirLhAX5dxQ"
name="D_FT" incoming="_NbtAgLJUEeaTirLhAX5dxQ"
outgoing="_N9kXcLJUEeaTirLhAX5dxQ _QEscALJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:DecisionNode" xmi:id="_ksMg8LJTEeaTirLhAX5dxQ"
name="D_DE" incoming="_XK0xMLJUEeaTirLhAX5dxQ"
outgoing="_Ysk30LJUEeaTirLhAX5dxQ _d8He0LJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:DecisionNode" xmi:id="_ltzH4LJTEeaTirLhAX5dxQ"
name="D_R" incoming="_h5c3ELJUEeaTirLhAX5dxQ"
outgoing="_j0jo0LJUEeaTirLhAX5dxQ _LlpAkLJUEeaTirLhAX5dxQ"/>
    <node xmi:type="uml:MergeNode" xmi:id="_9vls8LJTEeaTirLhAX5dxQ"
name="M" incoming="_pJgkULJUEeaTirLhAX5dxQ _vA0V8LJUEeaTirLhAX5dxQ"
outgoing="_1tIc8LJUEeaTirLhAX5dxQ"/>
</packagedElement>
</uml:Model>

```

Appendix F

Part A – Validation – PN Visual Check (Token Game)

This section covers the MATLAB code for the visual check of the PN model generated for recycling IT asset process, as discussed in Chapter 5.

```
n=1000;
x=rand(n,1);
% path1 - [AT VI (pass) FT (pass) DE (pass) CD M]

if x(1,:) <= prob(1,1) % Asset_Track (pass)
    if x(1,:) <= prob(3,1) % D_VI (pass)
        if x(1,:) <= prob(5,1) % D_FT (pass)
            if x(1,:) <= prob(9,1) % D_DE (pass)
                if x(1,:) <= prob(11,1) % Cleaning_De_Labelling
                    (pass)
                        if x(1,:) <= prob(12,1) % M (pass)
                            Mn=M0+(M_AT_CD*T_AT);
                            Mnew=Mn;
                            Mn=Mnew+(M_AT_CD*T_VI);
                            Mnew=Mn;
                            Mn=Mnew+(M_AT_CD*T_D_VI);
                            Mnew=Mn;
                            Mn=Mnew+(M_AT_CD*T_FT);
                            Mnew=Mn;
                            Mn=Mnew+(M_AT_CD*T_D_FT);
                            Mnew=Mn;
                            Mn=Mnew+(M_AT_CD*T_DE);
                            Mnew=Mn;
                            Mn=Mnew+(M_AT_CD*T_D_DE);
                            Mnew=Mn;
                            Mn=Mnew+(M_AT_CD*T_CD);
                            Mnew=Mn;
                            Mn=Mnew+(M_AT_CD*T_M);
                            Mnew=Mn
                        end
                    end
                end
            end
        end
    end
end

% path2 - [AT VI (pass) FT (pass) DE (fail) SS M]
if x(1,:) <= prob(1,1) % Asset_Track (pass)
    if x(1,:) <= prob(3,1) % D_VI (pass)
        if x(1,:) <= prob(5,1) % D_FT (pass)
            if x(1,:) > prob(9,1) % D_DE (fail)
                if x(1,:) <= prob(10,1) % Strip_Scrap (pass)
                    if x(1,:) <= prob(12,1) % M (pass)
                        Mn=M0+(M_AT_SS*T_AT);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_SS*T_VI);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_SS*T_D_VI);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_SS*T_FT);
```

```

Mnew=Mn;

Mn=Mnew+(M_AT_SS*T_D_FT);
Mnew=Mn;
Mn=Mnew+(M_AT_SS*T_DE);
Mnew=Mn;
Mn=Mnew+(M_AT_SS*T_D_DE);
Mnew=Mn;
Mn=Mnew+(M_AT_SS*T_SS);
Mnew=Mn;
Mn=Mnew+(M_AT_SS*T_M);
Mnew=Mn;
end
end
end
end
end
end

% path3 - [AT VI(fail) SS M]
if x(1,:) <= prob(1,1) % Asset_Track (pass)
    if x(1,:) > prob(3,1) % D_VI (fail)
        if x(1,:) <= prob(10,1) % Strip_Scrap (pass)
            if x(1,:) <= prob(12,1) % M (pass)
                Mn=M0+(M_AT_VIf_SS*T_AT);
                Mnew=Mn;
                Mn=Mnew+(M_AT_VIf_SS*T_VI);
                Mnew=Mn;
                Mn=Mnew+(M_AT_VIf_SS*T_D_VI);
                Mnew=Mn;
                Mn=Mnew+(M_AT_VIf_SS*T_SS);
                Mnew=Mn;
                Mn=Mnew+(M_AT_VIf_SS*T_M);
                Mnew=Mn;
            end
        end
    end
end

% path4 - [AT VI(pass) FT(fail) R(fail) SS M]
if x(1,:) <= prob(1,1) % Asset_Track (pass)
    if x(1,:) <= prob(3,1) % D_VI (pass)
        if x(1,:) > prob(5,1) % D_FT (fail)
            if x(1,:) > prob(7,1) % D_R (fail)
                if x(1,:) <= prob(10,1) % Strip_Scrap (pass)
                    if x(1,:) <= prob(12,1) % M (pass)
                        Mn=M0+(M_AT_Rf_SS*T_AT);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_Rf_SS*T_VI);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_Rf_SS*T_D_VI);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_Rf_SS*T_FT);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_Rf_SS*T_D_FT);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_Rf_SS*T_R);
                        Mnew=Mn;
                        Mn=Mnew+(M_AT_Rf_SS*T_D_R);
                    end
                end
            end
        end
    end
end

```

```

Mnew=Mn;

Mn=Mnew+(M_AT_Rf_SS*T_SS);
Mnew=Mn;
Mn=Mnew+(M_AT_Rf_SS*T_M);
Mnew=Mn;
end
end
end
end
end

% path5 - [AT VI(pass) FT(fail) R(pass) VI(pass) FT(pass) DE(pass)
CD M]
if x(1,:) <= prob(1,1) % Asset_Track (pass)
    if x(1,:) <= prob(3,1) % D_VI (pass)
        if x(1,:) > prob(5,1) % D_FT (fail)
            if x(1,:) <= prob(7,1) % D_R (pass)
                Mn=M0+(M_AT_Rp_CD*T_AT);
                Mnew=Mn;
                Mn=Mnew+(M_AT_Rp_CD*T_VI);
                Mnew=Mn;
                Mn=Mnew+(M_AT_Rp_CD*T_D_VI);
                Mnew=Mn;
                Mn=Mnew+(M_AT_Rp_CD*T_FT);
                Mnew=Mn;
                Mn=Mnew+(M_AT_Rp_CD*T_D_FT);
                Mnew=Mn;
                Mn=Mnew+(M_AT_Rp_CD*T_R);
                Mnew=Mn;
                Mn=Mnew+(M_AT_Rp_CD*T_D_R);
                Mnew=Mn;
            if x(1,:) <= prob(3,1) % D_VI (pass)
                if x(1,:) <= prob(5,1) % D_FT (pass)
                    if x(1,:) <= prob(9,1) % D_DE (pass)
                        if x(1,:) <= prob(11,1) %
Cleaning_De_Labelling (pass)
                                if x(1,:) <= prob(12,1) % M
                                    Mn=Mnew+(M_AT_Rp_CD2*T_VI);
                                    Mnew=Mn;
                                    Mn=Mnew+(M_AT_Rp_CD2*T_D_VI);
                                    Mnew=Mn;
                                    Mn=Mnew+(M_AT_Rp_CD2*T_FT);
                                    Mnew=Mn;

Mn=Mnew+(M_AT_Rp_CD2*T_D_FT);

                                    Mnew=Mn;
                                    Mn=Mnew+(M_AT_Rp_CD2*T_DE);
                                    Mnew=Mn;

Mn=Mnew+(M_AT_Rp_CD2*T_D_DE);

                                    Mnew=Mn;
                                    Mn=Mnew+(M_AT_Rp_CD2*T_CD);
                                    Mnew=Mn;
                                    Mn=Mnew+(M_AT_Rp_CD2*T_M);
                                    Mnew=Mn;
                                end
                            end
end

```


Part B – Validation – PN Model Numerical Simulation

This section covers the MATLAB code for the numerical simulation of the PN model generated for recycling IT asset process, as discussed in Chapter 5.

```
n=1000;
x=rand(n,1); % generate random n numbers
y=rand(n,1);
z=rand(n,1);
a=rand(n,1);
b=rand(n,1);
% real_activity_time %
% create Asset Track time based on MATLAB table data
activity_difference=max_time-min_time;

c_activity_dif=double(activity_difference);
c_AT=x*(c_activity_dif(1,1));

c_activity=double(min_time);
t_Asset_Track=c_AT+(c_activity(1,1));
% estimate the average time for the Asset Track activity
avg_t_AT= mean(t_Asset_Track);

% create Visual Inspection time based on MATLAB table data
c_VI=x*(c_activity_dif(2,1));
t_Visual_Inspection=c_VI+(c_activity(2,1));
% estimate the average time for the Visual_Inspection activity
avg_t_VI= mean(t_Visual_Inspection);

% create Functional Test time based on MATLAB table data
c_FT=x*(c_activity_dif(4,1));
t_Functional_Test=c_FT + (c_activity(4,1));
% estimate the average time for the FT activity
avg_t_FT= mean(t_Functional_Test);

% create Data Erasure time based on MATLAB table data
c_DE=x*(c_activity_dif(8,1));
t_Data_Erasure =c_DE+(c_activity(8,1));
% estimate the average time for the DE activity
avg_t_DE= mean(t_Data_Erasure);

% create Repair time based on MATLAB table data
c_R=x*(c_activity_dif(6,1));
t_Repair=c_R+(c_activity(6,1));
% estimate the average time for the R activity
avg_t_R= mean(t_Repair);

% create Cleaning Delabelling time based on MATLAB table data
c_CD=x*(c_activity_dif(11,1));
t_Cleaning_Delabelling=c_CD+(c_activity(11,1));
% estimate the average time for the CD activity
avg_t_CD= mean(t_Cleaning_Delabelling);

% create Strip & Scrap time based on MATLAB table data
c_SS=x*(c_activity_dif(10,1));
t_Strip_Scrap=c_SS+(c_activity(10,1));
% estimate the average time for the SS activity
```

```

avg_t_SS= mean(t_Strip_Scrap);

% interval_activity_time %
interval_difference_pass=max_interval_pass-min_interval_pass;
interval_difference_fail=max_interval_fail-min_interval_fail;
c_interval_dif_pass=double(interval_difference_pass);
c_interval_dif_fail=double(interval_difference_fail);

c_interval_pass=double(min_interval_pass);
c_interval_fail=double(min_interval_fail);

% create Asset Track time based on MATLAB table data
c_interval_AT=x*(c_interval_dif_pass(1,1));
t_interval_Asset_Track=c_interval_AT+(c_interval_pass(1,1));
% estimate the average time for the AT interval
avg_int_AT= mean(t_interval_Asset_Track);

% create D_VI pass time based on MATLAB table data
c_interval_VI_pass=x*(c_interval_dif_pass(3,1));
t_interval_Visual_Inspection_pass=c_interval_VI_pass+(c_interval_pass
(3,1));
% estimate the average time for the D_VI interval
avg_int_VI_pass= mean(t_interval_Visual_Inspection_pass);

% create D_VI fail time based on MATLAB table data
c_interval_VI_fail=x*(c_interval_dif_fail(3,1));
t_interval_Visual_Inspection_fail=c_interval_VI_fail+(c_interval_fail
(3,1));
% estimate the average time for the D_VI interval
avg_int_VI_fail= mean(t_interval_Visual_Inspection_fail);

% create D_FT pass time based on MATLAB table data
c_interval_FT_pass=x*(c_interval_dif_pass(5,1));
t_interval_Functional_Test_pass=c_interval_FT_pass +
(c_interval_pass(5,1));
% estimate the average time for the D_FT interval
avg_int_FT_pass= mean(t_interval_Functional_Test_pass);

% create D_FT fail time based on MATLAB table data
c_interval_FT_fail=x*(c_interval_dif_fail(5,1));
t_interval_Functional_Test_fail=c_interval_FT_fail +
(c_interval_fail(5,1));
% estimate the average time for the D_FT interval
avg_int_FT_fail= mean(t_interval_Functional_Test_fail);

% create D_DE pass and fail time based on MATLAB table data
c_interval_DE_pass=x*(c_interval_dif_pass(9,1));
t_interval_Data_Erasure =c_interval_DE_pass+(c_interval_pass(9,1));
% estimate the average time for the D_DE interval
avg_int_DE= mean(t_interval_Data_Erasure);

% create D_R time based on MATLAB table data
c_interval_R_pass=x*(c_interval_dif_pass(7,1));
t_interval_Repair=c_interval_R_pass+(c_interval_pass(7,1));
% estimate the average time for the D_R interval
avg_int_R= mean(t_interval_Repair);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pass_Probability=double(Pass_Probability);

```



```

% retrieve pass probabilities from excel file
if x(1,:) <= Pass_Probability(3,1) % VI pass
    Total_Asset_Track1 = avg_t_AT + avg_int_AT;
    Total_Visual_Inspection_p1 = avg_t_VI + avg_int_VI_pass;
    path1_VI = n * Pass_Probability(3,1);

    if y(1,:) <= Pass_Probability(5,1) % D_FT pass
        Total_Functional_Test_p1 = avg_t_FT + avg_int_FT_pass;
        path_FT_p = path1_VI * Pass_Probability(5,1);

        if z(1,:) <= Pass_Probability(9,1) % D_DE pass
            Total_Data_Erasure1 = avg_t_DE + avg_int_DE;
            avg_t_CD1 = mean(t_Cleaning_Delabelling);
            path1_DEp = path_FT_p * Pass_Probability(9,1);

            elseif z(1,:) > Pass_Probability(9,1) % D_DE fail
                Total_Data_Erasure_f2 = avg_t_DE + avg_int_DE;
                avg_t_SS2 = mean(t_Strip_Scrap);
                path2_DEf = path_FT_p * (1 - Pass_Probability(9,1));
            end

        elseif y(1,:) > Pass_Probability(5,1) % D_FT fail
            Total_Functional_Test_f3 = avg_t_FT + avg_int_FT_fail;
            path4_FT = path1_VI * (1 - Pass_Probability(5,1));

            if a(1,:) <= Pass_Probability(7,1) % D_R pass
                if b(1,:) <= Pass_Probability(9,1) % D_DE pass
                    Total_Repair3 = avg_t_R + avg_int_R;
                    Total_Visual_Inspection_p3 = avg_t_VI + avg_int_VI_pass;
                    Total_Functional_Test_p3 = avg_t_FT + avg_int_FT_pass;
                    Total_Data_Erasure3 = avg_t_DE + avg_int_DE;
                    avg_t_CD3 = mean(t_Cleaning_Delabelling);

                    path4_R = path4_FT * (Pass_Probability(7,1));
                    path4_VI = path4_R * (Pass_Probability(3,1));
                    path4_FTp = path4_VI * (Pass_Probability(5,1));
                    path4_DE = path4_FTp * (Pass_Probability(9,1));

                    elseif b(1,:) > Pass_Probability(9,1) % D_DE fail
                        Total_Repair4 = avg_t_R + avg_int_R;
                        Total_Visual_Inspection_p4 = avg_t_VI + avg_int_VI_pass;
                        Total_Functional_Test_p4 = avg_t_FT + avg_int_FT_pass;
                        Total_Data_Erasure4 = avg_t_DE + avg_int_DE;
                        avg_t_SS4 = mean(t_Strip_Scrap);
                        path5_DE = path4_FTp * (1 - Pass_Probability(9,1));
                    end

                    elseif a(1,:) > Pass_Probability(7,1) % D_R fail
                        Total_Repair5 = avg_t_R + avg_int_R;
                        avg_t_SS5 = mean(t_Strip_Scrap);
                        path5_R = path4_FT * (1 - Pass_Probability(7,1));
                    end
                end

            end

        elseif x(1,:) > Pass_Probability(3,1) % D_VI fail
            Total_Visual_Inspection_f6 = avg_t_VI + avg_int_VI_fail;
            avg_t_SS6 = mean(t_Strip_Scrap);
            path6_VI = n * (1 - Pass_Probability(3,1));
        end
end

```

Part C – Validation – PN Model Performance Analysis

This section presents the MATLAB code for the performance analysis of the PN model generated for recycling IT asset process, as discussed in Chapter 5. From this code, results for the average time of each transition and the number of visits to PN places can be obtained.

```
% time needed for each path to be completed
% path1 - [AT VI(pass) FT(pass) DE(pass) CD M]
t_path1=(Total_Asset_Track1+Total_Visual_Inspection_p1+Total_Functional_Test_p1+Total_Data_Erasure1+avg_t_CD1)
% % path2 - [AT VI(pass) FT(pass) DE(fail) SS M]

t_path2=(Total_Asset_Track1+Total_Visual_Inspection_p1+Total_Functional_Test_p1+Total_Data_Erasure_f2+avg_t_SS2)
% path3 - [AT VI(pass) FT(fail) R(pass) VI(pass) FT(pass) DE(pass) CD M]
t_path3=(Total_Asset_Track1+Total_Visual_Inspection_p1+Total_Functional_Test_f3+Total_Repair3+Total_Visual_Inspection_p3+Total_Functional_Test_p3+Total_Data_Erasure3+avg_t_CD3)
% path4 - [AT VI(pass) FT(fail) R(pass) VI(pass) FT(pass) DE(fail) SS M]
t_path4=(Total_Asset_Track1+Total_Visual_Inspection_p1+Total_Functional_Test_f3+Total_Repair3+Total_Visual_Inspection_p3+Total_Functional_Test_p3+Total_Data_Erasure3+avg_t_SS4)
% path5 - [AT VI(pass) FT(fail) R(fail) SS M]
t_path5=(Total_Asset_Track1+Total_Visual_Inspection_p1+Total_Functional_Test_f3+Total_Repair5+avg_t_SS5)
% path6 - [AT VI(fail SS M)]
t_path6=(Total_Asset_Track1+Total_Visual_Inspection_f6+avg_t_SS6)

