Karlsruhe Institute of Technology

Data model extraction through adaption of business process models

Master Thesis

by

David Philipp Diener

Degree Course: Business Informatics M.Sc.

Matriculation Number: 2396965

Institute of Applied Informatics and Formal Description Methods (AIFB)

KIT Department of Economics and Management

Advisor:	Prof. Dr. Andreas Oberweis
Second Advisor:	Prof. Dr.-Ing. Johann Marius Zöllner
Supervisor:	Selina Schüler
Submitted:	August 29, 2023

Abstract

This thesis explores the extraction of a data model from a business process model, addressing challenges related to the complexity and time-consuming nature of manual data model creation.

To achieve this, a new variant of High-Level Petri Nets, XML-Data-Nets, on the basis of Petri Net Markup Language, is introduced, which extends Petri Nets to represent more complex relationships and objects, thereby facilitating more accurate data model extraction.

To extract a data model from a business process model, certain rulesets which contain the extraction logic have to be defined. This work presents three rulesets dealing with the extraction of classes, associations, and attributes for the target data model. The resulting data model is a logical UML Class Diagram, which can then be used as a foundation for the creation or adaption of a physical data model in the context of software development.

This work also discusses the implementation of a prototype application demonstrating the practical application of the concept. The implementation serves as a foundation for the evaluation of the data model extraction concept as well as future research on the practical application of data model extraction based on business process models. The evaluation of the developed application, including both quantitative measures of performance and a qualitative user survey, provides evidence of the effectiveness and usability of the proposed solution. The results indicate that the application can save time and resources while maintaining high accuracy.

Contents

List of Abbreviations

BPM business process model.

BPML business process modeling language.

BPMN Business Process Model and Notation.

CDM Conceptual Data Model.

DIF Diagram Interchange Format.

DME Data Model Extractor.

DTD Document Type Definition.

EPC Event-driven Process Chain.

EPML EPC markup language.

HLPNG High-level Petri Nets Graph.

OMG Object Management Group.

PNML Petri Net Markup Language.

PNTD Petri Net Type Definition.

POM process-oriented model.

RNG REgular LAnguage for XML Next Generation.

UML Unified Modeling Language.

UML AD Unified Modeling Language Activity Diagram.

UML CD Unified Modeling Language Class Diagram.

XDN XML-Data-Net.

XMI XML Metadata Interchange.

XML Extensible Markup Language.

XSD XML Schema Definition.

YAWL Yet Another Workflow Language.

List of Figures

List of Tables

1 Introduction

Business process modeling and data modeling are two of the main tasks in the requirements engineering phase, an early phase of software development [Sommerville(2015)]. The design of those artifacts usually is created independently, which is a tedious process.

Business process modeling, a technique employed to visualize the sequence of activities in a business process, encapsulates rich and valuable information about the organization's workflows, roles, and data interactions. On the other hand, data modeling, which involves defining, structuring, and documenting data requirements for a system, forms the foundation upon which the system's data architecture is built. As such, these models individually hold significant relevance and importance. However, the potential value at the intersection of these two areas often remains underexplored and underutilized.

This paper combines those design processes and presents an approach for extracting a data model from a business process model. It offers a methodology that combines business process modeling and data modeling, thereby proposing the extraction of a data model from a business process model. This integration bridges the gap between process and data perspectives and improves the use of information within the business process model to enrich the data model.

The data model derived from the business process model serves as a foundation in the early phases of software development. This approach seeks to enhance the efficiency of the design process by reducing the time and effort spent on data model design.

1.1 Objective of this Thesis

This thesis introduces an approach for data model extraction from business process models (BPMs). To achieve this goal, the thesis systematically explores the process of extracting a business process model into a data model.

The design of the extraction process requires the analysis of various business process modeling languages (BPMLs) and their respective diagram interchange formats. The need to enhance one of these languages requires this comprehensive evaluation, as it provides a foundation for understanding their structure and features. Following this, a particular language is selected for enhancement to improve its expressiveness and to better support the extraction algorithm to yield more accurate data models. The next stage of the process involves selecting the target data model, for which the Unified Modeling Language Class Diagram (UML CD) is chosen.

For the creation of the extraction process, we turn to existing literature to derive insights from related approaches. These works provide an initial set of guidelines and rules for data model extraction. However, recognizing that there is room for improvement, this ruleset is iteratively refined and enhanced based on the findings while implementing the solution, eventually forming a ruleset specifically designed to extract a data model from a business process model.

Finally, the selected BPML is modified to incorporate all necessary features identified by the ruleset. This modification ensures the language is fully customized for the extraction process.

The solution presented in this work aims to support the data model extraction process while also proposing a practical implementation of the presented concept. By using the enhanced expressiveness of a BPML, this work promises to save time and resources in the early phases of the software development process.

1.2 Structure of this Thesis

The structure of this thesis is divided into seven chapters. The organization of the chapters is as follows:

Beginning with Chapter 2, an overview of the foundational elements are presented, including a thorough examination of various business process modeling languages and data model abstraction types, setting the theoretical stage for this work. Moving forward, Chapter 3 introduces the existing related work in the field of data model extraction. This is followed by Chapter 4, which presents the adopted waterfall research methodology and proposes a potential solution approach for this study. Subsequently, Chapter 5 transitions into the practical implementation, describing the implementation of a prototype application that realizes the data model extraction algorithm. Building upon this, Chapter 6 undertakes a comprehensive evaluation of the prototype application, incorporating both quantitative analyses of the resulting data model and qualitative analysis based on survey data. Finally, in chapter 7, we discuss the contribution and limitations of the thesis.

2 Foundations

Since this work is based on adapting an established BPML to meet the requirements of the extraction algorithm, we first need to think about a source language that will serve as the basis for the adapted language. This chapter contains an overview of the languages and technologies considered for this work. In chapter 2.1, the established BPML standards and their respective diagram exchange format, which can serve as a possible basis for a source language, are described. The Diagram Interchange Format (DIF) plays a special role here, as it acts as a link between the source BPM and the target data model from a technical point of view. Finally, chapter 2.2 describes the different types of data models and an established notation, the UML CD, for the target data model.

2.1 Source BPML

According to the foundational work of Weske [Weske(2019)] there are five essential modeling techniques and notations for business process modeling. These contain Petri nets, workflow nets, Business Process Model and Notation (BPMN), Event-driven Process Chains (EPCs), and Yet Another Workflow Language (YAWL). Brdjanin and Maric have divided the common source BPMLs in the context of data model synthesis into four categories: function-oriented, process-oriented, communication-oriented, and goal-oriented [Brdjanin and Maric(2014)]. This taxonomy is particularly important for this work since it deals specifically with data model synthesis based on BPMs. Process-oriented models (POMs) represent the largest category of BPMLs in terms of the number of papers dealing with the synthesis of data models based on these languages. The POMs include Petri nets, BPMN and EPCs, which intersect with the selection of the most essential BPMLs by Weske [Weske(2019)]. Based on popularity as well as the topicality, the Unified Modeling Language Activity Diagram (UML AD) was also selected for further inspection in this work. In the following sections, the five languages Petri nets, BPMN, UML AD, EPCs, and YAWL are briefly described with a general description, their entities, their DIF, and an example.

2.1.1 Petri nets

Description Petri nets are a visual formalism for describing the behavior of a system introduced by Carl Adam Petri in 1962 [Petri(1962)]. They are becoming increasingly popular in the application of business process modeling. Modern BPM analysis techniques even use Petri nets as internal representations (see [van der Aalst(2015)]). In addition to Petri's original design of Petri Nets, there are more advanced versions of the Petri Net model, High-level Petri Nets

[International Organization for Standardization(2000)], such as Colored Petri Nets [Jensen(1987)], where the tokens can be made distinguishable by the further specification. Another advanced form of Petri nets are workflow nets, which Weske [Weske(2019)] classifies as one of the most essential modeling techniques.

Entities According to Reisig [Reisig(1982)], a Petri net consists of places, transitions, and arcs. Places correspond to a state in the process and are visualized graphically by circles, while transitions represent actions of a process and are visualized graphically by a bar or rectangle. Arcs are directed arrows that connect places and transitions. The places that point into a transition form the pre-set of the transition. The places that originate from a transition form the post-set of the transition. In Petri Nets, the marking assigns each place a number of abstract, indistinguishable tokens represented by black dots. The transition is enabled if each place in the pre-set of a transition holds at least one token. Transitions are enabled only when each pre-set contains at least one token. Firing a transition consumes a token in the pre-sets of a transition and creates a token in the post-sets of the transition.

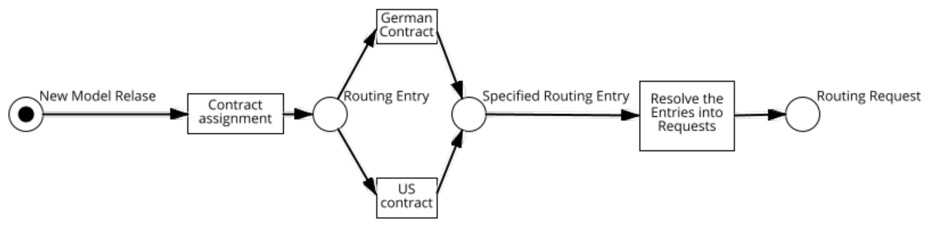


Figure 1: Example of a parts routing process as a Petri net model

Example Figure 1 shows an example of a parts routing process in the automotive sector modeled as a Petri net. When a new model is released, the routing system assigns the contract accordingly. Afterward, the process splits into two different tasks based on the contract types for the German and US markets. After the routing entries for the contract have been specified, they are resolved into Routing Requests by the system. Once the Routing Requests have been created, the process is complete.

The Petri net contains the places **New Model Release**, **Routing Entry**, **Specified Routing Entry** and **Routing Request**, as well as the transitions **Contract Assignment**, **German Contract**, **US Contract** and **Resolve Routing Requests into Requests**. **New Model Release** is the first place of the Petri net and contains an initial marking of one token. Using the transition **Contract assignment** as an example, the pre-set consists of the place **New Model Release**, and the post-set consists of the place **Routing Entry**.

Diagram Interchange In 2003, Weber and Kindler presented the Petri Net

Markup Language (PNML), an XML-based interchange format for Petri nets [Weber and Kindler(2003)]. PNML is capable of capturing basic Petri nets (a labeled graph with places and transitions), including the graphical information (2-dimensional coordinates of objects), dedicated tool information, and the page structure of Petri nets. PNML also supports various types of Petri nets, such as P/T nets, Colored Petri nets, and High-level Petri nets. The official syntax can be extended using custom Petri Net Type Definitions (PNTDs). A PNTD is a document that defines the Extensible Markup Language (XML) syntax of the elements in a PNML document like a Document Type Definition (DTD). An example of the use of a PNTD is the work of Werf et al. They defined a modified version of PNML to serialize custom Petri nets in their project (see [Van der Werf, Jan Martijn E.M. and Post(2004)]).

XML-Nets In order to provide the extraction algorithm presented in this work with information about the data view of the process, the source BPML must support the addition of information about the data view into the BPM.

In 2001, Lenz and Oberweis proposed XML nets, a variant of High-level Petri Nets, that is used to model interorganizational workflows [Lenz and Oberweis(2001)]. The idea is to add XML Schema Definitions (XSDs) to the places in the Petri net that specify the type of documents a place can contain. Therefore, the places of XML nets are interpreted as a collection of XML documents. XML nets also involve the use of a graphical modeling language (GXSL) to define these XSDs. Additionally, the arcs of the Petri net hold so-called extended-XSDs to express the transformation of the documents in the transition to the adjacent places. Those transformations are defined by a query language called XManiLa. The transitions may contain a logical expression over the specified variables appearing in the adjacent arcs' extended-XSD.

An example of XML nets is the toolset INCOME2010 [Klink et al.(2008)], a tool for the graphical modeling of custom Petri nets by using the concept of XML-Nets, from which the open source software toolset Horus has evolved [Thomas Karle et al.(2006)]. While this works explains the architecture of the software implementation, it does not go into detail on the implementation of the XML-Nets.

2.1.2 BPMN

Description BPMN is a graphical modeling language maintained by the Object Management Group (OMG). It was first released as BPMN 1.0 in May 2004 and currently represents the industry standard for modeling business processes in its latest version BPMN 2.0.2 [OMG(2013)].

Entities BPMN offers four basic types of elements for the representation of business processes - flow objects, connecting objects, artifacts, and swimlanes (see [Weske(2019)]). Flow objects are divided into activities, events, and gateways. Artifacts include data Objects, groups, and annotations, while connecting objects can represent sequence flows, message flows, and associations. By using these elements, especially the swimlanes, a BPMN graph is able to represent complex business processes across multiple systems or stakeholders.

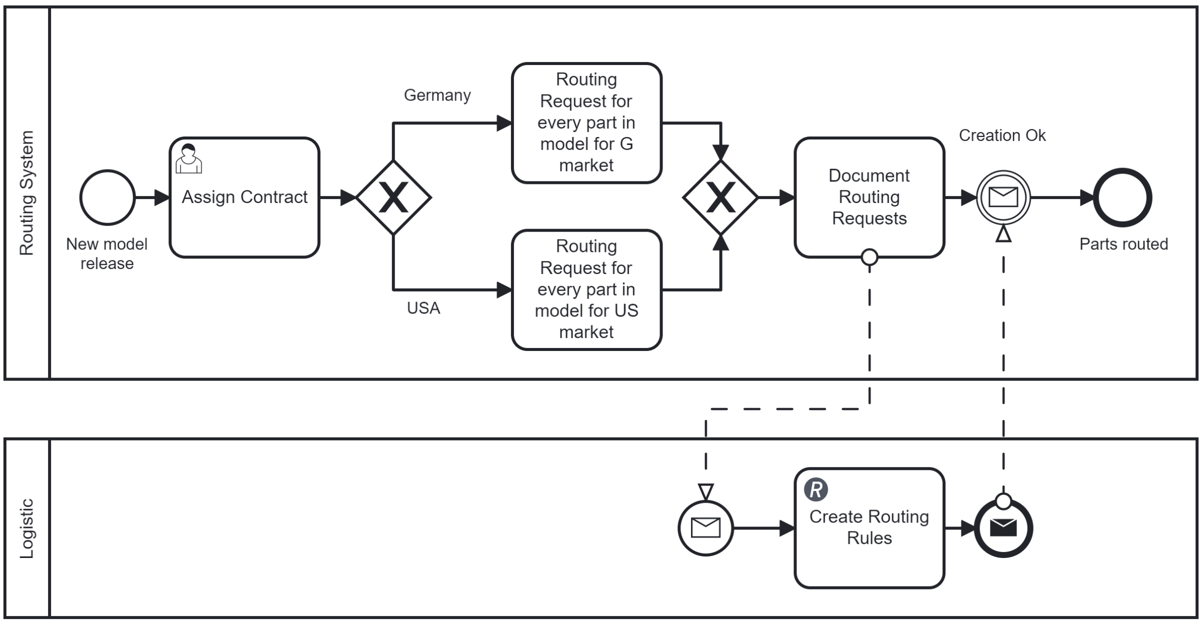

Routing System
New model release
Assign Contract
Germany
USA
Routing Request for every part in model for G market
Routing Request for every part in model for US market
Document Routing Requests
Creation Ok
Parts routed
Logistic
Create Routing Rules

Figure 2: Example of a parts routing process as a BPMN diagram

Example Figure 2 shows the same example as Figure 1, modeled with BPMN. The main difference between the two examples is the changed representation of the XOR element and the additional dimension added by the swimlane. In this example, the creation of the routing request can be modeled with additional information using the swimlanes and a REST API activity element. After the routing system confirms the routing rule creation, the process ends with a mandatory end event.

Diagram Interchange OMG provides the Diagram Definition standard, which describes a semantical and graphical language for exchanging BPMN diagrams [OMG(2012)]. The standard is divided into two models, Diagram Interchange and Diagram Graphics. Diagram Interchange describes the graphical information over which the user has control, such as the position of nodes. In contrast, the Diagram Graphics describes the graphical objects themselves as primitive graphical shapes. The Diagram Definition is used to standardize the use of OMG languages across tools.

Since version 2.0, the BPMN specification also describes its own XML-based interchange format. This format can be used either by following the specifications of a corresponding XSD or XML Metadata Interchange (XMI) document [OMG(2013)]. Both formats preserve the semantics of the model, for example, the individual entities of the model. The graphical layout (positioning of the elements) is also saved. The formats are similar in expression power, but the XSD-based format is more popular in practice [Kurz et al.(2022)].

2.1.3 UML Activity Diagram

Description UML ADs are a kind of behavioral diagram maintained by the OMG. It is one of several types of modeling tools included in the Unified Modeling Language (UML) specification [OMG(2017)]. They represent a series of actions or a flow of controls in a system and can thus model both computational and organizational processes.

Entities UML provides the elements states, activities, decision nodes, and control flows for activity diagrams. For the control flows, there are the concepts of forks and joins. A fork corresponds to the splitting and parallelization of the control flow. A join puts two control flows back together and synchronizes them. The UML AD specification also describes the concept of activity partitions (analogous to swimlanes), which is used to model loosely coupled systems, roles, or hierarchies.

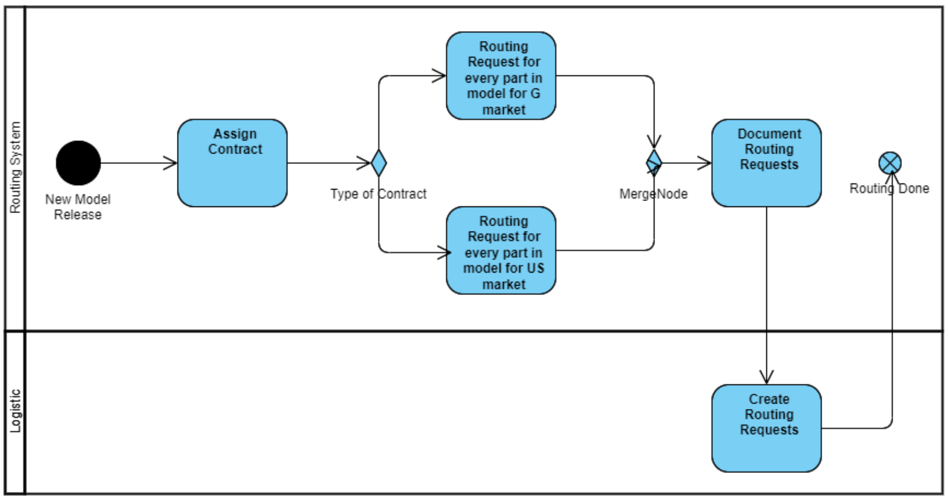

Routing System
Logistic
New Model Release
Assign Contract
Type of Contract
Routing Request for every part in model for G market
Routing Request for every part in model for US market
MergeNode
Document Routing Requests
Routing Done
Create Routing Requests

Figure 3: Example of a parts routing process as a UML AD

Example Figure 3 shows the same example as Figure 1, modeled as a UML AD. While it models the same basic control flow, the BPMN example can express more detail on the interface between the two partitions Logistic and **Routing System** by using activity partitions. The fact that the activity **Create Routing Request** implements a REST API and the **Routing System** receives a message can not be modeled as a UML AD.

Diagram Interchange OMG provides the same Diagram Definition standard for the UML as introduced in 2.1.2. The standard exchange format for UML AD is XMI, which was also developed by OMG [OMG(2022)].

2.1.4 EPC

Description The EPC is an ordered graph of events and functions. It was introduced in 1992 by Keller et al. [Keller et al.(1992)]. EPCs are used for the graphical representation of processes.

Entities EPCs have six main types of entities: *events*, *functions*, *connectors*, *combined connectors* and *control flows* (see [Weske(2019)]). *Events* are passive operators and are graphically represented as a hexagon. They describe the states of the process. The start and end entities of an EPC must be an event. *Functions* are transitions between states and are represented graphically as a rounded rectangle. Roles in an EPC can be modeled by either a *process owner* or a *organizational unit*, represented as either a square or ellipse with a vertical line. *Resources* such as information, material, or general resources are represented graphically as a rectangle. They serve as input or output data for functions.

There are three types of logical connectors in EPCs. XOR connectors are used for the branching and merging, while AND connectors are used for the forking and joining of the process flow. There is also a logical OR operator for EPC diagrams.

Control flows connect events with functions or logical connectors and are displayed as dashed arrows. Other types of flows include *information flow*, which connects functions to input and output data, and *organization unit assignments*, which connects organizations and the functions they are responsible for.

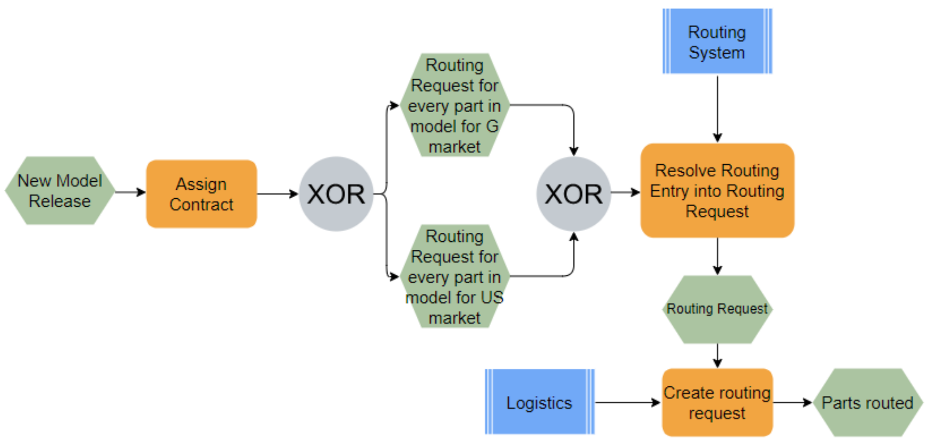

Routing System
Routing Request for every part in model for G market
New Model Release
Assign Contract
XOR
XOR
Resolve Routing Entry into Routing Request
Routing Request for every part in model for US market
Routing Request
Logistics
Create routing request
Parts routed

Figure 4: Example of a parts routing process as a EPC diagram

Example The example in figure 4 describes the same example as figure 1. While the EPC lacks the existence of swimlanes, it can still express the different partitions using the organizational unit assignments.

Diagram Interchange EPC diagrams can be interchanged between different tools by using the EPC markup language (EPML). EPML was first introduced by Mendling and Nüttgens in 2006 [Mendling and Nüttgens(2006)]. It is a platform-independent XML-based interchange format. In addition to the definition of the syntax of the EPC and the graphical positioning of the elements, EPML can also be used to define the graphical representation of the individual elements.

2.1.5 YAWL

Description YAWL was developed to address the limitations of Petri nets in expressing the full range of control flow patterns in business processes. YAWL is based on a variant of workflow nets called YAWL nets, which extend traditional workflow nets by introducing direct arcs between transitions, explicit split and join behavior, nonlocal behavior, and handling of multiple instances tasks. YAWL specifications consist of a set of YAWL nets, with multiple YAWL nets connected to each other through composite tasks. [Weske(2019)]

Entities YAWL comprises several entities, including conditions, tasks, arcs, and split and join behaviors. Conditions are represented by circles, tasks by rectangles, and arcs by directed arrows. Tasks can have specific split and join behaviors, such as AND, XOR, and OR. Multiple instances tasks can have a minimum and maximum number of instances, a threshold for continuation, and dynamic or static creation of instances. [Weske(2019)]

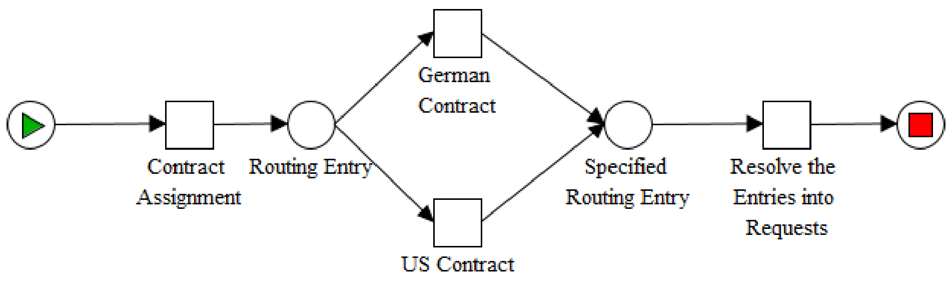


Figure 5: Example of a parts routing process as a YAWL diagram

Example The example in figure 5 describes the same example as figure 1.

Diagram Interchange YAWL does not include a DIF definition. However, in the work of Hofstede (see [Hofstede et al.(2010)]), which describes tools and an environment for working with YAWL, it is described that YAWL can be serialized as a XML document. For this purpose, a XML namespace was created. The namespace provides XML element definitions for the elements of YAWL while also providing elements for the graphical positioning of those elements in the underlying diagram.

2.2 Target Data Modeling Languages

Since the goal of this work is to automatically generate a data model from a BPM, the different types of data models are introduced in chapter 2.2.1. The types of data models refer to the level of abstraction of the data model. The data model type must be taken into account when choosing the data model notation since the data model notation must meet the requirements specified by the data model type.

In chapter 2.2.2, the data model notation for the graphical representation of the target data model is presented. In this work, the notation UML CD is used as the data model notation since it fulfills the data model type requirements expressed in chapter 2.2.1, and the data model compared in parts of the evaluation also uses this notation.

2.2.1 Data Model Types

A definition for data model types is given by the American National Standards Institute (ANSI) [Steel, Jr., Thomas B.(1975)]. In this definition, a distinction is made between the conceptual, logical, and physical data model types. These three types of data models essentially differ in the degree of abstraction.

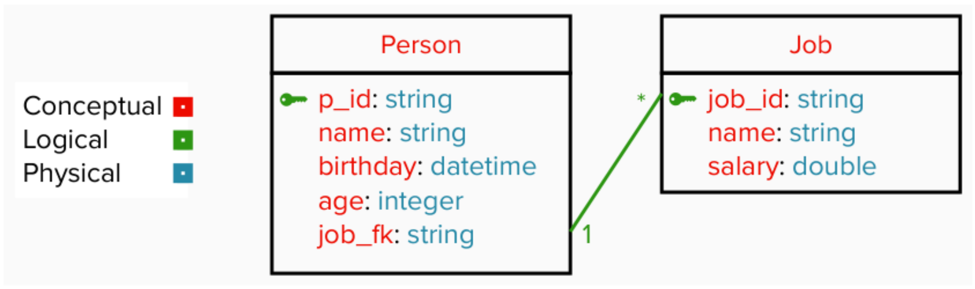


Figure 6: Visualization of the different data model types

Figure 6 shows an example of a data model describing the entities, person, and job. The different elements of the data model are colored based on if they meet the minimum specification for the respective data model type. The three types of data models are described below:

Conceptual Data Model (CDM) The CDM is the highest abstraction level of the three types and provides a high-level overview of the data elements in the system without going into specific details. The CDM is typically created during the initial stages of the software development process when the goal is to gain an understanding of the system's requirements and functionality. This model helps identify the entities and their attributes within the system and the relationships between them.

