universität wien

MASTERARBEIT

Titel der Masterarbeit

Modelling Languages and Algorithms: A Matching Mechanism

An Empirical Application of Petri Nets

verfasst von

Nesat Efendioglu

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2014

Studienkennzahl lt. Studienblatt:	A 066 926
Studienrichtung lt. Studienblatt:	Masterstudium Wirtschaftsinformatik
Betreuerin / Betreuer:	o. Univ.-Prof. Dr. Prof. Dimitris Karagiannis

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mir den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Datum	Unterschrift
19.03.2014	Nesat Efendioglu

Acknowledgement

I would like to express my greatest gratitude to Prof. Karagiannis for being my mentor during my academic and professional career. I would also like to give my sincere thanks to my supervisor and colleague Dr. Robert Woitsch, who supported me with his comprehensive knowledge and guided me with numerous delighting and proficient discussions. Without his contribution, his willingness to share his knowledge with me, his endless patience during the discussions and his moral support this work would never have been managed to be so conceptually and technically fruitful for the open source community.

Furthermore, I would like to thank my dear collegial friends Vedran Hrgovcic, Sabin Popescu, Wilfrid Utz, and Ronald Quirchmayr, who did not hesitate to spend time for advising me in their fields of expertise, as well as my friend Christos Lekaditis, who helped me to correct misspellings in this work with his excellent English-knowledge.

Mostly, I would like to thank my family, who supported me during my studies. They have always been there for me during this long journey.

Finally, I would like to devote this thesis to my precious wife Emine Efendioglu and my precious son Ege Efendioglu, who appreciated the time that I have to spend to finish this work, and supported me with all their heart.

Table of Contents

Eidesstattliche Erklärung i
Acknowledgement ii
Table of Contents iii
Table of Figures vi
Table of Tables viii
Table of Code Snippet ix
Table of Abbreviation x
Abstract 1
1 Introduction 2
2 Modelling Methods 8
2.1 Generic Modelling Method Specification Framework 8
2.1.1 Modelling Languages 9
2.1.2 Modelling Procedures 11
2.1.3 Mechanisms and Algorithms 11
2.2 The ADOxx® Metamodelling Platform 12
2.2.1 Model Hierarchy 13
2.2.2 Specification of Modelling Languages with ADOxx® 13
2.2.3 Specification of Mechanisms and Algorithms 19
2.3 Formal Description of Meta Models Based on ADOxx® 19
3 Petri Nets 21
3.1 Introduction in Petri Nets 21
3.2 List of identified Modelling Languages 24
3.2.1 Elementary Petri Nets Modelling Language 24
3.2.2 Place/Transition Petri Nets Modelling Language 29
3.2.3 Coloured Petri Nets Modelling Language 34
3.3 List of identified Mechanisms and Algorithms 40
3.3.1 Fast Simulation 40
3.3.2 Step by Step Simulation 41
3.3.3 Net Statistics 42
3.3.4 PNML Transformation 43
3.3.5 Genetic Algorithm 45

3.3.6 ADOxx® Path Analysis Algorithm 45
4 Description of Mechanisms & Algorithms for Matching with Modelling Languages 48
4.1 Matching Dimensions for Mechanisms and Algorithms 49
4.2 Semantic Service Description: state-of-the-art 53
4.2.1 WSDL: Web Service Description Language 53
4.2.2 OWL 2 Web Ontology Language 55
4.2.3 OWL-S: Semantic Markup for Web Services 58
4.2.4 WSMO: Web Service Modelling Ontology 59
4.2.5 SAWSDL: Semantic Annotations for WSDL and XML Schema 60
4.2.6 Lessons Learned from State-of-the-Art Analysis 61
4.3 Describing Mechanisms & Algorithms Formally 62
4.3.1 Formal Description of Mechanisms 62
4.3.2 Formal Description of Goals 70
4.3.3 Taxonomy as Classification Schema 74
4.3.4 Petri Nets Ontology 76
5 Realization of Matching Mechanism 79
5.1 Semantic Enriched Mechanism and Algorithm Description Language 79
5.1.1 Model Type for Describing Mechanisms: Mechanism Description Model Type 81
5.1.2 Model Type for Building Ontology: OWL Model Type 84
5.2 Similarity Calculation between Semantic Concepts 87
5.3 Matching Mechanism 89
5.3.1 Matching Algorithm 90
5.3.2 Matching Engine 92
5.3.3 Architecture of Matching Mechanism 98
6 Prototype: OMI Service 100
7 Evaluation 106
8 Conclusion and Discussion 109
8.1 Outlook and Findings 109
8.2 Input for Open Source Community 112
Bibliography 113
Zusammenfassung 120
Appendix 121
Appendix-1: Utilized Petri Nets Ontology 121

Appendix-2:Log Created for scenario; matching simulation mecnhanism & algorithms for Place/Transition Petri Nets.....122

Curriculum Vitae 127

Table of Figures

Figure 1-1Introduction of Approach 4
Figure 1-2 Structure of the Work and Dependencies between Core Chapters 6
Figure 2-1 Components of Modelling Methods Modelling Languages (Karagiannis, Grossmann and Höfferer 2008) p. 65 9
Figure 2-2 ADOxx® functionalities, variation of configuration and extension of functionalities of ADOxx® (ADOxx.org 2013) p. 4 12
Figure 2-3 Model Hierarchy (Karagiannis and Kühn 2002) 13
Figure 2-4 Model Hierarchy of ADOxx® (adapted from (Kühn 2004) 14
Figure 2-5 Metamodel of the Metamodelling Language (Kühn, Junginger, et al. 1999) p.79 15
Figure 2-6 ADOxx® Metamodel for Customizing Graph-based Modelling languages 16
Figure 2-7 Metamodel of the Mechanism Definition Language (Kühn 2004) p. 19
Figure 3-1 Classification of Petri Nets based on the proposal made by (Trompedeller 1995) 23
Figure 3-2 Metamodel of Elementary Petri Nets in the Model Hierarchy 27
Figure 3-3 Elementary Petri Nets Modelling Editor and a Sample Model "Traffic Lights" ... 29
Figure 3-4 Metamodel of Place/Transition Petri Nets in the Model Hierarchy 32
Figure 3-5 Place/Transition Nets Modelling Editor and a Sample Model "Automatic Biscuit Seller" 34
Figure 3-6 Metamodel of Coloured Petri Nets and Model Hierarchy 38
Figure 3-7 Colored Petri Nets Modelling Editor and a Sample Model "Simple Transport Protocol for a Unreliable Network" 39
Figure 4-1 Matching Dimension: Function Groups of Mechanism and Algorithms 50
Figure 4-2 Matching Dimension: Mechanisms & Algorithms Type 50
Figure 4-3 Matching Dimension: Model Hierarchy 50
Figure 4-4 Cohesion between Matching Dimensions 51
Figure 4-5 WSDL 2.0 Infoset (Booth und Liu 2005) 54
Figure 4-6 Entities, Literals, and Anonymous Individuals in OWL 2 (Motik, et al. 2012) 55
Figure 4-7 The Axioms of OWL 2 (Motik, et al. 2012) 56
Figure 4-8 Class Axioms in OWL 2 (Motik, et al. 2012) 56
Figure 4-9 Object Property Axioms in OWL 2 (Motik, et al. 2012) 57
Figure 4-10 Data Property Axioms in OWL 2 (Motik, et al. 2012) 58
Figure 4-11 Object type Taxonomy of Petri Nets extended with Synonyms of Place and Transition 76
Figure 4-12 Petri Nets (Core) Ontology (Gašević and Devedžić 2006) 77
Figure 4-13 Petri Nets Ontology adapted with ADOxx® Concepts 78
Figure 5-1 Static Structure of Semantically Enriched Mechanism Description Language 81
Figure 5-2 Metamodel of Mechanism Description Model Type 82
Figure 5-3 SeMD Modelling Toolkit 87
Figure 5-4 Use Case Diagram of Modelling Language and Mechanism Matching Mechanism 90
Figure 5-5 Matching Engine Class Diagram 93

Figure 5-6 The High-level Architecture of Modelling Language and Mechanism Matching Mechanism 99
Figure 6-1 Adapted Petri Nets Ontology with excepted Concepts from ADOxx® Specific Petri Net Ontology and Extended Petri Net Ontology 100
Figure 6-2 Petri Nets Ontology Adapted with ADOxx® Specific Concepts 100
Figure 6-3 Specification of a Mechanism 101
Figure 6-4 Annotation of Input/Output Classes and Properties of the Mechanism using Concepts from Petri Nets Ontology 102
Figure 6-5 Overview of Semantic Description of a Mechanism 103
Figure 6-6 Defining the Dependencies among the Mechanisms 103
Figure 6-7 Exporting SeMD Model as XML File 104
Figure 6-8 Defining the Goal using Web Client of the Matching Engine 104
Figure 6-9 Log file exposed by the Matching Engine 105
Figure 7-1: Matching PTN Modelling Language with Mechanism & Algorithms with using Matching Engine 106
Figure 7-2: Part of Results after Processing Request for Matching Simulation Mechanism & Algorithms with PTN Modelling Language 107
Figure 7-3: Successful Execution of Fast Simulation Mechanism on a Model modeled by PTN Modelling Language 108
Figure 8-1 Classification of the Matching Mechanism as a Knowledge Management Tool. 111
Figure Appendix-0-1 ADOxx specific Petri Nets Taxonomy 121