% number of visits in PN places based on paths identified
% path1 - [AT VI(pass) FT(pass) DE(pass) CD M]
device_path1=floor (path1_DEp)
% path2 - [AT VI(pass) FT(pass) DE(fail) SS M]
device_path2=floor (path2_DEf)
% path3 - [AT VI(pass) FT(fail) R(pass) VI(pass) FT(pass) DE(pass) CD M]
device_path3=floor (path4_DE)
% path4 - [AT VI(pass) FT(fail) R(pass) VI(pass) FT(pass) DE(fail) SS M]
device_path4=floor (path5_DE)
% path5 - [AT VI(pass) FT(fail) R(fail) SS M]
device_path5=floor (path5_R)
% path6 - [AT VI(fail SS M)]
device_path6=floor (path6_VI)
```

Appendix G

Part A – AD Examples in Chapter 6 (XMI Files)

Appendix G, part A covers the XMI files obtained from the three AD examples discussed in Chapter 6, shown in Figures 6.2, 6.4 and 6.6 respectively.

XMI file obtained from the first AD example in Figure 6.2

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
xmi:id="_dPhH0F2YEeeC05B8er0jow" name="RootElement">
  <packagedElement xmi:type="uml:Activity" xmi:id="_dPqRwF2YEeeC05B8er0jow"
name="1st_example" node="_gC3ZMF2YEeeC05B8er0jow
_jM0MsF2YEeeC05B8er0jow _rC9pkF2YEeeC05B8er0jow"
group="_jM0MsF2YEeeC05B8er0jow">
    <edge xmi:type="uml:ControlFlow" xmi:id="_M1-z8F2gEeeC05B8er0jow"
target="_KB7X8F2gEeeC05B8er0jow" source="_Hw02YF2gEeeC05B8er0jow"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="___VcMF2gEeeC05B8er0jow"
target="_gbxocF2YEeeC05B8er0jow" source="_pH_KoF2YEeeC05B8er0jow"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_fU6K4OBZEeeIaM4coD475g"
target="_qNgoQF2YEeeC05B8er0jow" source="_oW66EF2YEeeC05B8er0jow"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_1z6PcODFEeemJ8jkM3-fqg"
target="_pH_KoF2YEeeC05B8er0jow" source="_gC3ZMF2YEeeC05B8er0jow"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_2mgxEODFEeemJ8jkM3-fqg"
target="_rC9pkF2YEeeC05B8er0jow" source="_qNgoQF2YEeeC05B8er0jow"/>
    <structuredNode xmi:type="uml:ExpansionRegion"
xmi:id="_jM0MsF2YEeeC05B8er0jow" name="ExpansionRegion2" mode="parallel"
outputElement="_qNgoQF2YEeeC05B8er0jow"
inputElement="_pH_KoF2YEeeC05B8er0jow">
        <node xmi:type="uml:OpaqueAction" xmi:id="_gbxocF2YEeeC05B8er0jow"
name="Write_Article" incoming="___VcMF2gEeeC05B8er0jow">
```

```

    <outputValue xmi:type="uml:OutputPin"
xmi:id="_HWO2YF2gEeeC05B8er0jow" outgoing="_M1-z8F2gEeeC05B8er0jow"
isControlType="true">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_HWpdCF2gEeeC05B8er0jow" value="1"/>
    </outputValue>
</node>
<node xmi:type="uml:OpaqueAction" xmi:id="_oW66EF2YEeeC05B8er0jow"
name="Review_Article" outgoing="_fU6K4OBZEeeIaM4coD475g">
    <inputValue xmi:type="uml:InputPin" xmi:id="_KB7X8F2gEeeC05B8er0jow"
name="" incoming="_M1-z8F2gEeeC05B8er0jow" isControlType="true">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_KB7_AF2gEeeC05B8er0jow" value="1"/>
    </inputValue>
</node>
<node xmi:type="uml:ExpansionNode" xmi:id="_pH_KoF2YEeeC05B8er0jow"
name="List_of_Topics" incoming="_1z6PcODFEeeMj8jkM3-fqg"
outgoing="___VcMF2gEeeC05B8er0jow" isControlType="true"
regionAsInput="_jM0MsF2YEeeC05B8er0jow">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_pH_xsF2YEeeC05B8er0jow" value="1"/>
    </node>
<node xmi:type="uml:ExpansionNode" xmi:id="_qNgoQF2YEeeC05B8er0jow"
name="List_Box_Pin" incoming="_fU6K4OBZEeeIaM4coD475g"
outgoing="_2mgxEODFEeeMj8jkM3-fqg" type="_dPqRwF2YEeeC05B8er0jow"
isControlType="true" regionAsOutput="_jM0MsF2YEeeC05B8er0jow">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_qNhPUF2YEeeC05B8er0jow" value="1"/>
    </node>
</structuredNode>
<node xmi:type="uml:OpaqueAction" xmi:id="_gC3ZMF2YEeeC05B8er0jow"
name="Choose_Topics" outgoing="_1z6PcODFEeeMj8jkM3-fqg"/>

```

```

    <node xmi:type="uml:OpaqueAction" xmi:id="_rC9pkF2YEeC05B8er0jow"
name="Publish_Newsletter" incoming="_2mgxEODFEeemJ8jkM3-fqg"/>
  </packagedElement>
</uml:Model>

```

XMI file obtained from the second AD example in Figure 6.4

```

<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
xmi:id="_MJToQF2tEeC05B8er0jow" name="RootElement">
  <packagedElement xmi:type="uml:Activity" xmi:id="_MJVdcF2tEeC05B8er0jow"
name="2nd_example" node="_NBPV0F2tEeC05B8er0jow
_NVix4F2tEeC05B8er0jow _Si6lsF2tEeC05B8er0jow
_XXc7sF2tEeC05B8er0jow">
    <edge xmi:type="uml:ControlFlow" xmi:id="_mSSjUF2tEeC05B8er0jow"
target="_Si6lsF2tEeC05B8er0jow" source="_XXc7sF2tEeC05B8er0jow"/>
    <edge xmi:type="uml:ObjectFlow" xmi:id="_4BTiwF2tEeC05B8er0jow"
target="_c2xrIF2tEeC05B8er0jow" source="_r6HugF2tEeC05B8er0jow">
        <guard xmi:type="uml:LiteralBoolean" xmi:id="_4Beh4F2tEeC05B8er0jow"
value="true"/>
        <weight xmi:type="uml:LiteralInteger" xmi:id="_4Beh4V2tEeC05B8er0jow"
value="1"/>
    </edge>
    <edge xmi:type="uml:ObjectFlow" xmi:id="_5CXXIF2tEeC05B8er0jow"
target="_xJt_4F2tEeC05B8er0jow" source="_uaxtYF2tEeC05B8er0jow">
        <guard xmi:type="uml:LiteralBoolean" xmi:id="_5CpD8F2tEeC05B8er0jow"
value="true"/>
        <weight xmi:type="uml:LiteralInteger" xmi:id="_5CpD8V2tEeC05B8er0jow"
value="1"/>
    </edge>
    <node xmi:type="uml:OpaqueAction" xmi:id="_NBPV0F2tEeC05B8er0jow"
name="Create_Order">

```

```

        <outputValue xmi:type="uml:OutputPin" xmi:id="_r6HugF2tEeeC05B8er0jow"
name="" outgoing="_4BTiwF2tEeeC05B8er0jow"/>
    </node>
    <node xmi:type="uml:OpaqueAction" xmi:id="_NVix4F2tEeeC05B8er0jow"
name="Close_Order">
        <inputValue xmi:type="uml:InputPin" xmi:id="_xJt_4F2tEeeC05B8er0jow"
name="" incoming="_5CXXIF2tEeeC05B8er0jow">
            <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_xJum8F2tEeeC05B8er0jow" value="1"/>
        </inputValue>
    </node>
    <node xmi:type="uml:AcceptEventAction" xmi:id="_Si6IsF2tEeeC05B8er0jow"
name="Receive_Payment" incoming="_mSSjUF2tEeeC05B8er0jow">
        <result xmi:type="uml:OutputPin" xmi:id="_uaxtYF2tEeeC05B8er0jow"
name="" outgoing="_5CXXIF2tEeeC05B8er0jow">
            <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_uayUcF2tEeeC05B8er0jow" value="1"/>
        </result>
        <trigger xmi:type="uml:Trigger" xmi:id="_k_Y7kF2uEeeC05B8er0jow"/>
    </node>
    <node xmi:type="uml:SendSignalAction" xmi:id="_XXc7sF2tEeeC05B8er0jow"
name="Send_Invoice" outgoing="_mSSjUF2tEeeC05B8er0jow"
signal="_auo58F2tEeeC05B8er0jow">
        <target xmi:type="uml:InputPin" xmi:id="_c2xrIF2tEeeC05B8er0jow"
incoming="_4BTiwF2tEeeC05B8er0jow" type="_auo58F2tEeeC05B8er0jow">
            <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_c2xrIV2tEeeC05B8er0jow" value="1"/>
            <upperValue xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_c2xrIl2tEeeC05B8er0jow" value="1"/>
        </target>
    </node>
</packagedElement>

```

```

    <packagedElement xmi:type="uml:Signal" xmi:id="_auo58F2tEeeC05B8er0jow"
name="Verify_CC_Funds"/>
</uml:Model>

```

XMI file obtained from the third AD example in Figure 6.6

```

<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
xmi:id="_MV_aYF2iEeeC05B8er0jow" name="RootElement">
    <packagedElement xmi:type="uml:Activity"
xmi:id="_MWBPkF2iEeeC05B8er0jow" name="3rd_example"
node="_p4tRYF2iEeeC05B8er0jow_sw3xUF2iEeeC05B8er0jow">
        <node xmi:type="uml:OpaqueAction" xmi:id="_p4tRYF2iEeeC05B8er0jow"
name="User_Cancels">
            <handler xmi:type="uml:ExceptionHandler"
xmi:id="_wcswkF2nEeeC05B8er0jow"
exceptionInput="_uQxEgF2nEeeC05B8er0jow"
exceptionType="_MWBPkF2iEeeC05B8er0jow"
handlerBody="_sw3xUF2iEeeC05B8er0jow"/>
        </node>
        <node xmi:type="uml:OpaqueAction" xmi:id="_sw3xUF2iEeeC05B8er0jow"
name="Account_Cancelled">
            <inputValue xmi:type="uml:InputPin" xmi:id="_uQxEgF2nEeeC05B8er0jow"
name="InputPin1">
                <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_uQxrkF2nEeeC05B8er0jow" value="1"/>
            </inputValue>
        </node>
    </packagedElement>
</uml:Model>

```

Part B –XSLT Files

This section includes the XSLT files which are applied to XMI files and generate an XMI and an XML file, as discussed in Chapter 6.

First XSLT document used for the first XMI transformation

<!--The root element of a style sheet. The root element declare the XSLT namespace to get access to the XSLT elements and attributes-->

<xsl:stylesheet **version**="1.0" **xmlns:xsl**="http://www.w3.org/1999/XSL/Transform">

<!--Specifies the format for the result document, i.e. xml format-->

<!--"yes" indicates that the output should be indented according to its hierarchic structure-->

<xsl:output **method**="xml" **indent**="yes"/>

<!--A template contains processing instructions and commands for nodes in the input document that match the specified XPath expression.

@* | node() is a match pattern composed of three single patterns / matches a root node, @* matches any attribute node and node() as a pattern matches any node other than an attribute node and root node-->

<xsl:template **match**="@* | node()"

<!--Copy creates a duplicate of the current node being processed.-->

<xsl:copy>

<!--Applies template rules based on a given XPath selection criteria. If no template is found the built in templates are used. Select chooses all attributes and immediate children of the context node-->

<xsl:apply-templates **select**="@* | node()"/>

</xsl:copy>

</xsl:template>

<!--this template removes from the target XML document any upperBound node element-->

<xsl:template **match**="upperBound"/>

<!--this template removes from the target XML document the name attribute from any existing inputValue and outputValue node-->

<xsl:template **match**="inputValue/@name"/>


```
<xsl:template match="outputValue/@name"/>
</xsl:stylesheet>
```

Second XSLT document used for the second XMI transformation

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" />
<!--Template to match any child element of the XMI root element such as
packagedElement, node, edge, etc.-->
<xsl:template match="@* | node()">
<!-- Creates a copy of the child element-->
<xsl:copy>
<!--Apply template: selects all the attributes of the context elements-->
<xsl:apply-templates select="@* | node()" />
</xsl:copy>
</xsl:template>
<!--Template to match the XMI edge elements-->
<xsl:template match="edge">
<!--Creates a copy of the edge element-->
<xsl:copy>
<!--Apply template: selects all the attributes of the context node, i.e. edge, and
transforms the attributes to elements-->
<xsl:apply-templates select="@*" mode="to-element" />
</xsl:copy>
</xsl:template>
<!--Template to match the XMI node elements-->
<xsl:template match="node">
<!--Creates a copy of the node element-->
<xsl:copy>
<!--Apply template: selects all the attributes of the context node, i.e. node, and
transforms the attributes to elements-->
<xsl:apply-templates select="@*" mode="to-element" />
</xsl:copy>
```

```

</xsl:template>
<!--XMI attributes with their values are transformed into XML child elements -->
<!--Template to match the edge and node elements of the 2 templates above-->
<!--The name of the mode is used to match up with the apply-templates mode
attribute.-->
<xsl:template match="@*" mode="to-element">
<!--Transforms each attribute, exists in an edge/node element, to an element in the
result document. Name corresponds to the element is created.-->
<xsl:element name="{name()}">
<!--Used to pull the data values (context) from the element selected in the last row.-->
<xsl:value-of select="." />
</xsl:element>
</xsl:template>
</xsl:stylesheet>

```

Part C – Java Code for the XMI Transformations

Part C covers the two files of Java code developed for the two XMI model transformations as explained in Chapter 6.

Java code for the first XMI transformation

```

package data;

//This package defines the generic (application programming interfaces) APIs for
processing transformation instructions, and performing a transformation from source
to result.

import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

public class Main2 {
    public void simpleMessage2() {

```

//A TransformerFactory instance can be used to create Transformer and Templates objects. The system property that determines which Factory implementation to create is named "javax.xml.transform.TransformerFactory".

//This property names a concrete subclass of the TransformerFactory abstract class.

```
TransformerFactory factory = TransformerFactory.newInstance();
```

// XSLT file that defines the rules for the XML transformation

```
StreamSource xslStream = new StreamSource("xmi2xmi.xsl");
```

//input model (XMI file from Activity Diagram)

```
StreamSource in = new StreamSource("Activity_Diagram.uml");
```

// output model (XML file)

```
StreamResult out = new StreamResult("new_xmi_file.xml");
```

```
try{
```

//An instance of this abstract class can transform a source tree into a result tree. An instance of this class can be obtained with the TransformerFactory.newTransformer method. This instance may then be used to process XML from a variety of sources and write the transformation output to a variety of sinks.

```
Transformer transformer = factory.newTransformer(xslStream);
```

//Transform the XML Source (in) to a Result (out)

```
transformer.transform(in, out);
```

//This class specifies an exceptional condition that occurred during the transformation process.

```
} catch (TransformerException e){
```

//Print the trace of methods from where the error originated. This will trace all nested exception objects, as well as this object.

```
e.printStackTrace();
```

```
}
```

```
}
```

```
}
```

Java code for the second XMI transformation

```
package data;
```

```
import javax.xml.transform.Transformer;
```

```
import javax.xml.transform.TransformerException;
```

```

import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
public class Main {
    public void simpleMessage() {
        TransformerFactory factory = TransformerFactory.newInstance()
;
// XSLT file that defines the rules for the XML transformation
        StreamSource xslStream = new StreamSource("xmi2xml.xsl");
//input model (XMI file from Activity Diagram)
        StreamSource in = new StreamSource("Activity_Diagram.uml");
// output model (XML file)
        StreamResult out = new StreamResult("new_xml_file.xml");
        try{
            Transformer transformer = factory.newTransformer(xslStream);
transformer.transform(in, out);
        } catch (TransformerException e){
            e.printStackTrace();
        }
    }
}

```

Part D – Outputs from XMI Model Transformation of AD in Figure 6.4

In this section, the two XMI and XML files generated after the two model transformations of the XMI file obtained from the AD in Figure 6.4 are shown.

XMI File obtained from the first XMI transformation for the AD in Figure 6.4

```

<?xml version="1.0" encoding="UTF-8"?><uml:Model
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001" xmi:version="20131001"
xmi:id="_MJToQF2tEeeC05B8er0jow" name="RootElement">
    <packagedElement xmi:type="uml:Activity" xmi:id="_MJVdcF2tEeeC05B8er0jow"
name="2nd_example" node="_NBPV0F2tEeeC05B8er0jow _NVix4F2tEeeC05B8er0jow
_Si61sF2tEeeC05B8er0jow _XXc7sF2tEeeC05B8er0jow">

```

```

    <edge xmi:type="uml:ControlFlow" xmi:id="_mSSjUF2tEeeC05B8er0jow"
target="_Si61sF2tEeeC05B8er0jow" source="_XXc7sF2tEeeC05B8er0jow"/>
    <edge xmi:type="uml:ObjectFlow" xmi:id="_4BTiwF2tEeeC05B8er0jow"
target="_c2xrIF2tEeeC05B8er0jow" source="_r6HugF2tEeeC05B8er0jow">
    <guard xmi:type="uml:LiteralBoolean" xmi:id="_4Beh4F2tEeeC05B8er0jow"
value="true"/>
    <weight xmi:type="uml:LiteralInteger"
xmi:id="_4Beh4V2tEeeC05B8er0jow" value="1"/>
    </edge>
    <edge xmi:type="uml:ObjectFlow" xmi:id="_5CXXIF2tEeeC05B8er0jow"
target="_xJt_4F2tEeeC05B8er0jow" source="_uaxtYF2tEeeC05B8er0jow">
    <guard xmi:type="uml:LiteralBoolean" xmi:id="_5CpD8F2tEeeC05B8er0jow"
value="true"/>
    <weight xmi:type="uml:LiteralInteger"
xmi:id="_5CpD8V2tEeeC05B8er0jow" value="1"/>
    </edge>
    <node xmi:type="uml:OpaqueAction" xmi:id="_NBPV0F2tEeeC05B8er0jow"
name="Create_Order">
    <outputValue xmi:type="uml:OutputPin"
xmi:id="_r6HugF2tEeeC05B8er0jow" outgoing="_4BTiwF2tEeeC05B8er0jow"/>
    </node>
    <node xmi:type="uml:OpaqueAction" xmi:id="_NVix4F2tEeeC05B8er0jow"
name="Close_Order">
    <inputValue xmi:type="uml:InputPin" xmi:id="_xJt_4F2tEeeC05B8er0jow"
incoming="_5CXXIF2tEeeC05B8er0jow">
    </inputValue>
    </node>
    <node xmi:type="uml:AcceptEventAction" xmi:id="_Si61sF2tEeeC05B8er0jow"
name="Receive_Payment" incoming="_mSSjUF2tEeeC05B8er0jow">
    <result xmi:type="uml:OutputPin" xmi:id="_uaxtYF2tEeeC05B8er0jow"
name="" outgoing="_5CXXIF2tEeeC05B8er0jow">
    </result>
    <trigger xmi:type="uml:Trigger" xmi:id="_k_Y7kF2uEeeC05B8er0jow"/>
    </node>
    <node xmi:type="uml:SendSignalAction" xmi:id="_XXc7sF2tEeeC05B8er0jow"
name="Send_Invoice" outgoing="_mSSjUF2tEeeC05B8er0jow"
signal="_auo58F2tEeeC05B8er0jow">
    <target xmi:type="uml:InputPin" xmi:id="_c2xrIF2tEeeC05B8er0jow"
incoming="_4BTiwF2tEeeC05B8er0jow" type="_auo58F2tEeeC05B8er0jow">

```

```

        <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_c2xrIV2tEeeC05B8er0jow" value="1"/>
        <upperValue xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_c2xrIL2tEeeC05B8er0jow" value="1"/>
    </target>
</node>
</packagedElement>
<packagedElement xmi:type="uml:Signal" xmi:id="_auo58F2tEeeC05B8er0jow"
name="Verify_CC_Funds"/>
</uml:Model>

```

XML file obtained from the second XMI transformation for the AD in Figure 6.4

```

<?xml version="1.0" encoding="UTF-8"?><uml:Model
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001" xmi:version="20131001"
xmi:id="_MJToQF2tEeeC05B8er0jow" name="RootElement">
    <packagedElement xmi:type="uml:Activity" xmi:id="_MJVdcF2tEeeC05B8er0jow"
name="2nd_example" node="_NBPV0F2tEeeC05B8er0jow _NVix4F2tEeeC05B8er0jow
_Si61sF2tEeeC05B8er0jow _XXc7sF2tEeeC05B8er0jow">
        <edge>
<xmi:type>uml:ControlFlow</xmi:type>
<xmi:id>_mSSjUF2tEeeC05B8er0jow</xmi:id>
<target>_Si61sF2tEeeC05B8er0jow</target>
<source>_XXc7sF2tEeeC05B8er0jow</source>
</edge>
        <edge>
<xmi:type>uml:ObjectFlow</xmi:type>
<xmi:id>_4BTiwF2tEeeC05B8er0jow</xmi:id>
<target>_c2xrIF2tEeeC05B8er0jow</target>
<source>_r6HugF2tEeeC05B8er0jow</source>
</edge>
        <edge>
<xmi:type>uml:ObjectFlow</xmi:type>
<xmi:id>_5CXXIF2tEeeC05B8er0jow</xmi:id>
<target>_xJt_4F2tEeeC05B8er0jow</target>
<source>_uaxtYF2tEeeC05B8er0jow</source>
</edge>
    </node>

```

```

<xmi:type>uml:OpaqueAction</xmi:type>
<xmi:id>_NBPV0F2tEeeC05B8er0jow</xmi:id>
<name>Create_Order</name>
</node>
  <node>
    <xmi:type>uml:OpaqueAction</xmi:type>
    <xmi:id>_NVix4F2tEeeC05B8er0jow</xmi:id>
    <name>Close_Order</name>
  </node>
  <node>
    <xmi:type>uml:AcceptEventAction</xmi:type>
    <xmi:id>_Si61sF2tEeeC05B8er0jow</xmi:id>
    <name>Receive_Payment</name>
    <incoming>_mSSjUF2tEeeC05B8er0jow</incoming>
  </node>
  <node>
    <xmi:type>uml:SendSignalAction</xmi:type>
    <xmi:id>_XXc7sF2tEeeC05B8er0jow</xmi:id>
    <name>Send_Invoice</name>
    <outgoing>_mSSjUF2tEeeC05B8er0jow</outgoing>
    <signal>_auo58F2tEeeC05B8er0jow</signal>
  </node>
</packagedElement>
  <packagedElement xmi:type="uml:Signal" xmi:id="_auo58F2tEeeC05B8er0jow"
name="Verify_CC_Funds"/>
</uml:Model>

```

Appendix H – Advanced Generic SQL Code [A^T]

Appendix H shows the advanced generic SQL code developed for the automated generation of the transpose of the PN incidence matrix [A^T], discussed in Chapter 6.