The CDM is not tied to any particular technology or database system. Instead, it is a high-level representation of the data assets that can be used to guide the development of more detailed data models.

Logical Data Model A logical data model is classified between a physical and a CDM in terms of the level of detail but is still abstracted from specific technical implementations. The logical data model provides further information about the attributes and relations of the entities by declaring primary and foreign keys. The logical data model is typically created after the CDM and is used to guide the implementation of the database system. This model is often used to develop the schema for the database, which is the blueprint for the database structure.

Physical Data Model The physical data model is the most detailed of the three data model types. It complements the logical data model by defining the attribute types for each entity in the data model. It serves as an instantiation for a specific database implementation, meaning that the data types are specified for each attribute in the context of the database system.

For the extraction of a data model from a BPM, the logical data model was chosen as the abstraction level for the target data model. The target data model will be used in the early phases of software development. The specific database implementation cannot be defined at this stage. Therefore, the specification of the additional features provided by a physical data model is not possible, without a very large effort, due to the large number of available features that would be chosen from. Another reason is that the evaluation of this project is done with a data model at the abstraction level of a logical data model.

2.2.2 UML Class Diagram

There are several data modeling notations for the representation of data models, which differ among other things also in the degree of abstraction, i.e., data model type. One of the most popular data modeling notations is the UML CD. It is accepted and maintained by the OMG, which makes it a robust standard for data modeling. It is capable of representing the abstraction level of the logical data model and has sufficient expressive power to represent the elements resulting from the data model extraction algorithm. The evaluation is carried out on the basis of an industrial project. Since the standard notation for capturing data models in this project is the UML CD, the manually created data model, against which the extracted target data model is evaluated, was created in this notation. By using the same data model notation, syntactical comparability between the data models is ensured.

The main entities of the UML CD are classes, interfaces, associations, and other syntactic elements of a system, as well as their relationships and dependencies.

Classes are the main building blocks of a class diagram. They represent a collection of similar objects that share common attributes, operations, methods, and relationships. Classes are depicted as rectangles with the class name at the top and the attributes and methods listed below. Attributes are the data elements of a class, while methods are the functions or operations that can be performed on a class. Based on the abstraction level of the UML AD, the attributes may also describe their visibility levels and data types.

The resulting target data model in the UML CD notation has the purpose of being a foundation for the early phases of software development. In object-oriented software development, inheritance is an important concept to unify classes that have a high degree of similarity. In a class diagram, inheritance is visualized by a solid line with an arrowhead pointing to the parent class. This arrowhead represents the direction of the inheritance relationship, which means that the child class inherits the attributes and methods of the parent class.

Interfaces are similar to classes, but they only contain methods that other classes, which inherit the interface, must implement. They are visualized by a rectangle with the interface name in addition to the term "interface" above it.

Associations represent the relationships between classes and are visualized by a line connecting the two related classes. On each end of the association line, the cardinality is displayed. Cardinality specifies the number of instances of one class that can be linked to a single instance of the associated class.

Aggregation and composition are two types of associations that represent a "whole-part" relationship. Aggregation is represented by a diamond shape on the end of the line pointing towards the "whole", while composition is represented by a filled diamond. Aggregation means that the "part" class can exist without the "whole" class, while composition means that the "part" class cannot exist without the "whole" class.

Figure 7 shows an example of a data model for a parts routing system as a UML CD. It contains the six classes **Routing Request**, **Part Family**, **Manual Routing Request**, **Launch Routing Request**, **Routing Cluster** and **Routing Entry**. Each class contains its associated attribute definitions and methods. **Manual Routing Request** and **Launch Routing Request** inherit the attributes and method from their parent class **Routing Request**. The classes **Manual Routing Request**, **Launch Routing Request**, and **Routing Cluster** have a 1 to N relation to the class **Routing Entry**. This

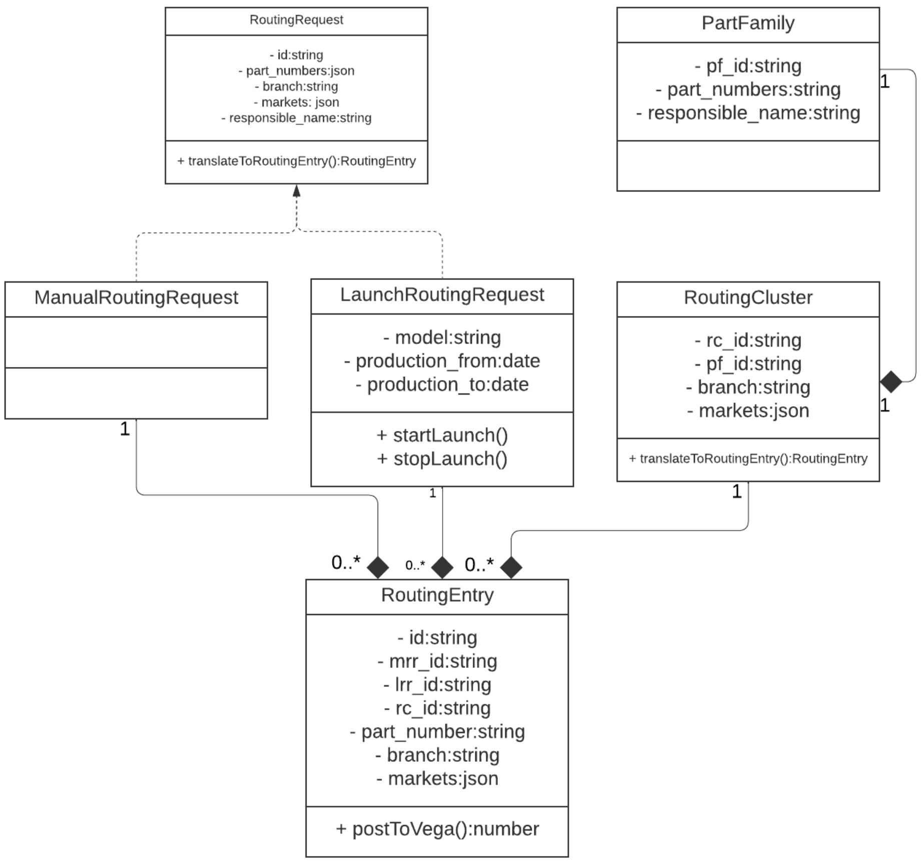


Figure 7: Example of a UML CD

means that every instantiation of those classes can have multiple instantiations of the **Routing Entry** class. The associations between those classes also represent an incoming composite relation to **Routing Entry**, meaning that an instance of **Routing Entry** cannot exist without the part classes. **Part Family** has a 1 to 1 relation to the class **Routing Cluster**, meaning that every instantiation of each class needs a matching element into the corresponding class. **Part Family** is the part class in the composite association to the whole class **Routing Cluster**, meaning that a **Routing Cluster** cannot exist without a **Part Family**.

3 Related Work

This chapter explores various approaches on how to extract data models from BPMs. The first approach proposed by Brdjanin et al. in 2012 [Brdjanin and Maric(2012)] focuses on using UML Activity Diagrams to automatically generate a CDM. The second approach, also by Brdjanin et al. in 2018 [Brdjanin et al.(2018)], uses a two-phase approach to extract data models from various BPMLs, identifying key concepts such as participants, roles, objects, message flows, and activations. Cruz et al. propose a systematic approach for extracting a data model from a BPMN 2.0 diagram [Cruz et al.(2012)], using three sets of rules to identify entities, relationships, and attributes. Finally, the chapter discusses the concept of XML-Nets, as already introduced in chapter 2.1.1, and their application in the BPML introduced in this work, which uses XSD-typed places to provide attributes required for the target data model.

3.1 An Approach to Automated Conceptual Database Design Based on the UML Activity Diagram

An approach to automate the creation of a CDM based on a UML AD was proposed by Brdjanin et al. in 2012 [Brdjanin and Maric(2012)]. In this work, 10 rules are formulated that, when applied to a UML AD, lead to the automated generation of a UML AD. The rules cover the extraction of participants, the extraction of objects, the associations between participants and objects, and the associations between objects and objects. Objects are generated based on the actions in the UML AD. The rule for generating objects distinguishes between existing objects and objects generated in the process. Brdjanin et al. implemented an automated CDM generator tool based on those rules. The tool represents the target data model by using two files, separating the XMI representation and diagram interchange of the UML CD.

3.2 An Online Business Process Model-driven Generator of the Conceptual Database Model

Another work on data model extraction on the basis of BPMs uses a two-phase approach to extract data models from various BPMLs [Brdjanin et al.(2018)]. The semantic capacity of the source BPMs are discussed here, with the goal of the synthesis of a CDM in mind. They identified participants, roles, objects, message flows, and activations of existing objects as the main concepts from BPMs that enable the automatic generation of classes in the target CDM.

A proprietary language is introduced to parse different BPMs to the same target data model format. In the first phase, the information from the source BPM is extracted and represented in the domain-specific language. Phase two then uses the representation of the source BPM in the domain-specific language to generate the target data model.

The two-phase approach for data model synthesis is also implemented into an application in the course of this work and was used for the evaluation of the approach. In an associated experiment, the application presented in this work was able to generate the target CDM with average completeness and precision over 80%.

3.3 From Business Process Modeling to Data Model: A Systematic Approach

Cruz et al. also describe an approach for the extraction of a data model by using a BPMN diagram as the source model [Cruz et al.(2012)]. More specifically, they are generating an early data model in a generic notation to be used in the early phases of software development. In this work, the data model is derived from a BPMN 2.0.0 diagram.

For the generation of the data model, they introduced three sets of rules on how to convert single elements of a BPMN 2.0 diagram into the single elements of the data model. The first set of rules deals with the identification of the entities in the BPMN diagram. The second group of rules identifies the relationships between the entities, and the third set of rules identifies the respective attributes for each entity.

The paper also distinguishes between persistent and non-persistent data. The persistent data is data that remains beyond the life cycle of the process. This distinction is made by considering the data representation elements of the BPMN. Data stores are considered permanent, while all other data representation elements are considered non-persistent. This makes it possible to identify classes that are maintained in a persistent manner.

The approach presented in this work is able to extract entities from data stores and participants in the source BPMN diagram. The associations between the extracted entities are derived from the message flows between participants and the activities that manipulate the data stores. Concerns are raised by the incompleteness of BPMN diagrams in regard to the identification of all activities that manipulate information stores in a data store and the correct linking of the participants to those activities. In conclusion, this approach may only partially identifies the relationships between the entities that represent the data stores in the target data model. Furthermore, due to the lack of information about participants, all entities that are derived from participants have the same attributes.

3.4 XML-Nets

XML-Nets were introduced by Lenz and Oberweis [Lenz and Oberweis(2001)] as an enhancement to Petri Nets. The BPML introduced in this project applies the concept of XSD-typed places from XML-Nets. With this feature, the source BPM holds the information about the data fields required for the attribute extraction in the target data model.

The XML-Nets approach enhances the BPM by allowing the incorporation of XSDs within a Petri net. This provides an additional layer of context to the Petri net, which can be used to model interorganizational workflows and specify the types of documents that each place can contain. Additionally, XML-Nets use a graphical modeling language (GXSL) to define XSDs and a query language called XManiLa to express transformations of documents in transitions.

By incorporating XSD-typed places from XML-Nets into the adapted BPML, the approach provides a way to specify the data fields required for attribute extraction in the target data model. This feature enhances the capabilities of existing approaches by providing additional context and structure to the BPM, resulting in a more comprehensive and precise understanding of the underlying data model in regard to the attributes.

In summary, the XML-Nets approach is an extension to Petri nets that allows for the modeling of complex interorganizational workflows and the specification of document types. By incorporating parts of this approach into the adapted business process modeling language, it is possible to provide a more complete and structured view of the underlying data model, resulting in a more accurate and effective approach to attribute extraction.

4 Designing an Approach for Data Model Extraction

This chapter establishes the foundation for the data model extraction approach utilized in this work, aiming to generate a target data model from a source BPM. The conception of this project is structured by considering the methodology, requirements for the source BPML, and the target data modeling notation. Furthermore, it explains the use of PNML as a DIF for the adapted BPML, the design of XML-Data-Nets (XDNs), the adapted BPML, and the data model extraction approach.

The methodology employed in this work is presented in Section 4.1, highlighting the systematic and linear nature of the waterfall method, which enables a step-by-step progression from requirements gathering to the final solution. Section 4.2 discusses the requirements for the source BPML and the target data modeling notation, respectively. This discussion provides clarity on the necessary components and characteristics a BPML must include to support the data model extraction approach and generate the target data model, as well as the semantic properties that are aimed to be reflected in the resulting target data model.

Section 4.3 introduces XDN, a BPML that extends the PNML High-Level Core Structure to support the specific requirements of the data model extraction approach. The use of PNML as the DIF for XDNs is examined in-depth, detailing the High-level Petri Nets Graph (HLPNG) type definition, its dependency tree, and the foundational elements of the PNML Core model. Additionally, the section covers the enhancements provided by the PNML High-Level Core Structure, allowing for the representation of more complex behavior in High-level Petri nets. Afterward, this section provides a comprehensive explanation of how token schemas are defined in places, role definitions are assigned to transitions, and cardinalities are specified using high-level elements for the definition of XDN in the PNML representation.

Lastly, Section 4.4 delves into the data model extraction algorithm, which includes three different groups of rules. These rules have been developed to extract the relevant information to map the information provided by an XDN model to a target data model. The section explains the rules and how they have been designed to work together.

4.1 Methodology

The Waterfall methodology is a traditional and widely-referenced software development approach that follows a linear and sequential process, where each phase of the project must be completed before moving on to the next one. This method is based on the principle of thorough planning, comprehensive documentation, and strict control over each phase. The waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development [Sommerville(2015)]. Figure 8 shows the waterfall model.

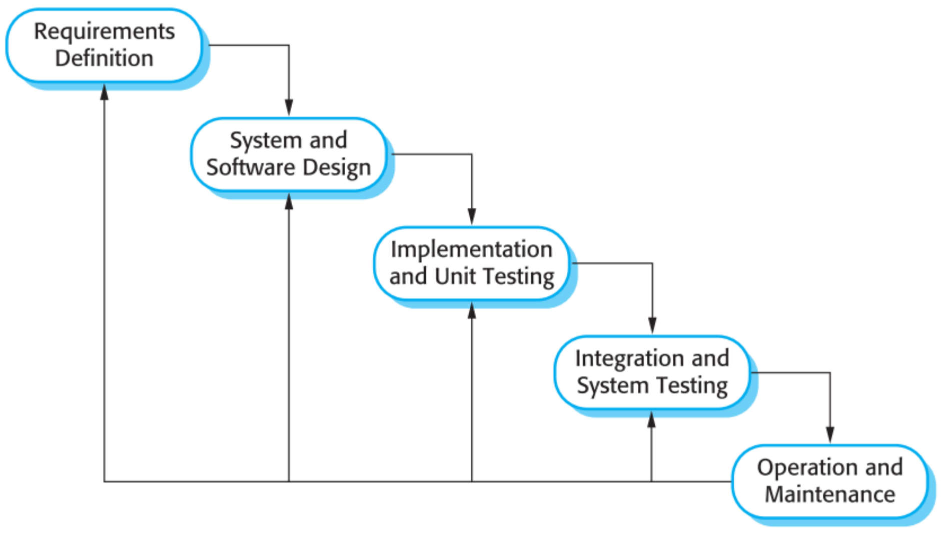

Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Figure 8: The waterfall model [Sommerville(2015)]

The Waterfall approach is characterized by several main phases: requirements analysis, system design, implementation, testing, operation, and maintenance [Sommerville(2015)]. Throughout these phases, developers and stakeholders work in a structured manner to ensure that the resulting software meets the predefined requirements and specifications.

1. **Requirements analysis and definition:** In this phase, project stakeholders collaborate to gather, analyze, and document the requirements for the software. This step is crucial in the Waterfall methodology, as it lays the foundation for the entire project and provides a clear understanding of the scope, objectives, and constraints.

2. **System and software design:** Based on the requirements gathered in the previous phase, a detailed system design is created, specifying the architecture, components, and data models to be used in the software. This design serves as a blueprint for the implementation phase.

3. **Implementation and unit testing:** In this phase, the system design is followed to build the software, with code being written and components integrated to realize the desired functionality. This step is focused on translating the design into a functional application.

4. **Integration and system testing:** Once the software has been implemented, it undergoes rigorous testing to identify and fix any bugs or issues. This phase ensures that the software meets the predefined requirements and functions as expected.

5. **Operation and maintenance:** After successful testing, the software is deployed to the target environment, making it available for use by end-users. Ongoing support is provided, addressing any issues that arise and making necessary updates or enhancements as required.

In summary, the Waterfall methodology is a structured and linear approach to software development that emphasizes thorough planning, comprehensive documentation, and strict control over each phase of the project [Sommerville(2015)]. By following a clear sequence of steps, the Waterfall method seeks to minimize risks and ensure that the final software product meets the project requirements. In the following, it will be discussed how the steps of this project can be adapted to align with the principles of the Waterfall methodology.

Based on the Waterfall methodology, in order to develop a sufficient prototype for the extraction of a data model from an adapted BPML, the following four steps, illustrated in figure 9, were taken.

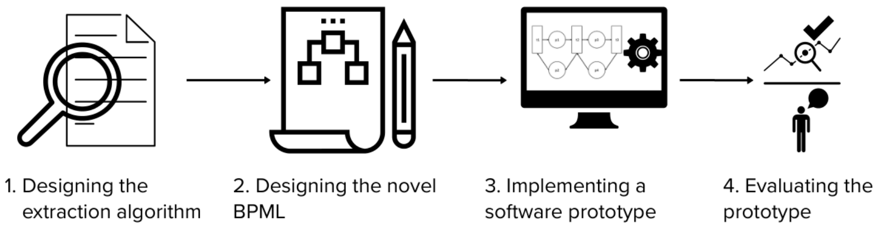

1. Designing the extraction algorithm
2. Designing the novel BPML
3. Implementing a software prototype
4. Evaluating the prototype

Figure 9: The 4 steps to develop the prototype for the extraction of a data model from an adapted BPML

Designing the extraction algorithm The first step involves the design of an extraction algorithm that can extract the target data model from a source BPM. To achieve this, existing BPMLs are analyzed, and a set of rules that can be used to extract the relevant information and transform it into the target data model notation is formulated. The rules are designed to capture the semantics of the process model and reflect them in the target data model. Concepts from related works as well as new concepts, are used to define the rules.

Designing the source BPML The second step involves designing the source BPML based on the extraction rules formulated in the previous step. This is achieved by identifying the key features and requirements that the language must support to ensure effective extraction of the target data model. A BPML is chosen and adapted so that it adheres to the requirements of the extraction algorithm.

Implementing a software prototype The third step involves implementing a software prototype that includes the concepts and rules developed in the previous steps. This is done by developing a user interface that enables users to visually model their business processes in the source BPML and implementing the data model extractor component that applies the extraction rules to generate the target data model notation and present it graphically.

Evaluating the prototype The final step involves evaluating the software prototype, including the design of the adapted BPML and its usefulness for target data model extraction. This is done by testing and analyzing the prototype's performance and usability using a real-world use case. The evaluation aims to demonstrate the effectiveness and efficiency of the source BPML and the associated data model extraction algorithm, as well as to identify areas for improvement. Furthermore, a qualitative evaluation is conducted with various users. The details about the evaluation process are outlined in chapter 6.

4.2 Requirements for the Data Model Extraction Approach

This section presents the requirements for the data model extraction approach, which aims to generate a target data model from a source BPML. The approach consists of two main components: the target data modeling notation and the source BPML. The section is organized into two chapters, providing an overview of the requirements and selection process for each component.

Chapter 4.2.1 focuses on the target data modeling notation, outlining the semantic properties that should be reflected in the resulting target data model. The requirements for the target data modeling notation are discussed, and the UML CD is justified as the appropriate notation for generating the logical data model.

Chapter 4.2.2 delves into the source BPML, providing a clear understanding of the necessary components and characteristics a BPML must include to support the data model extraction approach. The requirements for the source BPML are presented, and Petri Nets are selected as the suitable language for generating the logical data model.

Together, these chapters establish the foundation for the data model extraction approach, ensuring that the selected target data modeling notation and source BPML are capable of accurately and effectively representing the underlying structure and relationships within a logical data model.

4.2.1 The Target Data Modeling Notation

This section describes the requirements that have to be met by the target data model. The target data model must fulfill the requirements for the abstraction level of a logical data model as explained in chapter 2.2.1. Chapter 4.2.1.1 sets out the requirements for the target data model while chapter 4.2.1.2 justifies the selection of the target data model.

4.2.1.1 Requirements for the Target Data Modeling Notation

The target data modeling notation is essential for effectively representing the underlying structure and relationships within a logical data model. It must be expressive, consistent and provide a clear representation of the data model to support the data model extraction approach. This chapter outlines the primary requirements for the target data modeling notation, which will guide the selection of a suitable notation.

The target data model has to suffice the requirements of a logical data model. The abstraction of a logical data model was selected because the goal of this work is to provide a data model in the early phases of software development. In this phase, the specific database implementation may not be defined. Therefore, the specification of the additional features provided by a physical data model would be complex because of the multiplicity of data types that can be possible.

The notation should therefore support the following concepts:

- classes
- attributes
- attribute data types
- relations
- primary keys
- foreign keys

4.2.1.2 Selection of the Target Data Modeling Notation

Based on the outlined requirements, this chapter will justify the selection of the UML CD as the target data modeling notation for generating the logical data model.

The UML CD on the abstraction level of a logical data model meets all of the requirements as described in chapter 4.2.1.1. Therefore it is used as the notation for the target data model in this work. The UML CD was also selected for the reason that the quantitative evaluation (see chapter 6) is performed by comparing it to a manually created UML

CD. By using the same data model notation, syntactical comparability between the data models is ensured.

4.2.2 The Source BPML

The existing BPMLs introduced in chapter 2.1 and those used in the approaches presented in chapter 3 have limitations when it comes to representing the specific needs and requirements for data model extraction. This can lead to difficulties in accurately capturing the semantics of the process and extracting meaningful data from it. In this chapter, the requirements a source BPML should meet to address some of these limitations and select a suitable base language that the source BPML can be derived from are discussed.

4.2.2.1 Requirements for the Source BPML

The source BPML was created by adapting an established BPML. The selection of this BPML was carried out based on the criteria **semantic content of the BPML**, **complexity of the DIF** and **backward compatibility of the BPML**. These criteria were chosen because they are important factors to consider when adapting an existing BPML to create a source BPML for data model extraction. The **semantic content of the BPML** is important because it determines how well the language can represent the specific needs and requirements for data model extraction. **The complexity of the DIF** is important because it affects how difficult it is to extract meaningful data from the process. **Backward compatibility of the BPML** is important because it ensures that existing models created with the original BPML can still be used with the adapted version. The following describes the criteria in detail, followed by the resulting decision on the choice of the basic BPML used as a foundation for the creation of the source BPML for data model extraction.

The **semantic content of the BPML** refers to the information, specifically the semantic properties of the elements, the BPM holds for the purpose of data model extraction. These elements need to reflect the elements of the target data model, presented in chapter 4.2.1.1. They include the following:

- Objects: Objects in the business process represent states or entities.
- Operations: Operations in the business process represent actions that change the state of objects.
- Associations: Associations capture and convey the meaning and relationships between the different aspects of the business process being modeled and describe the dependencies or relationships between objects and operations.

- Roles: Roles are useful for the extraction of classes in the data model, which describes a role, person, or organization. For example, in the BPMN language, roles are represented as swimlanes.
- Cardinalities: For the generation of cardinalities between the classes in the target data model, the BPML has to express the relation between different process elements by using a quantitative weight as a relation.
- Fields: The BPM should contain information about the fields of a data object used in the respective process step. This information can be used when generating the attributes of a class in the data model.

The **complexity of the DIF** refers to the scope, as well as the maturity of the DIF. Since the DIF is primarily used as a basis for adapting the BPML, the DIF should be as small as possible, i.e., it should tend to contain fewer definitions of elements. The maturity of the DIF should be at an advanced stage, meaning the DIF is well documented, and provides examples of the implementation.

Finally, the BPML, and thus the DIF, should still be **backward compatible** with the original format after adapting it to fit the additional requirements for data model extraction. This can be achieved by using a DIF whose customization only requires adding new features to the DIF instead of changing existing properties of the base format.

4.2.2.2 Selection of the Source BPML

In this work, Petri nets are used as the base language from which the adapted source language is derived. Some concepts of XML-nets, mainly the use of XSD to define the schema of valid tokens in places, are also included in the definition of the customized BPML. From this point of view, XML-nets are very close to the definition of the needed BPML. However, since XML nets do not support the definition of roles for transitions, it is necessary to extend the concept. Furthermore, the specification of XML-nets is very abstract, which is why this project focuses mainly on PNML as the basis for the BPML. The result is a high-level variant of Petri nets to be used as a source language for the extraction of data models. Petri nets contain a comparatively limited selection of process elements. Of the criteria presented regarding the semantic content of the BPML, Petri nets fulfill the requirements for objects (places), operations (transitions), associations (arcs), and cardinalities (arc weights). The DIF for Petri nets, called PNML, is well documented and has multiple publications, documentations, and examples[1]. The different grammar models are documented in the REgular LAnguage for XML Next Generation (RNG) syntax, a similar but, by today's standards, not as popular format as XSD. By using

[1]https://www.pnml.org/

a DIF built on top of PNML, a XML-based standard for Petri net model interchange, for serializing the adapted Petri net model, backward compatibility with the High-level Petri nets Core Structure grammar is ensured. However, this is subject to the condition that only new tags are introduced. Changing or removing existing tags breaks backward compatibility.

4.3 XML-Data-Nets

This chapter introduces a BPML, XDNs, which is a type of High-level Petri Net that allows for a more sophisticated data model extraction on instances of XDNs. The graphical representation used to visualize XDN models is presented in section 4.3.1. Section 4.3.2 then deals with PNML, the DIF on which XDN is built. The chapter serves to clarify to the reader which PNML concepts are used to build XDN and how it differs from the HLPNG grammar. Section 4.3.3 then proceeds to outline the features that were used to define XDN in its PNML representation.

4.3.1 Graphical Representation of XML-Data-Nets

This chapter shows how XDNs models are represented graphically.