Table of Tables

Table 2-1 Definitions of pre-defined classes in ADOxx® Metamodel for Customizing Graph-based Modelling Languages 17
Table 2-2 Data Types allowed by ADOxx® to define Attributes 18
Table 3-1 ADOxx® specific Specification of the Constructs of the Metamodel of Elementary Petri Nets 29
Table 3-2 ADOxx® specific Specification of the Constructs of the Metamodel of Place/Transition Nets 34
Table 3-3 ADOxx® specific Specification of the Constructs of the Metamodel of the Coloured Petri Nets 39
Table 3-4 Specification of Mechanism "Fast Simulation" 41
Table 3-5 Specification of Mechanism "StepByStep Simulation" 42
Table 3-6 Specification of Mechanism "Net Statistics" 43
Table 3-7 Specification of Mechanism "PNML Transformation" 45
Table 3-8 Specification of "Genetic Algorithm for Petri Nets" 45
Table 3-9 Specification of Mechanism "ADOxx® Path Analysis" 47
Table 4-1 Synonyms of Place and Transition 75
Table 5-1 Specification of Mechanism Description Model Type 84
Table 5-2 Specification of OWL Model Type 87

Table of Code Snippet

Code Snippet 5-1 Pre-selection of Candidate Mechanism & Algorithms according to their Function Group and Type 94
Code Snippet 5-2 Get all annotations distinctly used to annotate required input classes required for each mechanism & algorithm 95
Code Snippet 5-3 Collection of required input class nodes from modelling language by annotation 95
Code Snippet 5-4 Selection of the Class in Modelling Language by Annotation 96
Code Snippet 5-5 Get Labels of Semantic Concept 96
Code Snippet 5-6 Get Sub-concepts of classes 97
Code Snippet 5-7 Search Required Property in Class in Modelling Language 97
Code Snippet 5-8 Calculation of Semantic Similarity 98

Table of Abbreviation

Abbreviation	Description
BPMN	Business Process Model and Notation
CPN	Colored Petri Nets
DSM	Domain-Specific Modelling
EPC	Event-driven Process Chain
EPN	Elementary Petri Nets
FDMM	A Formalism for Describing ADOxx Meta Models and Models
IPOE	Inputs, Pre-conditions, Outputs, Effects
IRI	Internationalized Resource Identifier
OMG	Open Model Group
OWL	Web Ontology Language
OWL 2	Web Ontology Language 2
OWL-S	Semantic Markup for Web Services
PTN	Place/Transition Nets
SAWSDL	Semantic Annotations for WSDL and XML Schema
SeMD	Semantic Enriched Mechanism and Algorithm Description
SUS	System Under Study
UML	Unified Modelling Language
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WSMO	Web Service Modeling Ontology
XML	Extensible Markup Language

XPATH	XML Path Language

Abstract

The modelling methods comply with the specific requirements and consist of concepts for specific application domain and functionality to create information value. The method engineering guides the individuals, who endeavour to conceptualize their own modelling methods. Since a modelling method on the one side shall provide concepts for the conceptual representation of reality in the context of a given application domain on the other side shall provide functionality that enables information value creation from conceptual models, development of such a modelling method is a highly complex process.

This master thesis proposes a conceptual framework on the generic level and a matching mechanism on the concrete level, which enable semantic matching between concepts defined in a modelling language and concepts in mechanism & algorithms, and the introduction of appropriate mechanisms & algorithms for the modelling language. The framework and the matching mechanism endeavour to guide method engineers to overcome the challenge caused by (1) complexity of the development process of modelling methods and (2) enrichment of simple model editors in order to develop full-fledged modelling tools, which consist of functionalities in the form of mechanism & algorithms.

Hence, this master thesis is aligned to the following ideas: First, investigate of method engineering approach as it is defined in the "Generic Modelling Method Specification Framework", realization approach with following ADOxx® Framework, and studies that made within these scope. Second, selection of Petri Nets modelling languages, and mechanism & algorithms. Next specification and formalization of those modelling languages and mechanism & algorithms with using conclusions obtained from the investigation of method engineering, so we can create application scenario which serves to prove concepts of the framework. Third, to analyze state-of-the-art in semantic service discovery that marks contemporary solutions and technologies in semantic description of service, which shall guide this work for developing approach for semantic description of mechanisms & algorithms. Forth, definition of formal description approach for mechanisms & algorithms, identification of matching dimensions in the light of application scenarios, and the formal description approach. Application of the formal description approach to mechanisms & algorithms which are afore selected. Moreover, investigation of a Petri Nets ontology and adaptation of this ontology with ADOxx® specific concepts. This ontology will be utilized for semantic enrichment of description of mechanisms & algorithms. Fifth, based on findings of previous investigations and mechanisms & algorithms description approach, the Matching Mechanism will be conceptually and technically specified. Two different components will be offered to users, which will constitute the matching mechanism, namely; (1) Semantic enriched Mechanism Description (SeMD) Modelling Editor and (2) the Matching Engine. Usage of those components will be introduced. A prototype of the Matching Mechanism with limited functionalities will be made available to the community for test purpose.

Keywords: Method Engineering, Metamodelling, Metamodelling Platforms, ADOxx, Semantic Service Description, Semantic Discovery, Semantic Mechanism and Algorithm Description.

1 Introduction

Since the standard modelling methods are not necessarily complying with the specific requirement of individual, and usually they are short of concepts for specific application domain, growing number of individuals around the world endeavours to conceptualize their own modelling methods, under the guidance of frameworks like the Generic Modelling Method Specification Framework proposed by (Karagiannis and Kühn 2002) (Kühn 2004). The models are not only the conceptual representation of abstracted reality but utilized to create information value, which requires certain functionality that could be applied to modelling language. Therefore a modelling method on the one side should provide required concepts for the conceptual representation of reality in the certain abstraction level on the other side should have functionality enabling information value creation from models. Hence the process of development of such a modelling method is highly complex. Even nowadays different communities provide varying approaches, guidelines and expose best – practices for developing modelling editors, the main challenge and question is how to enrich those model editors in order to developed full-fledged modelling tools that consists of functionalities in form of mechanism & algorithms that creates information value by processing conceptual models.

Modelling method consists - according to (Karagiannis and Kühn 2002) - modelling technique that itself consists of modelling language and modelling procedure as well as mechanisms & algorithms. To establish a modelling method a method engineer is responsible for providing a properly and consistent defined modelling method, while language engineer defines modelling language with adequate definition of syntax, semantic and notation.

The realization of modelling methods currently relies heavily on the expertise of the method engineer with less formalized support. Current research like FDMM (Fill, Redmond and Karagiannis 2012) investigates formalized representations for specifying modelling methods. Current specification mainly focus on modelling languages, few publications are currently available also considering specification of modelling procedures and mechanisms & algorithms.

This may be based on the fact that current modelling tools are simple editors that are limited in functionality directly required to manipulate modelling languages and do not consider full fledged functionality of technology supporting modelling methods.

From the viewpoint stated in (Karagiannis and Kühn 2002) this fact is surprising, as the added value of model-based approaches is the possibility to apply IT-supported processing. Here mentioned algorithms in case of predictable output – meaning without human user interaction – or mechanisms without predictable output – meaning including human user interaction.

The challenge is, therefore, to introduce functionality in the form of mechanisms and algorithms in such a way that the upgrade of the model editor to a full fletched modelling tool

can be efficiently performed. The better such an upgrade can be performed, the easier it is to provide functionality for information value creation, and hence the applicability of conceptual models is expected to rise.

This master thesis deals with the challenge to simplify the enrichment of model editors with mechanisms and algorithms to upgrade them to full-fletched modelling tools. The conceptual framework is worked out on a generic level and a concrete showcase to proof the concept is provided with Petri Nets as concrete application scenario.

The selected approach is to adapt and apply a semantic matching that is well known from the semantic service discovery in the domain of conceptual modelling mechanisms. Hence the question of this thesis is, how mechanism & can be described, to enable a (semi-) automatic match between the modelling languages and required or available mechanisms.

Typical questions are:

- Can modelling language X use the mechanism Y?
- What semantic primitives are required in modelling language X in order to execute the mechanism Y?
- Which mechanism can be performed by modelling language X?

Although these questions are generic, this master thesis focuses on the concrete implementation of Petri Nets (Petri 1966). Hence available modelling languages of Petri Nets are collected in form of a literature research and three different Petri Nets Modelling Languages from three different level of Petri Nets are selected namely; (1) Elementary Petri Nets from level one with the lowest level of notions, (2) Place/Transition Nets from level two with higher level of notions than level one and (3) Colored-Petri Nets from level three with highest level of notions. Available mechanisms are searched, listed and categorized. During the selection of mechanisms & algorithms, the mechanisms & algorithms which have been already implemented following ADOxx® Framework are taken firstly under consideration.

Both lists – modelling languages for Petri Nets, as well as mechanisms, – use a catalogue framework that enables the semi-automatic matching between these two lists. There are different realization possibilities to implement such catalogue framework. It will be one task of the master thesis to search, analyze and select an appropriate framework.

Independent of its technical realization such catalogue need a detailed description of both the required semantic primitives as well as available processing.

Hence a bottom-up approach will be realized by searching and listing available Petri Net algorithms and analyze their characteristics. Knowledge extraction will be used to identify such characteristics e.g. asking Petri Net experts why certain algorithms can be applied for one modelling language and not for another one. A vector is defined, based on the mentioned criteria considering different dimensions and different elements per dimension.

Such requirement and solution vectors are manually defined for the selected list of modelling languages and algorithms, and a matching is explained using the aforementioned dimensions and criteria.