```
package data;
import java.sql.*;

public class multiple_execution_of_classes{
    static String t0;
    static String t1;
    static String t2;
    static String t3;
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://127.0.0.1:3306/sql";
    // Database credentials
    static final String USER = "root";
    static final String PASS = "Xristina23";

    @SuppressWarnings({ "unused", "resource" })
    public static void main(String[] args) throws SQLException {

        Main MainObject = new Main();
        MainObject.simpleMessage();
        Main2 Main2Object = new Main2();
        Main2Object.simpleMessage2();

        Statement stmt = null;
        PreparedStatement pst = null;
        String cs =
"jdbc:mysql://localhost:3306/sql?allowMultiQueries=true";
        Connection conn1 = null;
        ResultSet rs = null;

        try {
            conn1 = DriverManager.getConnection(cs, USER, PASS);
            String code1 = "drop table if exists edge_xmi;"
                + " CREATE TABLE edge_xmi (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY,`xmi:type` VARCHAR(200) NULL,`xmi:id`
VARCHAR(200) NULL, `name` VARCHAR(200) NULL, `source` varchar(200)null,
`target` VARCHAR(200) NULL);";
            String code2 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xml_file.xml' INTO TABLE edge_xmi
ROWS IDENTIFIED BY '<edge>'; "
                + "update edge_xmi as t1 inner join edge_xmi as t2
on (t1.`name`=t2.`name`) and t1.`xmi:id` <> t2.`xmi:id` set
t1.name='place_';"
                    + "update edge_xmi as t1 inner join
edge_xmi as t2 on (t1.`name`=t2.`name`) and t1.`xmi:id` <> t2.`xmi:id` set
t1.name='place_';"
                        + "drop table if exists place_name;"
                        + "CREATE TABLE place_name as SELECT id,
`xmi:type`, `xmi:id`, CONCAT(name,',', id) AS name, source, target FROM
edge_xmi where edge_xmi.name='place_';" + "drop table if exists
edge_place_xmi;"
```



```

        + "CREATE TABLE edge_place_xmi SELECT *
FROM place_name UNION SELECT * FROM edge_xmi;"
        + "ALTER IGNORE TABLE `edge_place_xmi` ADD
UNIQUE (id, `xmi:id`);"
        + "drop table if exists
in_outputValue_xmi;"
        + "CREATE TABLE in_outputValue_xmi (id int
NOT NULL AUTO_INCREMENT PRIMARY KEY,`xmi:type` VARCHAR(200) NULL,`xmi:id`
VARCHAR(200) NULL, name VARCHAR(200) NULL,incoming VARCHAR(200) NULL,
outgoing varchar(200)null);"
        String code3 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xmi_file.xml' INTO TABLE
in_outputValue_xmi ROWS IDENTIFIED BY '<inputValue>';"
        String code4 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xmi_file.xml' INTO TABLE
in_outputValue_xmi ROWS IDENTIFIED BY '<outputValue>';"
        + "UPDATE in_outputValue_xmi t1 INNER JOIN
edge_place_xmi t2 ON t1.incoming = t2.`xmi:id` SET t1.incoming = t2.name;"
        String code5 = "UPDATE in_outputValue_xmi t1 INNER JOIN
edge_place_xmi t2 ON t1.outgoing = t2.`xmi:id` SET t1.outgoing = t2.name;"
        + "ALTER TABLE in_outputValue_xmi
CHANGE incoming place_before_node varchar(200) null;"
        + "ALTER TABLE in_outputValue_xmi
CHANGE outgoing place_after_node varchar(200) null;"
        + "ALTER TABLE in_outputValue_xmi
CHANGE `name` name_primary varchar(200) null;"
        + "DELETE FROM in_outputValue_xmi
WHERE place_after_node is null and place_before_node is null;"
        + "UPDATE in_outputValue_xmi t,
(SELECT DISTINCT place_before_node, name_primary, place_after_node FROM
in_outputValue_xmi) t1 SET t.place_after_node = t1.place_after_node WHERE
t.name_primary = t1.name_primary;"
        + "DELETE FROM in_outputValue_xmi
WHERE place_after_node is null or place_before_node is null;"
        + "drop table if exists
in_outputValue_final;"
        + "CREATE TABLE in_outputValue_final
(place_before_node VARCHAR(200) NULL, name_primary VARCHAR(200) NULL,
place_after_node varchar(200)null) as select place_before_node,
name_primary, place_after_node from in_outputValue_xmi;"
        String code5a = "drop table if exists node_xmi_send_accept;"
        + "create table node_xmi_send_accept as select *
from node_xmi where `element_type_nested`='target' or
`element_type_nested`='result';UPDATE node_xmi_send_accept t1 INNER JOIN
edge_place_xmi t2 ON t1.incoming_nested =t2.`xmi:id` SET t1.incoming_nested
= t2.name;UPDATE node_xmi_send_accept t1 INNER JOIN edge_place_xmi t2 ON
t1.outgoing_nested = t2.`xmi:id` SET t1.outgoing_nested = t2.name;"
        + "ALTER TABLE node_xmi_send_accept CHANGE
incoming_nested place_before_node varchar(200) null;"
        + "ALTER TABLE node_xmi_send_accept CHANGE
outgoing_nested place_after_node varchar(200) null;"
        + "ALTER TABLE node_xmi_send_accept CHANGE `name`
name_primary varchar(200) null;"
        + "drop table if exists send_accept_final;"
        + "CREATE TABLE send_accept_final
(place_before_node VARCHAR(200) NULL, name_primary VARCHAR(200) NULL,
place_after_node varchar(200)null) as select place_before_node,
name_primary, place_after_node from node_xmi_send_accept;"
        + "drop table if exists expansionNode_xmi;"

```

```

+ "CREATE TABLE expansionNode_xmi (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY, `xmi:type` VARCHAR(200) NULL, `xmi:id`
VARCHAR(200) NULL, name VARCHAR(200) NULL, incoming VARCHAR(200) NULL,
outgoing varchar(200)null, inputElement VARCHAR(200) NULL, outputElement
varchar(200)null, regionAsInput varchar(200) null);";

String code5b = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xmi_file.xml' INTO TABLE
expansionNode_xmi ROWS IDENTIFIED BY '<node>';"
+ "DELETE from expansionNode_xmi where
inputElement is null and outputElement is null;";
String code5c = "DELETE from expansionNode_xmi
where incoming is null and outgoing is null;";
+ "UPDATE expansionNode_xmi t1 INNER
JOIN edge_place_xmi t2 ON t1.inputElement = t2.`source` SET t1.inputElement
= t2.name;";
+ "UPDATE expansionNode_xmi t1 INNER
JOIN edge_place_xmi t2 ON t1.outputElement = t2.`target` SET
t1.outputElement = t2.name;";
+ "UPDATE expansionNode_xmi t1 INNER
JOIN edge_place_xmi t2 ON t1.incoming = t2.`xmi:id` SET t1.incoming =
t2.name;";
+ "UPDATE expansionNode_xmi t1 INNER
JOIN edge_place_xmi t2 ON t1.outgoing = t2.`xmi:id` SET t1.outgoing =
t2.name;";
+ "drop table if exists
expansionNode_xmi_for_delete;";
+ "create table
expansionNode_xmi_for_delete as select * from expansionNode_xmi where
`xmi:type`='uml:OpaqueAction';";
+ "UPDATE expansionNode_xmi t, (SELECT
DISTINCT * FROM expansionNode_xmi) t1 SET t.incoming = t1.incoming WHERE
t.inputElement = t1.outgoing and t.outgoing = t1.outputElement and
t.`xmi:type`='uml:OpaqueAction';";
+ "UPDATE expansionNode_xmi t, (SELECT
DISTINCT * FROM expansionNode_xmi) t1 SET t.outgoing = t1.outgoing WHERE
t.outputElement = t1.incoming and t1.`xmi:type`='uml:ExpansionNode' and
t.`xmi:type`='uml:OpaqueAction';";
+ "ALTER TABLE expansionNode_xmi
CHANGE incoming place_before_node varchar(200) null;";
+ "ALTER TABLE expansionNode_xmi
CHANGE outgoing place_after_node varchar(200) null;";
+ "ALTER TABLE expansionNode_xmi
CHANGE `name` name_primary varchar(200) null;";
+ "drop table if exists
expansionNode_final;";
+ "CREATE TABLE expansionNode_final
(place_before_node VARCHAR(200) NULL, name_primary VARCHAR(200) NULL,
place_after_node varchar(200)null) as select place_before_node,
name_primary, place_after_node from expansionNode_xmi where
`xmi:type`='uml:OpaqueAction';";

String code6 = "drop table if exists exceptionHandler_xmi;";
+ "CREATE TABLE exceptionHandler_xmi (id
int NOT NULL AUTO_INCREMENT PRIMARY KEY, `xmi:type` VARCHAR(200)
NULL, `xmi:id` VARCHAR(200) NULL, name VARCHAR(200) NULL, incoming
VARCHAR(200) NULL, outgoing varchar(200)null, `signal`
varchar(200)null, `element_type_nested` varchar(200)null);";

```

```

String code7 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xmi_file.xml' INTO TABLE
exceptionHandler_xmi ROWS IDENTIFIED BY '<handler>';"
        +"UPDATE exceptionHandler_xmi SET
`element_type_nested` = 'handler';";
String code8 = "drop table if exists result_xmi;"
        +"CREATE TABLE result_xmi (id int NOT
NULL AUTO_INCREMENT PRIMARY KEY, `xmi:type` VARCHAR(200) NULL,`xmi:id`
VARCHAR(200) NULL, name VARCHAR(200) NULL, incoming VARCHAR(200)
NULL,outgoing varchar(200)null,`signal`
varchar(200)null,`element_type_nested` varchar(200)null);";
String code9 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xmi_file.xml' INTO TABLE result_xmi
ROWS IDENTIFIED BY '<result>';"
        +"UPDATE result_xmi SET
`element_type_nested` = 'result';";
String code10 = "drop table if exists target_xmi;"
        +"CREATE TABLE target_xmi (id int NOT
NULL AUTO_INCREMENT PRIMARY KEY, `xmi:type` VARCHAR(200) NULL,`xmi:id`
VARCHAR(200) NULL, name VARCHAR(200) NULL, incoming VARCHAR(200)
NULL,outgoing varchar(200)null,`signal`
varchar(200)null,`element_type_nested` varchar(200)null);";
String code11 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xmi_file.xml' INTO TABLE target_xmi
ROWS IDENTIFIED BY '<target>';"
        +"UPDATE target_xmi SET
`element_type_nested` = 'target';";
String code12 = "drop table if exists node_xmi;"
        + "CREATE TABLE node_xmi SELECT * FROM
exceptionHandler_xmi UNION SELECT * FROM target_xmi UNION SELECT * FROM
result_xmi;";
String code13 = "drop table if exists node_xmi;"
        + "CREATE TABLE node_xmi SELECT * FROM
exceptionHandler_xmi UNION SELECT * FROM target_xmi UNION SELECT * FROM
result_xmi;"
        + "ALTER TABLE `node_xmi` CHANGE COLUMN `xmi:id`
`xmi:id_nested` VARCHAR(255) NULL; ALTER TABLE `node_xmi` CHANGE COLUMN
`xmi:type` `xmi:type_nested` VARCHAR(255) NULL; ALTER TABLE `node_xmi`
CHANGE COLUMN `incoming` `incoming_nested` VARCHAR(255) NULL;"
        + "ALTER TABLE `node_xmi` CHANGE COLUMN `outgoing`
`outgoing_nested` VARCHAR(255) NULL; ALTER TABLE `node_xmi` CHANGE COLUMN
`signal` `signal_nested` VARCHAR(255) NULL;"
        + "ALTER TABLE `node_xmi` DROP COLUMN`id`; ALTER
TABLE `node_xmi` ADD id INT PRIMARY KEY AUTO_INCREMENT first;";
String code14 = "update node_xmi as t1 inner join node_xmi
as t2 on (t1.`name`=t2.`name`) and t1.`xmi:id_nested` <> t2.`xmi:id_nested`
set t1.name='transition_';"
        + "drop table if exists transition_name;"
        + "CREATE TABLE transition_name as SELECT id,
`xmi:type_nested`, `xmi:id_nested`, CONCAT(name,'', id) AS name,
incoming_nested, outgoing_nested, signal_nested, element_type_nested FROM
node_xmi where node_xmi.name='transition_';"
        + "drop table if exists node_transition_xmi;"
        + "CREATE TABLE node_transition_xmi SELECT * FROM
transition_name UNION SELECT * FROM node_xmi;"
        + "ALTER IGNORE TABLE `node_transition_xmi` ADD
UNIQUE (id, `xmi:id_nested`);";
String code15 = "drop table if exists node_xml;";

```

```

        + "CREATE TABLE node_xml (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY, `xmi:type` VARCHAR(200) NULL, `xmi:type_nested`
VARCHAR(200) NULL, `xmi:id` VARCHAR(200) NULL, `xmi:id_nested`
VARCHAR(200) NULL, name VARCHAR(200) NULL, incoming VARCHAR(200) NULL,
outgoing varchar(200)null, `signal` varchar(200)null, `element_type`
varchar(200)null, `element_type_nested` varchar(200)null);";
        String code16 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xmi_file.xml' INTO TABLE node_xml
ROWS IDENTIFIED BY '<node>';";
        String code17 = "UPDATE node_xml SET `element_type_nested` =
'node';"

        + "update node_xml as t1 inner join node_xml as
t2 on (t1.`name`=t2.`name`) and t1.`xmi:id` <> t2.`xmi:id` set
t1.name='transition1';"
        + "drop table if exists transition1_name;"
        + "CREATE TABLE transition1_name as SELECT id,
`xmi:type`, `xmi:type_nested`, `xmi:id`, `xmi:id_nested`, CONCAT(name, '',
id) AS name, incoming, outgoing, `signal`, `element_type`,
`element_type_nested` FROM node_xml where node_xml.name='transition1';"
        + "drop table if exists node_transition_xml;"
        + "CREATE TABLE node_transition_xml SELECT * FROM
transition1_name UNION SELECT * FROM node_xml;"
        + "ALTER IGNORE TABLE `node_transition_xml` ADD
UNIQUE (id, `xmi:id`);"
        + "drop table if exists node_xmi_xml;"
        + "create table node_xmi_xml as SELECT c1.id,
c1.`xmi:type`, c2.`xmi:type_nested`, c1.`xmi:id`, c2.`xmi:id_nested`,
c1.name, c1.incoming, c1.outgoing, c1.signal, c1.`element_type`,
c2.`element_type_nested` FROM node_transition_xml c1 INNER JOIN node_xmi c2
ON c1.name = c2.name or c2.incoming_nested = c1.incoming or
c2.outgoing_nested = c1.outgoing ORDER BY c1.id;"
        + "UPDATE node_xmi_xml T SET T.incoming =(SELECT
incoming_nested FROM node_xmi A WHERE A.name = T.name or A.incoming_nested
= T.incoming);"
        + "UPDATE node_xmi_xml T SET T.outgoing =(SELECT
outgoing_nested FROM node_xmi A WHERE A.name = T.name or A.outgoing_nested
= T.outgoing );"
        + "UPDATE node_xmi_xml T SET T.signal =(SELECT
signal_nested FROM node_xmi A WHERE A.name = T.name);"
        + "drop table if exists final_node_xmi_xml; "
        + "CREATE TABLE final_node_xmi_xml SELECT * FROM
node_xmi_xml UNION SELECT * FROM node_transition_xml;"
        + "ALTER IGNORE TABLE `final_node_xmi_xml` ADD
UNIQUE (id); "
        + "ALTER TABLE final_node_xmi_xml DROP COLUMN
`signal`;"
        + "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1', 'decision') where `xmi:type` like
'uml:DecisionNode';"
        + "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1', 'merge') where `xmi:type` like 'uml:MergeNode';"
        + "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1', 'fork') where `xmi:type` like 'uml:ForkNode';"
        + "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1', 'join') where `xmi:type` like 'uml:JoinNode';"
        + "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1', 'pin') where `xmi:type` like 'uml:InitialNode';"

```

```

+ "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1','flow_pout') where `xmi:type`like
'uml:FlowFinalNode';"
+ "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1','pout') where `xmi:type`like
'uml:ActivityFinalNode';"
+ "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1','parameter_node') where `xmi:type`like
'uml:ActivityParameterNode';"
+ "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1','buffer_node') where `xmi:type`like
'uml:CentralBufferNode';"
+ "update `final_node_xmi_xml` set `name`= replace
(`name`, 'transition1','datastore_node') where `xmi:type`like
'uml:DataStoreNode';"
+ "UPDATE final_node_xmi_xml SET incoming =
REPLACE(incoming, '_', '');"
+ "UPDATE final_node_xmi_xml SET incoming =
REPLACE(incoming, ' ', '_');"
+ "UPDATE final_node_xmi_xml SET outgoing =
REPLACE(outgoing, '_', '');"
+ "UPDATE final_node_xmi_xml SET outgoing =
REPLACE(outgoing, ' ', '_');"

String code18 = "drop table if exists double_nodes_outgoing
;"
+ "create table double_nodes_outgoing as select *
from final_node_xmi_xml where outgoing like '%_%_%' ;"
+ "drop table if exists double_nodes_outgoing ;"
+ "create table double_nodes_outgoing as select *
from final_node_xmi_xml where outgoing like '%_%_%' ;"
+ "drop table if exists numbers ;"
+ "create table numbers (n int not null) ;"
+ "insert into numbers (n) values (1), (2), (3),
(4), (5), (6), (7), (8), (9),(10), (11), (12), (13), (14), (15) ;"
+ "drop table if exists double_separate_nodes ;"
+ "create table double_separate_nodes as select
double_nodes_outgoing.id, `xmi:type`, `xmi:type_nested`, `xmi:id`,
`xmi:id_nested`, `name`, incoming, `element_type`, `element_type_nested`,
SUBSTRING_INDEX(SUBSTRING_INDEX(double_nodes_outgoing.outgoing, '_',
numbers.n), '_', -1) outgoing from numbers inner join double_nodes_outgoing
on CHAR_LENGTH(double_nodes_outgoing.outgoing)-
CHAR_LENGTH(REPLACE(double_nodes_outgoing.outgoing, '_', ''))>=numbers.n-1
order by id, n,`xmi:type`, `xmi:type_nested`, `xmi:id`, `xmi:id_nested`,
`name`, incoming, outgoing,`element_type`, `element_type_nested`;
+ "DELETE FROM double_separate_nodes WHERE outgoing
= '' ;"
+ "update double_separate_nodes set outgoing =
concat('_', outgoing) ;"
+ "alter table `double_separate_nodes` change
column outgoing outgoing varchar(255) after `id` ;"
+ "drop table if exists lessthan17 ;"
+ "create table lessthan17 (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY) as SELECT `outgoing`, `xmi:type`,
`xmi:type_nested`, `xmi:id`, `xmi:id_nested`, `name`, incoming,
`element_type`, `element_type_nested` FROM double_separate_nodes WHERE
LENGTH(outgoing) < 22;"
+ "drop table if exists greaterthan11 ;"

```