A XDN consists of places, transitions, and arcs. Arcs are directed associations that connect a place and transition with each other. Places are graphically visualized by circles, transitions by rectangles, and arcs by directed arrows.

Places are containers that can be interpreted as a data store, memories, or a state within the business process. The places contain tokens that can hold specific information about the data stored in the token. For this purpose, a token may have a name, any number of superclasses, and any number of attribute elements assigned. Each attribute element assigned to the token is further specified by a data type and a boolean value indicating if the attribute is part of the primary key of the data structure stored in the token.

Transitions represent activities in the business process. They have a pre-set and a post-set of places, each consisting of the set of incoming and outgoing places of the transition, respectively. A transition has the ability to fire, which means, that it consumes a token from each place of its pre-set and generates a token in each place of its post-set. This firing of the transition can be interpreted as a generation of data based on some input data which is consumed.

The arcs that connect places and transition represent the flow of the process and, therefore, the movement of the tokens across the process. Each arc is further specified by an arc weight, which can hold the value “1” or “*” and is visually represented by the respective character on the directed arrow. The weight of an arc specifies the number of tokens that are consumed or generated in places in the event of a connected transition firing.

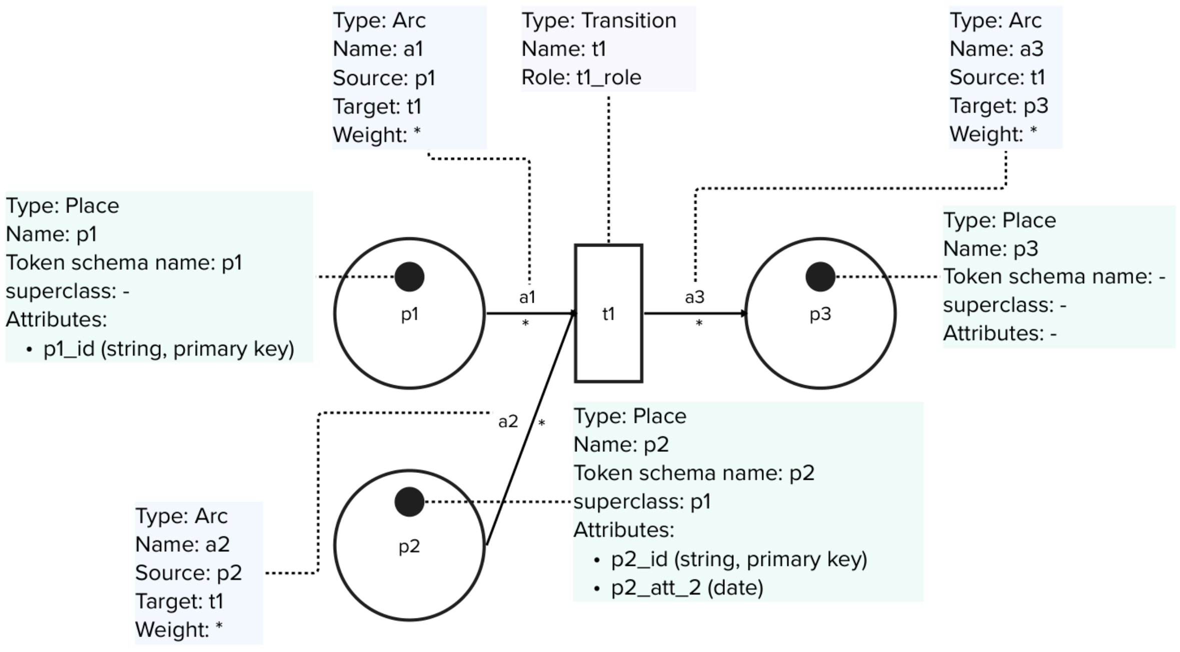


Figure 10: Example of a basic XDN model.

Figure 10 shows a simple XDN model with three places, one transition, and three arcs. The arcs connect the places to the transitions. The example shows a possible configuration of a XDN model. For example, the place p1, visualized by a circle with the text p1 in it, has a token assigned, which is further specified with a token name “p1” and an attribute “p1_id”. The place p2 has a token name “p2”, the superclass “p1”, and two attributes specified, one of them being a primary key.

The transition t1, visualized by a rectangle, has the name “t1” and a role “t1_role” specified.

Figure 10 also contains three arcs. For example, arc a1 is visualized by a directed arrow from the place p1 to the transition t1. The arc also visualizes its weight specification by a “*” displayed on the directed arrow.

4.3.2 PNML as the Diagram Interchange Format

This chapter presents PNML as the DIF for the previously introduced BPML, XDN. By utilizing PNML and its HLPNG type definition, XDN can efficiently represent complex business processes. This chapter provides an overview of PNML, its HLPNG type definition, and the dependency tree of the HLPNG type definition. It also highlights the key elements and attributes introduced in the PNML Core model and the PNML High-Level Core Structure, which play a crucial role in supporting the expressiveness and extensibility of the XDN language.

PNML is an XML-based interchange format for Petri nets (as introduced in 2.1.1), which describes different grammar models. The HLPNG also includes other grammars as a dependency.

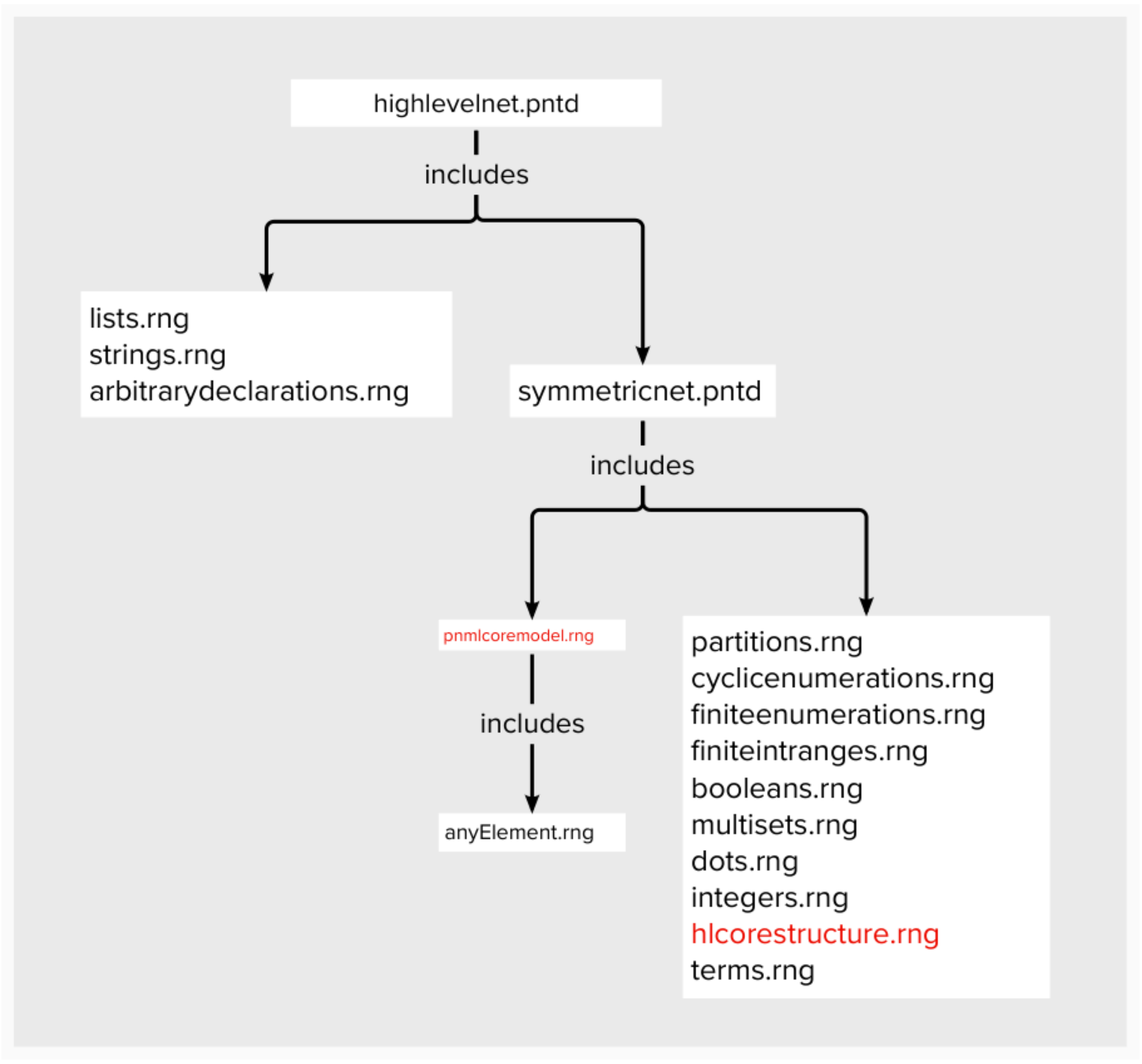


Figure 11: Dependency Tree of the PNML HLPNG

Figure 11 shows the dependency tree of the HLPNG type definition. These dependencies can be traced by following the dependencies in the grammar files starting from HLPNG. Except for some RNG declarations, the HLPNG inherits all dependencies of the Symmetric Net. The Symmetric Net, in turn, contains the PNML Core model and the High-Level Core Structure, among other definitions. These two RNG declarations contain the main XML schemas used in XDN.

The **PNML Core model** is the foundational model of the PNML. It defines a set of elements and attributes that can be used to describe Petri nets. The elements include:

- Net: This is the top-level element that represents the entire Petri net. It contains one or more places, transitions, and arcs.
- Page: This element serves as a container for a section of the Petri net.
- Place: This element represents a place in the Petri net, which can hold a certain number of tokens.
- Transition: This element represents a transition in the Petri net, which can fire and change the number of tokens in connected places.
- Arc: This element represents a directed arc between a place and a transition or between a transition and a place.

In addition to these elements, the PNML Core model defines several attributes that can be used to specify additional information about the Petri net, such as its name, type, and author. The PNML Core model is a simple model that can be used to represent a wide range of Petri nets. However, it is also extensible, which means that it can be customized to include additional information or properties specific to a particular type of Petri net or application domain using the “toolspecific” tag.

```
<place id="p1">
    <toolspecific tool="dme" version="1.0.0">
        <tokenSchema name="Schadteil">
            <xs:element name="referenznummer"/>
            <xs:element name="Hauptverantw."/>
        </tokenSchema>
    </toolspecific>
    <!-- additional place attributes -->
</place>
```

Listing 1: Usage of the “toolspecific” tag.

Listing 1 shows an example of a “toolspecific” element. This element can be the child of a place, transition, or arc and can contain any type of content that can be handled by the specified tool.

The **PNML High-Level Core Structure** extends the PNML Core model, enabling the representation of High-level Petri nets. High-level Petri nets allow the use of data types, complex expressions, and advanced constructs beyond the basic marking of places and transitions in basic Petri nets. The High-Level Core Structure enhances the PNML Core model by including components and attributes that allow Petri nets with more complex behavior to be defined.

Key elements introduced in the PNML High-Level Core Structure include:

- hlinscription: This element is used to represent arc weights and other expressions associated with High-level Petri nets. It is a child element of the arc element and can contain different types of expressions, such as constants, variables, or functions.
- type: This element allows the definition of data types for places, transitions, and arcs. It is crucial for describing the structure of tokens and the behavior of transitions in High-level Petri nets.
- operator: This element is used to define complex operations or functions that can be applied to tokens or expressions in High-level Petri nets.

The High-Level Core Structure also enables the use of advanced constructs like guards, enabling functions, and complex arc expressions. These constructs provide better control over firing transitions and manipulating tokens in a High-level Petri net.

In summary, the PNML HLPNG type definition serves as the foundation for defining and serializing XDNs, enabling the representation of complex behavior in High-level Petri nets. The PNML Core model and the PNML High-Level Core Structure provide the necessary elements and attributes for constructing business process models in XDN to be introduced in chapter 4.3.3.

4.3.3 PNML Definition of XML-Data-Nets

To create the DIF for XDN, the PNML High-Level Core Structure was used as a basis to support the specific requirements of the data model extraction approach. XDN is designed to support all required elements as presented as the semantic content of the BPML in chapter 4.2.2. The PNML High-Level Core Structure is already able to represent *Operations* (Transitions) and *Associations* (Arcs). To support *objects*, *roles*, *cardinalities* and *fields*, special features were added, which include:

1. **Token schema definition**: A new element, “tokenSchema”, has been introduced within the “toolspecific” tag for places. This element allows users to define the schema of tokens in a place using the XML schema notation. Defining token schemas allows users to explicitly specify the structure and attributes of the tokens stored in each place. This feature represents the *objects* and *fields* in the BPM.

   The reason why the “tokenSchema” can effectively represent a class in the target data model lies in the interpretation of places as memories within the process model. In this context, places act as repositories for storing objects, represented by token schemas, which in turn encapsulate the structure and attributes of the corresponding classes. As these token schemas embody the class schemas, they serve as blueprints for creating instances of those classes. By associating each place in the process model with a token schema, it becomes possible to generate the target data model classes from the process model itself.

2. **Role definition for transitions**: A new element, “role”, has been introduced within the “toolspecific” tag for transitions. This element allows users to assign roles to transitions that can be helpful in determining the system, user, or organization responsible for firing the transition. These roles can be used by the extraction algorithm to create additional classes and associations in the target data model. This feature represents the *roles* in the BPM.

   The reason why a transition can effectively represent a role in the target data model is due to the fact that activities within a process model can be executed by various entities, such as persons or organizations, which represent roles. Transitions in the process model symbolize these activities and as such, can include a role specification. By associating each transition with a role, it becomes possible to create a more comprehensive understanding of the responsibilities and interactions between different entities in the process model. Consequently, this information can be used to generate data model classes from the process model that accurately reflect the relationships and dependencies between various roles and other objects in the system.

3. **Cardinality inscriptions for arcs**: The existing “hlinscription” element is used to support cardinality inscriptions for arcs. This allows users to specify the minimum and maximum number of tokens that can flow through an arc, providing additional control over the flow of tokens in the net and, ultimately, the cardinalities in the target data model. This feature represents the *cardinalities* in the BPM.XDN restricts the content of cardinality inscriptions to be either “1” or “*”.

   The reason why arcs have a weight assigned in the process model is to indicate the quantitative association between elements, such as places and transitions, within the process model. By assigning arc weights to each arc in the process model, it becomes possible to define how many objects of the pre-set must be consumed to generate a specific number of objects in the post-set of a transition. This added layer of information allows for a more precise representation of the relationship between the involved elements and helps to accurately model the flow of tokens throughout the process.

   When generating associations for classes in the data model, these arc weights can be utilized to establish meaningful cardinalities between the corresponding classes. By considering the weights of arcs, the extraction algorithm can generate associations that better reflect the actual relationships and constraints present in the BPM.

These specifications for the PNML High-Level Core Structure form the basis for XDN specifically designed for data model extraction from BPMs. By including token schema definitions, role definitions for transitions, and cardinality inscriptions for arcs, XDN provides a more expressive language tailored to the requirements of the data model extraction algorithm.

4.3.3.1 Token Schemas in Places

To define custom elements in PNML, the “toolspecific” tag can be utilized. This section explains how to use the “toolspecific” element to define token schemas for places in XDN. It also covers how the XML namespace “xmlns:xs” is utilized to define the child elements of the token schema, representing the attributes of the associated data object.

The “toolspecific” element allows users to define custom attributes and elements that are specific to their application or tool. In the case of XDN, this flexibility is used to define token schemas for places in BPM. Token schemas represent the structure and attributes of tokens stored in a specific place. They are essential to the data model extraction process because they provide the information necessary to create classes and attributes for the target data model.

To define a token schema, a “tokenSchema” element nested inside a “toolspecific” element is created. The “tokenSchema” element has the following attributes:

- name: A unique identifier for the token schema.
- superClass: An attribute to define a list of superclasses for the token schema. This allows the creation of hierarchical data structures where the token schema can inherit properties from the superclasses. The superclass can also be an empty string if no inheritance relationship is defined.

The token schema can also have 0 to n child elements. These child elements are defined using the “xs:element” notation from the XML Schema namespace[2]. Each “xs:element” child has the following attributes:

- name: The name of the attribute represented by this element.
- type: The data type of the attribute, specified with one of the XSD types.
- isPrimaryKey: A boolean value that indicates whether the attribute is a primary key. If it is not specified, it defaults to "false".

[2]http://www.w3.org/2001/XMLSchema

```
<place id="p1">
    <toolspecific tool="dme" version="1.0.0">
        <tokenSchema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            name="Routing Request" superClass="">
            <xs:element name="RR_ID" type="xs:string"
               isPrimaryKey="true"/>
        </tokenSchema>
    </toolspecific>
    <!-- additional place attributes -->
</place>
```

Listing 2: Usage of the “tokenSchema” tag.

Listing 2 shows an example of a token schema defined within a “toolspecific” element. In this example, a token schema with the name “Routing Request” and no superclass is defined. The schema has one attribute, “RR_ID”, which is of type “xs:string” and is a primary key.

By partially using token schemas, XDN allows users to explicitly define the structure and attributes of tokens in BPM. This information is crucial to the data model extraction process, as it helps the algorithm accurately map the BPM elements to the target data model.

4.3.3.2 Role Definitions in Transitions

This section discusses the use of the “toolspecific” element to define roles for transitions in XDN.

Roles are important in XDN as they represent the responsible entities for the generation of tokens by a firing transition. By associating roles with transitions, it is possible to specify which entity is responsible for executing the corresponding task in the process and therefore generating a token associated with a place.

To define a role for a transition, the “toolspecific” element is used with a “role” child element. The “role” element has a child element “text”, which contains the name of the role as text content.

Listing 3 shows an example of a role definition within a “toolspecific” element. In this example, a role named “t1_role” is defined for the transition. The role is specified within the “toolspecific” element with its child element “role” and its child element “text” containing the role name.

```
<transition id="t1">
    <toolspecific tool="dme" version="1.0.0">
        <role>
            <text>t1_role</text>
        </role>
    </toolspecific>
    <!-- additional transition attributes -->
</transition>
```

Listing 3: Usage of the “role” tag.

By using role definitions in transitions, XDN allows to explicitly assign responsibilities to functions involved in the business process. This information can be valuable for a more accurate mapping of BPM elements to the target data model during the data model extraction process.

4.3.3.3 Cardinalities with High-Level Inscriptions for Arcs

In this section, the use of the “hlinscription” tag from the High-Level Core Structure to add weights (which later translate to cardinalities in the target data model) to the arcs in a XDN is discussed.

In the High-Level Core Structure, elements such as places, transitions, and arcs can have attributes and elements that allow the definition of complex behavior. One such element is “hlinscription”, which is used to represent arc weights and other expressions associated with High-level Petri nets. The “hlinscription” tag is utilized to specify the weight or expression associated with an arc. The “hlinscription” element is a child element of the arc element and may contain various types of expressions, such as constants, variables, or functions.

```
<arc id="a1" source="p1" target="t1">
  <hlinscription>
    *
  </hlinscription>
  <!-- additional arc attributes -->
</arc>
```

Listing 4: An arc with a “hlinscription”.

Listing 4 demonstrates the use of the “hlinscription” tag within an arc element in a High-level Petri net modeled with the PNML High-Level Core Structure. The arc element connects a source (place) to a target (transition). The “hlinscription” tag is nested within the arc element and is used to specify the weight or expression associated with the arc.

The weight (1 or *) is placed inside the “hlinscription” tag.

By using the “hlinscription” tag to define weights for arcs, XDN allows users to specify cardinalities for arcs in the net. This information is essential for accurately modeling the behavior of business processes and can contribute to a more accurate mapping of the BPM elements to the target data model during the data model extraction process.

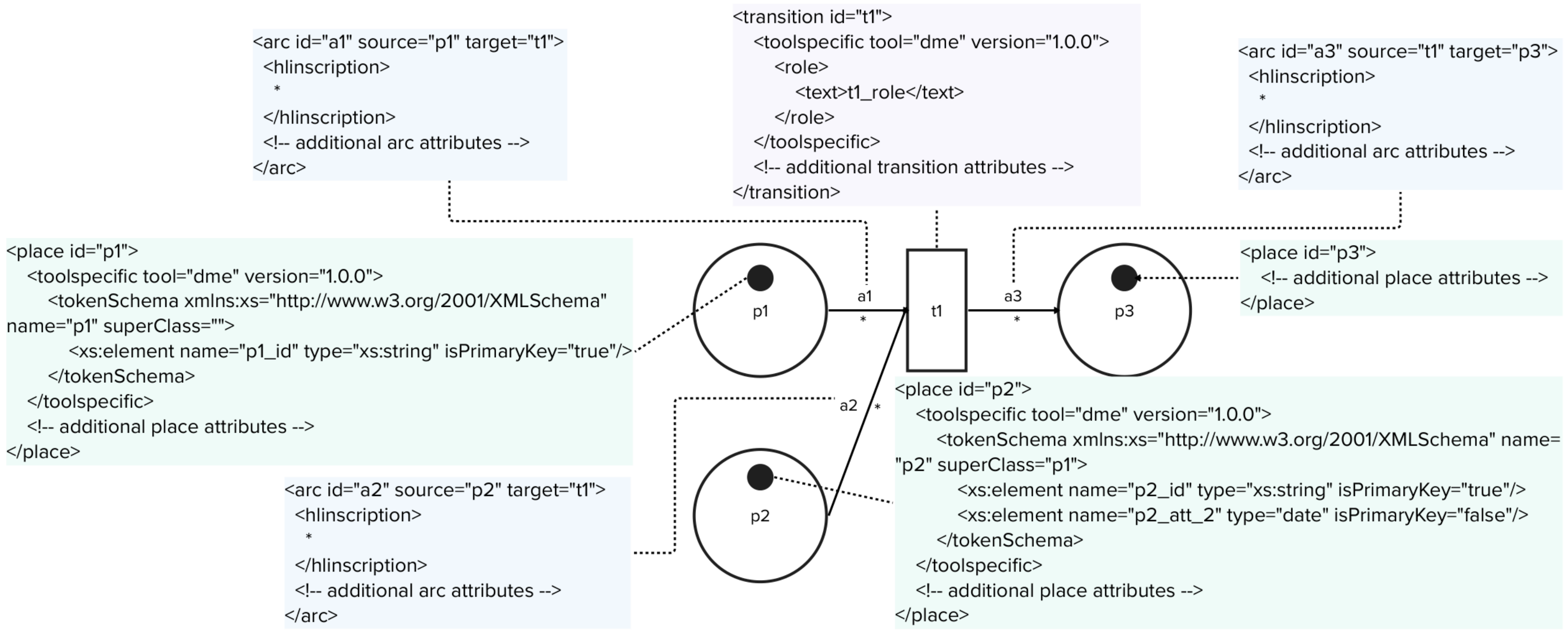
4.3.3.4 Example of an XDN model

Figure 12 shows a simple XDN model with three places, one transition, and three arcs. The arcs connect the places to the transition as shown in figure 12. This example shows the graphical representation of a XDN with the underlying defined concepts of token schemas, roles, and hlinscriptions.

The PNML representation is associated with the elements by the boxes connected by dashed lines. Thus a token schema can be stored in the toolspecific element within a place (see places p1, p2, and p3). Respectively a role definition can be stored inside the toolspecific element of the transition t1. Arcs use the hlinscription element from the High-Level Core Structure of PNML to store the cardinality inscription. The cardinality is also graphically visualized on the arc element in the XDN model.

4.4 An Extraction Algorithm to Extract a Logical Data Model From a BPM

This chapter presents an extraction algorithm to extract a logical data model, specifically a UML CD from a BPM. In order to generate a complete UML CD, a set of rules was designed based on existing works ([Brdjanin and Maric(2012)] and [Cruz et al.(2012)]) and iterative testing. These rules describe the extraction of a UML CD from an XDN model as shown in figure 13.



```
<arc id="a1" source="p1" target="t1">
 <hlinscription>
  *
 </hlinscription>
 <!-- additional arc attributes -->
</arc>
```

```
<transition id="t1">
  <toolspecific tool="dme" version="1.0.0">
    <role>
      <text>t1_role</text>
    </role>
  </toolspecific>
  <!-- additional transition attributes -->
</transition>
```

```
<arc id="a3" source="t1" target="p3">
 <hlinscription>
  *
 </hlinscription>
 <!-- additional arc attributes -->
</arc>
```

```
<place id="p1">
  <toolspecific tool="dme" version="1.0.0">
    <tokenSchema xmlns:xs="http://www.w3.org/2001/XMLSchema"
name="p1" superClass="">
      <xs:element name="p1_id" type="xs:string" isPrimaryKey="true"/>
    </tokenSchema>
  </toolspecific>
  <!-- additional place attributes -->
</place>
```

```
<place id="p3">
  <!-- additional place attributes -->
</place>
```

```
<arc id="a2" source="p2" target="t1">
 <hlinscription>
  *
 </hlinscription>
 <!-- additional arc attributes -->
</arc>
```

```
<place id="p2">
  <toolspecific tool="dme" version="1.0.0">
    <tokenSchema xmlns:xs="http://www.w3.org/2001/XMLSchema" name=
"p2" superClass="p1">
      <xs:element name="p2_id" type="xs:string" isPrimaryKey="true"/>
      <xs:element name="p2_att_2" type="date" isPrimaryKey="false"/>
    </tokenSchema>
  </toolspecific>
  <!-- additional place attributes -->
</place>
```

Figure 12: Example of a basic XDN model with the underlying PNML representation of the model.

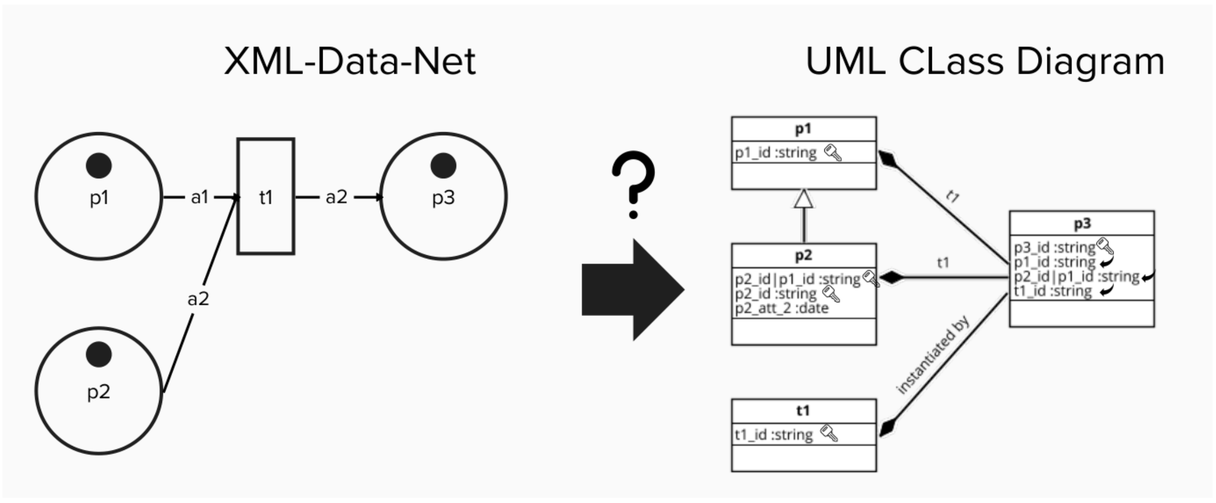


Figure 13: How to design rulesets in order to extract a UML CD from a source XDN?