Realization framework is a technological implementation of such vectors. A range of implementations is available, whereas semantic service discovery mechanisms using query templates and a similarity matching currently seems most appropriate.

Query template is an XML template representing each of the agreed dimensions, whereas the solution vector is again filled in XML template. Semantic comparing of these two XML document results in a similarity degree.

A prototype realizes such a matching that can be used while developing a modelling method in ADOxx®.

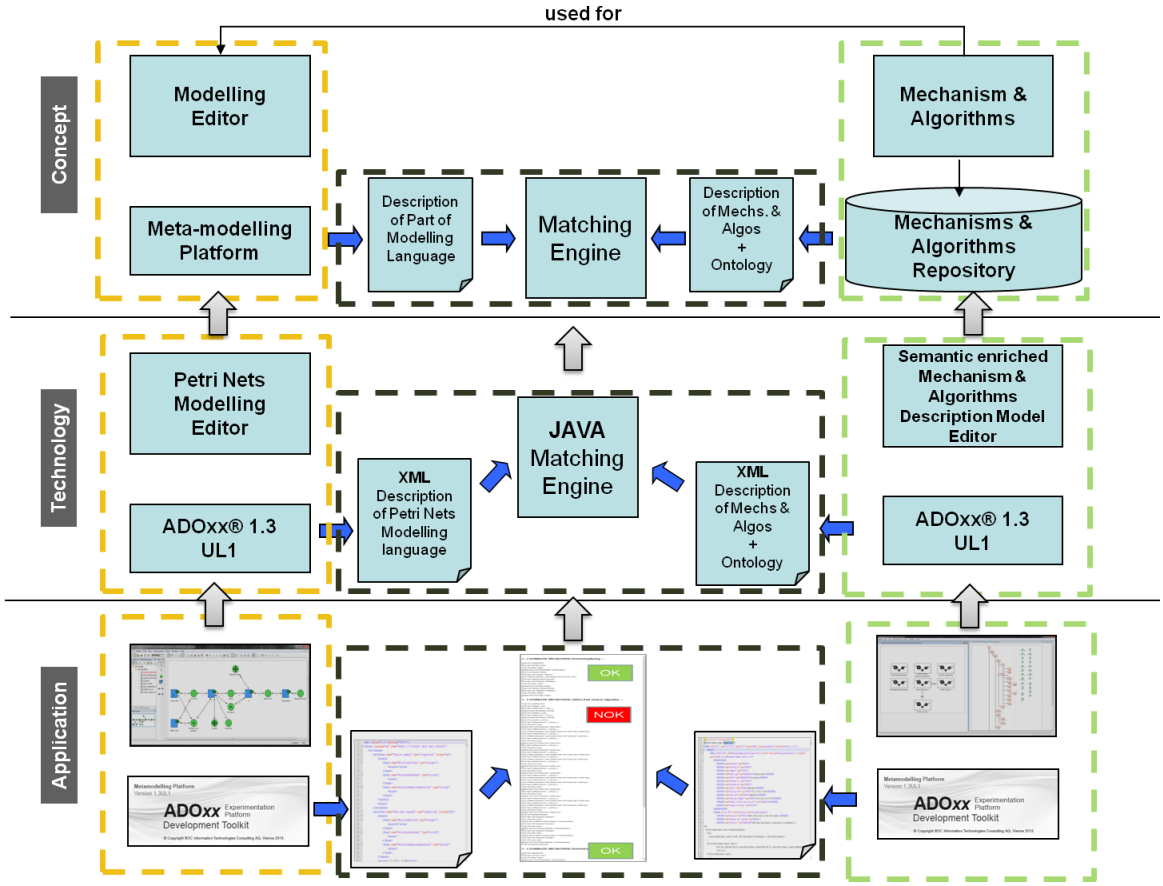


Figure 1-1Introduction of Approach

This thesis proposed following approach as depicted in Figure 1-1. At the conceptual level, the work defines a framework, which matches mechanisms & algorithms that can be used for a modelling language. The matching concept is implemented on the metamodel level to be inherited to all models. The framework proposes a matching mechanism, by comparing concepts in the description of relevant parts of the modelling language which are retrieved from meta-modelling platform and in the description in the mechanisms & algorithms, which are retrieved from the mechanisms & algorithms repository.

The technology level elaborates the realization of the matching mechanism. The metamodel has to be implemented in the form of a so-called ABL file in ADOxx® version 1.3 UL1, which is exported in XML format using standard functionality of ADOxx®. The mechanisms & algorithms are described by a mechanisms & algorithms description editor, which is implemented as a modelling tool on ADOxx® version 1.3 UL1. This modelling tool for mechanisms and algorithms enables semantic enrichment of the description of mechanisms & algorithms. This description of mechanisms and algorithms is also exported in XML format using standard functionality of the ADOxx platform. The matching engine implemented in java executes the matching via querying the two XML files, one containing the description of the relevant parts of the modelling language and the second containing the description of mechanisms & algorithms including the semantic enrichment.

At the application level, this work proofs the concepts by the so-called "Semantic enriched Mechanism & Algorithm Description Editor" implemented on ADOxx 1.3UL1, which enables semantic description of mechanisms & algorithms.

To evaluate the modelling languages, the Petri Net modelling editors and mechanisms & algorithms, which are already implemented on ADOxx®, have been used. Therefore, besides the implemented Petri Net modelling editor by the community "Open Models Initiative" (Efendioglu, Lekaditis, et al. 2011), additional Petri Net modelling editors have been implemented on top of ADOxx to proof the concept. Following mechanisms & algorithms are described: the "Fast Simulation", "Step by Step Simulation", "PNML Transformation", "Genetic Algorithm" and "ADOxx Path Analysis". The application layer is hence concerned with the matching of aforementioned PetriNet modellers and any models like the biscuit-teller machine modelled with this tools and the correct matching of aforementioned algorithms and mechanisms.

Considering the challenges and the procedure -introduced above- to get through to solution, we can outline structure of this work and dependencies among the core chapters as depicted in the Figure 1-2.

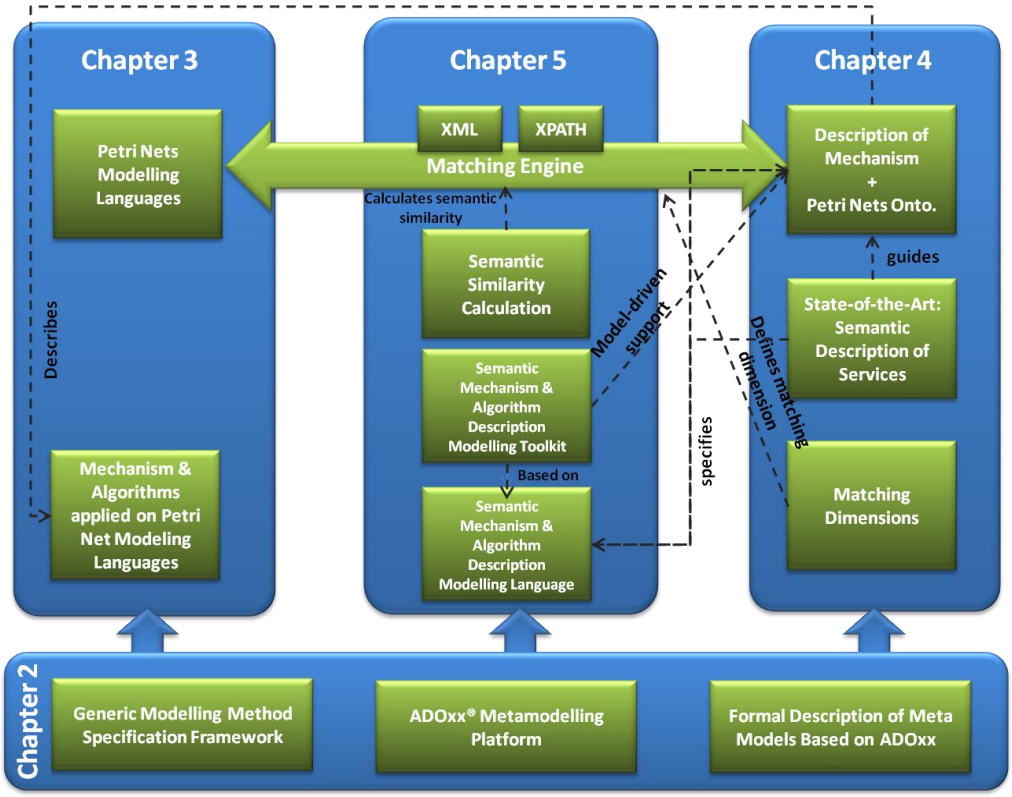


Figure 1-2 Structure of the Work and Dependencies between Core Chapters

The chapter 2 investigates the (1) Generic Modelling Method Specification Framework proposed by (Karagiannis and Kühn 2002) (Kühn 2004), which guides the method engineers to conceptualize and to build a modelling method, then the (2) ADO*xx*®[i] Metamodelling Platform, which enables realization of conceptualized modelling method, hence the chapter also investigate how to specify conceptualized modelling method specific to the ADOxx,at last but not least (3) Formal Description of Metamodels based on ADOxx®, which enables formal description of concepts used in metamodels based on ADOxx. The chapter 2 guides the work in other core chapters and provides fundamental knowledge like for the chapter 3 how to conceptualize and describe formally the Petri Nets Modelling Languages, and how to specify mechanism & algorithms ADOxx specific, for the chapter 4 input to find out the matching dimensions and to describe mechanisms & algorithms formally, for the chapter 5 knowledge to conceptualize and to implement the required modelling editor within the matching mechanism.