```

        + "create table greaterthan11 (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY) as SELECT `outgoing`, `xmi:type`,
`xmi:type_nested`, `xmi:id`, `xmi:id_nested`, `name`, incoming,
`element_type`, `element_type_nested` FROM lessthan17 WHERE
LENGTH(outgoing) >11 ;"
        + "DELETE FROM lessthan17 where LENGTH(outgoing) >
11 ;"
        + "alter table `lessthan17` drop column id ;"
        + "alter table `lessthan17` add column id int NOT
NULL AUTO_INCREMENT PRIMARY KEY first;"
        + "drop table if exists merge_1_table;"
        + "create table merge_1_table select * from
greaterthan11 union all select * from lessthan17; "
        + "drop table if exists merge_shorter_than22; "
        + "create table merge_shorter_than22 (id int NOT
NULL AUTO_INCREMENT PRIMARY KEY) as SELECT GROUP_CONCAT(outgoing SEPARATOR
''), `xmi:type`, `xmi:type_nested`, `xmi:id`, `xmi:id_nested`, `name`,
incoming, `element_type`, `element_type_nested` FROM lessthan17 GROUP BY
name;"
        + "ALTER TABLE `merge_shorter_than22` CHANGE COLUMN
`GROUP_CONCAT(outgoing SEPARATOR '')` `outgoing` VARCHAR(255) NOT NULL "
        + "drop table if exists union_1 ;"
        + "create table union_1 select * from
merge_shorter_than22 union all select * from double_separate_nodes ;"
        + "DELETE FROM union_1 where LENGTH(outgoing) < 21
;"
        + "drop table if exists double_nodes_incoming ;"
        + "create table double_nodes_incoming as
select * from final_node_xmi_xml where incoming like '%_ % _%' ; "
        + "drop table if exists
double_separate_nodes ;"
        + "create table double_separate_nodes as
select double_nodes_incoming.id, `xmi:type`, `xmi:type_nested`, `xmi:id`,
`xmi:id_nested`, `name`, outgoing, `element_type`, `element_type_nested`,
SUBSTRING_INDEX(SUBSTRING_INDEX(double_nodes_incoming.incoming, '_',
numbers.n), '_', -1) incoming from numbers inner join double_nodes_incoming
on CHAR_LENGTH(double_nodes_incoming.incoming)-
CHAR_LENGTH(REPLACE(double_nodes_incoming.incoming, '_', ''))>=numbers.n-1
order by id, n, `xmi:type`, `xmi:type_nested`, `xmi:id`, `xmi:id_nested`,
`name`, incoming, outgoing, `element_type`, `element_type_nested`;"
        + "DELETE FROM double_separate_nodes WHERE
incoming = ' ';"
        + "update double_separate_nodes set incoming
= concat('_', incoming) ;"
        + "alter table `double_separate_nodes`
change column incoming incoming varchar(255) after `id` ;"
        + "drop table if exists lessthan17 ;"
        + "create table lessthan17 (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY) SELECT `outgoing`, `xmi:type`,
`xmi:type_nested`, `xmi:id`, `xmi:id_nested`, `name`, incoming,
`element_type`, `element_type_nested` FROM double_separate_nodes WHERE
LENGTH(incoming) < 22;"
        + "drop table if exists greaterthan11 ;"
        + "create table greaterthan11 (id int NOT
NULL AUTO_INCREMENT PRIMARY KEY) as SELECT `outgoing`, `xmi:type`,
`xmi:type_nested`, `xmi:id`, `xmi:id_nested`, `name`, incoming,
`element_type`, `element_type_nested` FROM lessthan17 WHERE
LENGTH(incoming) >11 ;"

```

```

+ "DELETE FROM lessthan17 where
LENGTH(incoming) > 11 ;"
+ "alter table `lessthan17` drop column id
;"
+ "alter table `lessthan17` add column id
int NOT NULL AUTO_INCREMENT PRIMARY KEY first;"
+ "drop table if exists merge_1_table;"
+ "create table merge_1_table select * from
greaterthan11 union all select * from lessthan17;"
+ "drop table if exists
merge_shorter_than22; "+ "create table merge_shorter_than22 (id int NOT
NULL AUTO_INCREMENT PRIMARY KEY) as SELECT GROUP_CONCAT(incoming SEPARATOR
''), `xmi:type`, `xmi:type_nested`, `xmi:id`, `xmi:id_nested`, `name`,
outgoing, `element_type`, `element_type_nested` FROM lessthan17 GROUP BY
name;"
+ "ALTER TABLE `merge_shorter_than22` CHANGE
COLUMN `GROUP_CONCAT(incoming SEPARATOR '')` `incoming` VARCHAR(255) NOT
NULL ; "
+ "drop table if exists union_2a ;"
+ "create table union_2a select * from
merge_shorter_than22 union all select * from double_separate_nodes ;"
+ "drop table if exists union_2 ;"
+ "create table union_2 as select id,
`outgoing`, `xmi:type`, `xmi:type_nested`, `xmi:id`, `xmi:id_nested`,
`name`, incoming, `element_type`, `element_type_nested` from union_2a ; "
+ "DELETE FROM union_2 where
LENGTH(incoming) < 21 ; ";

String code19b= "drop table if exists unique_activities;"
+ "UPDATE final_node_xmi_xml SET incoming = Concat('_',
incoming); "
+ "UPDATE final_node_xmi_xml SET outgoing = Concat('_',
outgoing); "
+ "create table unique_activities as select id, outgoing,
`xmi:type`, `xmi:type_nested`, `xmi:id`, `xmi:id_nested`, `name`, incoming,
`element_type`, `element_type_nested` from final_node_xmi_xml ;"
+ "DELETE FROM unique_activities where LENGTH(incoming) >
22 ; "
+ "DELETE FROM unique_activities where LENGTH(outgoing) >
22 ; "
+ "UPDATE edge_place_xmi SET `xmi:id` = REPLACE(`xmi:id`,
'_', '') ; "
+ "UPDATE edge_place_xmi SET `xmi:id` = Concat('_',
`xmi:id`) ; "
+ "UPDATE edge_place_xmi SET `source` = REPLACE(`source`,
'_', '') ; "
+ "UPDATE edge_place_xmi SET `source` = Concat('_',
`source`) ; "
+ "UPDATE edge_place_xmi SET target = REPLACE(`target`,
'_', '') ; "
+ "UPDATE edge_place_xmi SET `target` = Concat('_',
`target`) ; "
+ "drop table if exists union_node;"
+ "UPDATE union_1 SET `incoming` = Concat('_',
`incoming`) ; "
+ "UPDATE union_2 SET `outgoing` = Concat('_',
`outgoing`) ; "
+ "create table union_node select * from union_1 union
select * from union_2 union select * from unique_activities;"

```

```

        + "alter table union_node drop column id; "
        + "alter table union_node add column id int NOT NULL
AUTO_INCREMENT PRIMARY KEY FIRST; "
        + "drop table if exists handler_xmi;"
        + "CREATE TABLE handler_xmi (id int NOT NULL
AUTO_INCREMENT PRIMARY KEY, `xmi:type` VARCHAR(200) NULL, `xmi:id`
VARCHAR(200) NULL, name VARCHAR(200) NULL, incoming VARCHAR(200) NULL,
outgoing varchar(200)null, `exceptionInput` varchar(200)null,
`exceptionType` varchar(200)null, `handlerBody` varchar(200)null);";

String code19 = "LOAD XML LOCAL INFILE
'c:/users/CHRISTINA/workspace/data/new_xmi_file.xml' INTO TABLE handler_xmi
ROWS IDENTIFIED BY '<handler>';"
        + "UPDATE handler_xmi SET `name`=CONCAT_WS('_', `name`,
'handler');" ;

String code20 = "drop table if exists handler_node_table;"
        + "CREATE TABLE handler_node_table
(`transition_before_node` VARCHAR(200) NULL, `transition_after_node`
VARCHAR(200) NULL) as SELECT * FROM handler_xmi where
(handler_xmi.`xmi:type`='uml:ExceptionHandler');"
        + "ALTER TABLE handler_node_table CHANGE COLUMN `xmi:id`
`xmi:id_primary` VARCHAR(255) NULL; "
        + "ALTER TABLE handler_node_table CHANGE COLUMN `name`
`name_primary` VARCHAR(255) NULL;"
        + "UPDATE handler_node_table t1 INNER JOIN
handler_node_node t2 ON t1.incoming = t2.`incoming` SET
t1.transition_before_node = t2.name;"
        + "drop table if exists handler_node_node; "
        + "drop table if exists handler_node_node; "
        + "CREATE TABLE handler_node_node AS SELECT m.*,
u2.`xmi:id`, u2.`name` FROM handler_node_table m INNER JOIN union_node u2
ON (m.`exceptionInput`= u2.`xmi:id`) or (m.`incoming`= u2.`incoming`);"
        + "UPDATE handler_node_table t1 INNER JOIN
handler_node_node t2 ON t1.`exceptionInput` = t2.`xmi:id` SET
t1.transition_after_node = t2.name; "
        + "UPDATE handler_node_table t1 INNER JOIN edge_place_xmi
t2 ON t1.incoming = t2.`xmi:id` SET t1.incoming = t2.name;"
        + "UPDATE handler_node_table t1 INNER JOIN union_node t2
ON t1.transition_after_node = t2.`name` SET t1.outgoing = t2.outgoing;"
        + "UPDATE handler_node_table t1 INNER JOIN edge_place_xmi
t2 ON t1.outgoing = t2.`xmi:id` SET t1.outgoing = t2.name;"
        + "drop table if exists handler_node_table_b; "
        + "create table handler_node_table_b as select
`transition_before_node`,`name_primary` from handler_node_table; "
        + "ALTER TABLE handler_node_table_b CHANGE COLUMN
`name_primary` `place_after_node` VARCHAR(255) NULL; "
        + "ALTER TABLE handler_node_table_b CHANGE COLUMN
`transition_before_node` `name_primary` VARCHAR(255) NULL; "
        + "ALTER TABLE handler_node_table_b ADD COLUMN
`place_before_node` VARCHAR(255) NULL FIRST;"
        + "drop table if exists handler_node_table_a; "
        + "create table handler_node_table_a as select
`name_primary`, `transition_after_node` from handler_node_table; "
        + "ALTER TABLE handler_node_table_a CHANGE COLUMN
`name_primary` `place_before_node` VARCHAR(255) NULL;"
        + "ALTER TABLE handler_node_table_a CHANGE COLUMN
`transition_after_node` `name_primary` VARCHAR(255) NULL; "

```



```

+ "ALTER TABLE handler_node_table_a ADD COLUMN
`place_after_node` VARCHAR(255) NULL;"
+ "drop table if exists handler_node_table_a_b;"
+ "create table handler_node_table_a_b select * from
handler_node_table_b union all select * from handler_node_table_a;"
+ "UPDATE handler_node_table_a_b t1 INNER JOIN
handler_node_table t2 ON t1.name_primary = t2.transition_before_node SET
t1.place_before_node = t2.incoming;"
+ "UPDATE handler_node_table_a_b t1 INNER JOIN
handler_node_table t2 ON t1.name_primary = t2.transition_after_node SET
t1.place_after_node = t2.outgoing;"
+ "drop table if exists initial_final_table;"
+ "CREATE TABLE initial_final_table
(`transition_before_node` VARCHAR(200) NULL, `transition_after_node`
VARCHAR(200) NULL) as SELECT id, `xmi:type`, `xmi:id`, `name` FROM
union_node where (union_node.`xmi:type`='uml:ActivityFinalNode') or
(union_node.`xmi:type`='uml:FlowFinalNode') or
(union_node.`xmi:type`='uml:InitialNode');"
+ "ALTER TABLE initial_final_table CHANGE COLUMN `xmi:id`
`xmi:id_primary` VARCHAR(255) NULL;"
+ "drop table if exists final_node;"
+ "CREATE TABLE final_node AS SELECT m.*, u1.target,
u1.source FROM initial_final_table m INNER JOIN edge_place_xmi u1 ON
(m.`xmi:id_primary`= u1.target); "
+ "ALTER TABLE `final_node` ADD COLUMN `xmi:id`
VARCHAR(255) NOT NULL;"
+ "ALTER TABLE `final_node` DROP COLUMN
`transition_after_node`;"
+ "UPDATE final_node t1 INNER JOIN edge_place_xmi t2 ON
t1.target = t2.target SET t1.`xmi:id` = t2.`xmi:id`; "
+ "UPDATE final_node INNER JOIN union_node ON
final_node.source = union_node.`xmi:id` SET
final_node.transition_before_node = union_node.name;"
+ "drop table if exists final_node_table;"
+ "create table final_node_table as select
`transition_before_node`,`name` from final_node; "
+ "ALTER TABLE final_node_table CHANGE COLUMN `name`
`place_after_node` VARCHAR(255) NULL;"
+ "ALTER TABLE final_node_table CHANGE COLUMN
`transition_before_node` `name_primary` VARCHAR(255) NULL;"
+ "ALTER TABLE final_node_table ADD COLUMN
`place_before_node` VARCHAR(255) NULL FIRST;"
+ "drop table if exists initial_node;"
+ "CREATE TABLE initial_node AS SELECT m.*, u2.target,
u2.source FROM initial_final_table m INNER JOIN edge_place_xmi u2 ON
(m.`xmi:id_primary`= u2.source); "
+ "ALTER TABLE `initial_node` ADD COLUMN `xmi:id`
VARCHAR(255) NOT NULL;"
+ "UPDATE initial_node t1 INNER JOIN edge_place_xmi t2 ON
t1.source = t2.source SET t1.`xmi:id` = t2.`xmi:id`;"
+ "ALTER TABLE `initial_node` DROP COLUMN
`transition_before_node`;"
+ "UPDATE initial_node INNER JOIN union_node ON
initial_node.target = union_node.`xmi:id` SET
initial_node.transition_after_node = union_node.name;"
+ "drop table if exists initial_node_table; create table
initial_node_table as select `name`, `transition_after_node` from
initial_node; "

```

```

        + "ALTER TABLE initial_node_table CHANGE COLUMN `name`
`place_before_node` VARCHAR(255) NULL; "
        + "ALTER TABLE initial_node_table CHANGE COLUMN
`transition_after_node` `name_primary` VARCHAR(255) NULL; "
        + "ALTER TABLE initial_node_table ADD COLUMN
`place_after_node` VARCHAR(255) NULL;"
        + "delete from edge_place_xmi WHERE `target` in (SELECT
DISTINCT `target` FROM `final_node`);"
        + "delete from edge_place_xmi WHERE `target` in (SELECT
DISTINCT `target` FROM `initial_node`);";

String code21 = "drop table if exists activity_parameter_table; "
        + "CREATE TABLE activity_parameter_table
(`transition_after_node` VARCHAR(200) NULL) as SELECT id, `xmi:type`,
`xmi:id`, `name`, outgoing FROM union_node where
(union_node.`xmi:type`='uml:ActivityParameterNode');"
        + "ALTER TABLE activity_parameter_table CHANGE COLUMN
`xmi:id` `xmi:id_primary` VARCHAR(255) NULL;"
        + "drop table if exists parameter_node; "
        + "CREATE TABLE parameter_node AS SELECT m.*, u2.target,
u2.source FROM activity_parameter_table m INNER JOIN edge_place_xmi u2 ON
(m.`xmi:id_primary`= u2.source); ALTER TABLE `parameter_node` ADD COLUMN
`xmi:id` VARCHAR(255) NOT NULL; UPDATE parameter_node t1 INNER JOIN
edge_place_xmi t2 ON t1.source = t2.source SET t1.`xmi:id` = t2.`xmi:id`;"
        + "drop table if exists par_node; "
        + "CREATE TABLE par_node AS SELECT * FROM union_node
where `xmi:type`='uml:ActivityParameterNode';"
        + "drop table if exists par_union_node; "
        + "CREATE TABLE par_union_node AS SELECT c.* FROM
union_node c inner join par_node a on c.incoming=a.outgoing;"
        + "drop table if exists par_union_union; "
        + "CREATE TABLE par_union_union SELECT * FROM par_node
union select * from par_union_node; "
        + "UPDATE activity_parameter_table t1 INNER JOIN
par_union_union t2 ON t1.outgoing = t2.incoming SET
t1.transition_after_node = t2.name;"
        + "delete from edge_place_xmi WHERE `target` in (SELECT
DISTINCT `target` FROM `parameter_node`);"
        + "drop table if exists activity_parameter_table_a; "
        + "create table activity_parameter_table_a as select
`name`, `transition_after_node` from activity_parameter_table; "
        + "ALTER TABLE activity_parameter_table_a CHANGE COLUMN
`name` `place_before_node` VARCHAR(255) NULL; "
        + "ALTER TABLE activity_parameter_table_a CHANGE COLUMN
`transition_after_node` `name_primary` VARCHAR(255) NULL;"
        + "ALTER TABLE activity_parameter_table_a ADD COLUMN
`place_after_node` VARCHAR(255) NULL; "
        + "drop table if exists activity_parameter_table1; "
        + "CREATE TABLE activity_parameter_table1
(`transition_before_node` VARCHAR(200) NULL) as SELECT id, `xmi:type`,
`xmi:id`, `name`, incoming FROM union_node where
(union_node.`xmi:type`='uml:ActivityParameterNode');"
        + "ALTER TABLE activity_parameter_table1 CHANGE COLUMN
`xmi:id` `xmi:id_primary` VARCHAR(255) NULL;"
        + "drop table if exists parameter_node1; "
        + "CREATE TABLE parameter_node1 AS SELECT m.*,
u2.target, u2.source FROM activity_parameter_table1 m INNER JOIN
edge_place_xmi u2 ON (m.`xmi:id_primary`= u2.target);"

```

```

+ "ALTER TABLE `parameter_node1` ADD COLUMN `xmi:id`
VARCHAR(255) NOT NULL;"
+ "UPDATE parameter_node1 t1 INNER JOIN edge_place_xmi t2
ON t1.target = t2.target SET t1.`xmi:id` = t2.`xmi:id`;"
+ "drop table if exists par_node1;"
+ "CREATE TABLE par_node1 AS SELECT * FROM union_node
where `xmi:type`='uml:ActivityParameterNode';"
+ "drop table if exists par_union_node1;"
+ "CREATE TABLE par_union_node1 AS SELECT c.* FROM
union_node c inner join par_node a on c.outgoing=a.incoming;"
+ "drop table if exists par_union_union1;"
+ "CREATE TABLE par_union_union1 SELECT * FROM par_node1
union select * from par_union_node1;"
+ "UPDATE activity_parameter_table1 t1 INNER JOIN
par_union_union1 t2 ON t1.incoming = t2.outgoing SET
t1.transition_before_node = t2.name;"
+ "delete from edge_place_xmi WHERE `source` in (SELECT
DISTINCT `source` FROM `parameter_node`);"
+ "drop table if exists activity_parameter_table1_b;"
+ "create table activity_parameter_table1_b as select
`transition_before_node`,`name` from activity_parameter_table1;"
+ "ALTER TABLE activity_parameter_table1_b CHANGE COLUMN
`name` `place_after_node` VARCHAR(255) NULL;"
+ "ALTER TABLE activity_parameter_table1_b CHANGE COLUMN
`transition_before_node` `name_primary` VARCHAR(255) NULL;"
+ "ALTER TABLE activity_parameter_table1_b ADD COLUMN
`place_before_node` VARCHAR(255) NULL FIRST;"
+ "drop table if exists activity_parameter_table_a_b;"
+ "create table activity_parameter_table_a_b select *
from activity_parameter_table1_b union all select * from
activity_parameter_table_a;"

+ "drop table if exists datastore_table;"
+ "CREATE TABLE datastore_table
(`transition_before_node` VARCHAR(200) NULL, `transition_after_node`
VARCHAR(200) NULL) as SELECT id, `xmi:type`, `xmi:id`, `name`, incoming,
outgoing FROM union_node where
(union_node.`xmi:type`='uml:DataStoreNode');"
+ "ALTER TABLE datastore_table CHANGE COLUMN `xmi:id`
`xmi:id_primary` VARCHAR(255) NULL;"
+ "drop table if exists datastore_node; CREATE TABLE
datastore_node AS SELECT m.*, u2.target, u2.source FROM datastore_table m
INNER JOIN edge_place_xmi u2 ON (m.`xmi:id_primary`= u2.source);"
+ "ALTER TABLE `datastore_node` ADD COLUMN `xmi:id`
VARCHAR(255) NOT NULL;"
+ "UPDATE datastore_node t1 INNER JOIN edge_place_xmi t2
ON t1.source = t2.source SET t1.`xmi:id` = t2.`xmi:id`;"
+ "drop table if exists data_node; CREATE TABLE data_node
AS SELECT * FROM union_node where `xmi:type`='uml:DataStoreNode';"
+ "drop table if exists data_union_node;"
+ "CREATE TABLE data_union_node AS SELECT c.* FROM
union_node c inner join data_node a on c.incoming=a.outgoing;"
+ "drop table if exists data_union_union;"
+ "CREATE TABLE data_union_union SELECT * FROM data_node
union select * from data_union_node;"
+ "UPDATE datastore_table t1 INNER JOIN data_union_union
t2 ON t1.outgoing = t2.incoming SET t1.transition_after_node = t2.name;"
+ "delete from edge_place_xmi WHERE `target` in (SELECT
DISTINCT `target` FROM `datastore_node`);"

```