Figure 14 shows the three sets of rules the algorithm uses to translate the semantic content of an XDN into elements of the target UML CD.

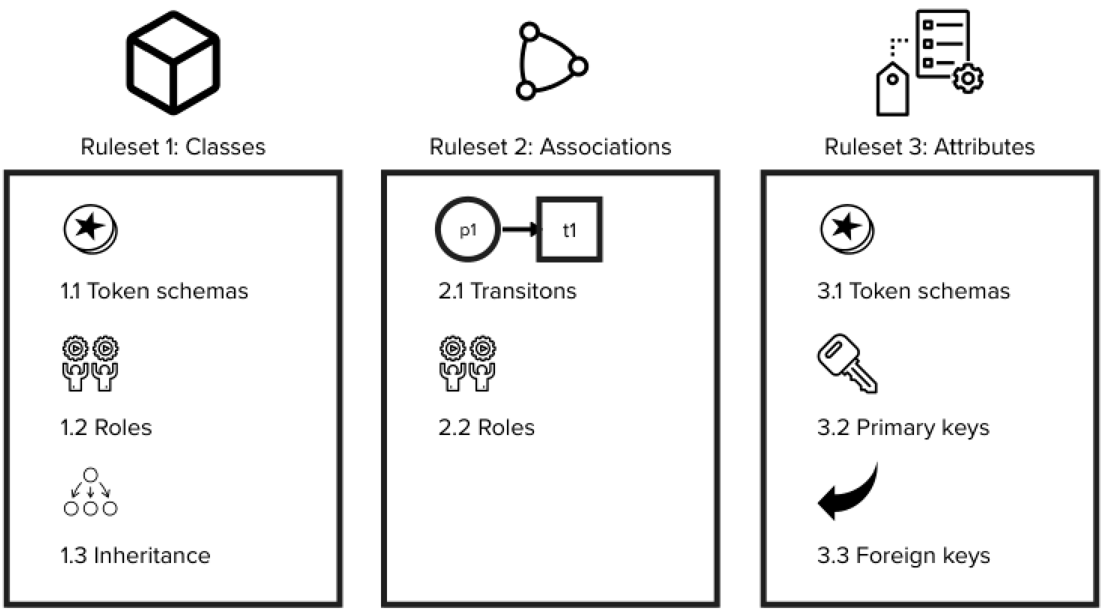


Figure 14: Overview of the three rulesets for the extraction of a CDM from a BPM

Ruleset 1: Classes - Consists of three sub-rules that define how to generate classes in the UML CD from the information in the XDN model. These sub-rules cover the generation of classes from token schemas and roles, the aggregation of classes with the same name, and the inheritance relationships between classes.

Ruleset 2: Associations - Contains two sub-rules that guide the generation of associations between classes in the UML CD based on the information provided in the XDN model.

The sub-rules describe how to create composition relationships based on the arcs around transitions and the relationships between roles and their post-set places.

Ruleset 3: Attributes - Focuses on generating attributes for the classes in the UML CD based on the information provided in the XDN model. This ruleset contains three sub-rules that address token schemas, primary keys, and foreign keys to ensure that the extracted UML CD accurately represents the data model derived from the XDN model.

4.4.1 Ruleset 1: Classes

The first ruleset in the extraction algorithm focuses on generating classes in the UML CD from the information contained within the XDN model. This ruleset contains three sub-rules. Figure 15 shows how the three sub-rules are applied to a basic XDN model to extract the classes from it.

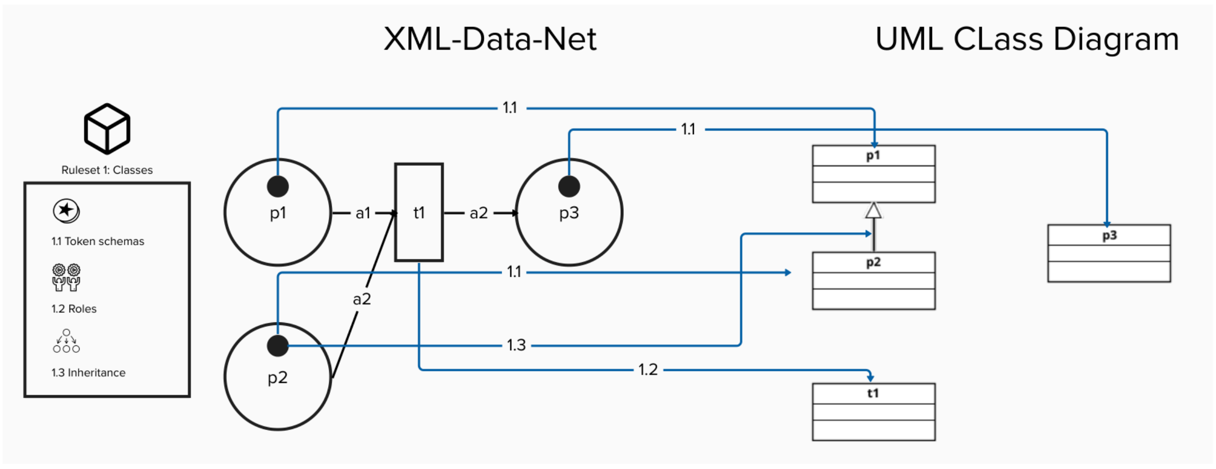


Figure 15: Extraction of classes from a XDN.

Rule 1.1: Every distinct token schema of a place results in a class in the data model. This rule states that for every unique token schema defined within a place in the XDN model, a corresponding class should be created in the target UML CD. If a token schema is not explicitly defined for a place, the place's name will be used as the token schema name. This is because places represent data objects in the process, and token schemas only provide a way to further define the structure, attributes, and inheritance relationships of these objects. This rule covers the generation of all classes in the data model derived from places.

Rule 1.2: Every distinct role of a transition results in a class in the data model. According to this rule, each unique role associated with a transition in the XDN model will result in a class being created in the target UML CD. When two classes generated by Rule 1.1 or 1.2 share the same name in the target data model, they will be aggregated into a single class in the resulting diagram. This approach allows for further specification of a class derived from a role by defining a token schema with the same name in the net. A similar rule was presented in Brdjanins' and Marics' work [Brdjanin and Maric(2012)] on the automated extraction of participants where each role represented by a swimlane in a UML AD leads to the creation of a class. By applying this concept to XDNs, every role defined within a transition leads to the creation of a class in the target data model.

Rule 1.3: Place classes can be inherited from already defined classes. This rule states that a class in the target UML CD, generated from a token schema or a role, can inherit properties from other classes that were already defined in the model. However, there are some important limitations:

- A class cannot inherit from itself.
- Abstract classes cannot be defined or extracted within this framework because the XDN model only holds instantiated data objects. Since token schemas are always concrete tokens, abstract classes cannot be used as token schemas for places.

This ruleset was introduced to cover the representation of inheritance relationships in the target UML CD.

In summary, Ruleset 1 focuses on defining the classes within the UML CD based on the XDN model's places and roles. The three sub-rules cover the generation of classes from token schemas and roles, the aggregation of classes with the same name, and the inheritance relationships between classes.

4.4.2 Ruleset 2: Associations

This ruleset focuses on generating associations between classes in the UML CD based on the information provided in the XDN model. There are two sub-rules in this ruleset. Figure 16 shows how the two sub-rules are applied to a basic XDN model to extract the associations from it.

Rule 2.1: Transition Associations. Arcs surrounding transitions in the XDN model translate into a composition relationship with the arc's cardinalities between the related classes in the data model. This relationship is a composition because the target object is dependent on the source object when it is generated by the transition. The cardinalities defined in the arcs provide essential information about the multiplicity of the relationships in the resulting class diagram.

Rule 2.2: Role Associations Each role in the XDN model creates a "1:n" composition relationship to the classes generated from its post-set places. These relationships represent

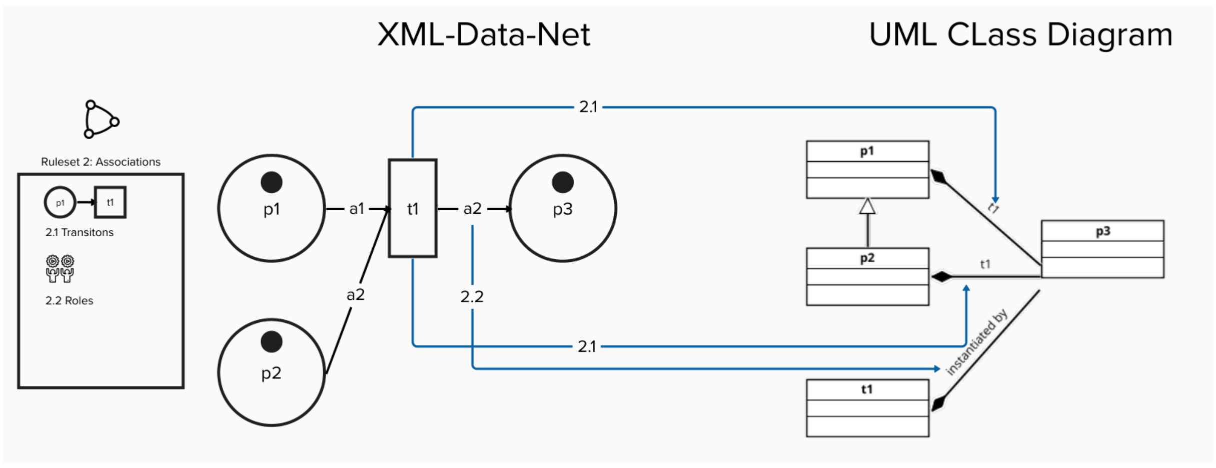


Figure 16: Extraction of associations from a XDN.

the association between roles and the data objects they generate, manipulate, or consume in the business process. A similar rule was defined by Cruz et al. [Cruz et al.(2012)] to relate participants with classes in the data model.

The extraction algorithm supports multiple associations between classes, as it uses the transition names for the association names in the UML CD. Only identical duplicate associations (i.e., having the same source, target, cardinality, and name) are omitted to avoid redundancy in the UML CD.

In summary, Ruleset 2 provides guidelines for generating associations between classes in the target UML CD, derived from the transitions and roles present in the XDN model. The two sub-rules describe how to create composition relationships based on the arcs around transitions and the relationships between roles and their post-set places. This ruleset ensures that the extracted UML CD accurately represents the relationships between data objects and roles within the business process, leading to a more precise representation of the underlying data model.

4.4.3 Ruleset 3: Attributes

Ruleset 3 focuses on generating attributes for the classes in the UML CD based on the information provided in the XDN model. It consists of three sub-rules that cover token schemas, primary keys, and foreign keys. Figure 17 shows how the three sub-rules are applied to a basic XDN model to extract the attributes, primary keys, and foreign keys from it.

Rule 3.1: Token Schemas. In the XDN model, places hold token schemas in XSD format. These token schemas provide detailed specifications for the structure and content of the data objects represented by the places. The extraction algorithm interprets these token schemas as classes in the UML CD. The structure of those tokens is introduced in

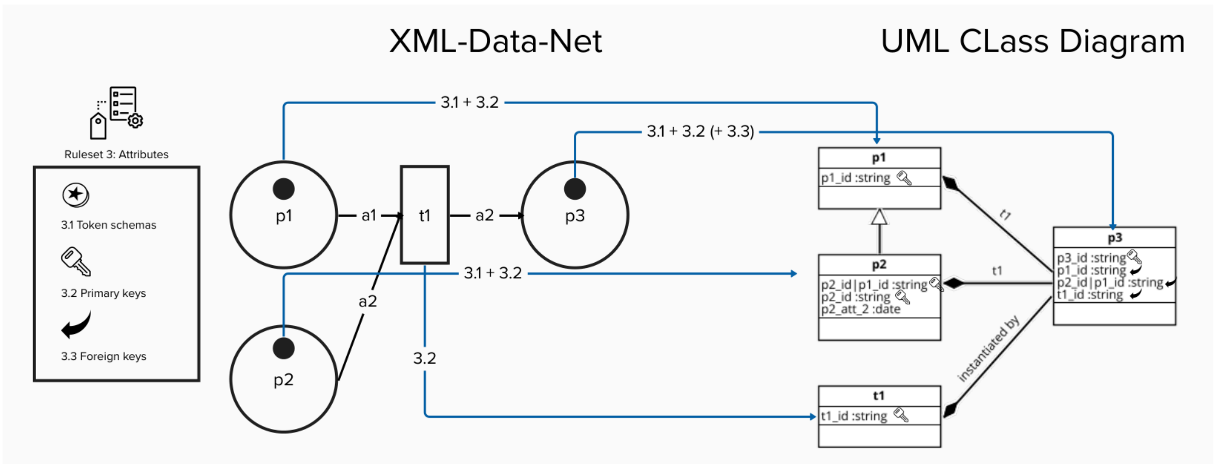


Figure 17: Extraction of attributes, primary keys, and foreign keys from a XDN.

chapter 4.3.3.1.

Rule 3.2: Primary Keys. Every class in the UML CD must have a unique identifier, known as a primary key. The user can specify one or more attributes as primary keys for the token schema. If multiple primary keys are defined, the extraction algorithm combines all primary keys to create a composite primary key for the class. The primary keys of all superclasses of the class are also considered during this process.

When circular references regarding inheritance are encountered, the extraction algorithm resolves superclass inheritance from top to bottom, inheriting only one time from each superclass.

If a primary key is not explicitly defined for a token schema, the algorithm assumes a default primary key with the format className_id.

Rule 3.3: Foreign Keys. The extraction algorithm automatically generates foreign keys in related classes to establish relationships between them. Foreign keys reference the primary keys of other classes, allowing for data consistency and referential integrity across the UML CD.

The algorithm identifies related classes based on the associations defined in Ruleset 2 and creates foreign keys in the source class that correspond to the primary key in the target class. It then establishes an association between the target primary key and the source foreign key.

By following these three sub-rules, the extraction algorithm generates a UML CD with well-defined attributes, primary keys, and foreign keys that represent the data model derived from the XDN model.

The rules presented here cover the extraction of classes, associations, and attributes of the UML CD. Thus, the methods of the UML CD classes were not treated. To extract the methods from the XDN, it would be conceivable to use the transition expression of XML-Nets, which would have to be integrated into XDN. Since UML CDs with method definitions were not used in any of the iterative scenarios to create the rules, a ruleset that handles the method generation is omitted. In addition, the quantitative evaluation (see 6) is also based on an UML CD without method definitions, so this concept was neglected in this work.

5 Realization

In this chapter, the practical implementation of the concepts discussed in chapters 4.3 and 4.4 is demonstrated, focusing on the development of a software prototype for visually modeling XDNs and extracting UML CDs from them. The chapter delves into the architecture of the software implementation and the creation of a prototype, which serves as an example of the data model extraction concept presented in this work.

Section 5.1 explores the architecture of the software implementation, designed to support the visual modeling of business processes as XDNs and the execution of the defined ruleset for data model extraction on XDNs to accurately represent the target data model. Following this, Section 5.2 discusses the development of an Angular-based web application prototype that implements the concept of data model extraction, which serves as the foundation for evaluating the proposed approach. In Section 5.3, an example is provided to demonstrate how a UML CD is extracted from an XDN, showcasing the practical application of the concepts presented in the previous chapters.

Overall, this chapter presents a software artifact that allows users to model business processes in the source BPML, XDN, and extract data models from them.

5.1 Architecture of the Software Implementation

In order to create a software artifact for the execution and evaluation of the data model extraction concept in this work, a prototypical application is presented within the scope of this work. This chapter explains the technology used and the architecture of the application.

Selecting an appropriate web application framework is crucial for the successful development of a prototype project. This section outlines the reasoning behind choosing the Angular front-end web application framework for the development of a client-side prototype application. The prototype application does not include a backend and operates solely on the client side in the browser. The key factors that influenced the decision to develop a web application, particularly using Angular as the main framework, are its popularity, accessibility in terms of ease of access, and cross-platform compatibility.

Popularity Angular is a widely-used and popular front-end web application framework, which ensures an extensive community (see [Cincović and Punt(2020)]).

Accessibility Secondly, Angular offers accessibility in the sense that web applications built using this framework can be easily accessed from anywhere with an internet connection. This makes it easier to share and gather feedback from a wide audience, which is essential for refining the application and ensuring its effectiveness.

Cross-platform Compatibility Lastly, Angular's cross-platform compatibility is a significant advantage. The framework supports the development of applications that can run on various platforms, including web, mobile, and desktop. This is achieved through the use of web standards, such as HTML, CSS, and JavaScript. This cross-platform compatibility ensures that the application can reach a wide range of users and devices, making it more versatile and accessible. By selecting Angular as the main framework for the prototype application, it is possible to enhance the application's reach by developing the application with a responsive layout.

In conclusion, Angular was chosen as the framework for this project due to its popularity, accessibility in terms of ease of access, and cross-platform compatibility. These factors contribute to the development of an effective and user-friendly application that can be easily adapted and improved as needed. In the next section, the architecture of the application will be discussed.

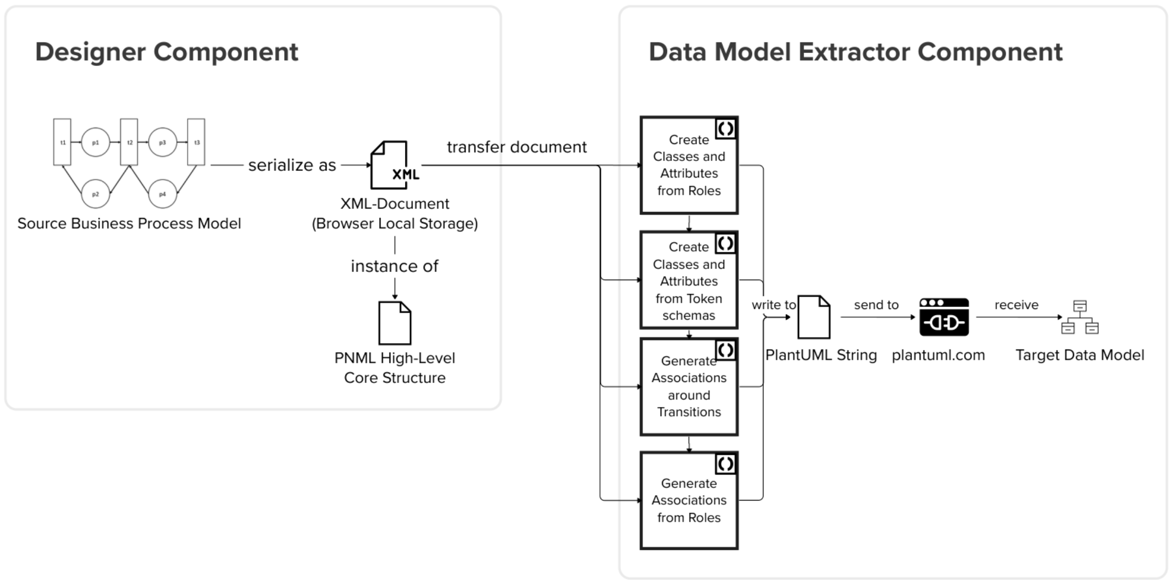

Designer Component
Data Model Extractor Component
Source Business Process Model
serialize as
XML
XML-Document (Browser Local Storage)
instance of
PNML High-Level Core Structure
transfer document
Create Classes and Attributes from Roles
Create Classes and Attributes from Token schemas
Generate Associations around Transitions
Generate Associations from Roles
write to
PlantUML String
send to
plantuml.com
receive
Target Data Model

Figure 18: Process flow of the data model extractor software application

The application is composed of two main components: a designer component and a data model extractor component. Figure 18 shows the two components and their process flow. The designer component provides a graphical interface for users to model business processes in the source BPML. Its primary function is to enable users to create XDN models. In addition to supporting the modeling process, the interface component also handles the XML serialization of the XDN, which is transformed into a PNML high-level core model-compatible format. The resulting XML document, which represents the enriched XDN, is saved as a text string in the browser's local storage.

By utilizing the browser's local storage as a cache, the designer component is designed to use the efficient transfer of data between the designer and data model extractor component. The local storage provides a fast and reliable means of accessing the stored XDNs during the modeling process without having to repeatedly retrieve the data from a backend server instead.

Furthermore, the browser's local storage also enables the persistence of the XDN model across sessions, allowing users to save their work and continue the modeling process at a later time. As the stored data remains in the local storage even after the browser is closed, the data can be retrieved by the user during subsequent sessions. However, it is important to note that this persistence of data is subject to the data retention policies of the browser, and data may be lost if the browser data is manually deleted or if the browser automatically deletes the data after a certain period of time.

Overall, the user interface component provides a platform for modeling business processes in XDN. By handling the XML serialization of the XDN and utilizing the browser's local storage as a cache, the designer component enables the user to model a XDN. The ability to persist the XDN across sessions further allows users to save their work and continue the modeling process at a later time.

The data model extractor component also features a graphical interface that displays the target data model extracted from the source XDN model. The component operates by loading the serialized XDN model from the browser's local cache and applying the extraction rules that are implemented as subsequent methods in the data model extractor component. The rules are designed to capture and retain the essential information of the XDN model. Each step of the data model extractor algorithm writes the resulting elements for the target data model to a string containing the PlantUML representation of the UML CD. PlantUML is an open-source tool that allows users to create UML diagrams using a simple and intuitive language. Instead of drawing diagrams manually, the diagrams can be written in code, and PlantUML will automatically generate a diagram from this code. In the context of the data model extractor component, PlantUML is utilized to generate UML CDs representing the extracted data model, where the output is a string in the PlantUML syntax that can be translated into a graphical representation of the data model. After all steps for the data model extraction are done, the PlantUML string will be sent to http://www.plantuml.com/, which returns the image file of the target UML CD, which is then displayed to the user in the graphical interface. Figure 19 shows an example UML CD generated based on a PlantUml string.

5.2 Implementation of DME

In this chapter, the implementation of a prototype application called Data Model Extractor (DME) that is designed to implement the data model extraction algorithm based on

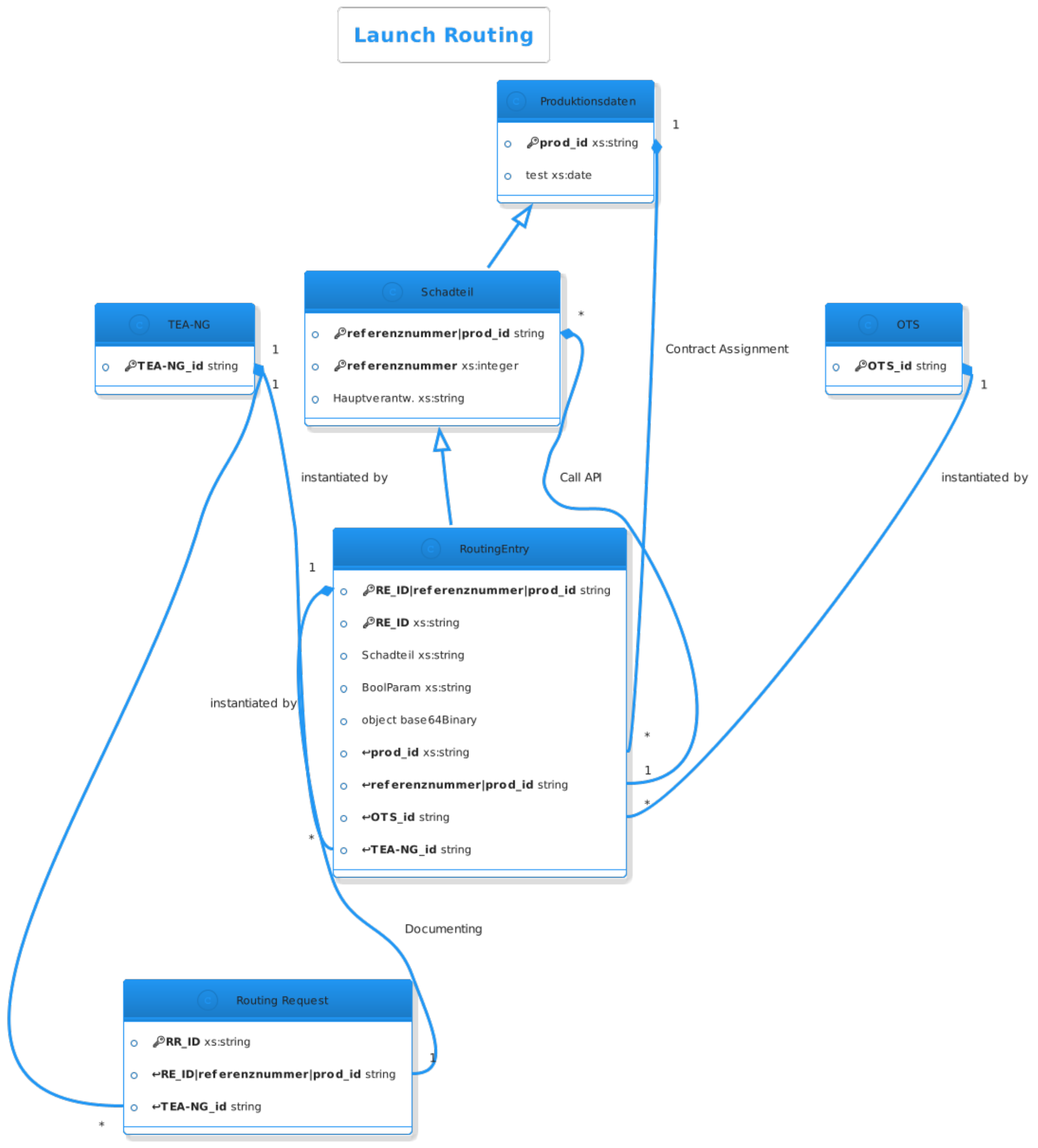


Figure 19: Example of a UML CD generated by PlantUml

XDNs, introduced in Chapter 4.3, is presented. The application takes into account all the rules introduced in Chapter 4.4. The DME application aims to support users in creating and extracting UML CDs from their XDNs.