The chapter 3 presents the results of the literature search about Petri Nets Modelling Languages and related mechanism & algorithms. Moreover that chapter represents the conceptualization and formal description of selected three Petri Nets Modelling Languages; (1) Elementary Petri Nets from level I Petri Nets Modelling Languages, (2) Place/Transition Nets from level II Petri Nets Modelling Languages and (3) Colored Petri Nets from level III Petri Nets Modelling Languages. Besides that, ADOxx® specific specifications of identified mechanisms & algorithms according to ADOxx® Framework are represented as well in that

[i] ADO*xx*® *is a Trade Mark of BOC AG and* ADO*xx*® Metamodelling Platform is a Product of BOC AG.

chapter. Those specifications are utilized as inputs in the formal description of identified mechanisms with using the approach proposed in chapter 4.

Since we have not come across any approach or work that studies matching modelling languages and mechanisms algorithms and can provide significant guidance for our work, we investigate in the chapter 4 state-of-the-art in semantic service discovery studies –particularly semantic description of services- and try to detect significant input that guides us to achieve our objective matching modelling languages and mechanism & algorithms and overcome the challenges caused semantic gaps between modelling language descriptions and mechanism descriptions. Besides that, in order to have the big picture, application scenarios are identified Then matching dimensions between modelling languages and mechanism & algorithms, matching points, which shall be considered on each dimension and then cohesion among them are identified. Moreover with guidance of the outcomes of state-of-the-art analysis and investigation for matching dimension, we identify formal description approach for mechanisms & algorithms. Afterwards, we classify concepts in the notions of Petri Nets and concepts in the notions of ADOxx®, in a taxonomy. Then we investigate a Petri Nets Ontology proposed by (Gašević and Devedžić 2006) and adapt this ontology with integrating the aforementioned taxonomy in order to have an Petri Net Ontology For that all, the outcomes of that chapter influenced the design and implementation of the SeMD Modelling Editor and the Matching, which compose together the matching mechanism, which are specified and introduced in the chapter 5, as well.

In the chapter 5, we introduce realization of the matching mechanism. First of all we specify and introduce the modelling language "Semantic enriched Mechanism Description Modelling Language", which offers model-driven mechanism & algorithm description enriched semantically. Based on the modelling language a modelling editor on ADOxx® implemented, called SeMD modelling editor The chapter also investigate an appropriate approach to calculate the semantic similarity between concepts from modelling languages and from the mechanisms & algorithms.

As a result, the matching engine provides a log exposes matching and not matching concepts and to what extent they are matching, moreover calculated semantic similarity between the concepts. For that all, the chapter introduces the architecture of the Matching Mechanism.

The chapter 6 introduces the prototype implemented for the concern to prove the approach proposed by this work. For that concern, the chapter introduces a procedure for the utilization of the matching engine. Furthermore, the chapter exposes the evaluation result of the prototype of the matching mechanism. For the evaluation of the prototype, Petri Nets modelling languages are implemented on ADOxx® based on the specification exposed in the chapter 3, and the mechanisms are identified in chapter 3 and described in chapter 4. Those modelling languages and mechanisms are used as input and the log produced by the prototype is assessed with the respect to the application scenarios defined in chapter 4.

The chapter 8 produces a summary of this work: of reached results, commitment of the work to the community and gives an overview for the future works.

2 Modelling Methods

Conceptual models are a well-known instrument to externalise knowledge and hence enable a processing of the models with the goal to create information value. Currently the conceptualisation process that is performed when developing a new modelling method is heavily investigated by the OMiLAB community.

The Master Thesis “Knowledge Management for Modelling Method Development” (Toader 2013) points out different ways to develop a modelling method that have been published in the tutorial slides on OMiLAB and ADOxx®.org. There are different ways to complete a modelling method, but the mapping between modelling languages and mechanisms has always to take place.

In the following, some rudimentary concepts are explained as a common taxonomy for the rest of the thesis.

2.1 Generic Modelling Method Specification Framework

The definition of a generic framework for specifying modelling methods has been an active scientific research field of the University of Vienna. Outcome of this research is the Generic Modelling Method Specification Framework (Karagiannis and Kühn 2002) (Kühn 2004). It is a generic framework, based on the analysis of modelling methods to specify modelling methods and is suggested by Open Models Initiative (Open Models Initiative n.d.).

The Figure 2-1 depicts the Generic Modelling Method Specification Framework proposed by (Karagiannis and Kühn 2002) (Kühn 2004) (Karagiannis, Grossmann and Höfferer 2008), according to which a modelling method consist of two main components: (1) modelling technique, which consists of (1a) modelling language and (1b) modelling procedure, and (2) mechanism & algorithm, also known as mechanism. A modelling language includes the elements which are used to describe a model. The modelling language has three main parts by which it is described; syntax, semantics and notation. The modelling procedure describes the steps which have to be followed to create results when using the modelling language. The mechanism & algorithm provide the functionalities that enable using, processing and evaluating the models, which are the instances of a modelling language.

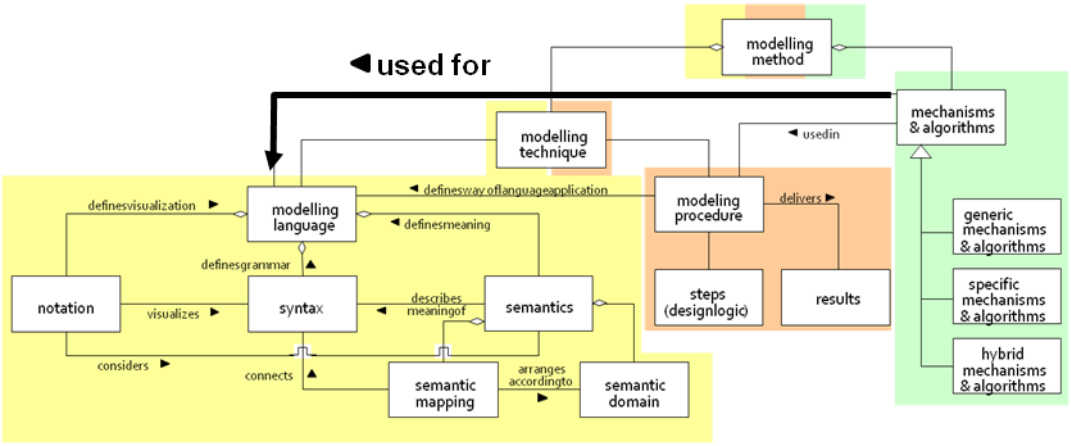


Figure 2-1 Components of Modelling Methods Modelling Languages (Karagiannis, Grossmann and Höfferer 2008) p. 65

2.1.1 Modelling Languages

As aforementioned, the - conceptual - modelling languages are described by their syntax semantics and in case of graphical modelling languages additionally be its notation. The syntax describes the elements and rules of a modelling language for creating models. The semantics, which consists of a semantic domain and semantic mapping, describes the meaning of a modelling language. The notation describes the visualization of a modelling language (Karagiannis and Kühn 2002).

Syntax means in Ancient Greek "arrangement". Chomsky defines syntax in terms of linguistic as " *the study of the principles and processes by which sentences are constructed in a particular language*" (Chomsky 1985). The syntax is defined in terms of logic; "*branch of modern logic that studies the various kinds of signs that occur in a system and the possible arrangements of those signs, complete abstraction being made of the meaning of the signs*" (Random House 2013). In terms of computer science "*the grammatical rules and structural patterns governing the ordered use of appropriate words and symbols for issuing commands, writing code, etc, in a particular software application or programming language*" (Random House 2013).

To describe syntax of modelling languages, there are two major approaches *graph grammars or metamodels* (Karagiannis and Kühn 2002).

Since we follow ADOxx® Framework, we concentrate us in this work on metamodels to describe syntax of a modelling language. A *Metamodel* is a model of models. A metamodel consists of entities defining the model elements and hence the modelling language, and also a metamodel aims to relate these model elements (Geisler, Klar and Pons 1998).

Syntax can be distinguished between two type of syntax;

- *Abstract syntax* specifies concepts of language, their properties and relationships among them, which are typically carried out with using meta-modelling. The basic

notions of the language and relationships among them are defined precisely with using structural constraints, multiplicities and implicit relationships (such as inheritance, refinement) (Kleppe 2007).

- *Concrete syntax* specifies visualization of concepts of language with assigning a visual symbol to the language concepts which are to be represented in diagrams (Baar 2008).

Semantics according to (Karagiannis and Kühn 2002) are used to describe the meaning of a modelling language. The semantic definition of a modelling language consists of two main parts; (1) semantic domain, which describes the meaning of modelling language constructs by using ontologies, mathematical expression etc., and (2) semantic mapping, which maps the syntactical constructs of a modelling language to their meaning in the semantic domain of the modelling language. There is a range of available approaches to define semantics of a modelling language, like following (Geisler, Klar and Pons 1998) (Karagiannis and Kühn 2002) (Kühn 2004)

- *Text Based Semantic Definition:* with the text based semantic, the meaning of a modelling language and its constructs can be defined in form of natural language definition
- *Denotational Semantic Definition:* the denotation semantics is defined by mapping syntactic constructs to mathematical objects from a semantic domain.
- *Operational Semantic Definition:* With operation semantics we understand that semantic is implicitly provided when using a certain operational schema. One can argue that the operational semantic is implicitly pre-defined, like a directed graph in mathematics where semantic of notes and relations are pre-defined in the syntax. The Operational semantic, therefore, comes from abstract execution machine that is executed or interpreted via syntax defining model.
- *Axiomatic Semantic Definition:* the axiomatic semantics based on mathematical logic is described via logical axioms and inference rules.
- *Algebraic Semantic Definition:* algebraic semantics is a form of axiomatic semantics based on algebraic. Algebraic semantics describes modelling language constructs via data types. The data types and their operators are described via algebraic

Notation describes the visual representation of concepts of a modelling language considering syntax and semantics of the modelling language (Kühn 2004). Again according to (Kühn 2004) notation can be distinguished between;

Static Notations which provides always same static visual representation of concepts independent with the state of concepts

Dynamic Notation, in contradiction to static notations, provides a visual representation of concepts dependent with the state of the concepts with the help of defined rules.