```

+ "drop table if exists datastore_node1; "
+ "CREATE TABLE datastore_node1 AS SELECT m.*,
u2.target, u2.source FROM datastore_table m INNER JOIN edge_place_xmi u2
ON (m.`xmi:id_primary`= u2.source); "
+ "ALTER TABLE `datastore_node1` ADD COLUMN `xmi:id`
VARCHAR(255) NOT NULL; "
+ "UPDATE datastore_node1 t1 INNER JOIN edge_place_xmi t2
ON t1.target = t2.target SET t1.`xmi:id` = t2.`xmi:id`; "
+ "drop table if exists data_union_node1; "
+ "CREATE TABLE data_union_node1 AS SELECT c.* FROM
union_node c inner join data_node a on c.outgoing=a.incoming;"
+ "drop table if exists data_union_union1; "
+ "CREATE TABLE data_union_union1 SELECT * FROM data_node
union select * from data_union_node1;"
+ "UPDATE datastore_table t1 INNER JOIN data_union_union1
t2 ON t1.incoming = t2.outgoing SET t1.transition_before_node = t2.name;"
+ "delete from edge_place_xmi WHERE `source` in (SELECT
DISTINCT `source` FROM `datastore_node1`);"
+ "drop table if exists datastore_table_b; "
+ "create table datastore_table_b as select
`transition_before_node`,`name` from datastore_table; "
+ "ALTER TABLE datastore_table_b CHANGE COLUMN `name`
`place_after_node` VARCHAR(255) NULL; "
+ "ALTER TABLE datastore_table_b CHANGE COLUMN
`transition_before_node` `name_primary` VARCHAR(255) NULL; "
+ "ALTER TABLE datastore_table_b ADD COLUMN
`place_before_node` VARCHAR(255) NULL FIRST;"
+ "drop table if exists datastore_table_a;"
+ "create table datastore_table_a as select `name`,
`transition_after_node` from datastore_table; "
+ "ALTER TABLE datastore_table_a CHANGE COLUMN `name`
`place_before_node` VARCHAR(255) NULL;"
+ "ALTER TABLE datastore_table_a CHANGE COLUMN
`transition_after_node` `name_primary` VARCHAR(255) NULL;"
+ "ALTER TABLE datastore_table_a ADD COLUMN
`place_after_node` VARCHAR(255) NULL;"
+ "drop table if exists datastore_table_a_b;"
+ "create table datastore_table_a_b select * from
datastore_table_b union all select * from datastore_table_a;"

+ "drop table if exists central_buffer_table; "
+ "CREATE TABLE central_buffer_table
(`transition_before_node` VARCHAR(200) NULL, `transition_after_node`
VARCHAR(200) NULL) as SELECT id, `xmi:type`, `xmi:id`, `name`, incoming,
outgoing FROM union_node where
(union_node.`xmi:type`='uml:CentralBufferNode');"
+ "ALTER TABLE central_buffer_table CHANGE COLUMN
`xmi:id` `xmi:id_primary` VARCHAR(255) NULL;"
+ "drop table if exists central_buffer_node;"
+ "CREATE TABLE central_buffer_node AS SELECT m.*,
u2.target, u2.source FROM central_buffer_table m INNER JOIN edge_place_xmi
u2 ON (m.`xmi:id_primary`= u2.source);"
+ "ALTER TABLE `central_buffer_node` ADD COLUMN `xmi:id`
VARCHAR(255) NOT NULL;"
+ "UPDATE central_buffer_node t1 INNER JOIN
edge_place_xmi t2 ON t1.source = t2.source SET t1.`xmi:id` = t2.`xmi:id`;"
+ "drop table if exists buffer_node; "
+ "CREATE TABLE buffer_node AS SELECT * FROM union_node
where `xmi:type`='uml:CentralBufferNode';"

```

```

+ "drop table if exists buffer_union_node; "
+ "CREATE TABLE buffer_union_node AS SELECT c.* FROM
union_node c inner join buffer_node a on c.incoming=a.outgoing;"
+ "drop table if exists buffer_union_union; CREATE TABLE
buffer_union_union SELECT * FROM buffer_node union select * from
buffer_union_node;"
+ "UPDATE central_buffer_table t1 INNER JOIN
buffer_union_union t2 ON t1.outgoing = t2.incoming SET
t1.transition_after_node = t2.name;"
+ "delete from edge_place_xmi WHERE `target` in (SELECT
DISTINCT `target` FROM `central_buffer_node`);"
+ "drop table if exists central_buffer_node1;"
+ "CREATE TABLE central_buffer_node1 AS SELECT m.*,
u2.target, u2.source FROM central_buffer_table m INNER JOIN edge_place_xmi
u2 ON (m.`xmi:id_primary` = u2.target);"
+ "ALTER TABLE `central_buffer_node1` ADD COLUMN `xmi:id`
VARCHAR(255) NOT NULL; UPDATE central_buffer_node1 t1 INNER JOIN
edge_place_xmi t2 ON t1.target = t2.target SET t1.`xmi:id` = t2.`xmi:id`;"
+ "drop table if exists buffer_union_node1;"
+ "CREATE TABLE buffer_union_node1 AS SELECT c.* FROM
union_node c inner join buffer_node a on c.outgoing=a.incoming;"
+ "drop table if exists buffer_union_union1;"
+ "CREATE TABLE buffer_union_union1 SELECT * FROM
buffer_node union select * from buffer_union_node1;"
+ "UPDATE central_buffer_table t1 INNER JOIN
buffer_union_union1 t2 ON t1.incoming = t2.outgoing SET
t1.transition_before_node = t2.name;"
+ "delete from edge_place_xmi WHERE `source` in (SELECT
DISTINCT `source` FROM `central_buffer_node1`);"
+ "drop table if exists central_buffer_table_b;"
+ "create table central_buffer_table_b as select
`transition_before_node`,`name` from central_buffer_table; "
+ "ALTER TABLE central_buffer_table_b CHANGE COLUMN
`name` `place_after_node` VARCHAR(255) NULL; "
+ "ALTER TABLE central_buffer_table_b CHANGE COLUMN
`transition_before_node` `name_primary` VARCHAR(255) NULL; "
+ "ALTER TABLE central_buffer_table_b ADD COLUMN
`place_before_node` VARCHAR(255) NULL FIRST;"
+ "drop table if exists central_buffer_table_a;"
+ "create table central_buffer_table_a as select `name`,
`transition_after_node` from central_buffer_table; "
+ "ALTER TABLE central_buffer_table_a CHANGE COLUMN
`name` `place_before_node` VARCHAR(255) NULL;"
+ "ALTER TABLE central_buffer_table_a CHANGE COLUMN
`transition_after_node` `name_primary` VARCHAR(255) NULL;"
+ "ALTER TABLE central_buffer_table_a ADD COLUMN
`place_after_node` VARCHAR(255) NULL;"
+ "drop table if exists central_buffer_table_a_b; "
+ "create table central_buffer_table_a_b select * from
central_buffer_table_b union all select * from central_buffer_table_a;";

```

```

String code22a = "drop table if exists
union_node_table; "
+ "CREATE TABLE union_node_table (`place_before_node`
VARCHAR(200) NULL, `place_after_node` VARCHAR(200) NULL) as SELECT id,
`xmi:type`, `xmi:id`, `name`, incoming, outgoing FROM union_node where
(union_node.`xmi:type`='uml:OpaqueAction') or
(union_node.`xmi:type`='uml:MergeNode') or
(union_node.`xmi:type`='uml:DecisionNode') or

```

```

(union_node.`xmi:type`='uml:ForkNode') or
(union_node.`xmi:type`='uml:JoinNode') or
(union_node.`xmi:type`='uml:AcceptEventAction') or
(union_node.`xmi:type`='uml:SendSignalAction') or
(union_node.`xmi:type`='uml:CallBehaviorAction');"
+ "ALTER TABLE union_node_table CHANGE COLUMN `xmi:id`
`xmi:id_primary` VARCHAR(255) NULL;"
+ "ALTER TABLE union_node_table CHANGE COLUMN `name`
`name_primary` VARCHAR(255) NULL;"
+ "drop table if exists union_node_node;"
+ "CREATE TABLE union_node_node AS SELECT m.*,
u2.`xmi:id`, u2.`name` FROM union_node_table m INNER JOIN edge_place_xmi
u2 ON (m.`incoming`= u2.`xmi:id`) or (m.`outgoing`= u2.`xmi:id`);"
+ "UPDATE union_node_table t1 INNER JOIN union_node_node
t2 ON t1.outgoing = t2.`xmi:id` SET t1.place_after_node = t2.name;"
+ "UPDATE union_node_table t1 INNER JOIN union_node_node
t2 ON t1.incoming = t2.`xmi:id` SET t1.place_before_node = t2.name;"
+ "delete from union_node_table WHERE (place_before_node
is null) and (place_after_node is null);"
+ "delete from edge_place_xmi WHERE `xmi:id` in (SELECT
DISTINCT `xmi:id` FROM `union_node_node`);"
+ "drop table if exists main_table;"
+ "CREATE TABLE main_table SELECT `place_before_node`,
`name_primary`, `place_after_node` FROM union_node_table union all select *
from initial_node_table union all select * from final_node_table union all
select * from handler_node_table_a_b union all select * from
activity_parameter_table_a_b union all select * from
central_buffer_table_a_b union all select * from datastore_table_a_b;"
+ "drop table if exists null_after;"
+ "CREATE TABLE null_after as SELECT * FROM main_table
where place_after_node is null and name_primary in (select name_primary
from main_table GROUP BY name_primary HAVING COUNT(*)>1);"
+ "drop table if exists null_before;"
+ "CREATE TABLE null_before as SELECT * FROM main_table
where place_before_node is null and name_primary in (select name_primary
from main_table GROUP BY name_primary HAVING COUNT(*)>1);"
+ "UPDATE null_after na, main_table mt SET
na.place_after_node = mt.place_after_node WHERE na.name_primary =
mt.name_primary and mt.place_after_node is not null and
mt.place_before_node is null;"
+ "UPDATE null_before na, main_table mt SET
na.place_before_node = mt.place_before_node WHERE na.name_primary =
mt.name_primary and mt.place_before_node is not null and
mt.place_after_node is null;"
+ "DELETE n1 FROM main_table n1 JOIN main_table n2 ON
n1.name_primary is null AND n1.place_before_node is null;"
+ "DELETE n1 FROM main_table n1 JOIN main_table n2 ON
n1.name_primary is null AND n1.place_after_node is null;"
+ "drop table if exists final_table;"
+ "CREATE TABLE final_table select * from null_after
union select * from null_before union select * from main_table;"
+ "drop table if exists finale_table;"
+ "create table finale_table as select * from
final_table;"
+ "DELETE n1 FROM finale_table n1 JOIN finale_table n2
ON n1.name_primary = n2.name_primary AND n1.place_before_node =
n2.place_before_node and n1.place_after_node is null;"

```

```

+ "DELETE n1 FROM finale_table n1 JOIN finale_table n2
ON n1.place_after_node = n2.place_after_node AND n1.place_before_node is
null AND n1.name_primary = n2.name_primary;"
+ "ALTER IGNORE TABLE `final_table` ADD UNIQUE
(place_before_node, place_after_node);"
+ "ALTER IGNORE TABLE `final_table` ADD UNIQUE
(place_after_node, name_primary);"
+ "ALTER IGNORE TABLE `final_table` ADD UNIQUE
(place_before_node, name_primary);"
+ "DELETE FROM final_table where place_after_node is not
null and place_before_node is not null;"

+ "drop table if exists place_tr_place1;"
+ "CREATE TABLE place_tr_place1 select * from
finale_table union all select * from final_table union all select * from
in_outputValue_final union all select * from expansionNode_final union all
select * from send_accept_final;"
+ "DELETE FROM place_tr_place1 where place_after_node is
null and place_before_node is null;"
+ "DELETE FROM place_tr_place1 where place_after_node is
null and name_primary is null;"
+ "DELETE FROM place_tr_place1 where name_primary is null
and place_before_node is null;"
+ "update place_tr_place1 as t1 inner join
place_tr_place1 as t2 on t1.name_primary = t2.name_primary and
t1.place_after_node <> t2.place_after_node and t2.place_before_node is null
and t1.place_after_node like 'place_%' set t1.place_after_node =
t2.place_after_node;"
+ "DELETE n1 FROM place_tr_place1 n1 JOIN
place_tr_place1 n2 ON n1.name_primary = n2.name_primary AND
n1.place_after_node = n2.place_after_node and n1.place_before_node is
null;"
+ "update place_tr_place1 as t1 inner join
place_tr_place1 as t2 on t1.name_primary = t2.name_primary and
t1.place_before_node <> t2.place_before_node and t2.place_after_node is null
and t1.place_after_node like 'place_%' set t1.place_before_node =
t2.place_before_node;"
+ "DELETE n1 FROM place_tr_place1 n1 JOIN
place_tr_place1 n2 ON n1.name_primary = n2.name_primary AND
n1.place_before_node = n2.place_before_node and n1.place_after_node is
null;"
+ "alter table place_tr_place1 add column id int
AUTO_INCREMENT primary key;"
+ "drop table if exists check_pan;"
+ "create table check_pan as select * FROM
place_tr_place1 p1 WHERE NOT EXISTS (SELECT place_before_node FROM
place_tr_place1 p2 WHERE p1.place_after_node = p2.place_before_node);"
+ "DELETE from check_pan where place_after_node like
'flow_pout%' or place_after_node like 'pout_%';"
+ "drop table if exists check_pbn;"
+ "create table check_pbn as select * FROM
place_tr_place1 p1 WHERE NOT EXISTS (SELECT place_before_node FROM
place_tr_place1 p2 WHERE p1.place_before_node = p2.place_after_node);"
+ "DELETE from check_pbn where place_before_node like
'pin%';"
+ "drop table if exists check_pan_pbn;"
+ "create table check_pan_pbn select * FROM check_pan
union select * from check_pbn;"

```



```

        + "DELETE n1 FROM place_tr_place1 n1 INNER JOIN
check_pan_pbn n2 ON n1.id = n2.id;"
        + "DELETE n1 FROM check_pan_pbn n1 INNER JOIN
expansionNode_xmi_for_delete n2 ON n1.place_before_node = n2.incoming and
n1.place_after_node = n2.outgoing and n1.name_primary = n2.name; "
        //+ "UPDATE check_pan_pbn SET place_after_node =
REPLACE(place_after_node, 'place', 'pout');"
        + "drop table if exists place_tr_place;"
        + "create table place_tr_place select * from
place_tr_place1 union all select * from check_pan_pbn;"
        + "UPDATE place_tr_place SET place_before_node
='initial_node_x' WHERE `place_before_node` is null; "
        + "UPDATE place_tr_place SET place_after_node
='final_node_x' WHERE `place_after_node` is null;"
        + "alter table place_tr_place drop column id;"
        + "alter table place_tr_place add column id int
AUTO_INCREMENT primary key;"
        + "UPDATE place_tr_place SET place_after_node = NULL
WHERE place_after_node like 'pout_%';"
        + "UPDATE place_tr_place SET place_after_node = 'pout'
WHERE place_after_node is null;"
        + "drop table negative_records;"
        + "create table negative_records ( id int not null
AUTO_INCREMENT PRIMARY KEY) as select distinct place_before_node,
name_primary from place_tr_place;"
        + "drop table positive_records;"
        + "create table positive_records ( id int not null
AUTO_INCREMENT PRIMARY KEY) as select distinct name_primary,
place_after_node from place_tr_place;";

```

```

        pst = conn1.prepareStatement(code1);
boolean isResult1 = pst.execute();
        pst = conn1.prepareStatement(code2);
boolean isResult2 = pst.execute();
        pst = conn1.prepareStatement(code3);
boolean isResult3 = pst.execute();
        pst = conn1.prepareStatement(code4);
boolean isResult4 = pst.execute();
        pst = conn1.prepareStatement(code5);
boolean isResult5 = pst.execute();
        pst = conn1.prepareStatement(code5a);
boolean isResult5a = pst.execute();
        pst = conn1.prepareStatement(code5b);
boolean isResult5b = pst.execute();
        pst = conn1.prepareStatement(code5c);
boolean isResult5c = pst.execute();
        pst = conn1.prepareStatement(code6);
boolean isResult6 = pst.execute();
        pst = conn1.prepareStatement(code7);
boolean isResult7 = pst.execute();
        pst = conn1.prepareStatement(code8);
boolean isResult8 = pst.execute();
        pst = conn1.prepareStatement(code9);
boolean isResult9 = pst.execute();
        pst = conn1.prepareStatement(code10);
boolean isResult10 = pst.execute();
        pst = conn1.prepareStatement(code11);
boolean isResult11 = pst.execute();

```



```

        pst = conn1.prepareStatement(code12);
boolean isResult12 = pst.execute();
        pst = conn1.prepareStatement(code13);
boolean isResult13 = pst.execute();
        pst = conn1.prepareStatement(code14);
boolean isResult14 = pst.execute();
        pst = conn1.prepareStatement(code15);
boolean isResult15 = pst.execute();
        pst = conn1.prepareStatement(code16);
boolean isResult16 = pst.execute();
        pst = conn1.prepareStatement(code17);
boolean isResult17 = pst.execute();
        pst = conn1.prepareStatement(code18);
boolean isResult18 = pst.execute();
        pst = conn1.prepareStatement(code19b);
boolean isResult19b = pst.execute();
        pst = conn1.prepareStatement(code19);
boolean isResult19 = pst.execute();
        pst = conn1.prepareStatement(code20);
        boolean isResult20 =
        pst.execute();
        boolean isResult21 =
        pst = conn1.prepareStatement(code21);
        pst.execute();
        pst = conn1.prepareStatement(code22a);
        boolean isResult22a
        = pst.execute();

String code111 = "drop table if exists negative;"
                + "SET group_concat_max_len=15000;"
                + "SELECT CONCAT('create table negative as
SELECT place_before_node,', GROUP_CONCAT(sums), 'FROM negative_records
GROUP BY id') FROM (SELECT distinct CONCAT('(case when
negative_records.name_primary = '', name_primary, '' then -1 else 0
end) as `', name_primary, ``')sums FROM negative_records GROUP BY id) s
INTO @sql;"

                + "PREPARE stmt FROM @sql;"
                + "EXECUTE stmt;"
                + "DEALLOCATE PREPARE stmt;"
        pst = conn1.prepareStatement(code111);
        boolean isResult111
        = pst.execute();

String code222 = "drop table if
exists positive;"
                + "SELECT CONCAT('create table
positive as SELECT place_after_node,', GROUP_CONCAT(sums), 'FROM
positive_records GROUP BY id') FROM (SELECT distinct CONCAT('(case when
positive_records.name_primary = '', name_primary, '' then 1 else 0 end)
as `', name_primary, ``')sums FROM positive_records GROUP BY id) s INTO
@sql;"

                + "PREPARE stmt FROM
@sql;"

                + "EXECUTE stmt;"
                + "DEALLOCATE
PREPARE stmt;"

                + "drop table if
exists overall;"

                + "create table
overall SELECT * FROM positive UNION SELECT * FROM negative;"
                + "drop table if
exists schema_table;"

```

```

schema_table as select * from overall;"
exists column_table; "
column_table (primary_id int NOT NULL AUTO_INCREMENT PRIMARY KEY) as select
column_name from information_schema.columns where table_name='overall';"
column_table where primary_id=1;"
exists matrix_pass_fail;"
exists incidence_matrix_single_device;"
exists table_union1";

pst = conn1.prepareStatement(code222);
boolean isResult222 = pst.execute();

String code334 = "set session sql_mode =
'NO_ENGINE_SUBSTITUTION';"
+ "DROP PROCEDURE IF EXISTS
`Te`;" + "SET group_concat_max_len= 150000;";
code334 += "CREATE PROCEDURE `Te`() ";
code334 += "BEGIN ";
code334 += "create table matrix_pass_fail (column_name varchar(150000)) ";
code334 += " SELECT
@query7:=GROUP_CONCAT(CONCAT('sum(`',column_name,`)`',column_name,`'`)) "
+ "AS column_name from
column_table order by CHAR_LENGTH(column_name); ";
code334 += "PREPARE stmt FROM @query7; ";
code334 += "EXECUTE stmt;";
code334 += "DEALLOCATE PREPARE stmt;";
code334 += "END ";

pst.execute(code334);
pst.close();

String query1 =
"Call Te()";

pst = conn1.prepareStatement(query1);
boolean isResultA = pst.execute();

}
catch(SQLException e){} try{ if (rs != null) rs.close(); if
(pst != null) pst.close();
if (conn1 != null) conn1.close();}catch(Exception e){}

try{
conn1

=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/sql",
"root","Xristina23");

pst =conn1.prepareStatement("select * from
matrix_pass_fail");
rs= pst.executeQuery();

while (rs.next())

```

```

        t1(("create table table_union1 as select place_after_node,") +
rs.getString("column_name")+ (" ") +("from overall group by
place_after_node;"));
        System.out.println(t1);
    }
    catch(SQLException e){} try{ if (rs != null) rs.close(); if (pst
!= null) pst.close();
        if (conn1 != null) conn1.close();}catch(Exception e){}
    try{
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");
        //STEP 3: Open a connection
        System.out.println("Connecting to a selected database...");
        conn1 = DriverManager.getConnection(DB_URL, USER, PASS);
        System.out.println("Connected database successfully...");
        //STEP 4: Execute a query
        System.out.println("Creating table in given database...");
        stmt = conn1.createStatement();
        String sql1 = t1;
        stmt.executeUpdate(sql1);

        System.out.println("Created table in given database...");
    }catch(SQLException se){
        //Handle errors for JDBC

        se.printStackTrace();
    }catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
    }finally{
        //finally block used to close resources
        try{
            if(stmt!=null)
                conn1.close();
        }catch(SQLException se){
        }// do nothing
        try{
            if(conn1!=null)
                conn1.close();
        }catch(SQLException se){
            se.printStackTrace();
        }//end finally try
        }//end try
    try {
        conn1 = DriverManager.getConnection(cs, USER, PASS);
        String t3=("create table incidence_matrix_single_device
as select * from table_union1 GROUP by place_after_node asc;");

        String query = "select * from
incidence_matrix_single_device;";
        pst = conn1.prepareStatement(t3);
        isResultt3 = pst.execute();
        pst = conn1.prepareStatement(query);
        isResult = pst.execute();
        do {
            rs = pst.getResultSet();
            ResultSetMetaData rsmd = rs.getMetaData();

```

```

        int columnsNumber = rsmd.getColumnCount();
        int col = rsmd.getColumnCount();
        for (int i = 1; i <= col; i++){
            String col_name = rsmd.getColumnName(i);
            System.out.print(col_name + "    ");
        }
        System.out.println(" ");
        // Iterate through the data in the result set and
display it.
        while (rs.next()) {
            //Print one row
            for(int i = 1 ; i <= columnsNumber; i++){
                System.out.print(rs.getString(i) + "
"); //Print one element of a row
            }
            System.out.println();
        }
        isResult = pst.getMoreResults();
    } while (isResult); } finally {
        if (rs != null) {
            rs.close(); }
        if (pst != null) {
            pst.close();}
        if (conn1 != null) {
            conn1.close();
        }
    }
    System.out.println("Goodbye!"); }
} //end main
//end JDBCExample

```

Appendix I – Advanced Generic SQL Code [M₀]

Appendix I presents the advanced generic SQL code developed for the automated generation of the PN initial marking matrix [M₀], discussed in Chapter 6.