The DME application is built as an Angular-based web application featuring two main components that work together to facilitate the data model extraction process:

Designer Component: This component serves as an XDN modeler tool, enabling users to create and edit XDN models. The component uses pixi.js [3], a 2D WebGL renderer, to design XDN models with visual interactive elements.

Data Model Extractor Component: The second component of the DME application is responsible for processing the XDN model and applying the extraction rules outlined in

[3]https://pixijs.com/

Chapter 4.4. The input for this component is a PNML XML document representing the XDN net. After processing the input file, the Data Model Extractor component generates a PlantUML string that represents the target UML CD. This string is then sent to the PlantUML web service [4], which returns a PNG file of the generated UML CD for the user to view.

By leveraging these two components, the DME application offers a streamlined approach to extracting UML CDs from BPMs using the XDN model and the data extraction algorithm. This chapter will provide detailed information about the two components. Before these two main components are presented, however, the entities and the XML Services that serve as the basis for the two components will be introduced in the next 2 chapters.

5.2.1 Entities

To visualize the components of the XDN, Pixi.js is used as the rendering engine in the DME prototype application. This section aims to provide a comprehensive analysis of the graphical modeling, with a focus on the Arc, Node, Place, and Transition classes. The Node class serves as a parent class to both the Place and Transition classes, which represent the elements with the same name in the XDN. Figure 20 shows the UML CD of the relevant entities classes. It includes five classes: Arc, Node, Place, Transition, and NodeType.

The following sections will delve into each class, detailing their properties and methods to offer a better understanding of their functionality within the graphical modeling component.

The Arc Class represents a directed connection between nodes in the application, serving as a graphical element for visually linking components. It is composed of three main components: a line, a triangle, and a text box. The constructor receives parameters, such as ID, start and target nodes, cardinality, parent sprite, and XML services. An Arc object stores its triangle texture, ID, start and target nodes, parent sprite, and services for XML manipulation.

Methods of the Arc Class:

- addArc(): This method adds the line and triangle sprite to the arc, positioning them accordingly before adding the arc to the viewport.
- addTextBox(text: string): Responsible for creating a text box with the specified text within the arc. This method also enables the onClick event handlers for the text box to be interactive.

[4]http://www.plantuml.com/,

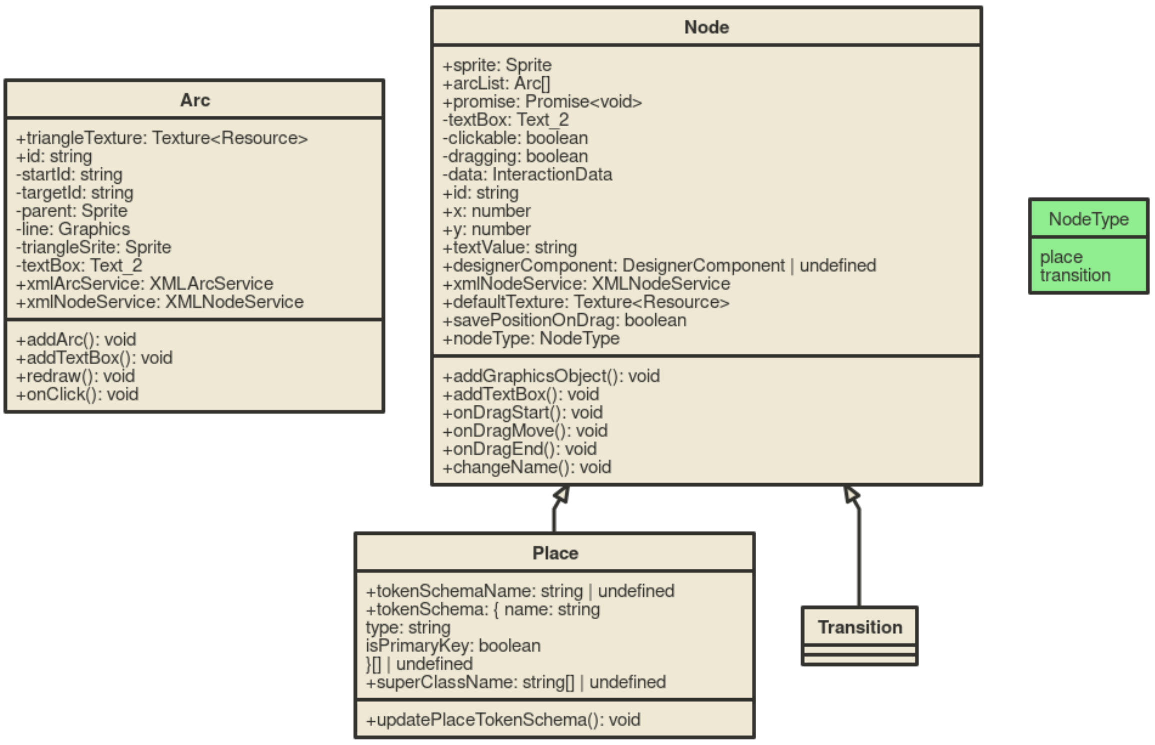


Figure 20: UML CD of the entities classes of the DME application

- redraw(): Updates the position and appearance of the arc, taking into account the current positions of the start and target nodes currently connected to the arc instance.
- onClick(event: InteractionEvent): Alters the cardinality of the arc when the text box is clicked, updating the XML representation accordingly.

The Node class is an abstract parent class designed for places and transitions in the application. It represents a node in the graph and includes a sprite, a text box, and a list of connected arcs. The constructor receives parameters such as ID, x and y coordinates, text value, designer component class instance, and XML services. NodeType is another parameter of the Node class which is automatically determined on creation based on the concrete class that instantiates the node.

Methods of the Node Class:

- addGraphicsObject(x: number, y: number, isInteractive: boolean, tint: number): Adds a graphical object (sprite) to the node, configuring its properties such as position, interactivity, and tint.
- addTextBox(text: string): Incorporates a text box to the node, displaying the specified text.

- onDragStart(event: InteractionEvent), onDragMove(event: InteractionEvent), and onDragEnd(event: InteractionEvent): Methods for handling dragging events on the node, allowing users to reposition nodes within the graph.
- changeName(newName: string): Renames the node and updates the XML node name to reflect the change.

The Place class is a child class of the Node class, representing a place in the graph. In addition to the parent class properties, it stores the token schema name, token schema, and superclass name. The constructor accepts parameters such as ID, x and y coordinates, text value, saveInXml flag, designer component class instance, and XML services.

Methods of the Place Class:

- Inherits all methods from the Node class.
- updatePlaceTokenSchema(dataObjectName: string, data: name: string; type: string, isPrimaryKey: boolean [], superClassName: string[], xmlPlaceService: XMLPlaceService): Updates the token schema and superclass name for the place, while also reflecting these changes in the XML representation.

The Transition class is another child class of the Node class, representing a transition in the graph. It shares the properties and methods of the Node class. The constructor receives parameters such as ID, x and y coordinates, text value, designer component class instance, and XML services.

Methods of the Transition Class:

- Inherits all methods from the Node class.

5.2.2 XML Services for the Manipulation of XDN's

In this section, the service layer created in this work for the manipulation of the XDN model in the DME application is discussed. These services work together to provide a comprehensive and organized approach to interacting with the XML document representing the XDN and making the necessary changes according to the application's requirements. By encapsulating the logic for XML manipulation in these services, the application's components can easily access and modify the XML document without having to deal with the intricacies of the XML DOM.

1. **XMLService:** This service creates and initializes a new XML document with the core PNML structure. The primary function, `createNewXMLDocument()`, is responsible for:

- Instantiating a new XML document and setting it as a global variable
- Creating the core PNML structure, including the 'pnml', 'net', 'name', 'text', and 'page' elements
- Assigning appropriate attributes to these elements, such as 'xmlns', 'id', 'type', and 'textContent'

2. **XMLNodeService:** This service manages common operations for both places and transitions, such as:
   - Creating a new node object using `createNode()`
   - Updating the graphics position of a node object with `updateNodePosition()`
   - Updating the name of a node using `updateNodeName()`
   - Retrieving the graphical position and name of a node using `getNodePosition()` and `getNodeNameById()`
   - Fetching the text assigned to a node with the given ID using `getNodeTextById()`
3. **XMLPlaceService:** This service handles operations related to places in the XML document. Key functionalities include:
   - Fetching place attributes: This is done using the getPlaceAttributesById() method, which returns an array of attribute objects for a given place ID.
   - Managing token schemas: The getPlaceTokenSchemaName() and updatePlace-TokenSchema() methods allow for retrieving and updating the token schema assigned to a place, respectively.
   - Handling superclasses: Methods such as getPlaceSuperClassNameById() and getDistinctSuperClassNameByName() help fetch assigned superclass names from the XML document.
   - Fetching token schema elements: The getDistinctTokenSchemaByName() method returns an array of unique schema objects for a given token schema name.
   - Other utility methods: getAllPlaces() and getPlaceById() enable fetching all places or a specific place from the XML document.
4. **XMLTransitionService:** This service manages the operations related to transitions in the XML document. Its main features include:
   - Fetching transitions: The getAllTransitions() method retrieves a collection of transition elements from the XML document.
   - Managing transition roles: The updateTransitionRole() and getTransition-Role() methods allow for updating and retrieving the role assigned to a transition with a given ID, respectively.

- Handling unique transition roles: The getTransitionRolesDistinct() method returns an array of unique transition role strings from the XML document.
- Fetching transition role elements: The getTransitionRoles() method retrieves an array of transition role elements from the XML document.

5. **XMLArcService:** This service handles operations related to arcs in the XML document. It includes methods for:

   - Creating a new arc object using `createArc()`
   - Updating the cardinality of an arc object with `updateArcCardinality()`
   - Fetching all arcs defined in the XML document via `getAllArcs()`
   - Returning all arcs with a specific source or target node using `getAllArcsWithSource()` and `getAllArcsWithTarget()`

In summary, the XML Services presented in this chapter offer a structured and detailed solution for handling the manipulation of the XML document in the application. The services enable efficient interaction with the XML document, allowing for maintainable and modular code throughout the application.

5.2.3 Designer Component

The Designer Component is the primary user interface for modeling XDNs within the DME application. It is responsible for providing an interactive canvas where users can create, edit, and visualize XDN models using graphical representations of places, transitions, and arcs. This section will discuss the structure, functionality, and code implementation of the Designer Component.

The Designer Component is a TypeScript class decorated with the @Component decorator, which allows it to be instantiated and utilized within the Angular framework. The component imports various dependencies and services, including ElementRef, ViewChild, MatSnackBar, and the different XML services introduced in 5.2.2 for managing the XML representation of the XDN model.

The class maintains an internal state, which includes various flags to track user interactions, such as placeSelected, transitionSelected, and createArcInProgress. The state also keeps track of the current node properties like name, role, tokenSchemaName, superClassSelected, and data for the properties panel. Additionally, the nodeReferenceList and promiseList arrays store references to the nodes and promises, respectively, required for the rendering and management of nodes and arcs.

The Designer Component implements the AfterViewInit lifecycle hook, which is called after the component's view has been fully initialized. Inside the ngAfterViewInit method, the component initializes the Pixi.js application, sets up the canvas, and loads any previously saved XDN models from the browser's local storage if they exist.

5.2.3.1 Functional Overview

The Designer Component is responsible for the following core functionalities:

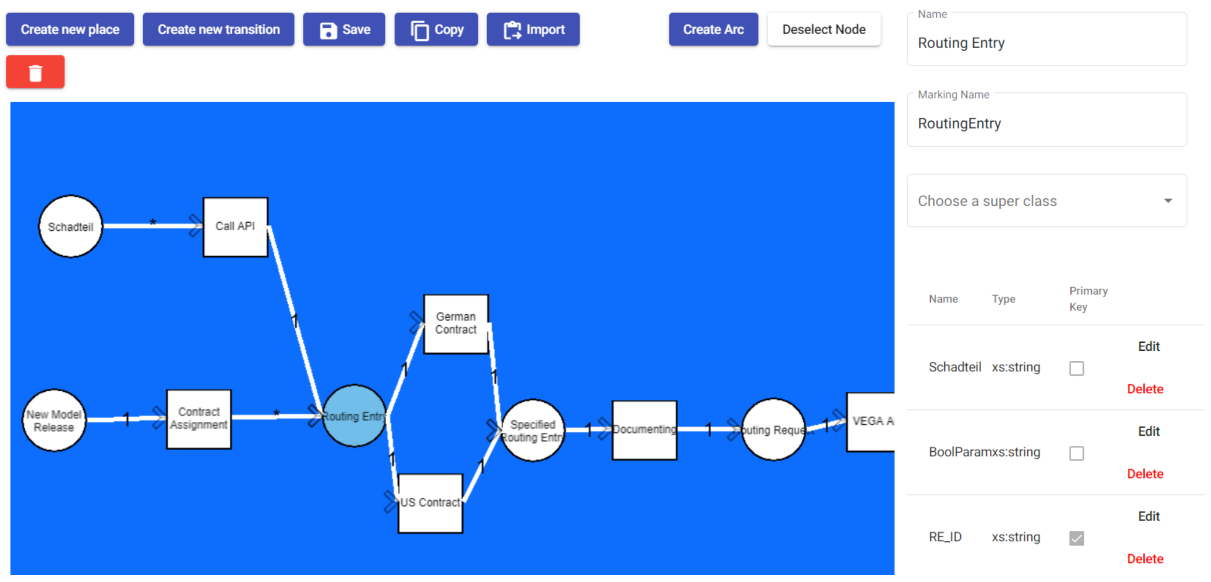
Visualization: The component renders the XDN elements, including places, transitions, arcs, and their labels. This graphical representation enables users to understand the structure and relationships within the XDN.

Editing: Users can interact with the XDN elements directly on the canvas, creating or modifying their properties. This approach streamlines the editing process, allowing users to focus on the XDN design rather than the underlying XML structure.

5.2.3.2 Components

The Designer Component is composed of several sub-components and features that work together to provide a cohesive user experience:

Figure 21 shows the UI of the DME application. It consists of the canvas for the graphical representation of the XDN model, a toolbar on top as well as the property panel on the right.



Name: Routing Entry

Marking Name: RoutingEntry

Choose a super class

Name	Type	Primary Key	
Schadteil	xs:string	☐	Edit Delete
BoolParam	xs:string	☐	Edit Delete
RE_ID	xs:string	✓	Edit Delete

Figure 21: UI of the Designer Component of the DME application

1. **Canvas:** The canvas serves as the primary workspace for visualizing and interacting with the XDN elements.

2. **Toolbar:** The toolbar presents a collection of tools and actions that users can apply to the XDN. The Designer Component provides various methods for creating and managing the XDN model elements, including adding places, transitions, and arcs:

   - addPlace(): Creates a new Place instance and adds it to the nodeReferenceList.
   - addTransition(): Creates a new Transition instance and adds it to the nodeReferenceList.
   - addArc(): Creates a new Arc instance and adds it to the nodeReferenceList.

   Other methods include saving the XDN model to the local storage or clipboard and reading the XDN model from the clipboard.

3. **Property Panel:** The property panel displays the properties of the currently selected node in the XDN. Users can edit these properties to update the nodes as needed. The properties panel is conditionally rendered based on whether a place or transition is selected. According to the XDN design presented in chapter 4.3, for both places and transitions, the name attribute can be edited. For transitions, a role attribute can be edited. For places, the marking name and superclasses can be edited. When a place with a token schema name is selected, a table is displayed, allowing users to add, edit, and delete rows representing the token schema's properties.

5.2.3.3 Underlying Mechanisms

The Designer Component uses several mechanisms and technologies to facilitate seamless interaction with the XDN:

1. **Drag and Drop:** Utilizing drag-and-drop functionality, users can add new XDN elements to the canvas or modify existing ones by dragging them to the desired location.

2. **Event Handling:** The component responds to user input events, such as clicks and drags, to offer a responsive and interactive editing experience. This includes adding, deleting, or updating elements and their properties.

3. **Data Binding:** By taking advantage of Angular's data binding mechanisms, the Designer Component keeps the XDN model and UI synchronized. This ensures that any changes made on the canvas are automatically reflected in the underlying XML document, and vice versa.

4. **Integration with Services:** The Designer Component relies on the XML Services described in Section 5.2.2 to handle the manipulation of the XML document. This

integration allows the component to focus on user interactions while delegating XML manipulation tasks to the appropriate services.

Overall, the Designer component offers a user interface for creating, editing, and managing graphical XDN models with places, transitions, and arcs while allowing users to modify the attributes of selected nodes. This component is essential in making the overall application a valuable tool for users creating well-formed XDN models.

5.2.4 Data Model Extractor Component

The Data Model Extractor Component is a component responsible for applying the rulesets presented in section 4.4 to the XDN and converting the result into a PlantUML format. Figure 22 presents a flowchart that visually outlines the process the Data Model Extractor Component implements. Starting from parsing the XML document to generating a PlantUML string as output, this process involves several sequential steps, which are described in detail below.

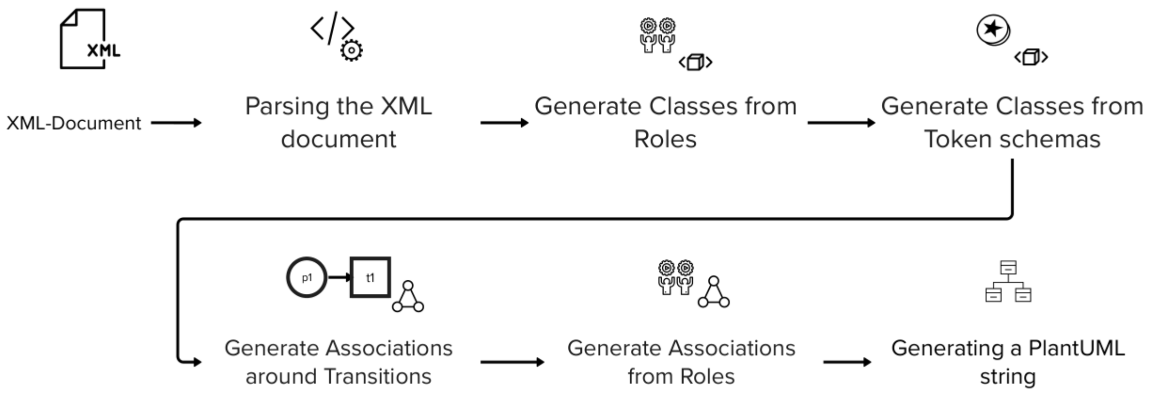

XML
XML-Document
Parsing the XML document
Generate Classes from Roles
Generate Classes from Token schemas
p1
t1
Generate Associations around Transitions
Generate Associations from Roles
Generating a PlantUML string

Figure 22: Flow chart of the data model extraction process.

5.2.4.1 Parsing the XML document The first step is parsing the XML document to extract the relevant information needed for the class diagram. The code makes use of the DOMParser to access the necessary elements and their properties.

The first step involves retrieving the XML document, which represents the XDN model, from the browser's local storage as a string. This document contains the relevant information needed for the UML CD. In order to process this information, the code utilizes the DOMParser to convert the string into a JavaScript object, specifically an XMLDocument[5]. This conversion enables the handling and manipulation of the document's elements and

[5]https://developer.mozilla.org/en-US/docs/Web/API/XMLDocument

their properties using the XML Services introduced in chapter 5.2.2, which will be used in the following steps.

5.2.4.2 Generate Classes from Roles

In this step, the XMLTransitionService is used to get an array of all distinct roles in the source XDN. For each distinct role element, it is checked if:

- the class is not already created (from another role).
- the class will not be created in the next step (5.2.4.3) from a token schema.

If both conditions are met, the class is pushed into the *classes* array, which holds the staging classes. This array holds elements with the schema shown in listing 5. The *classes* array stages all classes and will later be manipulated when searching for possible foreign keys in the chapters 5.2.4.4 and 5.2.4.5. The foreign keys are then added to the attributes array.

```
name: string;
superClasses: string[];
attributes: {
    name: string;
    type: string;
    isPrimaryKey: boolean;
    isPrimaryKeyCombi: boolean;
}[];
```

Listing 5: Object schema of a class in the classes array

The name of the class equals the defined role name of the transition (see rule 1.2 in 4.4.1). The *superClasses* array stays empty.

The attribute array is initialized with the corresponding primary key for the role using the *getPKCombination* method. This method calculates the primary key for a class with the given name. If there is a superclass and/or multiple primary keys defined in this class, we build the combination of all primary keys available for this class. The method searches for all defined primary keys in the token schemas of the XDN with the given class name. Additionally, it searches for all defined primary keys in the superclasses of all token schemas with the given class name and concatenates them together (see rule 3.2 in 4.4.3). If no primary can be found, a default primary key with the schema of className+"_id" is given. This is, for example, the case if there is no token schema with an identical name as the role or the user simply did not define a primary key for a token schema.

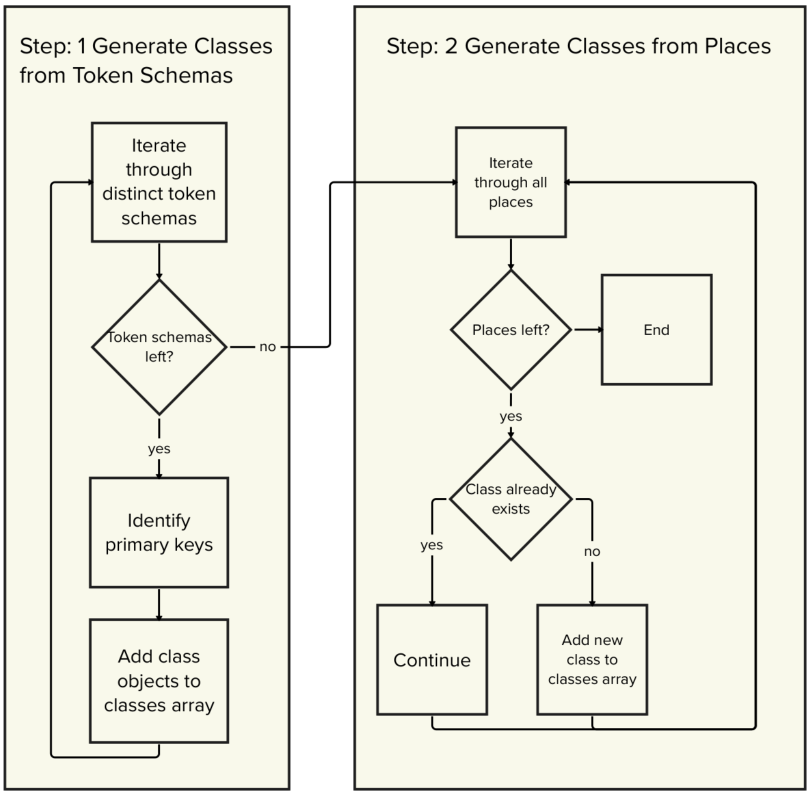

Step: 1 Generate Classes from Token Schemas
Iterate through distinct token schemas
Token schemas left?
no
yes
Identify primary keys
Add class objects to classes array
Step: 2 Generate Classes from Places
Iterate through all places
Places left?
End
yes
Class already exists
yes
no
Continue
Add new class to classes array

Figure 23: Flowchart of the generateClassesFromTokenSchemas method.

5.2.4.3 Generate Classes from Token schemas In this step, the XMLPlaceService is used to get a distinct array of all token schema names defined in the XDN. Figure 23 shows a flowchart describing the process of how the classes are generated from the token schemas of the parsed XDN. The process is divided into two steps. First, all distinct token schemas are processed. In the second step, all places which don't have a token schema specified in the XDN are handled.

For each token schema name, the primary key is calculated using the method *getPKCombination* (introduced in 5.2.4.2). In the next step, all token schema elements, meaning the attributes for the given token schema name, are identified using the *getDistinctTokenSchemaByName* method. This method returns all distinct token schema elements in the XDN for a given token schema name.

The attribute array is then supplemented by the generated primary key. This is necessary in the case the primary key was not included by default in the token schema definition itself but was rather created from a combination of primary keys, inherited from a parent class, or generated due to the lack of a primary key definition.

The superclasses of the class are then identified using the *getDistinctSuperClassNameByName* method (see rule 1.3 in 4.4.1).

The name, superclasses, and attributes (including the primary key) are then added to the staging *classes* array.

In the second step, all the leftover places which don't have an explicit token schema definition are created. Those classes are created by using the place name or alternatively "undefinedClass" if not even a place name is defined as the class name. If a class with the same name does not already exists (because there could be classes with the same name created from roles or token schemas), the class is added to the staging *classes* array.

5.2.4.4 Generate Associations around Transitions

The *generateCardinalitiesAroundTransitions* method is designed to systematically generate associations between classes based on transitions and the arcs around them within the XDN (see Rule 2.1 in 4.4.2). Figure 24 shows a flowchart describing the process of how the associations are generated by iterating through all transitions of the parsed XDN.

The algorithm starts by iterating through all available transitions, subsequently identifying arcs with the current transition as their target. For each of these predecessor arcs, the method continues to examine arcs with the current transition as their source, referred to as successor arcs.

Upon validating the presence of both predecessor and successor arcs, the associated predecessor and successor places are retrieved according to their respective arc's source and target attributes. The algorithm then verifies the validity of these places. If they are valid, the names of the predecessor and successor classes are extracted using the appropriate function from the XMLPlaceService. In cases where a token schema is not defined for either the predecessor or successor, the place name is utilized as a fallback, with the assignment of "undefinedClass" in the absence of a name, according to Rule 1.1 (see 4.4.1).

Following this, the cardinalities for both predecessor and successor arcs are obtained. In the event of missing or invalid cardinalities, a default value of "*" is assigned. The algorithm proceeds to compare the names of the predecessor and successor classes, and when a distinction is identified, a composition relation between the two classes is established using the *addComposition* method

The *addComposition* method is responsible for adding a composition relation between two classes, considering their cardinalities, names, and association text. The method first checks for the existence of a duplicate relation in the *associationList*. If a duplicate is not found and both *sourceName* and *targetName* are valid, it proceeds to establish the relation. To achieve this, the primary key combination for the source class is retrieved using the *getPKCombination* method.