2.1.2 Modelling Procedures

A modelling procedure is the other main part of a modelling technique (as depicted in Figure 2-1). The modelling procedure defines the required steps to valid models and the corresponding results (Karagiannis, Grossmann and Höfferer 2008). According to (Kühn 2004) a modelling procedure enables the creation of a model while ensuring the completeness of necessary aspects as well as the syntactical correctness.

2.1.3 Mechanisms and Algorithms

"The processing of models is done with the help of mechanism & algorithms that provide functionality to use and evaluate" (Karagiannis and Kühn 2002).*Tthe mechanism & algorithms are classified in three type, namely; (1) generic, (2) specific, and (3) hybrid mechanism & algorithms.*.

Generic mechanism & algorithms: are mechanism & algorithms that can be applied to any modelling technique independent from semantics of modelling technique as long as they fulfil the certain defined syntactical rules, those mechanisms are realized on the meta²model[ii] level of a modelling language (Kühn 2004) (Karagiannis, Grossmann and Höfferer 2008) (Kühn 2004). Such examples for this mechanism & algorithm would be; import and export mechanism of ADOxx® and Dijsktra Algorithm, which solves the shortest path between nodes problem. Specific mechanism & algorithms: are mechanism & algorithms, which are dependent to modelling technique (Kühn 2004). In order to process a model of a modelling language correctly mechanism & algorithm needs to understand the semantics of the domain. Such a mechanism would be the workflow engine which follows business rules –defines semantics of the domain- to execute business processes correctly (Karagiannis, Grossmann and Höfferer 2008).

Hybrid mechanism & algorithms: are mechanism & algorithms, which are ,in principle, independent from semantics of modelling technique and they are realized on meta²model level of modelling language, but they need to be adapted so the mechanism & algorithm can work also with concepts of modelling language. Such an example would be query mechanism of ADOxx®, which works in the principle with the concept on the meta²model, but in order retrieve certain information from model instance it requires some adaptations, so it can identify the concepts of the modelling language.

[ii] Meta²model describes the concepts for building metamodels (Geisler, Klar and Pons 1998)

Figure 2-2 ADOxx® functionalities, variation of configuration and extension of functionalities of ADOxx® (ADOxx.org 2013) p. 4

The Figure 2-1 depicts core functionalities of ADOxx®, varying possibilities of configuration and extension of those functionalities, in the other words mechanism and algorithms. The core functionalities of ADOxx® are; (1) Database functionalities, (2) Visualisation functionalities, (3) querying functionalities, (4) transformation and (5) simulation. The ADOxx.org community describes this figure as onion diagram the each more exterior circle consists of more flexible configuration and extension possibilities, but the less supported possibilities by ADOxx® community (ADOxx.org 2013). Therefore this work aims mostly to support the matching of mechanisms, which are implemented with the possibilities of the two outer circles; (1) External coupling and Add-on implementation, where the user required more support

2.2 The ADOxx® Metamodelling Platform

Development of a modelling method and a full fletched modelling tool based on it requires a development environment, which provides required means and capabilities (Karagiannis, Fill and Zivkovic, et al. 2012).

There are modelling languages accepted such as UML, EPC , BPMN , which are accepted by industry as standard, are based on fixed metamodels. Since the languages are based on fixed metamodels, they are lack possibilities to express explicitly variation of domain specific concepts with pointing to certain modelling constructs in order to full fill the requirements of the domain (Pohjonen and Tolvanen 2005).

ADOxx® – metamodelling platform is a metamodelling-based development and configuration environment to create domain-specific modelling products, developed by BOC group. The ADOxx® metamodelling platform is based on the ADOxx® metamodelling

approach, which has been arisen during the development of the ADONIS®[iii] business process management toolkit in 1995 (Junginger, et al. 2000) (Fill, Redmond and Karagiannis, FDMM: A Formalism for Describing ADOxx® Meta Models and Models 2012).

The following sections in this chapter, especially sections 2.2.2, and 2.2.3 based on Dissertation of Kühn (Kühn 2004) and ADOxx® tutorials.

2.2.1 Model Hierarchy

Creation of a model is done by using a modelling language that is described by a metamodel. As aforementioned, the metamodel is defined as "*model of models"* (Object Management Group 2003). The creation of a metamodel also requires using a modelling language, which is called the *metamodelling language* (Karagiannis and Kühn 2002). The metamodelling language is defined by model so-called meta-metamodel or meta²-model (Strahringer 1996) (Geisler, Klar and Pons 1998) (Karagiannis and Kühn 2002).

For the definition of metamodels, a methodology is widely used, which is based on a four-level approach as depicted in Figure 2-3 (Geisler, Klar and Pons 1998) (Karagiannis and Kühn 2002) (Kühn 2004). The lowest level is the original SUS (Subject-Under-Study) from which a model is abstracted on the second level, third level metamodel level describes the concepts for building models while fourth level meta²-model level describes the concepts for building metamodels (Geisler, Klar and Pons 1998) (Karagiannis and Kühn 2002).

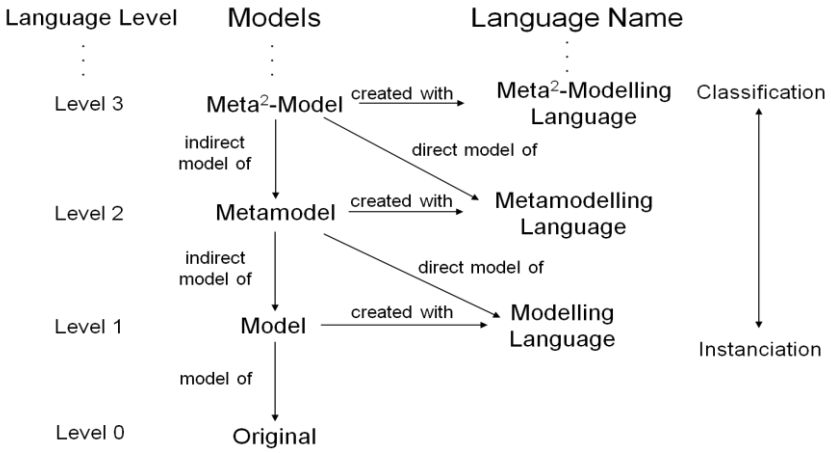


Figure 2-3 Model Hierarchy (Karagiannis and Kühn 2002)

2.2.2 Specification of Modelling Languages with ADOxx®

Since we follow the ADOxx® approach in this work to describe the building of a modelling language, we investigate in this section how to specify a modelling language with ADOxx®.

[iii] ADONIS® *is a Trade Mark of BOC AG* ADONIS® Business Process Management Tool Kit *i*s a Product of BOC AG

In Figure 2-4, the model hierarchy of ADOxx® approach is depicted. The ADOxx® Meta²model is implemented in the C++ programming language. The ADOxx® meta model is instantiated from the ADOxx® Meta²model and provides a set of predefined constructs that are detailed in section 2.2.2.1, and domain specific metamodels created by users are derived from predefined classes of ADOxx® Metamodel, and they are described in ADOxx® Library Language (ALL). The ALL provides the concepts to ´describe metamodels by using the constructs defined in the ADOxx® Meta²model (Fill and Karagiannis 2013). The models are created as the instances of domain specific metamodels created by users and they can be described in ADOxx® export format ADL.

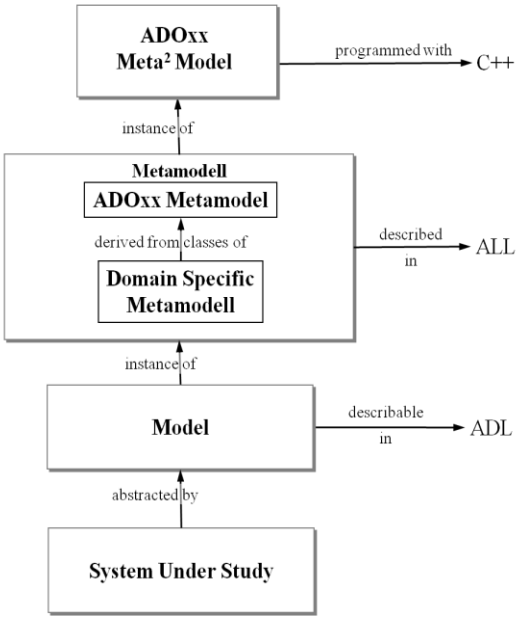


Figure 2-4 Model Hierarchy of ADOxx® (adapted from (Kühn 2004)

Creation of a metamodel -to describe syntactical constructs of the modelling language- requires a metamodelling language. The Figure 2-5 depicts the metamodel of the metamodelling language, with which metamodel of (domain specific-) modelling languages can be created (as described in section 2.2.1). The metamodel of a metamodelling language is also known as "Meta²-model" of a modelling language. According to (Kühn 2004)a metamodel can consist of one or more *metamodel part*. A metamodel part composes at least one *class* and optionally one *relation type*. The properties of classes and relation types can be described by using *attributes*. The relation types can bind classes and they have exactly one *fromclass* which represents source class and exactly one *toclass* which represents target class. The metamodel parts specialize to *modeltype*, to *views* and to *design patterns*. As aforementioned properties of classes and relation types are described by using attributes which are distinguished between *class attribute* and *instance attribute*. While class attributes receive one value for every instance of the class, instance attributes receive one value of each instance or relation. Thus class attributes are context-independent against that instance attributes are context-dependent. *Attribute profile* is special form of context-dependent attribute. An attribute profile is bundle of one or more instance attributes. Moreover attribute types are differentiated between *atomic*- and *composed- type*. While atomic type are data

types like “integer”, “string”, “date” etc., composed types are used to define new data types by using existing atomic data types. *Facets* are used to describe property of attributes. Special form of facets; *regular expressions* make possible to define new data type in context free language.