```
package step1_initial_marking;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
public class initial_marking {
    public static void main(String[] args) throws SQLException {
        Connection con = null;
        PreparedStatement pst = null;
        ResultSet rs=null;
        String cs =
"jdbc:mysql://localhost:3306/sql?allowMultiQueries=true";
        String user = "root";
        String password = "Xristina23";
        try {
            con = DriverManager.getConnection(cs, user, password);
            String code1 = "SET SQL_SAFE_UPDATES=0;"
                + "drop table if exists initial_marking;"
                + "create table initial_marking (primary_id int not null
auto_increment primary key, activity varchar(250),
process_number_of_devices int);"
                + "insert into initial_marking (activity) select
place_after_node from incidence_matrix_single_device;"
                + "drop table if exists m0_marking;"
                + "create table m0_marking (primary_id int not null
auto_increment primary key) SELECT IFNULL(process_number_of_devices, 0)
FROM initial_marking;"
                + "ALTER TABLE `m0_marking` CHANGE COLUMN
`IFNULL(process_number_of_devices, 0)` `process_number_of_devices` int;"
                + "alter table initial_marking drop column
process_number_of_devices;"
                + "drop table if exists initial_marking_final;"
                + "CREATE TABLE initial_marking_final AS (SELECT
initial_marking.*, m0_marking.process_number_of_devices FROM
initial_marking INNER JOIN m0_marking ON initial_marking.primary_id =
m0_marking.primary_id);"
                + "UPDATE initial_marking_final SET
initial_marking_final.process_number_of_devices = "
                + "REPLACE(initial_marking_final.process_number_of_devices,
'0', '1') "
                + " where initial_marking_final.activity like 'pin%' or
initial_marking_final.activity like 'initial_place%' or
initial_marking_final.activity like 'initial_node_x%';"
                + "drop table if exists find_initial_node;"
                + "create table find_initial_node as select * from
place_tr_place;"
                + " DELETE n1 FROM find_initial_node n1 JOIN
find_initial_node n2 ON n1.place_before_node = n2.place_after_node;"
        }
```

```

        + "ALTER TABLE find_initial_node DROP column name_primary,
drop place_after_node, drop id;"
        + " ALTER TABLE find_initial_node ADD COLUMN `primary_id`
int NOT NULL AUTO_INCREMENT PRIMARY KEY; "
        + "ALTER TABLE find_initial_node  ADD COLUMN
`process_number_of_devices` INT;"
        + "ALTER TABLE find_initial_node CHANGE place_before_node
activity char(250);"
        + "UPDATE find_initial_node SET process_number_of_devices =
1 WHERE process_number_of_devices IS NULL;"
        + "ALTER TABLE find_initial_node MODIFY activity
varchar(250) AFTER primary_id;"
        + "drop table if exists initial_marking_final_1; "
        + "create table initial_marking_final_1 select * from
find_initial_node UNION select * from initial_marking_final;"
        + "DELETE n1 FROM initial_marking_final_1 n1 JOIN
find_initial_node n2 ON n1.activity=n2.activity AND
n1.process_number_of_devices=0;"
        + "ALTER TABLE initial_marking_final_1 DROP column
primary_id;"
        + "drop table if exists process_device_number;"
        + "CREATE TABLE process_device_number (number_of_devices
int, activity varchar(50), min_time int, max_time int, probability_pass
double, probability_fail double, min_interval_pass int, max_interval_pass
int, min_interval_fail int, max_interval_fail int, initial_marking int);";
        String code2 = "LOAD DATA LOCAL INFILE
'E:/online_shopping/excel_data_number_of_devices.csv' INTO TABLE
process_device_number FIELDS TERMINATED BY ';' LINES TERMINATED BY '\r\n'
(number_of_devices, activity, min_time, max_time, probability_pass,
probability_fail, min_interval_pass, max_interval_pass, min_interval_fail,
max_interval_fail, initial_marking);";
        String code3 = "ALTER TABLE process_device_number ADD primary_id
int NOT NULL AUTO_INCREMENT PRIMARY KEY ;"
        + "DELETE FROM process_device_number where primary_id=1;"
        + "drop table if exists initial_marking_table;"
        + "create table initial_marking_table (primary_id int NOT
NULL AUTO_INCREMENT PRIMARY KEY) as select initial_marking from
process_device_number;";
        String code4 = "UPDATE initial_marking_final, initial_marking_table SET
initial_marking_final.process_number_of_devices =
REPLACE(initial_marking_final.process_number_of_devices,'1',
initial_marking_table.initial_marking) where
initial_marking_table.primary_id='1';"
        + "ALTER IGNORE TABLE initial_marking_final_1 ADD UNIQUE (activity,
process_number_of_devices);";

        pst = con.prepareStatement(code1);                boolean isResult1 =
        pst.execute();
        pst = con.prepareStatement(code2);                boolean isResul2t =
        pst.execute();
        pst = con.prepareStatement(code3);                boolean isResult3 =
        pst.execute();
        pst = con.prepareStatement(code4);                boolean isResult4 =
        pst.execute();

        String query = "select * from initial_marking_final;";
        pst = con.prepareStatement(query);
        boolean isResult = pst.execute();
        do {

```

```

        rs = pst.getResultSet();
        ResultSetMetaData rsmd = rs.getMetaData();
        int columnsNumber = rsmd.getColumnCount();
        int col = rsmd.getColumnCount();
        for (int i = 1; i <= col; i++){
            String col_name = rsmd.getColumnName(i);
            System.out.print(col_name + " ");
        }
        System.out.println(" ");
        // Iterate through the data in the result set and display
it.
        while (rs.next()) {
            //Print one row
            for(int i = 1 ; i <= columnsNumber; i++){
                System.out.print(rs.getString(i) + " ");
            //Print one element of a row
            }
            System.out.println();
        }
        isResult = pst.getMoreResults();
    }
    while (isResult);
    } finally {
        if (rs != null) {
            rs.close();
        }
        if (pst != null) {
            pst.close();
        }
        if (con != null) {
            con.close();
        }
    }
}
}

```

Appendix J

Appendix J covers the XMI files for the two AD examples shown in Chapter 7.

Part A- Production System Example (XMI File)

XMI obtained from the AD for the Production System, shown in Figure 7.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
xmi:id="__mtawCH2Eeilppsn_eFPfg" name="RootElement">
  <packagedElement xmi:type="uml:Activity" xmi:id="__tTyACH2Eeilppsn_eFPfg"
name="Activity1" node="__tcQnECH3Eeilppsn_eFPfg _vGJWACH3Eeilppsn_eFPfg
_25ogcCH3Eeilppsn_eFPfg _95hqwCH3Eeilppsn_eFPfg _FaRfYCH4Eeilppsn_eFPfg
_So3AkCH4Eeilppsn_eFPfg _bRFaoCH4Eeilppsn_eFPfg _cdQS0CH4Eeilppsn_eFPfg
_tXuP4CH4Eeilppsn_eFPfg _-95cYCH4Eeilppsn_eFPfg _F39rgCH5Eeilppsn_eFPfg
_G82P8CH5Eeilppsn_eFPfg _JucN0CH5Eeilppsn_eFPfg _Qy02kCH5Eeilppsn_eFPfg
_WQZ5kCH5Eeilppsn_eFPfg _YwFBACH5Eeilppsn_eFPfg _tP9lkCH5Eeilppsn_eFPfg
_ufdekCH5Eeilppsn_eFPfg _yrbvwCH5Eeilppsn_eFPfg _AL2n0CH6Eeilppsn_eFPfg
_C619oCH6Eeilppsn_eFPfg _GpmuYCH6Eeilppsn_eFPfg _JrgZICH6Eeilppsn_eFPfg
_ND_X4CH6Eeilppsn_eFPfg _T4TkUCH6Eeilppsn_eFPfg _VvAnQCH6Eeilppsn_eFPfg
_WuXygCH6Eeilppsn_eFPfg _fZyqgCH6Eeilppsn_eFPfg _gUwd8CH6Eeilppsn_eFPfg
_7YpCYCH3Eeilppsn_eFPfg _Gr_9QCIJEeilppsn_eFPfg _ZOymMCINEeilppsn_eFPfg
_sqt-UCJYEeimD_ix8LXVAA">
    <edge xmi:type="uml:ControlFlow" xmi:id="_uzeU0CH7Eeilppsn_eFPfg"
target="__vGJWACH3Eeilppsn_eFPfg" source="__tcQnECH3Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_wBAe8CH7Eeilppsn_eFPfg"
target="__25ogcCH3Eeilppsn_eFPfg" source="__vGJWACH3Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_xTTNQCH7Eeilppsn_eFPfg"
target="__25ogcCH3Eeilppsn_eFPfg" source="__95hqwCH3Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_ymKjUCH7Eeilppsn_eFPfg"
target="__7YpCYCH3Eeilppsn_eFPfg" source="__25ogcCH3Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_zYyTECH7Eeilppsn_eFPfg"
target="__95hqwCH3Eeilppsn_eFPfg" source="__7YpCYCH3Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_0guLgCH7Eeilppsn_eFPfg"
target="__Gr_9QCIJEeilppsn_eFPfg" source="__7YpCYCH3Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_8-XtQCH7Eeilppsn_eFPfg"
target="__bRFaoCH4Eeilppsn_eFPfg" source="__So3AkCH4Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_9stZ8CH7Eeilppsn_eFPfg"
target="__cdQS0CH4Eeilppsn_eFPfg" source="__bRFaoCH4Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_-UqKwCH7Eeilppsn_eFPfg"
target="__tXuP4CH4Eeilppsn_eFPfg" source="__bRFaoCH4Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_cEHsCH8Eeilppsn_eFPfg"
target="__-95cYCH4Eeilppsn_eFPfg" source="__cdQS0CH4Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_IQ-p8CH8Eeilppsn_eFPfg"
target="__F39rgCH5Eeilppsn_eFPfg" source="__tXuP4CH4Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_JC52ICH8Eeilppsn_eFPfg"
target="__G82P8CH5Eeilppsn_eFPfg" source="__F39rgCH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_J3yPsCH8Eeilppsn_eFPfg"
target="__JucN0CH5Eeilppsn_eFPfg" source="__F39rgCH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_MT_zcCH8Eeilppsn_eFPfg"
name="Filling_with_base" target="__Qy02kCH5Eeilppsn_eFPfg"
source="__G82P8CH5Eeilppsn_eFPfg"/>
```



```

    <edge xmi:type="uml:ControlFlow" xmi:id="_PdIuoCH8Eeilppsn_eFPfg"
name="Filling_with_additive" target="_WQZ5kCH5Eeilppsn_eFPfg"
source="_JucN0CH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_SUxDECH8Eeilppsn_eFPfg"
target="_YwFBACH5Eeilppsn_eFPfg" source="_WQZ5kCH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_TctHICH8Eeilppsn_eFPfg"
target="_tP9lkCH5Eeilppsn_eFPfg" source="_YwFBACH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_UpSDYCH8Eeilppsn_eFPfg"
target="_ufdekCH5Eeilppsn_eFPfg" source="_tP9lkCH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_VWThcCH8Eeilppsn_eFPfg"
target="_ufdekCH5Eeilppsn_eFPfg" source="_Qy02kCH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_WBvEECH8Eeilppsn_eFPfg"
target="_yrbvwCH5Eeilppsn_eFPfg" source="_tP9lkCH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_mS6zkCH8Eeilppsn_eFPfg"
target="_AL2n0CH6Eeilppsn_eFPfg" source="_ufdekCH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_nHLKsCH8Eeilppsn_eFPfg"
name="Mixing_in_M1" target="_C619oCH6Eeilppsn_eFPfg"
source="_AL2n0CH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_rRdxMCH8Eeilppsn_eFPfg"
name="Emptying_M1" target="_GpmuYCH6Eeilppsn_eFPfg"
source="_C619oCH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_u3F0ECH8Eeilppsn_eFPfg"
target="_JrgZICH6Eeilppsn_eFPfg" source="_GpmuYCH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_zKXAECCH8Eeilppsn_eFPfg"
target="_T4TkUCH6Eeilppsn_eFPfg" source="_JrgZICH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_0eAZQCH8Eeilppsn_eFPfg"
target="_ND_X4CH6Eeilppsn_eFPfg" source="_T4TkUCH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_BExFcCH9Eeilppsn_eFPfg"
target="_VvAnQCH6Eeilppsn_eFPfg" source="_yrbvwCH5Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_CTKyQCH9Eeilppsn_eFPfg"
target="_WuXygCH6Eeilppsn_eFPfg" source="_VvAnQCH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_IwmIsCH9Eeilppsn_eFPfg"
target="_fZyqgCH6Eeilppsn_eFPfg" source="_WuXygCH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_Jf768CH9Eeilppsn_eFPfg"
target="_gUwd8CH6Eeilppsn_eFPfg" source="_fZyqgCH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_Ms1UwCH9Eeilppsn_eFPfg"
target="_T4TkUCH6Eeilppsn_eFPfg" source="_fZyqgCH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_WrgkYCH9Eeilppsn_eFPfg"
target="_sqt-UCJYEeimD_iX8LXVAA" source="_ND_X4CH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_zHTPkCH9Eeilppsn_eFPfg"
name="Valve_open" target="_VvAnQCH6Eeilppsn_eFPfg" source="_-
95cYCH4Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_L_fkcCIAEeilppsn_eFPfg"
target="_sqt-UCJYEeimD_iX8LXVAA" source="_7YpCYCH3Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_LS8TMCIEeilppsn_eFPfg"
target="_FaRfYCH4Eeilppsn_eFPfg" source="_Gr_9QCIJEeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_az7YACINEeilppsn_eFPfg"
target="_ZOymMCINEeilppsn_eFPfg" source="_gUwd8CH6Eeilppsn_eFPfg"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_2mHQcCJYEeimD_iX8LXVAA"
target="_So3AkCH4Eeilppsn_eFPfg" source="_sqt-UCJYEeimD_iX8LXVAA"/>
    <node xmi:type="uml:InitialNode" xmi:id="_tcQnECH3Eeilppsn_eFPfg"
name="" outgoing="_uzeU0CH7Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_vGJWACH3Eeilppsn_eFPfg"
name="Production_Order_1_asks_Receipt_P1_to_produce_P1"
incoming="_uzeU0CH7Eeilppsn_eFPfg" outgoing="_wBAe8CH7Eeilppsn_eFPfg"/>
    <node xmi:type="uml:MergeNode" xmi:id="_25ogcCH3Eeilppsn_eFPfg"
name="M1" incoming="_wBAe8CH7Eeilppsn_eFPfg" xTTNQCH7Eeilppsn_eFPfg"
outgoing="_ymKjUCH7Eeilppsn_eFPfg"/>

```

```

    <node xmi:type="uml:DecisionNode" xmi:id="_7YpCYCH3Eeilppsn_eFPfg"
name="D1" incoming="_ymKjUCH7Eeilppsn_eFPfg"
outgoing="_zYyTECH7Eeilppsn_eFPfg _0guLgCH7Eeilppsn_eFPfg
_L_fkCIAEeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_95hqwCH3Eeilppsn_eFPfg"
name="Production_of_a_batch_of_P1_in_M2(similar_to_M1)"
incoming="_zYyTECH7Eeilppsn_eFPfg" outgoing="_xTTNQCH7Eeilppsn_eFPfg"/>
    <node xmi:type="uml:ActivityFinalNode" xmi:id="_FaRfYCH4Eeilppsn_eFPfg"
incoming="_LS8TMCIJ7Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_So3AkCH4Eeilppsn_eFPfg"
name="Receipt_P1_requests_M1_to_Interface_CM1"
incoming="_2mHQcCJYEimD_iX8LXVAA" outgoing="_8-XtQCH7Eeilppsn_eFPfg"/>
    <node xmi:type="uml:ForkNode" xmi:id="_bRFaoCH4Eeilppsn_eFPfg" name=""
incoming="_8-XtQCH7Eeilppsn_eFPfg" outgoing="_9stZ8CH7Eeilppsn_eFPfg _-
UqKwCH7Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_cdQS0CH4Eeilppsn_eFPfg"
name="Receipt_P1_asks_Interface_VT2--1_to_open_VT2-1"
incoming="_9stZ8CH7Eeilppsn_eFPfg" outgoing="_CEhHsCH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_tXuP4CH4Eeilppsn_eFPfg"
name="Interface_CM1_requests_M1_to_controller_CM1" incoming="_-
UqKwCH7Eeilppsn_eFPfg" outgoing="_IQ-p8CH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_-95cYCH4Eeilppsn_eFPfg"
name="Interface_VT2-1_opens_VT2-1" incoming="_CEhHsCH8Eeilppsn_eFPfg"
outgoing="_zHTPkCH9Eeilppsn_eFPfg"/>
    <node xmi:type="uml:ForkNode" xmi:id="_F39rgCH5Eeilppsn_eFPfg" name=""
incoming="_IQ-p8CH8Eeilppsn_eFPfg" outgoing="_JC52ICH8Eeilppsn_eFPfg
_J3yPsCH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_G82P8CH5Eeilppsn_eFPfg"
name="CM1_opens_VM1-1" incoming="_JC52ICH8Eeilppsn_eFPfg"
outgoing="_MT_zcCH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_JucN0CH5Eeilppsn_eFPfg"
name="CM1_opens_VM1-2" incoming="_J3yPsCH8Eeilppsn_eFPfg"
outgoing="_PdIuoCH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_Qy02kCH5Eeilppsn_eFPfg"
name="CM1_closes_VM1-1" incoming="_MT_zcCH8Eeilppsn_eFPfg"
outgoing="_VWThcCH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_WQZ5kCH5Eeilppsn_eFPfg"
name="CM1_closes_VM1-2" incoming="_PdIuoCH8Eeilppsn_eFPfg"
outgoing="_SUxDECH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_YwFBACH5Eeilppsn_eFPfg"
name="CM1_informs_end_of_additive_loading_to_Interface_CM1"
incoming="_SUxDECH8Eeilppsn_eFPfg" outgoing="_TCThICH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:ForkNode" xmi:id="_tP9LkCH5Eeilppsn_eFPfg" name=""
incoming="_TCThICH8Eeilppsn_eFPfg" outgoing="_UpSDYCH8Eeilppsn_eFPfg
_WBvEECH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:JoinNode" xmi:id="_ufdekCH5Eeilppsn_eFPfg"
incoming="_UpSDYCH8Eeilppsn_eFPfg _VWThcCH8Eeilppsn_eFPfg"
outgoing="_mS6zkCH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_yrbvwCH5Eeilppsn_eFPfg"
name="Interface_CM1_informs_end_of_additive_loading_to_Receipt_P1"
incoming="_WBvEECH8Eeilppsn_eFPfg" outgoing="_BExFcCH9Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_AL2n0CH6Eeilppsn_eFPfg"
name="CM1_starts_mixing_in_M1" incoming="_mS6zkCH8Eeilppsn_eFPfg"
outgoing="_nHLKsCH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_C619oCH6Eeilppsn_eFPfg"
name="CM1_stops_mixing_and_starts_emptying_M1"
incoming="_nHLKsCH8Eeilppsn_eFPfg" outgoing="_rRdxMCH8Eeilppsn_eFPfg"/>