In the next step, the target class is identified. If it exists, the method checks if a foreign key attribute corresponding to the source class's primary key is already present in the target class. If not, it adds a new foreign key attribute to the target class with the format

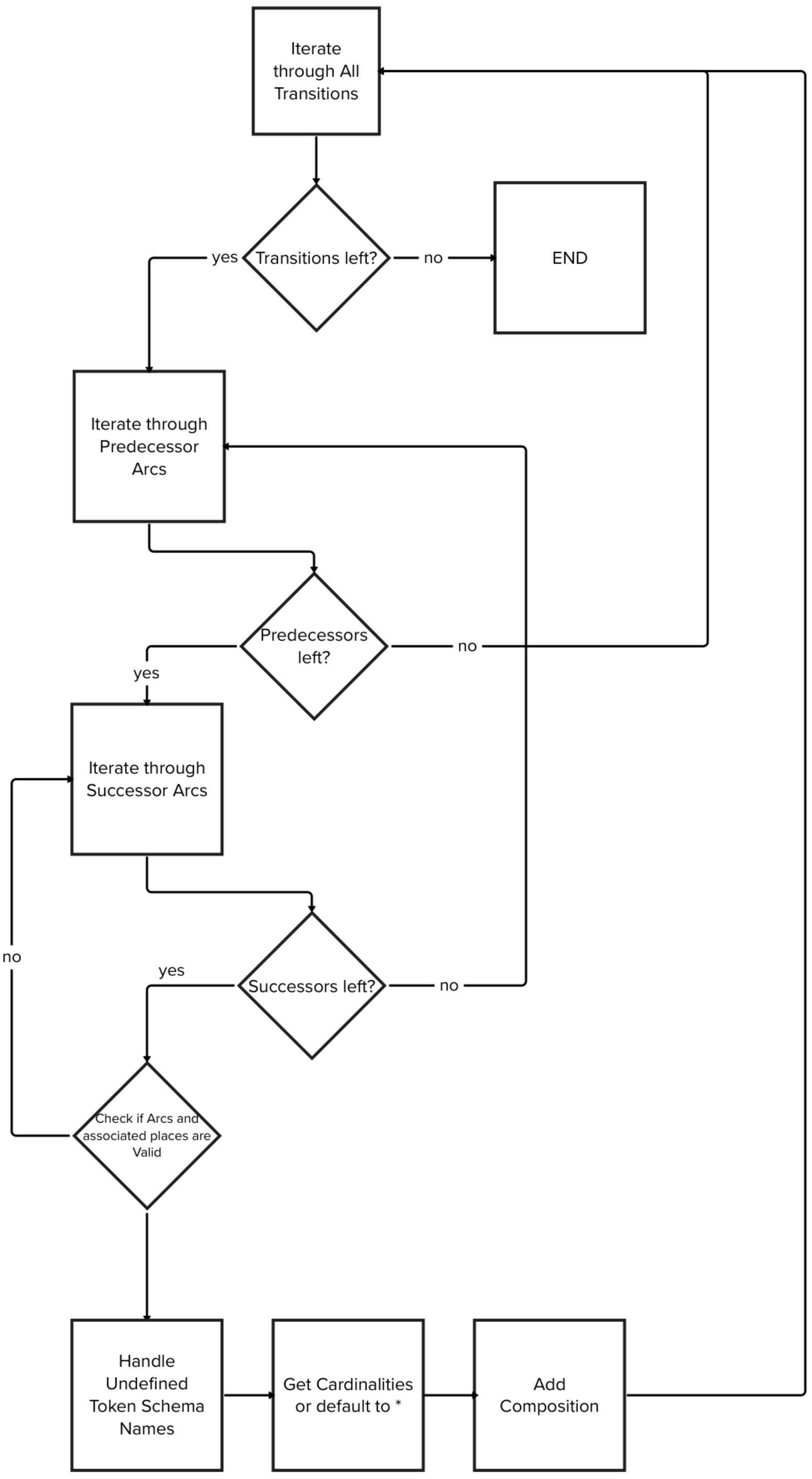

Iterate through All Transitions
Transitions left?
yes
no
END
Iterate through Predecessor Arcs
Predecessors left?
yes
no
Iterate through Successor Arcs
Successors left?
yes
no
no
Check if Arcs and associated places are Valid
Handle Undefined Token Schema Names
Get Cardinalities or default to *
Add Composition

Figure 24: Flowchart of the generateAssociationsAroundTransitions method.

“foreign_key(primary_key.name)” and the appropriate primary key type (see rule 3.3 in 4.4.3).

Subsequently, the *addComposition* method adds the relation information as an object to the array *associationList.* This is done so that in the upcoming iterations, it can be checked if the relation already exists. It also constructs a string representation of the relation, which includes the source class name, primary key, source cardinality, direction (if provided), target cardinality, target class name, and association text. This string is then appended to the staging *associations* array.

5.2.4.5 Generate Associations from Roles

The *generateCardinalitiesFromRoles* method systematically generates relations between classes predicated on the transition roles present within the XDN.

The algorithm initiates by iterating through all transition roles, subsequently identifying arcs with the current transition role as their source. For each of these arcs, the method verifies if the target node is a valid place.

Upon establishing the validity of the target node, the name of the target class is retrieved. In cases where a token schema is not defined for the target class, the place name is employed as a fallback, with the assignment of “undefinedClass” in the absence of a name (according to Rule 1.1 in 4.4.1). The algorithm then proceeds to create a composition relation between the transition role and the target class using the *addComposition* method, passing the according parameters for a 1:n relationship every time (see Rule 2.2 in 4.4.2).

5.2.4.6 Generating a PlantUML string

Finally, the *flushToPlantUML* method is responsible for converting the extracted data into a PlantUML string.

It starts by adding the necessary headers, including the diagram title. Then, it iterates through the classes, adding class headings and attributes, taking into account primary keys and primary key combinations. Afterward, it adds the associations between the classes.

The *flushToPlantUML* method is designed to transform the previously staged classes and associations into a PlantUML formatted string, subsequently generating a corresponding diagram. The algorithm initializes the *plantUMLString* variable with rudimentary PlantUML structure, theme, primary and foreign key macros, and the title derived either from the XDN or the default “Generated Class Diagram”.

The method proceeds to iterate through each class element within the *classes* array, examining the presence of superclasses and formulating the inheritance string by concatenating superclass names. The constructed class header, inclusive of the inheritance string, is subsequently appended to the *plantUMLString*. Afterward, the method sorts class attributes based on their status as primary key combinations or primary keys, assigning priority to primary key combinations and primary keys in succession. The algorithm iterates through each attribute, appending it to the *plantUMLString*.

Upon the conclusion of processing each class in the *classes* array, the class definition is sealed with a closing brace. The method then iterates through the *associations* array, incorporating each association element into the *plantUMLString*. Finally, the method appends the "@enduml" string to signify the conclusion of the PlantUML definition and proceeds to generate a PlantUML diagram. This is achieved by encoding the *plantUMLString* utilizing the PlantUML Encoder and designating the encoded string as the source for an HTML image src attribute. The image then points to the PlantUML website and retrieves the final UML CD at runtime.

Figure 25 shows the UI of the DME application. It consists of the image element for the graphical representation of the UML CD, a button to copy the source PlantUML string to the user's clipboard, as well as a link to the PlantText[6] homepage.

In conclusion, the Data Model Extractor Component is able to convert an XDN document into a UML CD represented in PlantUML format. The component handles the parsing of the XDN document and extracts the necessary information to generate a UML CD according to the extraction algorithm presented in chapter 4.4.

5.3 Example based on XDN

Listing 6 shows an example XDN. Figure 26 shows the associated XDN in its graphical representation modeled within the designer component of the DME application. This simple example consists of three places and one transition and is designed to trigger all of the presented rulesets introduced in chapter 4.4. It should be noted that the XDN shown in listing 6 is not a valid PNML document, as the core tags and graphical information are omitted for readability of this example.

```
<place id="p1">
        <name><text>p1</text></name>
        <toolspecific tool="dme" version="1.4.2">
                <tokenSchema
                        xmlns:xs="http://www.w3.org/2001/
                          XMLSchema" name="p1" superClass="p3">
```

[6]https://www.planttext.com/

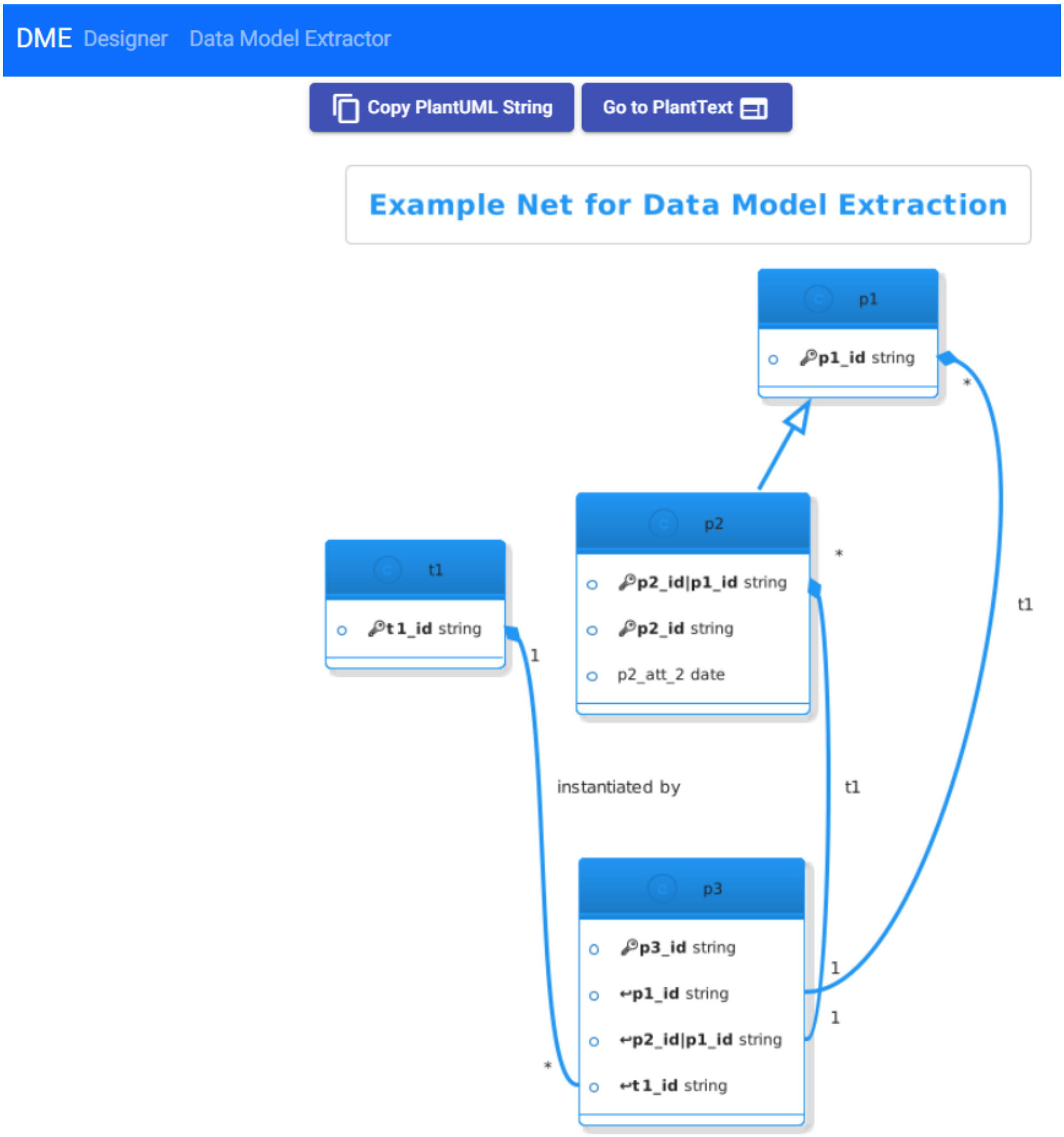


Figure 25: UI of the Data Model Extractor Component of the DME application

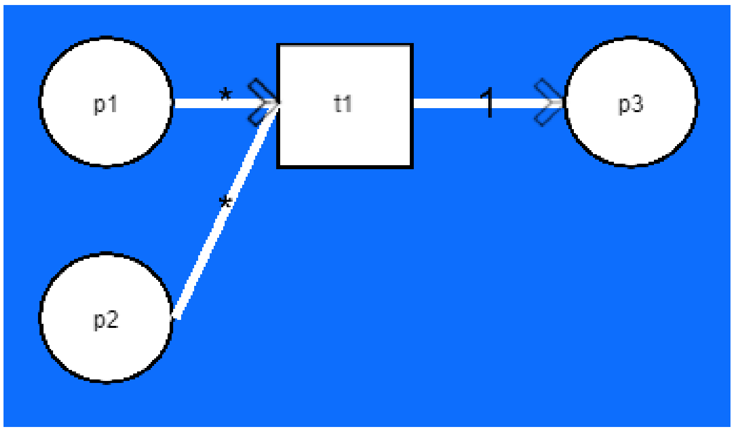


Figure 26: A XDN example modeled within the designer component.

```
                        <xs:element name="p1_att_1" type="string
                            " isPrimaryKey="true"/>
                        <xs:element name="p1_att_2" type="date"
                            isPrimaryKey="false"/>
                </tokenSchema>
        </toolspecific>
</place>
<place id="p2">
        <name><text>p2</text></name>
        <toolspecific tool="dme" version="1.4.2">
                <tokenSchema
                        xmlns:xs="http://www.w3.org/2001/
                            XMLSchema" name="p2" superClass="">
                        <xs:element name="p2_att_1" type="string
                            " isPrimaryKey="true"/>
                        <xs:element name="p2_att_2" type="date"
                            isPrimaryKey="false"/>
                </tokenSchema>
        </toolspecific>
</place>
<place id="p3">
        <name><text>p3</text></name>
        <toolspecific tool="dme" version="1.4.2">
                <tokenSchema
                        xmlns:xs="http://www.w3.org/2001/
                            XMLSchema" name="p3" superClass=""/>
                </toolspecific>
```

```
</place>
<transition id="t1">
        <name><text>t1</text></name>
        <toolspecific tool="dme" version="1.4.2">
                <role><text>System_1</text></role>
        </toolspecific>
</transition>
<arc id="a1" source="p1" target="t1">
    <hlinscription>*</hlinscription>
</arc>
<arc id="a2" source="t1" target="p3">
    <hlinscription>1</hlinscription>
</arc>
<arc id="a3" source="p2" target="t1">
    <hlinscription>*</hlinscription>
</arc>
```

Listing 6: Example of a XDN

Figure 27 shows the resulting UML CD after applying the three extraction rulesets to the XDN. The following extractions are performed on the XDN in listing 6:

- Ruleset 1: Classes
  - **Rule 1.1: Every distinct token schema of a place results in a class in the data model**
    - Tokenschema name of place p1 creates class p1.
    - Tokenschema name of place p2 creates class p2.
    - Tokenschema name of place p3 creates class p3.
  - **Rule 1.2: Every distinct role of a transition results in a class in the data model**
    - Role of transition t1 creates class System_1.
  - **Rule 1.3: Place classes can be inherited from already defined classes**
    - Superclass p3 of place p1 creates an inheritance relationship from p1 to p3.
    - Superclass p3 of place p1 leads to inheritance of the primary key p3_id for class p1.
- Ruleset 2: Associations

- **Rule 2.1: Transition Associations**
  * Transition t1 creates a 1:n composite association between classes p1 and p2 based on arcs a1 and a2.
  * Transition t1 creates a 1:n composite association between classes p3 and p2 based on arcs a3 and a2.
- **Rule 2.2: Role Associations**
  * The role System_1 of transition t1 creates a composite 1:n relation to its successor place p2 (based on arc a2).

• Ruleset 3: Attributes

- **Rule 3.1: Token Schemas**
  * Token schema p1 of place p1 contains two xs:elements. Those lead to the generation of the attributes att_1 and att_2 in class p1.
  * Token schema p2 of place p2 contains one xs:element. This leads to the generation of the attribute p2_att_1 in class p2.
- **Rule 3.2: Primary Keys**
  * xs:element p1_att_1 of place p1 is marked as isPrimaryKey=true. This leads to the creation of the primary key p1_att_1 in class p1.
  * xs:element p2_att_1 of place p2 is marked as isPrimaryKey=true. This leads to the creation of the primary key p2_att_1 in class p2.
  * No primary keys are provided in the token schema of place p3. This leads to the automatic generation of the primary key p3_id in class p3.
  * No primary keys are provided for the role generated class System_1. This leads to the automatic generation of the primary key System_1_id in class System_1.
  * The inheritance relationship between the classes p1 and p3 also leads to class p1 inheriting the primary key p3_id of class p3. The result is a combined primary key of p1_att_1|p3_id.
- **Rule 3.3: Foreign Keys**
  * Based on the association t1 between classes p1 and p2, a foreign key p1_att_1|p3_id is created for class p2.
  * Based on the association t1 between classes p3 and p2, a foreign key p3_id is created for class p2.
  * Based on the association “instantiated_by” between classes System_1 and p2, a foreign key System_1_id is created for class p2.

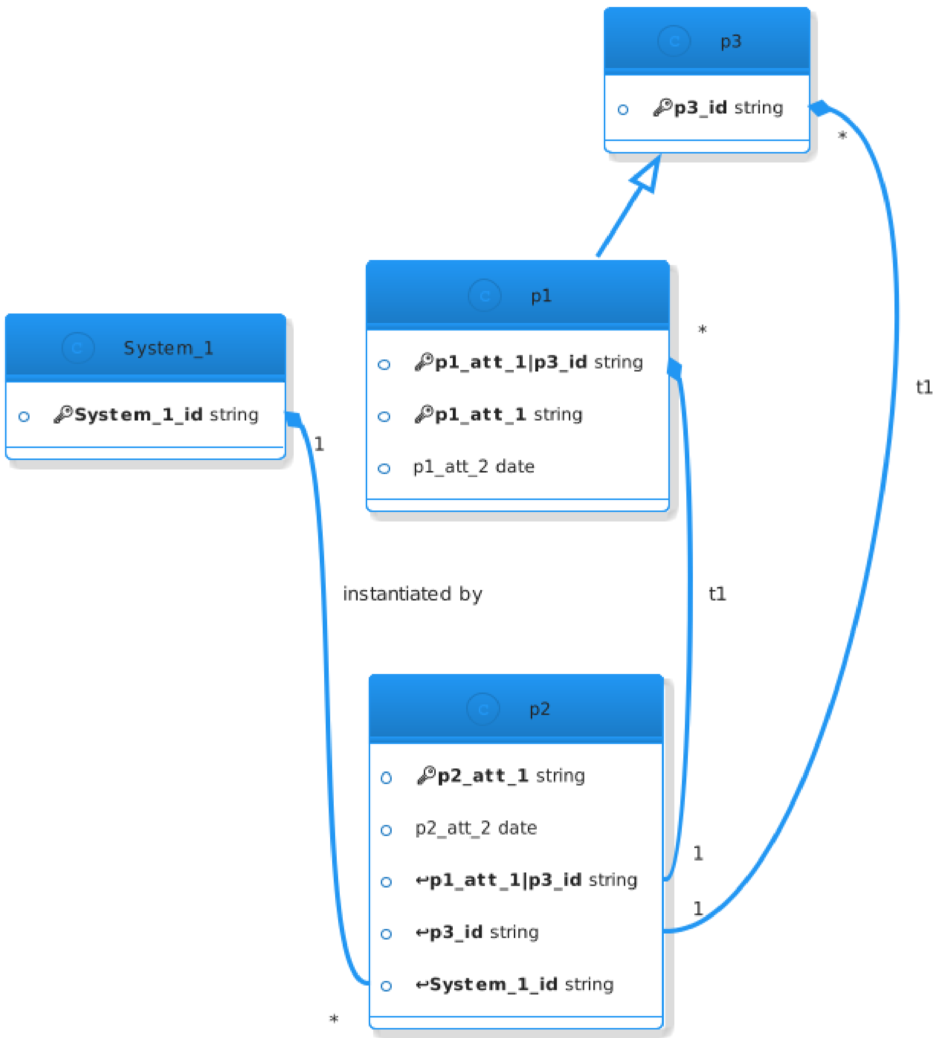


Figure 27: UML CD generated by applying the extraction rules to the XDN provided in listing 6

6 Evaluation and Implications

This chapter presents a comprehensive evaluation of the prototype application developed, including the concept of extracting data models from BPMs, specifically XDNs, as introduced in chapter 4.3. The evaluation process is divided into three main sections. In section 6.1, the DME application's performance is assessed through measurable metrics, offering an objective understanding of its effectiveness in extracting data models from XDNs. Section 6.2 focuses on gathering insights from users with varying levels of expertise in the field, providing an overview of the application's usability, effectiveness, and potential for improvement. Finally, Section 6.3 addresses the validation process for the XDN model, ensuring that the resulting PNML file generated by the DME application is valid and compatible with third-party tools. The PNML Document Checker is employed to validate the PNML XML string produced by the designer component, with an example output illustrating the validation process. This section emphasizes the significance of adhering to standard specifications to enable the use of XDN across various tools and applications.

By evaluating the application across these three dimensions, this chapter aims to demonstrate its effectiveness in extracting data models from XDNs while exploring areas for future development and enhancement and the flexibility of the XDN definition.

6.1 Quantitative Analysis: DME vs. Manual Data Model

This section aims to provide an objective assessment of the DME application's performance in extracting accurate data models by comparing the generated data model to a manually created data model in a real-world example. Section 6.1.1 introduces the setup of the quantitative evaluation, introducing various metrics and an example setting. Section 6.1.2 will then present the metrics observed through the analysis of two data models and will interpret the results.

6.1.1 Planning

The quantitative evaluation is designed on the example of an experiment conducted by Banjac et al. [Banjac et al.(2018)]. They introduced a set of key figures and metrics for the evaluation of automatically generated conceptual database models based on BPMs. The five key figures are:

- $N_{generated}$: The total number of automatically generated concepts
- $N_{correct}$: The number of correctly generated concepts that can be kept in the target

model

- $N_{incorrect}$: The number of incorrectly generated concepts that cannot be kept in the target model
- $N_{excessive}$: The number of excessively generated concepts that should not be kept in the target model
- $N_{missing}$: The number of missing concepts that should be in the target model but are not generated

The key figures were obtained by comparing the automatically generated data model with a manually created data model. Using the key figures $N_{correct}$, $N_{missing}$ and $N_{incorrect}$, the metrics *Recall* and *Precision* can be calculated:

$$Recall = \frac{N_{correct}}{N_{correct} + N_{missing}} \quad Precision = \frac{N_{correct}}{N_{correct} + N_{incorrect}}$$

Recall describes the percentage of the concepts in the target data model that are automatically generated. *Precision* describes the percentage of the correctly generated concepts in the target data model.

6.1.2 Execution

The quantitative evaluation is based on an industrial project in the automotive industry. The project was carried out in cooperation with Mercedes-Benz USA, LLC. To improve the precision and time spent selecting defective parts in the field, a new software application is planned. The application needs to select defective parts, on the one hand, for the regress of warranty goods with the suppliers of Mercedes-Benz and, on the other hand, for the quality analysis and thus the fix of the issues in production.

The routing of parts is carried out based on three different use cases:

- Regress-View
- Quality-View
- Launch Routing

Because of the complexity of the application, we only consider a part of the application process for evaluation in this work. The use-case used for the evaluation of this work describes the process of the launch routing view of the application.

The launch routing describes a special phase in the selection of defective parts. In contrast to the regress and quality views, which handle the selection of parts during the runtime

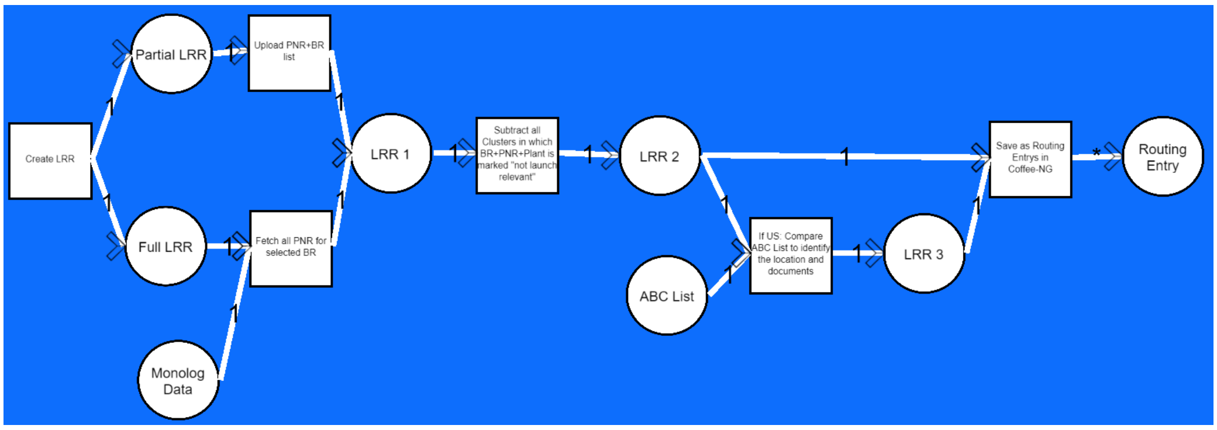

Create LRR
Partial LRR
Full LRR
Monolog Data
Upload PNR+BR list
Fetch all PNR for selected BR
LRR 1
Subtract all Clusters in which BR+PNR+Plant is marked "not launch relevant"
LRR 2
ABC List
If US: Compare ABC List to identify the location and documents
LRR 3
Save as Routing Entrys in Coffee-NG
Routing Entry
1
*

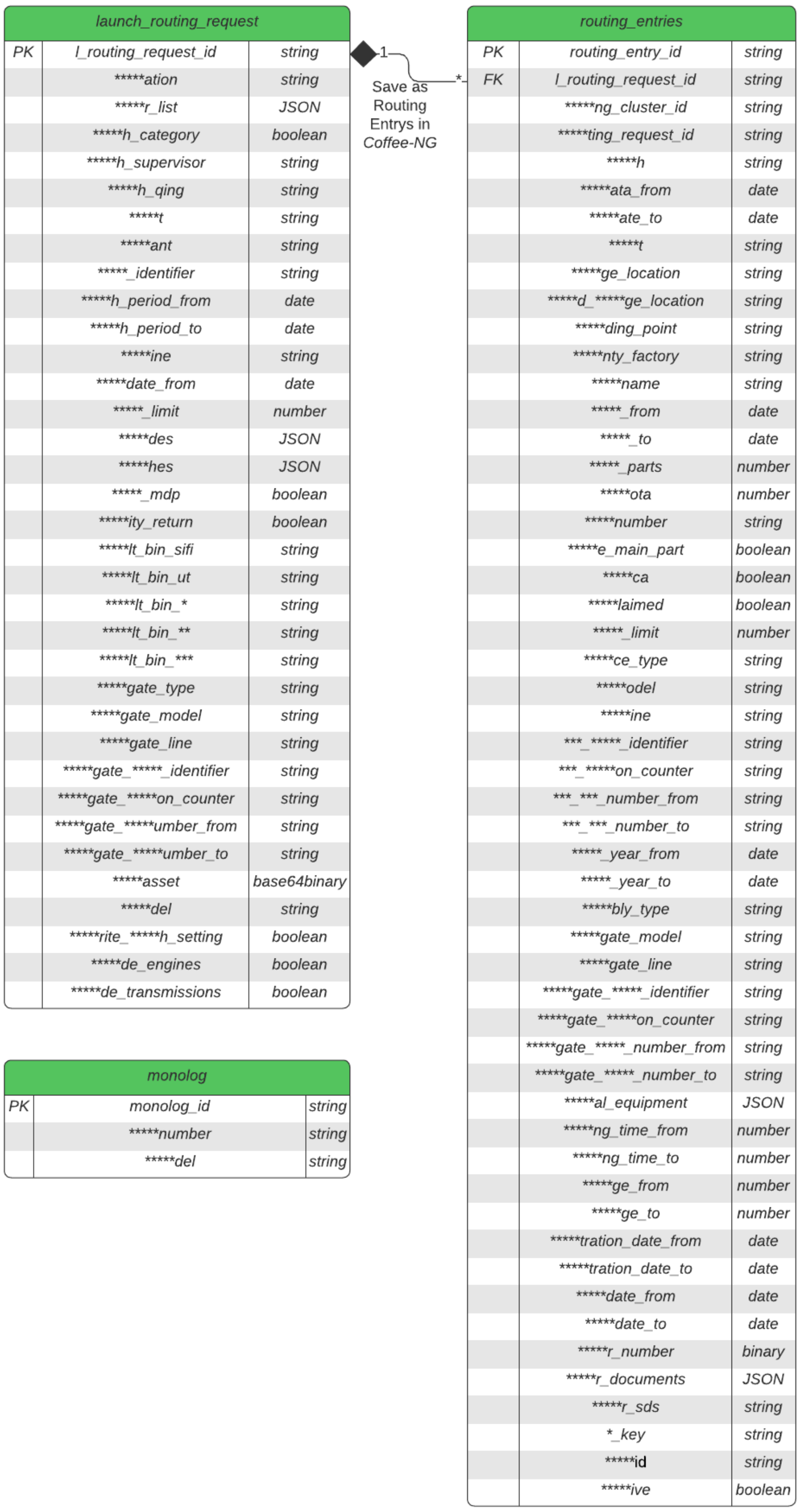
Figure 28: XDN model of the launch routing process.

of a released car model, the launch routing view overrules those views in the first months after launching a new model. This is necessary because the regress and quality view is designed to only select either a sample of all parts in the field or select the defective parts based on certain quality metrics. In the launch period of a model, however, it is necessary to route in all of the defective parts of the model to identify issues faster and get more information on certain issues.