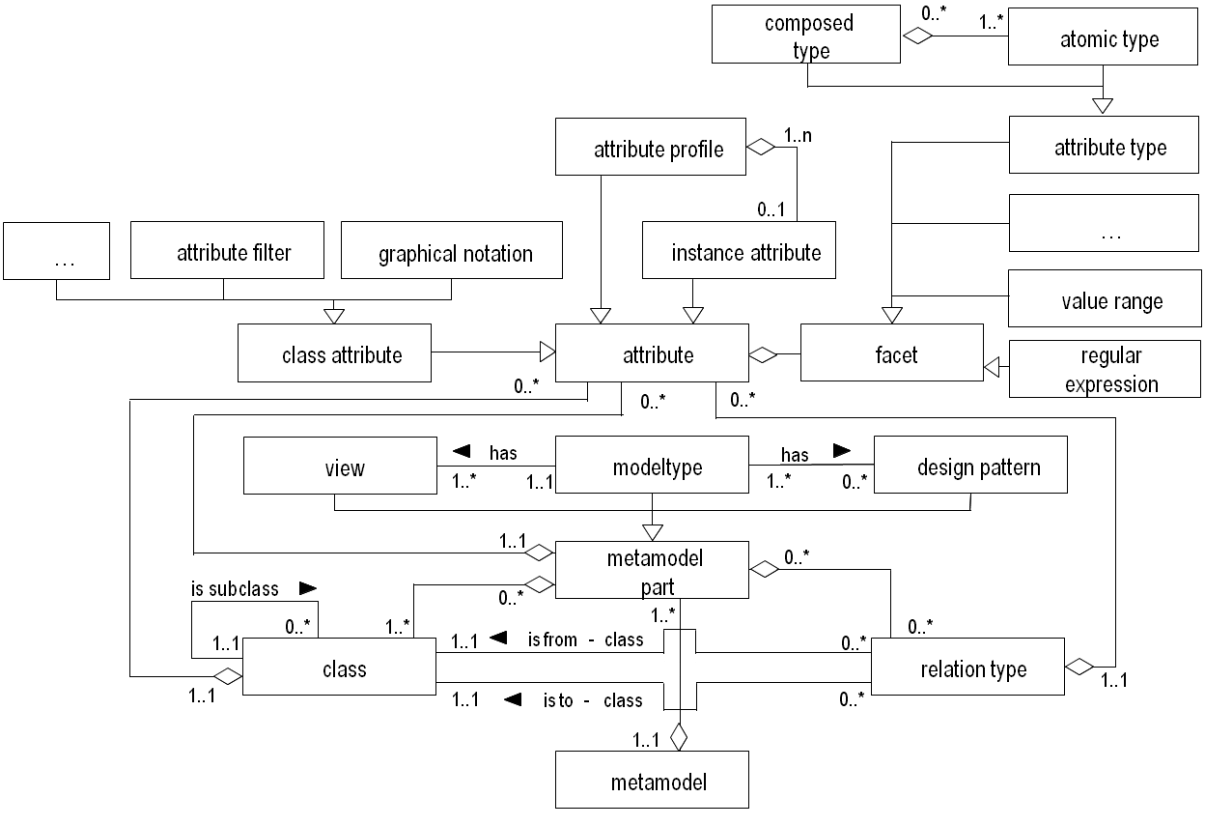


Figure 2-5 Metamodel of the Metamodelling Language (Kühn, Junginger, et al. 1999) p.79

2.2.2.1 ADOxx® Metamodel Classes

ADOxx® offers predefined abstract classes in its metamodel with given semantics and basic syntax in the form of attributes, which can be used to inherit the predefined syntax and attributes for defining abstract classes or classes of metamodel of domain specific modelling language. Hence those pre-defined abstract classes exist in every meta model based on ADOxx® (ADOxx.org 2013).

ADOxx® provides functionalities which are working dependently to aforementioned predefined abstract classes, thus those provided functionalities can be consumed due to inheritance.

There are two different metamodels with pre-defined abstract classes provided by ADOxx®; (1) the meta model pre-defined ADOxx® classes to customize graph-based modeling languages and (2) and to customize tree-based modeling languages. In this work, we deal with graph-based various petri nets modeling language; therefore, we introduce pre-defined abstract classes for customizing graph-based modeling languages as depicted in Figure 2-6.

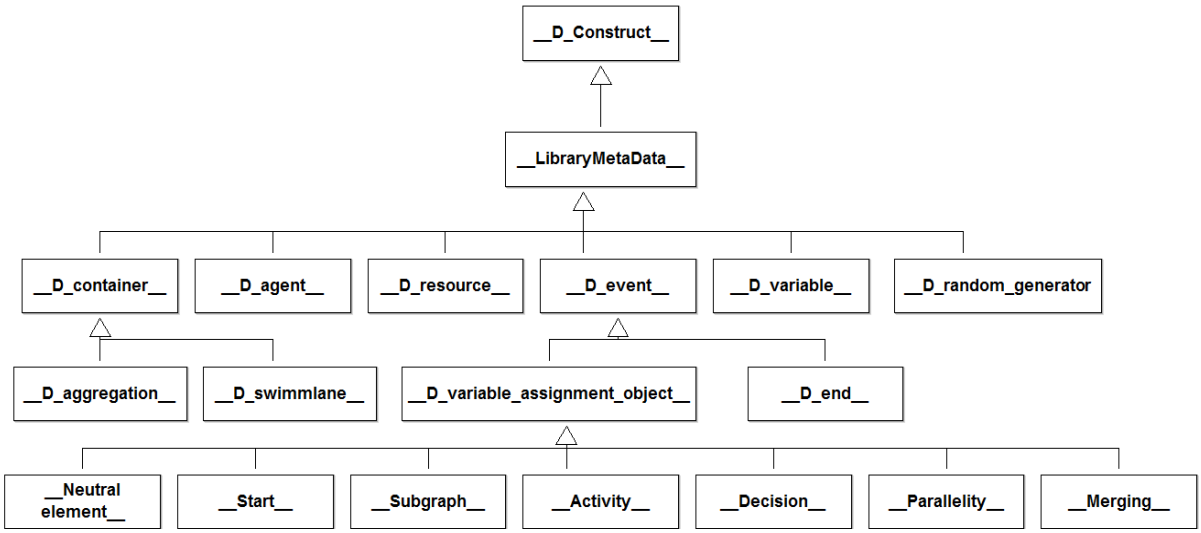


Figure 2-6 ADOxx® Metamodel for Customizing Graph-based Modelling languages

Considering that the matching mechanism is going to match modelling languages implemented on ADOxx® with inheriting syntax and semantic from pre-defined metamodel, it is useful to investigate constructs in that metamodel closely. The class definitions are given by (ADOxx.org 2013) as listed in Table 2-1 ;

Class Name	Definition
__D_Construct__	Super class for graph-based pre-defined meta model
__D_Container__	Container class provide the relation “is.inside”, which allows any container object of a container class, inherited from __D_Container to contain every object drawn within coordinates of drawing the area of the container by using “is-inside” relation.
__D_aggregation__	This class is derived from __D_Container hence inherits properties of __D_Container that provides “is-inside” relation and additional resizable self-defined drawing area.
__D_swimmlane_	This class is derived from __D_Container hence inherits properties of __D_Container provides “is-inside” relation, but it allows either rows or columns as possible resizable drawing area.
__D_Event__	This class encapsulates all possible nodes of a graph and distinguishes between “D_variable_assignment_objec” and “D_end”.
__D_variable_assignment_objects__	This class enables the change of the state. Each of following sub-classes of this class have the potential to change the status if variable within a graph: __Neutral element__, __Start__, __Subgraph__, __Activity__,

Class Name	Definition
	__Decision__, __Parallelity__, __Merging__.
__Neutral element__	Neutral elements do not participate in executing the graph but only display references or state the status.
__Start__	Start is the starting node of the graph
__Subgraph__	Due to readability issues of complex graphs sub-graphs are used. Technically the subgraph is a pointer to another graph.
__Activity__	Activity is a node within the graph that represents typical actions the graph is designed for. Activities transform input into output.
__Decisions__	Decisions split the graph into several alternative paths.
__Parallelity__	Parallelity starts a synchronized paths of a graph.
__Merging__	Merging ends a synchronized paths of a graph
__D_end__	The end concludes the graph and finishes state changes
__D_variable__	Objects of this class store a certain status of the graph. Different objects of this class can be defined to describe different aspects of a graph.
__D_random_generator__	Random generator is thought for simulation, it creates random figures which can be assigned to object of variables class.
__D_resources__	Resources are properties of graph nodes represented in an own class hierarchy. Therefore descriptive properties need not only be defined as attributes of graph nodes but can be described as classes using the class hierarchy from resources.