```

```

    <node xmi:type="uml:OpaqueAction" xmi:id="_GpmuYCH6Eeilppsn_eFPfg"
name="CM1_detects_M1_empty" incoming="_rRdxMCH8Eeilppsn_eFPfg"
outgoing="_u3F0ECH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_JrgZICH6Eeilppsn_eFPfg"
name="CM1_informs_end_of_batch_to_Interface_CM1"
incoming="_u3F0ECH8Eeilppsn_eFPfg" outgoing="_zKXAECH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_ND_X4CH6Eeilppsn_eFPfg"
name="Interface_CM1_informs_end_of_batch_to_Receipt_P1"
incoming="_0eAZQCH8Eeilppsn_eFPfg" outgoing="_WrgkYCH9Eeilppsn_eFPfg"/>
    <node xmi:type="uml:JoinNode" xmi:id="_T4TkUCH6Eeilppsn_eFPfg" name=""
incoming="_zKXAECH8Eeilppsn_eFPfg _Ms1UwCH9Eeilppsn_eFPfg"
outgoing="_0eAZQCH8Eeilppsn_eFPfg"/>
    <node xmi:type="uml:JoinNode" xmi:id="_VvAnQCH6Eeilppsn_eFPfg" name=""
incoming="_BExFcCH9Eeilppsn_eFPfg _zHTPkCH9Eeilppsn_eFPfg"
outgoing="_CTKyQCH9Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_WuXygCH6Eeilppsn_eFPfg"
name="Receipt_P1_requests_Interface_VT2--1_to_close_VT2-1"
incoming="_CTKyQCH9Eeilppsn_eFPfg" outgoing="_IwmIsCH9Eeilppsn_eFPfg"/>
    <node xmi:type="uml:ForkNode" xmi:id="_fZyqgCH6Eeilppsn_eFPfg" name=""
incoming="_IwmIsCH9Eeilppsn_eFPfg" outgoing="_Jf768CH9Eeilppsn_eFPfg
_Ms1UwCH9Eeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_gUwd8CH6Eeilppsn_eFPfg"
name="Interface_VT2_closes_VT2" incoming="_Jf768CH9Eeilppsn_eFPfg"
outgoing="_az7YACINEeilppsn_eFPfg"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_Gr_9QCIJEeilppsn_eFPfg"
name="Terminate_production_P1" incoming="_0guLgCH7Eeilppsn_eFPfg"
outgoing="_LS8TMCIEeilppsn_eFPfg"/>
    <node xmi:type="uml:FlowFinalNode" xmi:id="_ZOymMCINEeilppsn_eFPfg"
name="" incoming="_az7YACINEeilppsn_eFPfg"/>
    <node xmi:type="uml:MergeNode" xmi:id="_sqt-UCJYEeimD_ix8LXVAA"
incoming="_L_fkCIEeilppsn_eFPfg _WrgkYCH9Eeilppsn_eFPfg"
outgoing="_2mHQcCJYEeimD_ix8LXVAA"/>
  </packagedElement>
</uml:Model>

```

Part B – Online Shopping Process (XMI File)

XMI obtained from the AD for the online shopping process, illustrated in Figure 7.4.

```

<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
xmi:id="__QzS0JR1EeebX0bshy6vLA" name="RootElement">
  <packagedElement xmi:type="uml:Activity" xmi:id="__h8GIJR1EeebX0bshy6vLA"
name="Activity1" node="_M-oUwJR2EeebX0bshy6vLA _ZigFAJR2EeebX0bshy6vLA
_izGGoJR2EeebX0bshy6vLA _kVjX4JR2EeebX0bshy6vLA _mIuLsJR2EeebX0bshy6vLA
_ogUvMJR2EeebX0bshy6vLA _qBiqJR2EeebX0bshy6vLA _uPNkoJR2EeebX0bshy6vLA
_xFJ3AJR2EeebX0bshy6vLA _5F6BYJR2EeebX0bshy6vLA _BYaDEJR3EeebX0bshy6vLA
_QZxwgJR3EeebX0bshy6vLA _Rr-_QJR3EeebX0bshy6vLA _Xq6b0JR3EeebX0bshy6vLA
_edbRcJR3EeebX0bshy6vLA __wTroJR6EeebX0bshy6vLA _NbgrwJR7EeebX0bshy6vLA
_aYOZ0JR7EeebX0bshy6vLA _ifxIsJR7EeebX0bshy6vLA _j6mvAJR7EeebX0bshy6vLA
_LNZMkJR7EeebX0bshy6vLA _pyfIYJR7EeebX0bshy6vLA _AI0bUJR8EeebX0bshy6vLA
_NomQEJR8EeebX0bshy6vLA _004CUJR8EeebX0bshy6vLA _7TENwJR8EeebX0bshy6vLA
_8nxw4JR8EeebX0bshy6vLA _PHurEJR9EeebX0bshy6vLA _rq_3QJR9EeebX0bshy6vLA _-
VKQUJR9EeebX0bshy6vLA _PT9TwJR-EeebX0bshy6vLA _aCiWUJR-EeebX0bshy6vLA

```

```

_0kXPMJR_EeebXObshy6vLA _DZXUkJR_EeebXObshy6vLA _DquxYJR_EeebXObshy6vLA
_ESTHsJR_EeebXObshy6vLA _YXy2UJR_EeebXObshy6vLA _J0hUoJSAEeebXObshy6vLA
_5kI0gJSBEeebXObshy6vLA _BpfyEJSCeebXObshy6vLA _DOQwcJSCeebXObshy6vLA
_D1IjMJSCeebXObshy6vLA _OxpUEJSCeebXObshy6vLA _UJ2u4JSCeebXObshy6vLA
_7TV_kJknEeeOIORO-VyZIA _S0_E8JkoEeeOIORO-VyZIA _gEeLEJkoEeeOIORO-VyZIA
_zb-c4JunEeeMJso6T-JfIQ _ZLMygJ7uEeel-svp6Z91RQ _LcXMsJ7vEeel-svp6Z91RQ"
group="_YXy2UJR_EeebXObshy6vLA">
    <ownedBehavior xmi:type="uml:Activity" xmi:id="_e76kQJR3EeebXObshy6vLA"
name="Activity1">
        <nestedClassifier xmi:type="uml:Signal" xmi:id="_VyjEcJuoEeeMJso6T-
JfIQ"/>
        <nestedClassifier xmi:type="uml:Signal" xmi:id="_uaqpsJuoEeeMJso6T-
JfIQ" name="Verify_CC_Funds"/>
    </ownedBehavior>
    <edge xmi:type="uml:ControlFlow" xmi:id="_vk2scJR3EeebXObshy6vLA"
target="_ZigFAJR2EeebXObshy6vLA" source="_M-oUwJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_wxh68JR3EeebXObshy6vLA"
target="_gEeLEJkoEeeOIORO-VyZIA" source="_ZigFAJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_xoAP8JR3EeebXObshy6vLA"
target="_kVjX4JR2EeebXObshy6vLA" source="_izGGoJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_y2k74JR3EeebXObshy6vLA"
target="_mIuLSJR2EeebXObshy6vLA" source="_izGGoJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_zrTkcJR3EeebXObshy6vLA"
target="_ogUvMJR2EeebXObshy6vLA" source="_kVjX4JR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_0qqvsJR3EeebXObshy6vLA"
target="_ogUvMJR2EeebXObshy6vLA" source="_mIuLSJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_1s9TIJR3EeebXObshy6vLA"
target="_uPNkoJR2EeebXObshy6vLA" source="_LcXMsJ7vEeel-svp6Z91RQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_7tsGgJR3EeebXObshy6vLA"
target="_BYaDEJR3EeebXObshy6vLA" source="_5F6BYJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_EnCFwJR4EeebXObshy6vLA"
name="Authorised" target="_QZxwgJR3EeebXObshy6vLA"
source="_BYaDEJR3EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_Gi3gUJR4EeebXObshy6vLA"
name="Not_Authorised" target="_Rr-QJR3EeebXObshy6vLA"
source="_BYaDEJR3EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_Ltr_YJR4EeebXObshy6vLA"
target="_Xq6b0JR3EeebXObshy6vLA" source="_QZxwgJR3EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_M-guEJR4EeebXObshy6vLA"
target="_Xq6b0JR3EeebXObshy6vLA" source="_Rr-QJR3EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_N5JKUJR4EeebXObshy6vLA"
target="_edbRcJR3EeebXObshy6vLA" source="_Xq6b0JR3EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_QEZZkJR7EeebXObshy6vLA"
name="Acount_Information_(pending)" target="_WTroJR6EeebXObshy6vLA"
source="_ZLMygJ7uEeel-svp6Z91RQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_VcBlkJR7EeebXObshy6vLA"
target="_NbgrwJR7EeebXObshy6vLA" source="_WTroJR6EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_cAnT8JR7EeebXObshy6vLA"
name="Verify_Account_by_Phone" target="_aY0Z0JR7EeebXObshy6vLA"
source="_WTroJR6EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_q540QJR7EeebXObshy6vLA"
target="_ifxIsJR7EeebXObshy6vLA" source="_aY0Z0JR7EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_rrpCYJR7EeebXObshy6vLA"
target="_j6mvAJR7EeebXObshy6vLA" source="_aY0Z0JR7EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_sorSYJR7EeebXObshy6vLA"
target="_LNZMkJR7EeebXObshy6vLA" source="_aY0Z0JR7EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_tVswcJR7EeebXObshy6vLA"
target="_pyfIYJR7EeebXObshy6vLA" source="_ifxIsJR7EeebXObshy6vLA"/>

```



```

    <edge xmi:type="uml:ControlFlow" xmi:id="_uU3HYJR7EeebXObsHy6vLA"
target="_pyfIYJR7EeebXObsHy6vLA" source="_j6mVAJR7EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_vH6U8JR7EeebXObsHy6vLA"
target="_pyfIYJR7EeebXObsHy6vLA" source="_LNZMkJR7EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_Ozc5oJR8EeebXObsHy6vLA"
target="_NomQEJR8EeebXObsHy6vLA" source="_AIObUJR8EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="__uP_oJR8EeebXObsHy6vLA"
target="_004CUJR8EeebXObsHy6vLA" source="_pyfIYJR7EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_BV4EwJR9EeebXObsHy6vLA"
name="Request_More_Information" target="_7TENwJR8EeebXObsHy6vLA"
source="_004CUJR8EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_JT9kJR9EeebXObsHy6vLA"
target="_8nxW4JR8EeebXObsHy6vLA" source="_004CUJR8EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_xAm0kJR-EeebXObsHy6vLA"
target="_-VKQUJR9EeebXObsHy6vLA" source="_rq_3QJR9EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_3PXc4JR-EeebXObsHy6vLA"
target="_0kXPMJR-EeebXObsHy6vLA" source="_-VKQUJR9EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_q2F9oJR-EeebXObsHy6vLA"
target="_c_FxkJR-EeebXObsHy6vLA" source="_aPQHMR-EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_rrKkcJR-EeebXObsHy6vLA"
target="_baUzMJR-EeebXObsHy6vLA" source="_c_FxkJR-EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_S8Dw8JSAEeebXObsHy6vLA"
target="_J0hUoJSAEeebXObsHy6vLA" source="_8nxW4JR8EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_UJILkJSBEeebXObsHy6vLA"
target="_NomQEJR8EeebXObsHy6vLA" source="_J0hUoJSAEeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_ZxRDUJSEeebXObsHy6vLA"
target="_BpfyEJSEeebXObsHy6vLA" source="_wJ6D0JSEeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_btyaYJSEeebXObsHy6vLA"
name="Monitor" target="_JdkyYJSEeebXObsHy6vLA"
source="_BpfyEJSEeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_fP6dUJSEeebXObsHy6vLA"
name="Computer" target="_IswZcJSEeebXObsHy6vLA"
source="_BpfyEJSEeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_2ciU8JSEeebXObsHy6vLA"
target="_7TV_kJknEeeOIORO-VyZIA" source="_K_IrwJSEeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="__U0kQJSEeebXObsHy6vLA"
target="_UJ2u4JSEeebXObsHy6vLA" source="_OxpUEJSEeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_Uwro4JSEeebXObsHy6vLA"
target="_NomQEJR8EeebXObsHy6vLA" source="_UJ2u4JSEeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_9x2pYJknEeeOIORO-VyZIA"
target="_OxpUEJSEeebXObsHy6vLA" source="_7TV_kJknEeeOIORO-VyZIA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_bDae8JkoEeeOIORO-VyZIA"
target="_S0_E8JkoEeeOIORO-VyZIA" source="_M-oUwJR2EeebXObsHy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_hj1fAJkoEeeOIORO-VyZIA"
target="_gEeLEJkoEeeOIORO-VyZIA" source="_S0_E8JkoEeeOIORO-VyZIA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_7aWKAJkoEeeOIORO-VyZIA"
target="_izGGoJR2EeebXObsHy6vLA" source="_gEeLEJkoEeeOIORO-VyZIA"/>
    <edge xmi:type="uml:ObjectFlow" xmi:id="_34ePoJuiEeeMJso6T-JfIQ"
target="_vSm7IJukEeeMJso6T-JfIQ" source="_Ti85MJR9EeebXObsHy6vLA">
    <guard xmi:type="uml:LiteralBoolean" xmi:id="_35aq0JuiEeeMJso6T-JfIQ"
value="true"/>
    <weight xmi:type="uml:LiteralInteger" xmi:id="_35aq0ZuiEeeMJso6T-
JfIQ" value="1"/>
  </edge>
  <edge xmi:type="uml:ControlFlow" xmi:id="_k1_AsJujEeeMJso6T-JfIQ"
target="_aCiWUJR-EeebXObsHy6vLA" source="_PT9TwJR-EeebXObsHy6vLA"/>
  <edge xmi:type="uml:ControlFlow" xmi:id="_FiBNuJukEeeMJso6T-JfIQ"
target="_7TV_kJknEeeOIORO-VyZIA" source="_MM4RQJSEeebXObsHy6vLA"/>

```

```

    <edge xmi:type="uml:ControlFlow" xmi:id="_pzIasJukEeeMJso6T-JfIQ"
target="_PHurEJR9EeebXObshy6vLA" source="_004CUJR8EeebXObshy6vLA"/>
    <edge xmi:type="uml:ObjectFlow" xmi:id="_TTcPoJumEeeMJso6T-JfIQ"
name="Order_Items" target="_9mQmIJSBEeebXObshy6vLA"
source="_KtnewJulEeeMJso6T-JfIQ">
    <guard xmi:type="uml:LiteralBoolean" xmi:id="_TUQvAJumEeeMJso6T-JfIQ"
value="true"/>
    <weight xmi:type="uml:LiteralInteger" xmi:id="_TUQvAZumEeeMJso6T-
JfIQ" value="1"/>
</edge>
    <edge xmi:type="uml:ControlFlow" xmi:id="_b_bdcJuoEeeMJso6T-JfIQ"
target="_wEoRIJuoEeeMJso6T-JfIQ" source="_-VKQJR9EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_DAIegJ7uEeel-svp6Z91RQ"
target="_5F6BYJR2EeebXObshy6vLA" source="_LcXMsJ7vEeel-svp6Z91RQ"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_LId3kJ7uEeel-svp6Z91RQ"
target="_xFJ3AJR2EeebXObshy6vLA" source="_uPNkoJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_L8ShMJ7uEeel-svp6Z91RQ"
target="_qBiqUJR2EeebXObshy6vLA" source="_xFJ3AJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_aoYiMJ7uEeel-svp6Z91RQ"
target="_ZLMygJ7uEeel-svp6Z91RQ" source="_qBiqUJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_g0YxcJ7uEeel-svp6Z91RQ"
target="_ZLMygJ7uEeel-svp6Z91RQ" source="_edbRcJR3EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_nhGroJ7vEeel-svp6Z91RQ"
target="_LcXMsJ7vEeel-svp6Z91RQ" source="_ogUvMJR2EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_kq7BwKg8Eeez4cgOfIXL4g"
target="_NomQEJR8EeebXObshy6vLA" source="_baUzMJR_EeebXObshy6vLA"/>
    <edge xmi:type="uml:ControlFlow" xmi:id="_LyGsQKg8Eeez4cgOfIXL4g"
target="_aPQHMJR_EeebXObshy6vLA" source="_0kXPMJR-EeebXObshy6vLA"/>
    <structuredNode xmi:type="uml:ExpansionRegion"
xmi:id="_YXy2UJR_EeebXObshy6vLA" name="ExpansionRegion1" mustIsolate="true"
mode="parallel" outputElement="_baUzMJR_EeebXObshy6vLA"
inputElement="_aPQHMJR_EeebXObshy6vLA">
    <node xmi:type="uml:ExpansionNode" xmi:id="_aPQHMJR_EeebXObshy6vLA"
name="ExpansionNode1" incoming="_LyGsQKg8Eeez4cgOfIXL4g"
outgoing="_q2F9oJR_EeebXObshy6vLA" isControlType="true"
regionAsInput="_YXy2UJR_EeebXObshy6vLA"
regionAsOutput="_afnHoKkpEeeSpPj0NkCV_Q">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_aPRVUJR_EeebXObshy6vLA" value="1"/>
    </node>
    <node xmi:type="uml:ExpansionNode" xmi:id="_baUzMJR_EeebXObshy6vLA"
name="ExpansionNode2" incoming="_rrKkcJR_EeebXObshy6vLA"
outgoing="_kq7BwKg8Eeez4cgOfIXL4g" isControlType="true"
regionAsOutput="_YXy2UJR_EeebXObshy6vLA">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_baVaQJR_EeebXObshy6vLA" value="1"/>
    </node>
    <node xmi:type="uml:OpaqueAction" xmi:id="_c_FxkJR_EeebXObshy6vLA"
name="Contact_Suppliers" incoming="_q2F9oJR_EeebXObshy6vLA"
outgoing="_rrKkcJR_EeebXObshy6vLA"/>
    <node xmi:type="uml:ExpansionRegion" xmi:id="_afnHoKkpEeeSpPj0NkCV_Q"
outputElement="_aPQHMJR_EeebXObshy6vLA"/>
</structuredNode>
    <node xmi:type="uml:InitialNode" xmi:id="_M-oUwJR2EeebXObshy6vLA"
outgoing="_vk2scJR3EeebXObshy6vLA _bDae8JkoEeeOIORO-VyZIA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_ZigFAJR2EeebXObshy6vLA"
name="Ask_to_Create_Account_for_New_Users"
incoming="_vk2scJR3EeebXObshy6vLA" outgoing="_wxh68JR3EeebXObshy6vLA"/>