Figure 28 shows the launch routing process modeled as an XDN within the DME applications designer component. The process starts with the creation of a launch routing request (LRR). This can be either a partial LRR or a full LRR. The difference between them is the full routing of a model's parts (full RR) or just the partial routing of a model's parts (partial LRR). After creating a partial LRR, the user has to upload a list of all the part numbers and model line combinations which should be routed in. In the case of a full LRR, this part number and model line information will be automatically identified using the Monolog dataset. Both the partial LRR and the full LRR then result in the creation of an LRR (LRR 1). The system will then subtract all part numbers from the LRR part number list, which are marked as "not launch relevant" in the systems master data. This results in a second version of the LRR (LRR 2). If the LRR is part of a US model launch, the ABC List (a list containing dealer information for US dealers) is used to identify additional information for the LRR. This results in an optional third version of the LRR (LRR 3). Finally, the LRR is translated into one or more Routing Requests, which represent a unique routing decision rule in the system.

The manually generated data model for the launch routing example is shown in figure 29. This data model was manually created by a group of three people in the parts routing team in the conception phase of the software project. The data model includes the following concepts:

- Classes: 3



launch_routing_request		
PK	l_routing_request_id	string
	*****ation	string
	*****r_list	JSON
	*****h_category	boolean
	*****h_supervisor	string
	*****h_qing	string
	*****t	string
	*****ant	string
	*****_identifier	string
	*****h_period_from	date
	*****h_period_to	date
	*****ine	string
	*****date_from	date
	*****_limit	number
	*****des	JSON
	*****hes	JSON
	*****_mdp	boolean
	*****ity_return	boolean
	*****lt_bin_sifi	string
	*****lt_bin_ut	string
	*****lt_bin_*	string
	*****lt_bin_**	string
	*****lt_bin_***	string
	*****gate_type	string
	*****gate_model	string
	*****gate_line	string
	*****gate_*****_identifier	string
	*****gate_*****on_counter	string
	*****gate_*****umber_from	string
	*****gate_*****umber_to	string
	*****asset	base64binary
	*****del	string
	*****rite_*****h_setting	boolean
	*****de_engines	boolean
	*****de_transmissions	boolean

monolog		
PK	monolog_id	string
	*****number	string
	*****del	string

routing_entries		
PK	routing_entry_id	string
FK	l_routing_request_id	string
	*****ng_cluster_id	string
	*****ting_request_id	string
	*****h	string
	*****ata_from	date
	*****ate_to	date
	*****t	string
	*****ge_location	string
	*****d_*****ge_location	string
	*****ding_point	string
	*****nty_factory	string
	*****name	string
	*****_from	date
	*****_to	date
	*****_parts	number
	*****ota	number
	*****number	string
	*****e_main_part	boolean
	*****ca	boolean
	*****laimed	boolean
	*****_limit	number
	*****ce_type	string
	*****odel	string
	*****ine	string
	_**_identifier	string
	_**on_counter	string
	__number_from	string
	__number_to	string
	*****_year_from	date
	*****_year_to	date
	*****bly_type	string
	*****gate_model	string
	*****gate_line	string
	*****gate_*****_identifier	string
	*****gate_*****on_counter	string
	*****gate_*****_number_from	string
	*****gate_*****_number_to	string
	*****al_equipment	JSON
	*****ng_time_from	number
	*****ng_time_to	number
	*****ge_from	number
	*****ge_to	number
	*****tration_date_from	date
	*****tration_date_to	date
	*****date_from	date
	*****date_to	date
	*****r_number	binary
	*****r_documents	JSON
	*****r_sds	string
	*_key	string
	*****id	string
	*****ive	boolean

Figure 29: UML CD model generated manually.

  - launch_routing_request: Attributes: 35
  - Monolog: Attributes: 3
  - routing_entries: Attributes: 53

- Associations: 1
  - 1 to * from launch_routing_request to routing_entries

This leads to an overall count of $N_{correct} = 95$.

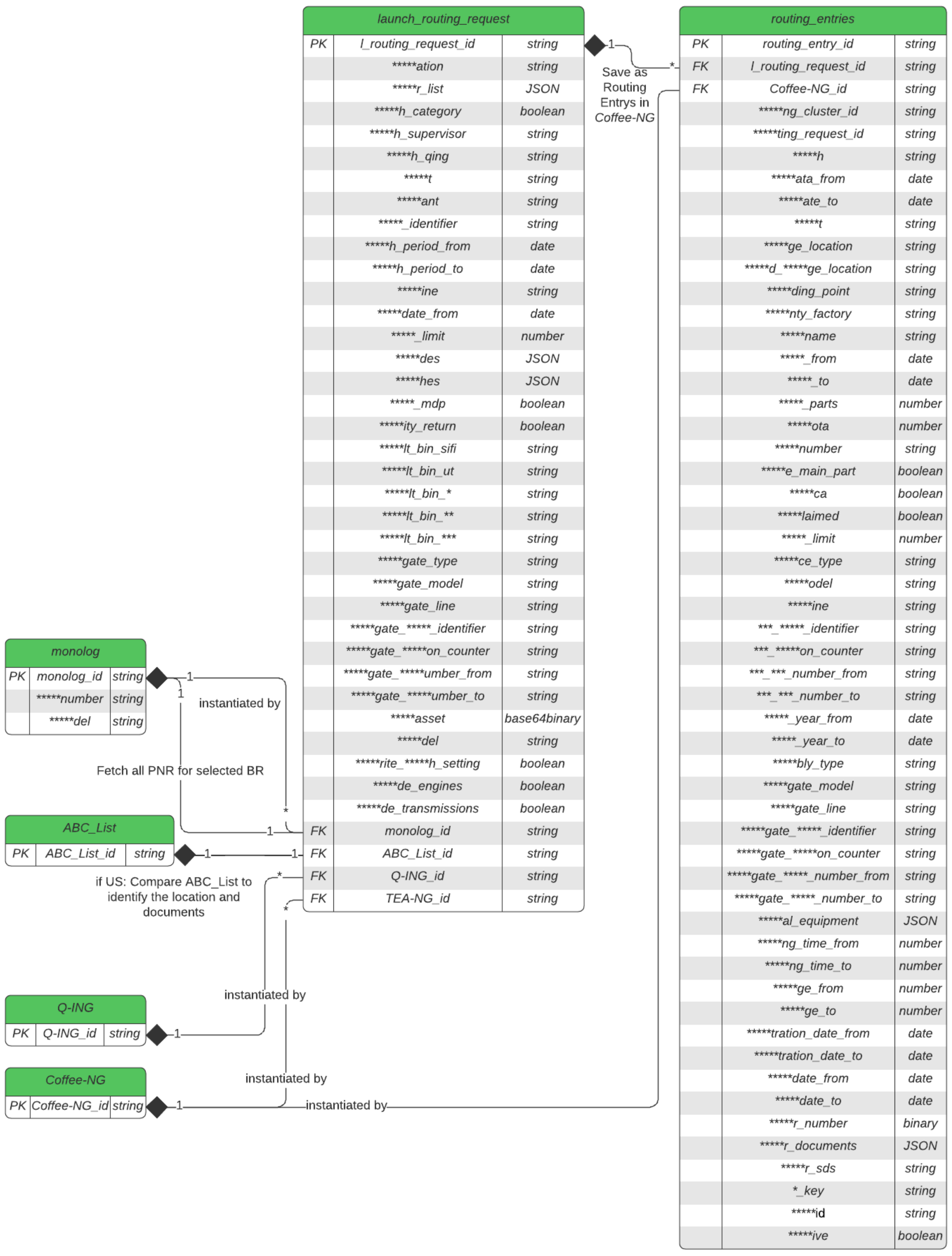
The data model generated by the DME application on the basis of an XDN for the launch routing example is shown in figure 30. This data model was generated by applying the XDN example shown in listing 7. The data model includes the following concepts:

- Classes: 6
  - launch_routing_request: Attributes: 39
  - Monolog: Attributes: 3
  - routing_entries: Attributes: 54
  - Coffee-NG: Attributes: 1
  - ABC_List: Attributes: 1
  - Q-ING: Attributes: 1
- Associations: 7
  - 1 to 0..n from launch_routing_request to routing_entries
  - 1 to 1 from Monolog to launch_routing_request
  - 1 to from Monolog to launch_routing_request
  - 1 to 1 from ABC_List to launch_routing_request
  - 1 to 1 from Q-ING to launch_routing_request
  - 1 to 1 from Coffee-NG to launch_routing_request
  - 1 to 1 from Coffee-NG to routing_entries

This leads to an overall count of $N_{generated} = 112$. The discrepancy of the attribute counts of launch_routing_request and routing_entries can be explained by the addition of five foreign keys created for these classes based on the additional associations.

Overall the following metrics were obtained for the data models:

- $N_{generated} = 112$


1
Save as Routing Entrys in Coffee-NG
*
1
1
instantiated by
Fetch all PNR for selected BR
*
1
1
1
if US: Compare ABC_List to identify the location and documents
*
*
instantiated by
1
instantiated by
1
instantiated by

launch_routing_request		
PK	l_routing_request_id	string
	*****ation	string
	*****r_list	JSON
	*****h_category	boolean
	*****h_supervisor	string
	*****h_qing	string
	*****t	string
	*****ant	string
	*****_identifier	string
	*****h_period_from	date
	*****h_period_to	date
	*****ine	string
	*****date_from	date
	*****_limit	number
	*****des	JSON
	*****hes	JSON
	*****_mdp	boolean
	*****ity_return	boolean
	*****lt_bin_sifi	string
	*****lt_bin_ut	string
	*****lt_bin_*	string
	*****lt_bin_**	string
	*****lt_bin_***	string
	*****gate_type	string
	*****gate_model	string
	*****gate_line	string
	*****gate_*****_identifier	string
	*****gate_*****on_counter	string
	*****gate_*****umber_from	string
	*****gate_*****umber_to	string
	*****asset	base64binary
	*****del	string
	*****rite_*****h_setting	boolean
	*****de_engines	boolean
	*****de_transmissions	boolean
FK	monolog_id	string
FK	ABC_List_id	string
FK	Q-ING_id	string
FK	TEA-NG_id	string

routing_entries		
PK	routing_entry_id	string
FK	l_routing_request_id	string
FK	Coffee-NG_id	string
	*****ng_cluster_id	string
	*****ting_request_id	string
	*****h	string
	*****ata_from	date
	*****ate_to	date
	*****t	string
	*****ge_location	string
	*****d_*****ge_location	string
	*****ding_point	string
	*****nty_factory	string
	*****name	string
	*****_from	date
	*****_to	date
	*****_parts	number
	*****ota	number
	*****number	string
	*****e_main_part	boolean
	*****ca	boolean
	*****laimed	boolean
	*****_limit	number
	*****ce_type	string
	*****odel	string
	*****ine	string
	_**_identifier	string
	_**on_counter	string
	__number_from	string
	__number_to	string
	*****_year_from	date
	*****_year_to	date
	*****bly_type	string
	*****gate_model	string
	*****gate_line	string
	*****gate_*****_identifier	string
	*****gate_*****on_counter	string
	*****gate_*****_number_from	string
	*****gate_*****_number_to	string
	*****al_equipment	JSON
	*****ng_time_from	number
	*****ng_time_to	number
	*****ge_from	number
	*****ge_to	number
	*****tration_date_from	date
	*****tration_date_to	date
	*****date_from	date
	*****date_to	date
	*****r_number	binary
	*****r_documents	JSON
	*****r_sds	string
	*_key	string
	*****id	string
	*****ive	boolean

monolog		
PK	monolog_id	string
	*****number	string
	*****del	string

ABC_List		
PK	ABC_List_id	string

Q-ING		
PK	Q-ING_id	string

Coffee-NG		
PK	Coffee-NG_id	string

Figure 30: UML CD model generated by the DME application.

- $N_{correct} = 95$
- $N_{incorrect} = 0$
- $N_{excessive} = 0$
- $N_{missing} = 0$
- $Recall = \frac{95}{95+0} = 1$
- $Precision = \frac{95}{95+0} = 1$

The metrics $N_{incorrect} = 0$ and $N_{missing} = 0$ result from the fact that the automatically generated data model represents a complete subset of the manually generated data model.

The metric $N_{excessive} = 0$ is derived from the assumption that the manually generated data model was created in the conception phase and is not necessarily complete. The additionally generated concepts of the automatically generated data model, such as the additional three classes Coffee-NG, ABC_List, and Q-ING (including the three additional primary keys), the five additional foreign keys as well as the six additional associations are not incorrect as such, and therefore not to be classified as excessive. These 17 concepts thus also explain the difference between $N_{generated} = 112$ and $N_{correct} = 95$.

The *Recall* and *Precision* values indicate that the automatically generated data model performs exceptionally well. A *Recall* value of 1 suggests that 100% of the concepts in the target data model were automatically generated, and none were missing. This demonstrates the effectiveness of the data model extraction process in capturing all necessary concepts from the input XDN.

A *Precision* value of 1 also reveals that 100% of the generated concepts were correct, and there were no incorrect concepts in the automatically generated data model. This result shows the high accuracy of the data model extraction process in producing a correct data model.

The execution of the quantitative evaluation process has shown that the data model generated by the DME application is both effective and accurate, as evidenced by the *Recall* and *Precision* values of 1. The data model extraction process is capable of capturing all the essential concepts and relationships present in the manually created data model on the basis of the source XDN model.

6.2 Qualitative User Feedback on DME Application

In this section, the focus shifts to the qualitative evaluation of the DME application for extracting data models from XDNs. This section seeks to gather valuable insights from users to better understand their experiences with the application, their perceptions of its usability, and its estimated impact on their work.

A survey was conducted with users from varying levels of expertise. This evaluation aims to gather information about the user experience that may not be evident through quantitative measures alone and therefore complements section 6.1. Section 6.2.1 introduces the survey setup, including the six survey questions, while section 6.2.2 aims to interpret the results of the survey.

6.2.1 Planning

The survey consisted of several questions aimed at gathering feedback on the prototype application. Respondents were asked to familiarize themselves with the DME application, including the tutorial and pre-built examples provided on the landing page. Once they had a good understanding of the application's functionality, they were asked to answer the following questions:

- How would you rate your level of expertise in working with business process models and UML Class Diagrams? Please select the option that best represents your experience.
  - a) Beginner - I have little to no experience in these topics and/or have just started learning about them.
  - b) Intermediate - I have a basic understanding and some experience working with these concepts, but still have a lot to learn.
  - c) Advanced - I have a deep understanding and significant experience in these areas, and feel confident in my abilities.
  - d) Expert - I am highly skilled in these topics, have extensive experience, and could be considered an authority in these fields.
- Based on the information entered in the process model, what other elements, relationships, or details would you have expected to see in the data model?
  - Textual answer
- Did the data model provide any additional insights that you had not considered before?
  - Textual answer
- Does the data model extraction logic of the application make sense to you? Were there any illogical transformations that you noticed in the resulting data model?

  - Textual answer

- Improvement suggestions: Are there any functions you would like to see added to the application? Were there any aspects of the software that you found confusing or difficult to use?
  - Textual answer
- Do you think that the ability to extract a UML class diagram as you just did would save you time and effort during the conception phase of software development?
  - Rating on a scale of 1 to 5, with 1 being the least time-saving and 5 being the most time-saving.

6.2.2 Execution

In the context of this project, a survey was conducted to gather feedback on the DME application. The primary aim of the survey was to evaluate the effectiveness of the application and identify potential areas for improvement. The survey results can be found in table 1.

How would you rate your level of expertise in working with business process models and UML Class Diagrams? Please select the option that best represents your experience. Respondents' expertise varied, with the majority identifying as beginners in working with BPMs and UML CDs (Respondents 1, 4, 5, and 6), and two respondents claiming intermediate experience (Respondents 2 and 3). This distribution of expertise provides a diverse perspective on the application's usability and effectiveness.

Based on the information entered in the process model, what other elements, relationships, or details would you have expected to see in the data model? The survey results revealed several suggestions for additional elements, relationships, or details that respondents expected to see in the data model. These included relationship types, keys, attributes, and class separation (Respondent 2). It can be assumed that the respondent has reversed the meaning of the question and has only named expected elements, as all those elements are represented in the target data model. Additionally, the respondent states the relationship type as an insight he has not considered before in the next question.

Did the data model provide any additional insights that you had not considered before? As already mentioned, Respondent 2 found the relationship type between classes an additional insight not considered before. Respondent 4 noted that the data model offered additional insights not considered before, specifically mentioning the association names on the relation arcs in the data model.

Does the data model extraction logic of the application makes sense to you? Were there any illogical transformations that you noticed in the resulting data model? As for the data model extraction logic, four respondents stated it makes sense (Respondents 1, 4, 5, and 6), while two stated it does not (Respondents 2, 3). For the two respondents who answered “no”, it is not clear if the answer provided is meant for the first or the second question of this point (“Does the data model extraction logic of the application make sense to you?” or “Were there any illogical transformations that you noticed in the resulting data model?”). But it can be assumed that they meant that no illogical transformations were noticed in the resulting data model as no further explanation was given.

Respondent 4 mentions that the data model extraction logic is hard to analyze but that the resulting target data model looks correct in general.

Improvement suggestions: Are there any functions you would like to see added to the application? Were there any aspects of the software that you found confusing or difficult to use? Several respondents provided suggestions for improving the application. These included allowing users to move objects around in the data model (Respondent 4), improving the size of shapes for texts (Respondent 3), adding functionality to create similar objects (Respondent 5), and improving usability on mobile devices (Respondent 6).

Do you think that the ability to extract a UML class diagram as you just did would save you time and effort during the conception phase of software development? In terms of time and effort savings during the conception phase of software development, the respondents generally believed that the ability to extract a UML CD would save time and effort. Ratings ranged from 2 (Respondent 1) to 5 (Respondents 3 and 6) on a scale of 1 to 5, with an average of 3.83.

The survey results suggest that the prototype application has potential, but improvements can be made in terms of additional elements in the data model, usability, and functionality. Respondents generally found the data model extraction logic to make sense and believed that the ability to extract a UML CD would save time and effort during the creation of a data model in the conception phase of software development. Future work on this application should focus on addressing the suggestions provided by the respondents to enhance its overall effectiveness and user experience.

6.3 PNML Validation

In this chapter, the validation of a XDN in PNMLs representation, which includes the concepts explained in the sections 4.3.3.1, 4.3.3.2 and 4.3.3.3, is discussed.

To ensure that the resulting PNML file, generated by the DME application, is valid and can be used with third-party tools, a validation check is performed. The PNML project website recommends several methods for validating PNML files [7]. In this case, the PNML Document Checker [8] is used for validating the PNML XML string resulting from the designer component.

Figure 31 shows an example output from the PNML Document Checker when validating the "Launch Routing" example (see listing 7):

```
HTTP/1.1 200 OK
Message: Your PNML document conforms to the standard specifications.
Model name: Launch Routing
Model type: High-level Petri Net
Number of nets: 1
Number of places: 5
Number of transitions: 6
Number of arcs: 11
Number of reference places: 0
Number of reference transitions: 0

08:28:58.936 [main] INFO  f.l.m.pnml.validation.ValidationMain - Checked by PNML DoC 1.2.4
08:28:58.938 [main] INFO  f.l.m.pnml.validation.ValidationMain - The PNML checking took
1.9363175 seconds.
```

Figure 31: Output of the PNML Document Checker checking the Launch Routing example.

The output shown in listing 31 indicates that the PNML document conforms to the standard specifications while providing information about the model name, type, and the number of elements within the net, such as places, transitions, and arcs. This validation process ensures that the XDN generated by the designer component conforms to the PNML high-level net standard and can be utilized in various tools and applications.

In summary, the development of XDN brings several advantages to the field of data model extraction from BPMs. These advantages include:

- Improved expressiveness: XDNs extensions to the PNML High-Level Core Structure enable users to represent more complex relationships and behavior in BPMs, facilitating more accurate data model extraction.

[7]https://www.pnml.org/validation.php

[8]http://pnml.lip6.fr/pnmlvalidation/index.html

- Improved compatibility: By adhering to the PNML specification, XDN is compatible with a wide range of existing tools and software that support PNML, making it easier to integrate into existing workflows and processes.

7 Conclusion

In this paper, we have presented a novel approach for the automatic extraction of data models from BPMs by introducing XDN and developing a prototype application for data model extraction. This work demonstrates the potential benefits of this approach in terms of time and resource savings, as well as maintaining high quality and accuracy in the extracted data models.

7.1 Summary

After the analysis of various BPMLs and their respective DIFs, Petri nets are selected as the source BPML. They were chosen for their essential elements and flexible DIF that allows for easy customization. Afterward, different levels of data models were discussed, with the UML CD based on a logical abstraction level chosen as the target data model notation. In order to find a solution how to define an extraction algorithm, existing literature was analyzed. this literature could show some basic principles for the extraction of data models. This serves as a basis for the conceptual design of the approach to data model extraction presented in this work.

The conception phase included creating XDN by extending the PNML High-Level Core Structure. A ruleset for data model extraction was then developed to guide the data model extraction process.

The implementation of this work serves as a foundation for the evaluation and future research on the practical implementation of data model extraction based on BPMs. In addition, a prototype application, including an XML service layer for the abstraction of the XML DOM for XDN documents, was developed.

The evaluation of the DME application involved quantitative measures, such as high Recall and Precision values, which demonstrated the effectiveness and accuracy of the data model extraction process. The qualitative evaluation, based on a survey conducted with users of varying expertise levels, provides insights into the application's usability and potential impact on software development. These results offer strong evidence supporting the data model extraction of logical database models based on BPMs, as it can save time and resources compared to manual creation methods while maintaining high quality and accuracy. Furthermore, the evaluation confirmed that XDN adheres to the PNML specification, ensuring compatibility with other third-party software implementing the PNML standard.

In conclusion, this work contributes significantly to the field of data model extraction from BPMs by presenting a novel approach through the creation of XDN, the development of a ruleset, and the implementation of a prototype application.

7.2 Contribution and limitations

By introducing XDN and a ruleset for the extraction of data models, this work offers an efficient approach to automating data model extraction on the basis of XDNs, leading to several key benefits:

1. Time and resource savings: The automation of data model extraction can greatly reduce the time and effort required in the conception phase of software development projects, enabling developers to focus on other critical aspects of the project.

2. Improved expressiveness: XDNs enables users to represent more complex relationships and behavior in BPMs, facilitating more accurate data model extraction and providing a richer representation of the underlying data structures. This can be especially helpful for the software development process.

3. Enhanced compatibility: By adhering to the PNML specification, XDNs are compatible with a wide range of existing tools and software that support PNML, making it easier to integrate into existing workflows and processes.

Despite the valuable contributions of this work, there are limitations that should be acknowledged and addressed in future research. One of these limitations is the limited user evaluation. The qualitative evaluation of the DME application was based on a relatively small sample of respondents with varying expertise levels. A more extensive user evaluation with a larger and more diverse sample could provide more robust insights into the application's usability and effectiveness.

Another limitation is the incomplete coverage of BPM elements. While XDN offers improved expressiveness, there may be elements and relationships within other BPM models that are not fully captured or translated by the current implementation. Future work should explore the inclusion of additional BPM elements and relationships to ensure comprehensive coverage. An example of this is the method generation for UML CDs. A potential solution to this limitation could involve incorporating the transition expressions found in XML-Nets. To achieve this, it would be necessary to implement the concept of transition expressions from XML-Nets in the XDN definition and establish a new rule for facilitating method generation in UML CDs. Another element included in Petri Nets are the firing rules for transitions which could be used to generate associated concepts in the target data model.

Furthermore, there are opportunities to improve the DME application's usability and functionality, as indicated by the survey results. Addressing these limitations will enhance the user experience and make the application more accessible to a wider audience.