Table 2-1 Definitions of pre-defined classes in ADOxx® Metamodel for Customizing Graph-based Modelling Languages

There are class attributes and instance attributes defined in root class of metamodel, in our case the root class is "__D_Construct::", hence inherited by each class, the most important class attributes and instance attributes are;

- *AttrRep:* a class attribute, which contains notebook-definition of classes.
- *GrapRep:* a class attribute, which contains the graphical representation of classes.
- *Modelpointer:* a class attribute, which contains relations to other models.
- *Class cardinality:* a class attribute, which contains relation constraints.
- *ClassName:* a class attribute, which contains name of the class
- Name: instance attribute, which contains name of object of the class

- *ClassAbstract*: a class attribute, which specifies if the class is abstract hence specifies that class is not instanciable
- *ClassVisible:* a class attribute, which specifies if the class is visible on modelling area, or not.

The another important thing to know is the following attributes are always provided on the model level, which are assigned in persistence layer per definition of application library; *ClassId, ObjectId, ModelId*:

ADOxx® allows defining attributes for class and relation classes with following data types listed in;

Data Type	Definition
INTEGER	Integer within -/+1.999.999.999
DOUBLE	Floating number within -/+999,999,999,999,999 without decimals or -/+999,999,999.999999 with at last 6 decimal digits
STRING	String with up to 3699 symbols
LONGSTRING	String with up to 32000 symbols
TIME	time
DATE	Date with format YYYY:MM:DD
DATETIME	Date and time with format YYYY:MM:DD HH:MM:SS
ENUMERATION	Enumeration for selecting a value from value set
ENUMERATIONLIST	Enumeration for selection one or more values from value set
PROGRAMMCALL	Enumeration for selection a program
RECORD	A table of attributes
EXPRESSION	A formula
INTERREF	A reference on a model or an instance on a model
ATTRPROFREFEFRENCE	A preset set of attribute values

Table 2-2 Data Types allowed by ADOxx® to define Attributes

2.2.3 Specification of Mechanisms and Algorithms

For the specification of mechanism and algorithms, Kühn proposed known as "Mechanism Definition Language" (Kühn 2004). The Figure 2-7 depicts syntactical constructs of metamodel of the "Mechanism Definition Language". Based on their relation to the meta²-model, mechanisms are classified as a generic mechanism, a specific mechanism or as a hybrid mechanism. One mechanism can have either null, one or more inputs, but mechanisms produce at least one output. One mechanism can use the other mechanism or mechanisms. The execution of a mechanism is started by a trigger. A trigger can be either synchronous or asynchronous. One mechanism can have one or more precondition and post-condition. A Mechanism can be described via natural language, via UML, via pseudo code and etc. The implementation of a mechanism can be achieved via a scriptural language, so-called "AdoScript", which has ADOxx® specific dialect, via liked external DLLs or executable programme.

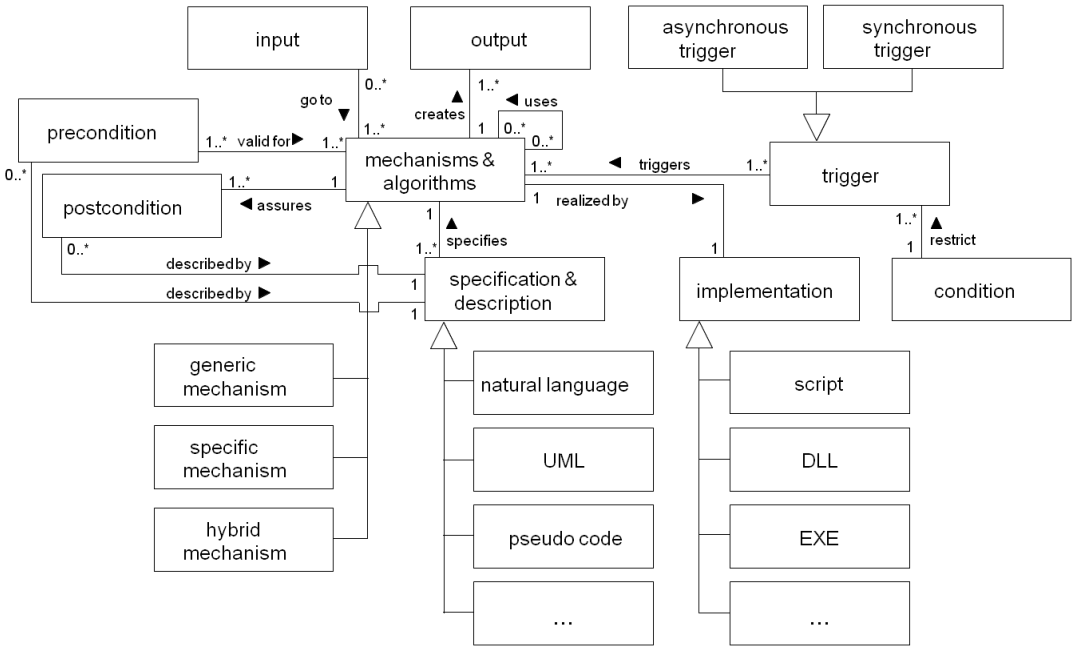


Figure 2-7 Metamodel of the Mechanism Definition Language (Kühn 2004) p.

2.3 Formal Description of Meta Models Based on ADOxx®

(Fill, Redmond and Karagiannis 2012) proposed a formalism to describe the core constituents of the ADOxx® meta modelling approach, namely "A Formalism for Describing ADOxx® Meta Models and Models" shortly called "FDMM".

The FDMM allows us to describe our modelling languages based on their informal or formal description, in such a way that ensures compliance of description of the modelling language with ADOxx® approach (Fill, Redmond and Karagiannis 2012).

Formalism to define ADOxx® metamodels proposed like following in (Fill, Redmond and Karagiannis 2012);

- $MM = \langle MT, \preceq, domain, range, card \rangle$
- $MT = \{MT_1, MT_2, \ldots . MT_n\}$
- $MT_i = \langle O_i^T, D_i^T, A_i \rangle$

Here MM represents the metamodel from which model instances mt are derived, MT represents the set of model types MT_i which consists of O_i^T representing a set of object types, D_i^T representing a set of data types, A_i representing a set of attributes. Each of the object types, data types and attributes represent a part of their respective total sets:

- $O^T = \bigcup_j O_j^T, D^T = \bigcup D_j^T, A = \bigcup A_j$

$\preceq$ represents an ordering among the set of objects, with what the inheritance hierarchy of ADOxx® classes can be expressed.

The $domain$ function fulfils restriction of what objects and attributes can map in the model instances:

- $domain: A \rightarrow P(\bigcup_j O_j^T)$

The $range$ function performs the mapping of one attribute to the power set of all pairs of object types and model types, all data types and all model types:

- $range: A \rightarrow P(\bigcup_j (O_j^T x \{MT_i\}) \cup D^T \cup MT)$

The $card$ function performs the mapping of pair of objects types and attributes to pairs of integers:

- $card: O^T x A \rightarrow P(\mathbb{N} x (\mathbb{N} \cup \{\infty\}))$

Here $\mathbb{N}$ represents the set of non-integers. Moreover FDMM defines the correctness criteria for metamodels: The set of object types, data types and attributes have to be pairwise disjointed:

- $O^T \cap D^T = \emptyset, O^T \cap A = \emptyset, D^T \cap A = \emptyset$

In addition the FDMM defines another equation that ensures that the $domain$ function relates an attribute and an object type, which are the part of the same model type definition

- $a \in A_i \Rightarrow domain(a) \subseteq O_i^T$

This work applies FDMM for description of selected Petri Nets modelling languages in chapter 3 andthe description of mechanisms in chapter 4 in order to investigate matching points among them.

3 Petri Nets

Petri Nets have wide application area like design; implementation and control of production systems (Silvam, et al. 1998), of computer supported cooperative work (De Michelis and Clarence A. 1998), design of digital hardware (Yakovlev and Koelmans 1998), design and analysis of intelligent networks (Capellmann, Dibold and herzog 1999), modelling distributed algorithms for networks of agents (Reisig, et al. 1998). For that concern, we have chosen the Petri Net Modelling languages and related mechanism and algorithms as the empirical application of this work.

3.1 Introduction in Petri Nets

The concept of petri nets was introduced by Carl Adam Petri in his dissertation (Petri 1966) . According to (Murata 1989) petri nets are defined as follows "*Petri Nets are a graphical and mathematical modelling tool applicable to many systems. They can be a useful tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, non – deterministic, and/or stochastic".* Again according to (Murata 1989) petri nets are a type of the directed graphs, which has an initial state called initial marking so-presented "M_0" This particular type of graph has two types of nodes namely *place* and *transition* and among them *arcs* either from place to transition or from transition to place. Due to commonly accepted representation, places are drawn as circles, transitions are drawn as boxes.

Figure 3-1 depicts the classification of Petri Nets based on the proposal made by (Trompedeller 1995) and a survey made (Bernardinello and Cindio 1992). This figure gives an overview of various kinds of Petri Nets that are available in literature. Actually "Petri Nets" is a generic name for various kind of net-based models which consist of a net and the rules of a token game executed on the net, an which can be classified into three main levels; (1)The level I represents the fundamental net-based models which are well suited for a thorough investigation of foundational issues of concurrent systems, (2) the level II represents the intermediate net-based models which represent systems in a more compact way by wrapping repetitive features of fundamental net-based models and (3) the level II represents the high level net-based models that provide compact net representation by using algebraic and logical tools (Rozenberg and Engelfriet 1998).