```

```

    <node xmi:type="uml:ForkNode" xmi:id="_izGGoJR2EeebXObshy6vLA"
incoming="_7aWKAJkoEeeOIORO-VyZIA" outgoing="_xoAP8JR3EeebXObshy6vLA
_y2k74JR3EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_kVjX4JR2EeebXObshy6vLA"
name="Enter_User_Name" incoming="_xoAP8JR3EeebXObshy6vLA"
outgoing="_zrTkcJR3EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_mIuIsJR2EeebXObshy6vLA"
name="Enter_Password" incoming="_y2k74JR3EeebXObshy6vLA"
outgoing="_0qqvsJR3EeebXObshy6vLA"/>
    <node xmi:type="uml:JoinNode" xmi:id="_ogUvMJR2EeebXObshy6vLA" name=""
incoming="_zrTkcJR3EeebXObshy6vLA _0qqvsJR3EeebXObshy6vLA"
outgoing="_nhGroJ7vEeel-svp6Z91RQ"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_qBiqUJR2EeebXObshy6vLA"
name="Verify_User_Data" incoming="_L8ShMJ7uEeel-svp6Z91RQ"
outgoing="_aoYiMJ7uEeel-svp6Z91RQ"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_uPNkoJR2EeebXObshy6vLA"
name="Find_User_Data" incoming="_1s9TIJR3EeebXObshy6vLA"
outgoing="_LIId3kJ7uEeel-svp6Z91RQ"/>
    <node xmi:type="uml:DataStoreNode" xmi:id="_xFJ3AJR2EeebXObshy6vLA"
name="User_Profile" incoming="_LIId3kJ7uEeel-svp6Z91RQ"
outgoing="_L8ShMJ7uEeel-svp6Z91RQ" isControlType="true">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_xFYggJR2EeebXObshy6vLA" value="1"/>
    </node>
    <node xmi:type="uml:OpaqueAction" xmi:id="_5F6BYJR2EeebXObshy6vLA"
name="Assign_User_Data" incoming="_DAIegJ7uEeel-svp6Z91RQ"
outgoing="_7tsGgJR3EeebXObshy6vLA"/>
    <node xmi:type="uml:DecisionNode" xmi:id="_BYaDEJR3EeebXObshy6vLA"
name="" incoming="_7tsGgJR3EeebXObshy6vLA"
outgoing="_EnCFwJR4EeebXObshy6vLA _Gi3gUJR4EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_QZxwgJR3EeebXObshy6vLA"
name="Create_Account" incoming="_EnCFwJR4EeebXObshy6vLA"
outgoing="_Ltr_YJR4EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_Rr-QJR3EeebXObshy6vLA"
name="Reject_Account" incoming="_Gi3gUJR4EeebXObshy6vLA" outgoing="_M-
guEJR4EeebXObshy6vLA"/>
    <node xmi:type="uml:MergeNode" xmi:id="_Xq6b0JR3EeebXObshy6vLA" name=""
incoming="_Ltr_YJR4EeebXObshy6vLA _M-guEJR4EeebXObshy6vLA"
outgoing="_N5JKUJR4EeebXObshy6vLA"/>
    <node xmi:type="uml:CallBehaviorAction"
xmi:id="_edbRcJR3EeebXObshy6vLA" name="Update_System"
incoming="_N5JKUJR4EeebXObshy6vLA" outgoing="_g0YxcJ7uEeel-svp6Z91RQ"
behavior="_e76kQJR3EeebXObshy6vLA"/>
    <node xmi:type="uml:DecisionNode" xmi:id="_WTroJR6EeebXObshy6vLA"
incoming="_QEZZkJR7EeebXObshy6vLA" outgoing="_VcBLkJR7EeebXObshy6vLA
_cAnT8JR7EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_NbgrwJR7EeebXObshy6vLA"
name="User_Cancels" incoming="_VcBLkJR7EeebXObshy6vLA">
    <handler xmi:type="uml:ExceptionHandler"
xmi:id="_HmLDQJR8EeebXObshy6vLA" exceptionInput="_DuU_UJR8EeebXObshy6vLA"
exceptionType="_h8GIJR1EeebXObshy6vLA"
handlerBody="_AIObUJR8EeebXObshy6vLA"/>
    </node>
    <node xmi:type="uml:ForkNode" xmi:id="_aY0Z0JR7EeebXObshy6vLA" name=""
incoming="_cAnT8JR7EeebXObshy6vLA" outgoing="_q540QJR7EeebXObshy6vLA
_rrpCYJR7EeebXObshy6vLA _sorSYJR7EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_ifxIsJR7EeebXObshy6vLA"
name="Verify_Email" incoming="_q540QJR7EeebXObshy6vLA"
outgoing="_tVswcJR7EeebXObshy6vLA"/>

```

```

    <node xmi:type="uml:OpaqueAction" xmi:id="_j6mvAJR7EeebXObshy6vLA"
name="Verify_Password" incoming="_rrpCYJR7EeebXObshy6vLA"
outgoing="_uU3HYJR7EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_LNZMkJR7EeebXObshy6vLA"
name="Verify_Phone_Number" incoming="_sorSYJR7EeebXObshy6vLA"
outgoing="_vH6U8JR7EeebXObshy6vLA"/>
    <node xmi:type="uml:MergeNode" xmi:id="_pyfIYJR7EeebXObshy6vLA" name=""
incoming="_tVswcJR7EeebXObshy6vLA _uU3HYJR7EeebXObshy6vLA
_vH6U8JR7EeebXObshy6vLA" outgoing="__uP_oJR8EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_AI0bUJR8EeebXObshy6vLA"
name="Account_Cancelled" outgoing="_Ozc5oJR8EeebXObshy6vLA">
    <inputValue xmi:type="uml:InputPin" xmi:id="_DuU_UJR8EeebXObshy6vLA"
name="InputPin">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_DuVmYJR8EeebXObshy6vLA" value="1"/>
    </inputValue>
    </node>
    <node xmi:type="uml:ActivityFinalNode" xmi:id="_NomQEJR8EeebXObshy6vLA"
name="" incoming="_Ozc5oJR8EeebXObshy6vLA _UJILkJSBEeebXObshy6vLA
_Uwro4JSDeebXObshy6vLA _kq7BwKg8Eeez4cg0fIXL4g"/>
    <node xmi:type="uml:DecisionNode" xmi:id="_004CUJR8EeebXObshy6vLA"
name="" incoming="__uP_oJR8EeebXObshy6vLA"
outgoing="_BV4EwJR9EeebXObshy6vLA _JTy9kJR9EeebXObshy6vLA
_pzIasJukEeeMJso6T-JfIQ"/>
    <node xmi:type="uml:FlowFinalNode" xmi:id="_7TENwJR8EeebXObshy6vLA"
name="" incoming="_BV4EwJR9EeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_8nxW4JR8EeebXObshy6vLA"
name="Customer_Complains" incoming="_JTy9kJR9EeebXObshy6vLA"
outgoing="_S8Dw8JSAEeebXObshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_PHurEJR9EeebXObshy6vLA"
name="Account_information_Approved" incoming="_pzIasJukEeeMJso6T-JfIQ">
    <outputValue xmi:type="uml:OutputPin"
xmi:id="_Ti85MJR9EeebXObshy6vLA" name="" outgoing="_34ePoJuiEeeMJso6T-
JfIQ">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_Ti85MZR9EeebXObshy6vLA" value="1"/>
    </outputValue>
    </node>
    <node xmi:type="uml:OpaqueAction" xmi:id="_rq_3QJR9EeebXObshy6vLA"
name="Receive_Order" outgoing="_xAm0kJR-EeebXObshy6vLA">
    <inputValue xmi:type="uml:InputPin" xmi:id="_vSm7IJukEeeMJso6T-JfIQ"
name="" incoming="_34ePoJuiEeeMJso6T-JfIQ">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_vSoJQJukEeeMJso6T-JfIQ" value="1"/>
    </inputValue>
    </node>
    <node xmi:type="uml:DecisionNode" xmi:id="_-VKQUJR9EeebXObshy6vLA"
name="" incoming="_xAm0kJR-EeebXObshy6vLA" outgoing="_3PXc4JR-
EeebXObshy6vLA _b_bdcJuoEeeMJso6T-JfIQ"/>
    <node xmi:type="uml:SendSignalAction" xmi:id="_PT9TwJR-EeebXObshy6vLA"
name="Verify_CC_Funds" outgoing="_k1_AsJujEeeMJso6T-JfIQ"
signal="_uaqpsJuoEeeMJso6T-JfIQ">
    <target xmi:type="uml:InputPin" xmi:id="_wEoRIJuoEeeMJso6T-JfIQ"
incoming="_b_bdcJuoEeeMJso6T-JfIQ" type="_PL28EJR-EeebXObshy6vLA"
isControlType="true">
    <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_wEoRIZuoEeeMJso6T-JfIQ" value="1"/>
    <upperValue xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_wEoRIpuoEeeMJso6T-JfIQ" value="1"/>

```



```

        </target>
    </node>
    <node xmi:type="uml:AcceptEventAction" xmi:id="_aCiWUJR-EeebX0bshy6vLA"
name="Received_Verification" incoming="_k1_AsJuJFeeMJso6T-JfIQ">
    <result xmi:type="uml:OutputPin" xmi:id="_KtnewJulEeeMJso6T-JfIQ"
name="" outgoing="_TTCPoJumEeeMJso6T-JfIQ" isControlType="true">
        <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_KtoF0JulEeeMJso6T-JfIQ" value="1"/>
    </result>
    <trigger xmi:type="uml:Trigger" xmi:id="_4LTycJuqEeeMJso6T-JfIQ"
name="Trigger"/>
</node>
    <node xmi:type="uml:OpaqueAction" xmi:id="_0kXPMJR-EeebX0bshy6vLA"
name="Out_of_Stock_Items" incoming="_3PXc4JR-EeebX0bshy6vLA"
outgoing="_LyGsQKg8Eeez4cg0fIXL4g"/>
    <node xmi:type="uml:CallBehaviorAction"
xmi:id="_DZXUkJR-EeebX0bshy6vLA" name="CallBehaviorAction1"
behavior="__h8GIJR1EeebX0bshy6vLA"/>
    <node xmi:type="uml:CallBehaviorAction"
xmi:id="_DquxYJR-EeebX0bshy6vLA" name="CallBehaviorAction2"
behavior="__h8GIJR1EeebX0bshy6vLA"/>
    <node xmi:type="uml:CallBehaviorAction"
xmi:id="_ESTHsJR-EeebX0bshy6vLA" name="CallBehaviorAction3"
behavior="__h8GIJR1EeebX0bshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_J0hUoJSAEeebX0bshy6vLA"
name="Customer_Complaint_Handled_(complaint_dept.)"
incoming="_S8Dw8JSAEeebX0bshy6vLA" outgoing="_UJILkJSBEeebX0bshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_5kI0gJSBEeebX0bshy6vLA"
name="Get_Products">
        <inputValue xmi:type="uml:InputPin" xmi:id="_9mQmIJSBEeebX0bshy6vLA"
name="" incoming="_TTCPoJumEeeMJso6T-JfIQ" isControlType="true">
            <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_9mQmIZSBEeebX0bshy6vLA" value="1"/>
        </inputValue>
        <outputValue xmi:type="uml:OutputPin"
xmi:id="_wJ6D0JSDEeebX0bshy6vLA" outgoing="_ZxRDUJSCeebX0bshy6vLA"
isControlType="true">
            <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_wJ6q4JSDEeebX0bshy6vLA" value="1"/>
        </outputValue>
    </node>
    <node xmi:type="uml:DecisionNode" xmi:id="_BpfyEJSCeebX0bshy6vLA"
name="" incoming="_ZxRDUJSCeebX0bshy6vLA"
outgoing="_btyaYJSCeebX0bshy6vLA _fP6dUJSCeebX0bshy6vLA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_DOQwcJSCeebX0bshy6vLA"
name="Test_Computer">
        <inputValue xmi:type="uml:InputPin" xmi:id="_IswZcJSCeebX0bshy6vLA"
incoming="_fP6dUJSCeebX0bshy6vLA" isControlType="true">
            <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_IswZcZSCeebX0bshy6vLA" value="1"/>
        </inputValue>
        <outputValue xmi:type="uml:OutputPin"
xmi:id="_K_IrwJSCeebX0bshy6vLA" outgoing="_2ciU8JSCeebX0bshy6vLA"
isControlType="true">
            <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_K_JS0JSCeebX0bshy6vLA" value="1"/>
        </outputValue>
    </node>

```

```

    <node xmi:type="uml:OpaqueAction" xmi:id="_D1IjMJSCeebX0bshy6vLA"
name="Test_Monitor">
    <inputValue xmi:type="uml:InputPin" xmi:id="_JdkyYJSCeebX0bshy6vLA"
incoming="_btyaYJSCeebX0bshy6vLA" isControlType="true">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_JdLZcJSCeebX0bshy6vLA" value="1"/>
    </inputValue>
    <outputValue xmi:type="uml:OutputPin"
xmi:id="_MM4RQJSCeebX0bshy6vLA" outgoing="_FibNUJukEeeMJso6T-JfIQ"
isControlType="true">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_MM4RQZSCeebX0bshy6vLA" value="1"/>
    </outputValue>
    </node>
    <node xmi:type="uml:CentralBufferNode" xmi:id="_OxpUEJSCeebX0bshy6vLA"
name="Product" incoming="_9x2pYJknEeeOIORO-VyZIA"
outgoing="__U0kQJSCeebX0bshy6vLA" isControlType="true">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_Oxp7IJSCEebX0bshy6vLA" value="1"/>
    </node>
    <node xmi:type="uml:OpaqueAction" xmi:id="_UJ2u4JSCeebX0bshy6vLA"
name="Send_Order_for_Shipment" incoming="__U0kQJSCeebX0bshy6vLA"
outgoing="_Uwro4JSDeebX0bshy6vLA"/>
    <node xmi:type="uml:MergeNode" xmi:id="_7TV_kJknEeeOIORO-VyZIA" name=""
incoming="_2ciU8JSCeebX0bshy6vLA _FibNUJukEeeMJso6T-JfIQ"
outgoing="_9x2pYJknEeeOIORO-VyZIA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_S0_E8JkoEeeOIORO-VyZIA"
name="Ask_to_Login_for_Registered_Users" incoming="_bDae8JkoEeeOIORO-VyZIA"
outgoing="_hj1fAJkoEeeOIORO-VyZIA"/>
    <node xmi:type="uml:MergeNode" xmi:id="_gFeLEJkoEeeOIORO-VyZIA" name=""
incoming="_wxh68JR3EebX0bshy6vLA _hj1fAJkoEeeOIORO-VyZIA"
outgoing="_7aWKAJkoEeeOIORO-VyZIA"/>
    <node xmi:type="uml:OpaqueAction" xmi:id="_zb-c4JunEeeMJso6T-JfIQ"
name="">
    <outputValue xmi:type="uml:OutputPin" xmi:id="_46hwQJunEeeMJso6T-
JfIQ" name="">
    <upperBound xmi:type="uml:LiteralInteger"
xmi:id="_46iXUJunEeeMJso6T-JfIQ" value="1"/>
    </outputValue>
    </node>
    <node xmi:type="uml:MergeNode" xmi:id="_ZLMygJ7uEeel-svp6Z91RQ"
incoming="_aoYiMJ7uEeel-svp6Z91RQ _g0YxcJ7uEeel-svp6Z91RQ"
outgoing="_QEZZkJR7EebX0bshy6vLA"/>
    <node xmi:type="uml:DecisionNode" xmi:id="_lcXMsJ7vEeel-svp6Z91RQ"
name="" incoming="_nhGroJ7vEeel-svp6Z91RQ" outgoing="_DAIegJ7uEeel-
svp6Z91RQ _1s9TIJR3EebX0bshy6vLA"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Signal" xmi:id="_PL28EJR-EeebX0bshy6vLA"
name="Signal1"/>
</uml:Model>

```


Part B

Table K.1 MySQL ‘initial marking’ Table for the Production System

264

Part C

This section includes the file retrieved from the transpose of the incidence matrix automatically generated for the online shopping process as discussed in Chapter 7.

[illegible]

Part D

This section presents the initial marking table automatically generated for the online shopping process as discussed in Chapter 7.

Table K.2 MySQL ‘initial_marking’ Table for the Online Shopping Process

activity	process_number_of_devices
Account Information (pending)	0
Authorised	0
Computer	0
flow out 30	0
Monitor	0
Not Authorised	0
Order Items	0
pin 5	1
place 11	0
place 12	0
place 13	0
place 15	0
place 17	0
place 18	0
place 19	0
place 2	0
place 20	0
place 21	0
place 22	0
place 24	0
place 26	0
place 27	0
place 28	0
place 3	0
place 31	0
place 33	0
place 36	0
place 4	0
place 41	0
place 42	0
place 43	0
place 44	0
place 45	0
place 46	0
place 48	0
place 49	0
place 5	0
place 52	0
place 53	0
place 54	0
place 55	0
place 56	0
place 6	0
place 7	0
place 8	0
pout	0
Product	0
User Profile	0
Verifv Account by Phone	0

Appendix L – Verification of the Production System and Online Shopping Process

Appendix L includes figures obtained from HiPS proving that the production system and online shopping process, discussed in Chapter 7, are (i) structurally and behaviourally bounded as can be seen in Figures L.1 and L.3 respectively; and (ii) behaviourally live and safe as can be seen in Figures L.2 and L.4 respectively.

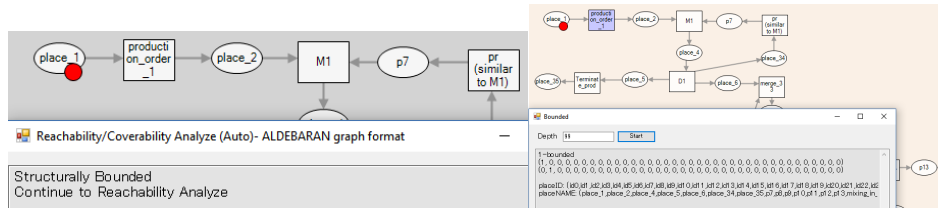


Figure L.1 Structurally and Behaviourally Bounded Check in HiPS for the Production System

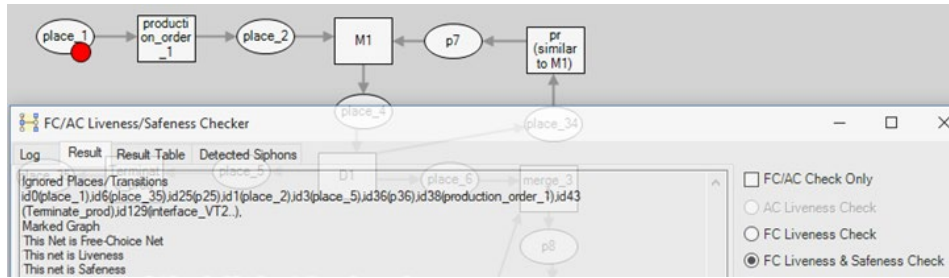


Figure L.2 Behavioural Liveness and Safeness Properties Check in HiPS for the Production System

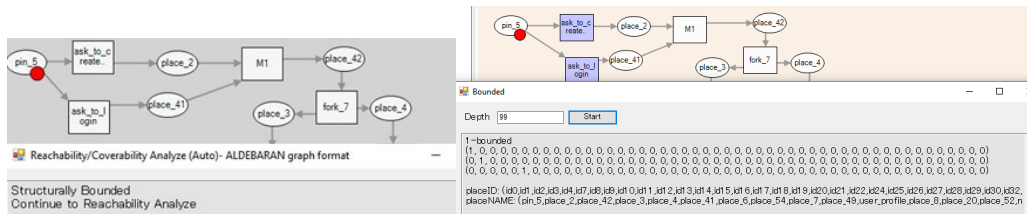


Figure L.3 Structurally and Behaviourally Bounded Check in HiPS for the Online Shopping Process

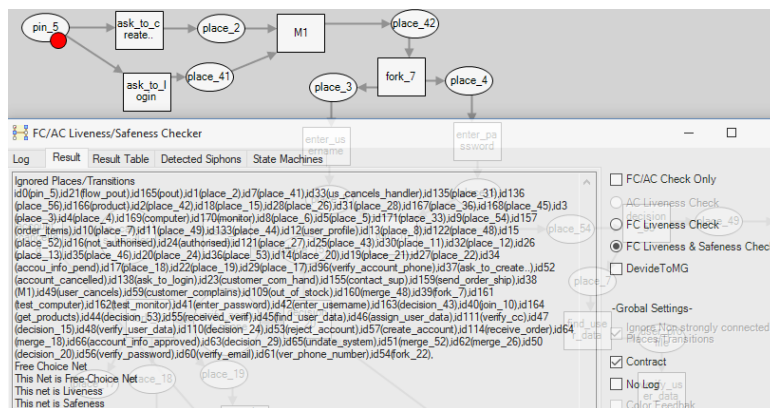


Figure L.4 Behavioural Liveness and Safeness Properties Check in HiPS for the Online Shopping Process