Lastly, the prototype application's reliance on a limited number of data types, specifically XML primitive types, can be restrictive in certain contexts. The lack of support for JSON data types may not fully address the needs of modern software development. To improve the application's versatility, it would be beneficial to expand the range of data types supported, including the addition of JSON data types.

7.3 Future Research

In conclusion, this work presents valuable contributions to the field of data model extraction from BPMs, providing a foundation for future research and development.

Possible topics for future research may include:

- Enhancing the presented rulesets and capabilities of XDN to support the creation of more concepts in the target data model.
- Synthesizing of the rulesets to work with different source BPMLs and target data model notations.

By acknowledging and addressing the limitations of the current work, future research can further refine the concepts and applications presented, resulting in a more robust and versatile solution for data model extraction in the early phases of software development projects.

A Appendix

A.1 XDN Examples

```
<pnml xmlns="http://www.pnml.org/version-2009/grammar/pnml">
        <net id="ne8aa6769-080c-452b-aaf4-2b163dd723ed" type="
           http://www.pnml.org/version-2009/grammar/highlevelnet
           ">
    <name>
        <text>Launch Routing</text>
    </name>
    <page id="a7ec6279c-021f-48c6-93cc-6105b439bb75">
        <place id="ka63229c9-68b3-46db-b9f2-d0782fc164b6">
            <graphics>
                <position x="174" y="140"/>
            </graphics>
            <name>
                <text>Partial LRR</text>
            </name>
            <toolspecific xmlns="" tool="dme" version="1.4.2">
                <tokenSchema xmlns:xs="http://www.w3.org/2001/
                   XMLSchema" name="LRR" superClass="">
                    <xs:element name="xlsx_asset" type="
                       base64Binary" isPrimaryKey="false"/>
                </tokenSchema>
            </toolspecific>
        </place>
        <place id="zd4715195-370b-471e-9156-47ed3e7196fe">
            <graphics>
                <position x="169" y="305"/>
            </graphics>
            <name>
                <text>Full LRR</text>
            </name>
            <toolspecific xmlns="" tool="dme" version="1.4.2">
                <tokenSchema xmlns:xs="http://www.w3.org/2001/
                   XMLSchema" name="LRR" superClass="">
                    <xs:element name="*****del" type="string"
                       isPrimaryKey="false"/>
                    <xs:element name="*****rite_*****h_setting"
                       type="boolean" isPrimaryKey="false"/>
                    <xs:element name="*****de_engines" type="
                       boolean" isPrimaryKey="false"/>
                    <xs:element name="*****de_transmissions"
                       type="boolean" isPrimaryKey="false"/>
                </tokenSchema>
            </toolspecific>
        </place>
        <transition id="e528fcd9f-2c21-4f61-ad1c-cb9a6858c28a">
            <graphics>
                <position x="300" y="140"/>
            </graphics>
            <name>
                <text>Upload PNR+BR list</text>
            </name>
            <toolspecific tool="dme" version="1.4.2">
                <role>
                    <text>Q-ING</text>
                </role>
            </toolspecific>
        </transition>
        <transition id="me5c65bc3-59d9-4d7f-bcd9-bca8de4e6eae">
            <graphics>
                <position x="70" y="230"/>
            </graphics>
            <name>
                <text>Create LRR</text>
            </name>
            <toolspecific tool="dme" version="1.4.2">
                <role>
                    <text>Q-ING</text>
                </role>
            </toolspecific>
        </transition>
        <arc id="z114500f7-7ec4-4b70-93d2-426041fbdd6f" source="
           me5c65bc3-59d9-4d7f-bcd9-bca8de4e6eae" target="
           ka63229c9-68b3-46db-b9f2-d0782fc164b6">
           <hlinscription>1</hlinscription>
```

```
</arc>
<arc id="j82a9527c-44a1-4d2a-86e3-69e16d6cdb8d" source="
   me5c65bc3-59d9-4d7f-bcd9-bca8de4e6eae" target="
   zd4715195-370b-471e-9156-47ed3e7196fe">
    <hlinscription>1</hlinscription>
</arc>
<arc id="jae0b19b2-4dc8-4254-8525-99e7243c02bf" source="
   ka63229c9-68b3-46db-b9f2-d0782fc164b6" target="
   e528fcd9f-2c21-4f61-ad1c-cb9a6858c28a">
    <hlinscription>1</hlinscription>
</arc>
<transition id="l6791c3ce-5024-4cb7-8bf3-32bde02407d8">
    <graphics>
        <position x="300" y="310"/>
    </graphics>
    <name>
        <text>Fetch all PNR for selected BR</text>
    </name>
    <toolspecific tool="dme" version="1.4.2">
        <role>
            <text>Monolog</text>
        </role>
    </toolspecific>
</transition>
<arc id="j51ad076d-2fb6-44cf-bf59-92a8137f5561" source="
   zd4715195-370b-471e-9156-47ed3e7196fe" target="
   l6791c3ce-5024-4cb7-8bf3-32bde02407d8">
    <hlinscription>1</hlinscription>
</arc>
<place id="ia5ce6bab-eb2e-49ff-9699-a35b9f700c1b">
    <graphics>
        <position x="180" y="420"/>
    </graphics>
    <name>
        <text>Monolog Data</text>
    </name>
    <toolspecific xmlns="" tool="dme" version="1.4.2">
        <tokenSchema xmlns:xs="http://www.w3.org/2001/
           XMLSchema" name="Monolog" superClass="">
```

```
                <xs:element name="*****number" type="string"
                     isPrimaryKey="false"/>
                <xs:element name="*****del" type="string"
                    isPrimaryKey="false"/>
            </tokenSchema>
        </toolspecific>
    </place>
    <arc id="u2bd72cdf-09f2-4c5e-a4ce-ae235df20168" source="
       ia5ce6bab-eb2e-49ff-9699-a35b9f700c1b" target="
       l6791c3ce-5024-4cb7-8bf3-32bde02407d8">
        <hlinscription>1</hlinscription>
    </arc>
    <place id="tca895ac0-1ebe-46eb-9f0f-59040395c7c2">
        <graphics>
            <position x="439" y="230"/>
        </graphics>
        <name>
            <text>LRR 1</text>
        </name>
        <toolspecific xmlns="" tool="dme" version="1.4.2">
            <tokenSchema xmlns:xs="http://www.w3.org/2001/
                XMLSchema" name="LRR" superClass="">
                <xs:element name="l_routing_request_id" type
                    ="string" isPrimaryKey="true"/>
                <xs:element name="*****ation" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*****r_list" type="string"
                     isPrimaryKey="false"/>
                <xs:element name="*****h_category" type="
                    boolean" isPrimaryKey="false"/>
                <xs:element name="*****h_supervisor" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****h_qing" type="string"
                     isPrimaryKey="false"/>
                <xs:element name="*****t" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*****ant" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*****_identifier" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****h_period_from" type="
                    date" isPrimaryKey="false"/>
                <xs:element name="*****h_period_to" type="
                    date" isPrimaryKey="false"/>
                <xs:element name="*****ine" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*****date_from" type="date
                    " isPrimaryKey="false"/>
                <xs:element name="*****_limit" type="decimal
                    " isPrimaryKey="false"/>
                <xs:element name="*****des" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*****hes" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*****_mdp" type="boolean"
                    isPrimaryKey="false"/>
                <xs:element name="*****ity_return" type="
                    boolean" isPrimaryKey="false"/>
                <xs:element name="*****lt_bin_sifi" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****lt_bin_ut" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****lt_bin_*" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****lt_bin_**" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****lt_bin_***" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****gate_type" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****gate_model" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****gate_line" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****gate_*****_identifier
                    " type="string" isPrimaryKey="false"/>
                <xs:element name="*****gate_*****on_counter"
                     type="string" isPrimaryKey="false"/>
```

```
            <xs:element name="*****gate_*****umber_from"
                type="string" isPrimaryKey="false"/>
            <xs:element name="*****gate_*****umber_to"
               type="string" isPrimaryKey="false"/>
        </tokenSchema>
    </toolspecific>
</place>
<arc id="cd54727d0-cc3f-4bb1-9f5b-2ec8c0720ee5" source="
   e528fcd9f-2c21-4f61-ad1c-cb9a6858c28a" target="
   tca895ac0-1ebe-46eb-9f0f-59040395c7c2">
    <hlinscription>1</hlinscription>
</arc>
<arc id="y6694475b-8ad3-4281-8246-6c87b905820a" source="
   l6791c3ce-5024-4cb7-8bf3-32bde02407d8" target="
   tca895ac0-1ebe-46eb-9f0f-59040395c7c2">
    <hlinscription>1</hlinscription>
</arc>
<transition id="y5859650c-c7b2-40da-b076-983caad1032b">
    <graphics>
        <position x="577" y="230"/>
    </graphics>
    <name>
        <text>Subtract all Clusters in which BR+PNR+
           Plant is marked "not launch relevant"</text>
    </name>
    <toolspecific tool="dme" version="1.4.2">
        <role>
            <text>Coffee-NG</text>
        </role>
    </toolspecific>
</transition>
<arc id="xa4349da5-7f43-4e00-ac00-9d031b69271d" source="
   tca895ac0-1ebe-46eb-9f0f-59040395c7c2" target="
   y5859650c-c7b2-40da-b076-983caad1032b">
    <hlinscription>1</hlinscription>
</arc>
<place id="g9defe9eb-b663-47a8-9461-34a147af7c3c">
    <graphics>
        <position x="700" y="230"/>
```

```
            </graphics>
            <name>
                <text>LRR 2</text>
            </name>
            <toolspecific xmlns="" tool="dme" version="1.4.2">
                <tokenSchema xmlns:xs="http://www.w3.org/2001/
                   XMLSchema" name="LRR" superClass=""/>
                </toolspecific>
        </place>
        <arc id="e2978e011-e5de-44f9-a353-d9d5e16519db" source="
           y5859650c-c7b2-40da-b076-983caad1032b" target="
           g9defe9eb-b663-47a8-9461-34a147af7c3c">
            <hlinscription>1</hlinscription>
        </arc>
        <transition id="p4561e490-0b4c-494b-ae9f-cf5d3abec814">
            <graphics>
                <position x="829" y="310"/>
            </graphics>
            <name>
                <text>If US: Compare ABC List to identify the
                   location and documents</text>
            </name>
            <toolspecific tool="dme" version="1.4.2">
                <role>
                    <text>Coffee-NG</text>
                </role>
            </toolspecific>
        </transition>
        <arc id="b7b24e3c5-c8c2-491b-9791-de1ba2628794" source="
           g9defe9eb-b663-47a8-9461-34a147af7c3c" target="
           p4561e490-0b4c-494b-ae9f-cf5d3abec814">
            <hlinscription>1</hlinscription>
        </arc>
        <place id="redc5ce81-0815-4ea9-8874-fd0541bca45f">
            <graphics>
                <position x="700" y="390"/>
            </graphics>
            <name>
                <text>ABC List</text>
```

```
          </name>
          <toolspecific xmlns="" tool="dme" version="1.4.2">
               <tokenSchema xmlns:xs="http://www.w3.org/2001/
                  XMLSchema" name="ABC_List" superClass=""/>
               </toolspecific>
     </place>
     <arc id="b61e848d8-b09a-4abc-865e-f97df13bf7c3" source="
        redc5ce81-0815-4ea9-8874-fd0541bca45f" target="
        p4561e490-0b4c-494b-ae9f-cf5d3abec814">
          <hlinscription>1</hlinscription>
     </arc>
     <transition id="k4a3aee63-8e0b-4d0b-8f6a-fea88ea852c8">
          <graphics>
               <position x="1040" y="230"/>
          </graphics>
          <name>
               <text>Save as Routing Entrys in Coffee-NG</text>
          </name>
          <toolspecific tool="dme" version="1.4.2">
               <role>
                    <text>Coffee-NG</text>
               </role>
          </toolspecific>
     </transition>
     <place id="p7437f8e5-6c1f-47da-9f09-b3b10980299a">
          <graphics>
               <position x="953" y="310"/>
          </graphics>
          <name>
               <text>LRR 3</text>
          </name>
          <toolspecific xmlns="" tool="dme" version="1.4.2">
               <tokenSchema xmlns:xs="http://www.w3.org/2001/
                  XMLSchema" name="LRR" superClass=""/>
               </toolspecific>
     </place>
     <arc id="zb30dd785-7631-4606-97cd-92e403ad6c06" source="
        p4561e490-0b4c-494b-ae9f-cf5d3abec814" target="
        p7437f8e5-6c1f-47da-9f09-b3b10980299a">
          <hlinscription>1</hlinscription>
     </arc>
     <arc id="b96716f9c-1b3a-40c6-a8bf-a146dd4f8fe9" source="
        p7437f8e5-6c1f-47da-9f09-b3b10980299a" target="
        k4a3aee63-8e0b-4d0b-8f6a-fea88ea852c8">
          <hlinscription>1</hlinscription>
     </arc>
     <arc id="d631d9224-f79b-40b9-9178-3969e007a4db" source="
        g9defe9eb-b663-47a8-9461-34a147af7c3c" target="
        k4a3aee63-8e0b-4d0b-8f6a-fea88ea852c8">
          <hlinscription>1</hlinscription>
     </arc>
     <place id="a5673094c-fafe-439f-b218-23f49e23de3a">
          <graphics>
               <position x="1169" y="230"/>
          </graphics>
          <name>
               <text>Routing Entry</text>
          </name>
          <toolspecific xmlns="" tool="dme" version="1.4.2">
               <tokenSchema xmlns:xs="http://www.w3.org/2001/
                  XMLSchema" name="RoutingEntry" superClass="">
                    <xs:element name="routing_entry_id" type="
                       string" isPrimaryKey="true"/>
                    <xs:element name="*****ng_cluster_id" type="
                       string" isPrimaryKey="false"/>
                    <xs:element name="*****ting_request_id" type
                       ="string" isPrimaryKey="false"/>
                    <xs:element name="*****h" type="string"
                       isPrimaryKey="false"/>
                    <xs:element name="*****ata_from" type="date"
                        isPrimaryKey="false"/>
                    <xs:element name="*****ate_to" type="date"
                       isPrimaryKey="false"/>
                    <xs:element name="*****t" type="string"
                       isPrimaryKey="false"/>
                    <xs:element name="*****ge_location" type="
                       string" isPrimaryKey="false"/>
                    <xs:element name="*****d_*****ge_location"
```

```
   type="string" isPrimaryKey="false"/>
<xs:element name="*****ding_point" type="
   string" isPrimaryKey="false"/>
<xs:element name="*****nty_factory" type="
   string" isPrimaryKey="false"/>
<xs:element name="*****name" type="string"
   isPrimaryKey="false"/>
<xs:element name="*****_from" type="date"
   isPrimaryKey="false"/>
<xs:element name="*****_to" type="date"
   isPrimaryKey="false"/>
<xs:element name="*****_parts" type="decimal
   " isPrimaryKey="false"/>
<xs:element name="*****ota" type="decimal"
   isPrimaryKey="false"/>
<xs:element name="*****number" type="string"
    isPrimaryKey="false"/>
<xs:element name="*****e_main_part" type="
   boolean" isPrimaryKey="false"/>
<xs:element name="*****ca " type="boolean"
   isPrimaryKey="false"/>
<xs:element name="*****laimed" type="boolean
   " isPrimaryKey="false"/>
<xs:element name="*****_limit" type="decimal
   " isPrimaryKey="false"/>
<xs:element name="*****ce_type" type="string
   " isPrimaryKey="false"/>
<xs:element name="*****odel" type="string"
   isPrimaryKey="false"/>
<xs:element name="*****ine" type="string"
   isPrimaryKey="false"/>
<xs:element name="***_*****_identifier" type
   ="string" isPrimaryKey="false"/>
<xs:element name="***_*****on_counter" type=
   "string" isPrimaryKey="false"/>
<xs:element name="***_***_number_from" type=
   "string" isPrimaryKey="false"/>
<xs:element name="***_***_number_to" type="
   string" isPrimaryKey="false"/>
```

```
<xs:element name="*****_year_from" type="
   date" isPrimaryKey="false"/>
<xs:element name="*****_year_to" type="date"
    isPrimaryKey="false"/>
<xs:element name="*****bly_type" type="
   string" isPrimaryKey="false"/>
<xs:element name="*****gate_model" type="
   string" isPrimaryKey="false"/>
<xs:element name="*****gate_line" type="
   string" isPrimaryKey="false"/>
<xs:element name="*****gate_*****_identifier
   " type="string" isPrimaryKey="false"/>
<xs:element name="*****gate_*****on_counter"
    type="string" isPrimaryKey="false"/>
<xs:element name="*****gate_*****
   _number_from" type="string" isPrimaryKey=
   "false"/>
<xs:element name="*****gate_*****_number_to"
    type="string" isPrimaryKey="false"/>
<xs:element name="*****al_equipment" type="
   string" isPrimaryKey="false"/>
<xs:element name="*****ng_time_from" type="
   decimal" isPrimaryKey="false"/>
<xs:element name="*****ng_time_to" type="
   decimal" isPrimaryKey="false"/>
<xs:element name="*****ge_from" type="
   decimal" isPrimaryKey="false"/>
<xs:element name="*****ge_to" type="decimal"
    isPrimaryKey="false"/>
<xs:element name="*****tration_date_from"
   type="date" isPrimaryKey="false"/>
<xs:element name="*****tration_date_to" type
   ="date" isPrimaryKey="false"/>
<xs:element name="*****date_from" type="date
   " isPrimaryKey="false"/>
<xs:element name="*****date_to" type="date"
   isPrimaryKey="false"/>
<xs:element name="*****r_number" type="
   string" isPrimaryKey="false"/>
```

```
                <xs:element name="*****r_documents" type="
                    string" isPrimaryKey="false"/>
                <xs:element name="*****r_sds" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*_key" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*****id" type="string"
                    isPrimaryKey="false"/>
                <xs:element name="*****ive" type="boolean"
                    isPrimaryKey="false"/>
            </tokenSchema>
        </toolspecific>
    </place>
    <arc id="c1ad4bf97-bd83-403e-8e6f-970940335aa0" source="
        k4a3aee63-8e0b-4d0b-8f6a-fea88ea852c8" target="
        a5673094c-fafe-439f-b218-23f49e23de3a">
        <hlinscription xmlns=""><text>*</text></
            hlinscription>
    </arc>
  </page>
  </net>
</pnml>
```

Listing 7: XDN representation of the Launch Routing example

A.2 Survey Results

Respondent	How would you rate your level of expertise in working with business process models and UML Class Diagrams? Please select the option that best represents your experience.	Based on the information entered in the process model, what other elements, relationships, or details would you have expected to see in the data model?	Did the data model provide any additional insights that you had not considered before?	Does the data model extraction logic of the application make sense to you? Were there any illogical transformations that you noticed in the resulting data model?	Improvement suggestions: Are there any functions you would like to see added to the application? Were there any aspects of the software that you found confusing or difficult to use?	Do you think that the ability to extract a UML class diagram as you just did would save you time and effort during the conception phase of software development?
1	a) Beginner - I have little to no experience in these topics and/or have just started learning about them.	Nothing	No	Makes sense	No	2
2	b) Intermediate - I have a basic understanding and some experience working with these concepts, but still have a lot to learn.	relationship type, keys, attributes, separation in classes	relationship type	No	in the beginning the connection of the objects was a bit unusual, but i quickly got used to it	4

3	b) Intermediate - I have a basic understanding and some experience working with these concepts, but still have a lot to learn.	,"no	but i guess it helps to get a better overview of the actuall process from start to end"	no	The shapes are generally to small for the texts	5
4	a) Beginner - I have little to no experience in these topics and/or have just started learning about them.	no other	I did not expect the association names on the relation arcs in the data model	makes sense	add function in DME to allow user to move objects around	3
5	a) Beginner - I have little to no experience in these topics and/or have just started learning about them.	I would have liked to define a type e.g. Bins as location and than have sub types with the same attributes. I saw the master class field, but that only works for existing fields.	Nope	Hard to analize due to the added complexity by the haphazard display of the tables. Looked right though	A lot of usability topics, but as it is a prototype not going into detail. As described above a function to create similar objects would be nice.	4

6	a) Beginner - I have little to no experience in these topics and/or have just started learning about them.	-	Yes	Makes sense. Furthermore is very easy to use and understand the application.	Sometimes its hard to use with a mobile phone but still possible and useable!	5

Table 1: Results of the survey on the DME prototype application

References

[Banjac et al.(2018)] Danijela Banjac, Drazen Brdjanin, Goran Banjac, and Slavko Maric. 2018. Evaluation of Automatically Generated Conceptual Database Model Based on Business Process Model: Controlled Experiment. Springer, Cham, 134–145. `https://doi.org/10.1007/978-3-319-68855-8{_}13`

[Brdjanin et al.(2018)] Drazen Brdjanin, Danijela Banjac, Goran Banjac, and Slavko Maric. 2018. An Online Business Process Model-driven Generator of the Conceptual Database Model. In *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*, Rajendra Akerkar, Mirjana Ivanović, Sang-Wook Kim, Yannis Manolopoulos, Riccardo Rosati, Miloš Savić, Costin Badica, and Miloš Radovanović (Eds.). ACM, New York, NY, USA, 1–9. `https://doi.org/10.1145/3227609.3227666`

[Brdjanin and Maric(2012)] Drazen Brdjanin and Slavko Maric. 2012. An approach to automated conceptual database design based on the UML activity diagram. *Computer Science and Information Systems* 9, 1 (2012), 249–283. `https://doi.org/10.2298/CSIS110318069B`

[Brdjanin and Maric(2014)] Drazen Brdjanin and Slavko Maric. 2014. Model-driven Techniques for Data Model Synthesis. *Electronics* 17, 2 (2014), 130–136. `https://doi.org/10.7251/ELS1317130B`

[Cincović and Punt(2020)] Jelica Cincović and Marija Punt. 2020. *Comparison: Angular vs. React vs. Vue. Which framework is the best choice?* `http://www.eventiotic.com/eventiotic/files/papers/url/50173409-699e-4b17-8edb-9764ecc53160.pdf`

[Cruz et al.(2012)] Estrela Ferreira Cruz, Ricardo J. Machado, and Maribel Y. Santos. 2012. From Business Process Modeling to Data Model: A Systematic Approach. In *2012 Eighth International Conference on the Quality of Information and Communications Technology*. IEEE, 205–210. `https://doi.org/10.1109/QUATIC.2012.31`

[Hofstede et al.(2010)] Arthur H. M. Hofstede, Wil M. P. Aalst, Michael Adams, and Nick Russell. 2010. *Modern Business Process Automation: YAWL and its Support Environment*. Springer, Berlin, Heidelberg. `https://doi.org/10.1007/978-3-642-03121-2`

[International Organization for Standardization(2000)] International Organization for Standardization. 2000. High-level Petri Nets - Concepts, Definitions and Graphical Notation. (2000).

[Jensen(1987)] Kurt Jensen. 1987. Coloured Petri Nets. In *Petri Nets: Central Models and Their Properties*. Springer, Berlin, Heidelberg, 248–299. `https://doi.org/10.1007/978-3-540-47919-2{_}10`

[Keller et al.(1992)] G. Keller, M. Nüttgens, and August-Wilhelm Scheer. 1992. *Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)"*. Institut für Wirtschaftsinformatik.

[Klink et al.(2008)] Stefan Klink, Yu Li, and Andreas Oberweis. 2008. INCOME2010 - a Toolset for Developing Process-Oriented Information Systems Based on Petri Nets. 14. `https://doi.org/10.1145/1416222.1416241`

[Kurz et al.(2022)] Matthias Kurz, Falko Menge, and Misiak Zbigniew. 2022. BPMN_Interchange. (2022).

[Lenz and Oberweis(2001)] Kirsten Lenz and Andreas Oberweis. 2001. Modeling Interorganizational Workflows with XML Nets. (2001).

[Mendling and Nüttgens(2006)] Jan Mendling and Markus Nüttgens. 2006. EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC). *Information Systems and e-Business Management* 4, 3 (2006), 245–263. `https://doi.org/10.1007/s10257-005-0026-1`

[OMG(2012)] OMG. 2012. Diagram Definition (DD). (2012). `https://www.omg.org/spec/DD/1.0/PDF`

[OMG(2013)] OMG. 2013. Business Process Model and Notation (BPMN), Version 2.0.2. `http://www.omg.org/spec/BPMN/2.0.2`

[OMG(2017)] OMG. 2017. Unified Modeling Language (UML), Version 2.5.1. `https://www.omg.org/spec/BPMN/2.0/About-BPMN`

[OMG(2022)] OMG. 2022. XML Metadata Interchange (XMI) Specification Version 2.4.1. `https://www.omg.org/spec/XMI/2.4.1/About-XMI`

[Petri(1962)] Carl Adam Petri. 1962. *Kommunikation mit Automaten.* Ph. D. Dissertation. TU Darmstadt. `https://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/`

[Reisig(1982)] Wolfgang Reisig. 1982. Petrinetze: Eine Einführung. `https://link.springer.com/book/10.1007/978-3-642-96705-4`

[Sommerville(2015)] Ian Sommerville. 2015. *Software Engineering.* `https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf`

[Steel, Jr., Thomas B.(1975)] Steel, Jr., Thomas B. 1975. Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems 75-02-08. *Bulletin of ACM SIGMOD* 7, 2 (1975), 1–140.

[Thomas Karle et al.(2006)] Thomas Karle, Björn Keuter, Stefan Klink, Daniel Ried, Yu Li, Marco Mevius, Markus Zaich, Timm Caporale, Murat Citak, and Andreas Oberweis. 2006. KIT-Horus. `https://www.aifb.kit.edu/web/KIT-Horus/en`

[van der Aalst(2015)] W. M. P. van der Aalst. 2015. Business process management as the "Killer App" for Petri nets. *Software & Systems Modeling* 14, 2 (2015), 685–691. `https://doi.org/10.1007/s10270-014-0424-2`

[Van der Werf, Jan Martijn E.M. and Post(2004)] Van der Werf, Jan Martijn E.M. and Reinier Post. 2004. EPNML 1.1 - an XML format for Petri nets. (2004).

[Weber and Kindler(2003)] Michael Weber and Ekkart Kindler. 2003. The Petri Net Markup Language. In *Petri Net Technology for Communication-Based Systems*. Springer, Berlin, Heidelberg, 124–144. `https://doi.org/10.1007/978-3-540-40022-6{_}7`

[Weske(2019)] Mathias Weske. 2019. *Business Process Management: Concepts, Languages, Architectures* (3rd ed.). Springer Berlin Heidelberg, Berlin, Heidelberg. `https://books.google.com/books?id=-D5tpT5Xz8oC`

Assertion

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, August 29, 2023 David Diener