Existing large number of various kinds of Petri Nets was the one of the reasons to take Petri Nets as the empirical application of this work. But nevertheless to restrain the scope of this work we take two three type of Petri Nets, one from each level, under consideration as a sample and specify their modelling languages in section 3.2 by following ADOxx® Metamodelling Approach as introduced in (OMILAB 2013) and using ADOxx®® Metamodelling Platform version 1.3. Those Petri Nets types are (1) Elementary Petri Nets from level I-fundamental, (2) Place/Transtion Petri Nets from level II-intermediate and (3) Coloured Petri Nets from level III.

We list and specify the mechanisms in section 3.3 that we want to investigate if they can match with any of three Petri Nets modelling languages.

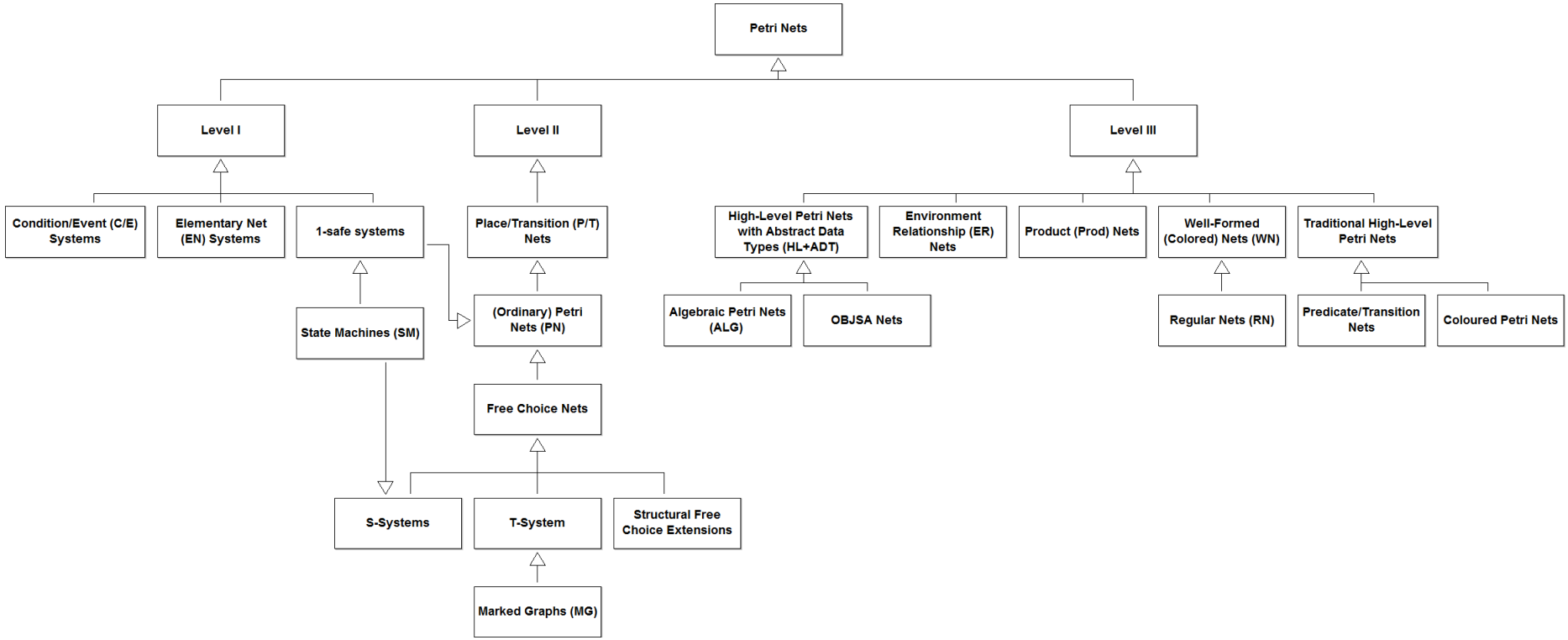


Figure 3-1 Classification of Petri Nets based on the proposal made by (Trompedeller 1995)

3.2 List of identified Modelling Languages

3.2.1 Elementary Petri Nets Modelling Language

In this section, we introduce elementary petri nets by giving a formal definition of elementary nets based on (Rozenberg and Engelfriet 1998) (W. Reisig 2010). Afterwards we introduce specified Elementary Petri Nets Modelling Language based on formal definition of elementary nets by presenting its syntax, semantic and notation which are defined according to ADOxx Framework.

One important part of elementary petri nets system is a net. A net has a structure represented is a 3-tuple form; $N = (P, T, F)$ (Rozenberg and Engelfriet 1998) where:

- P and T are finite set with $P \cap T = \emptyset$
- $X = P \cup T$
- $x \in X$, ${}^{\bullet}x = \{y \in X \mid (x, y) \in F\}$ is input set, $x^{\bullet} = \{y \in X \mid (x, y) \in F\}$ is output set of x
- $F \subseteq (PxT) \cup (TxP)$,
- for every $t \in T$ there exist $p, q \in P$ such that $(p, t), (t, q) \in F$
- for every $t \in T$ and $p, q \in P$, $if (p, t), (t, q) \in F$, then $p \neq q$

Here P represents the places, T represents transition; F represents flow relation of net (N) and X represents elements of net (N).
Passive states (or also known as local states) of the concurrent system that are consisted by a global system are represented by places. Active states (or so-called local transition) are represented by transition.

Besides a net, an elementary net system consists of an initial configuration or also so-called initial marking, which represented by marking corresponding places with tokens.

- $M_0 : P \rightarrow \mathbb{N}$, initial marking of a elementary net system
- $M: P \rightarrow \{0,1\}$, in elementary nets system each place p can have no or one token.
- Each Place p have $M(p)$ marks

Furthermore each arc f has a weight $w(p, t)$ which emphasizes how many tokens the underlying arc f can carry from place p to transition t (Murata 1989). In elementary nets system, each arc has weight of "1". Formal definition of weight function in elementary nets system is as follows:

- $W: F \rightarrow \{1\}$, each arc has exact weight of "1"

Is the $M(p) \geq 1$ for each place $p \in {}^{\bullet}t$, then marking M enables transition t, basically it means for the occurrence of the transition t, the transition t has to have one token on its input places. For the step $M \xrightarrow{t} M'$, defined for each place p as following (W. Reisig 2010);

Consequently we can define elementary net system (EN) in a 5-tuple form $EN = (P, T, F, W, M_0)$

Having a formal description of elementary net system according to the literature, it is easy to identify constituents of elementary Petri Nets modelling language and describe it with using FDMM.

We start defining the model types of the modelling language. The elementary Petri Nets modelling language consists of just one model type "Elementary Net" that can be described formally by using FDMM as following:

- $MT_{EN} = \langle O_{EN}^T, D_{EN}^T, A_{EN} \rangle$

Details of set of object types O_{EN}^T:

- $O_{EN}^T = \{Place, Transition, Place2TransitionArc, Transition2PlaceArc\}$

Details of set of object typesD_{EN}^T:

- $D_{EN}^T = \{String, Integer_{place}, Integer_{weight}, Enumeration_{State}\}$
- $Integer_{place} = \{0,1\}$
- $Integer_{weight} = \{1\}$
- $Enumeration_{State} = \{not\ enabled@enabled\}$

Details of set of attributes A_{EN}:

- $A_{EN} = \{Name, Number\ of\ Tokens, State, Weight, Place2TransitionArc - from,\ Place2TransitionArc - to, Transition2PlaceArc - from, Transition2PlaceArc - to\}$

Details of $domain$, $range$ and $card$ functions:

- $domain(Name) = \{Place, Transition\}$
- $range(Name) = \{String\}$
- $domain(Number\ of\ Tokens) = \{Place\}$
- $range(Number\ of\ Tokens) = \{Integer_{place}\}$
- $domain(State) = \{Transition\}$
- $range(State) = \{Enumeration_{State}\}$
- $domain(Weight) = \{Place2TransitionArc, Transition2PlaceArc\}$
- $range(Weight) = \{Integer_{weight}\}$
- $domain(Place2TransitionArc - from) = \{Place2TransitionArc\}$
- $range(Place2TransitionArc - from) = \{Place\}$
- $card(Place, Place2TransitionArc - from) = \langle 1,1 \rangle$
- $domain(Place2TransitionArc - to) = \{Place2TransitionArc\}$

- $range(Place2TransitionArc - to) = \{Transition\}$
- $card(Place, Place2TransitionArc - to) = \langle 1,1 \rangle$
- $domain(Transition2PlaceArc - from) = \{Transition2PlaceArc\}$
- $range(Transition2PlaceArc - from) = \{Transition\}$
- $card(Transition, Transition2PlaceArc - from) = \langle 1,1 \rangle$
- $domain(Transition2PlaceArc - to) = \{Transition2PlaceArc\}$
- $range(Transition2PlaceArc - to) = \{Place\}$
- $card(Transition, Transition2PlaceArc - to) = \langle 1,1 \rangle$

The Figure 3-2 depicts the metamodel of the Elementary Petri Nets, which is implemented on ADOxx®. The figure depicts inheritance relation between the metamodel of Elementary Petri Nets and pre-defined metamodel, as well as meta²model of ADOxx®.

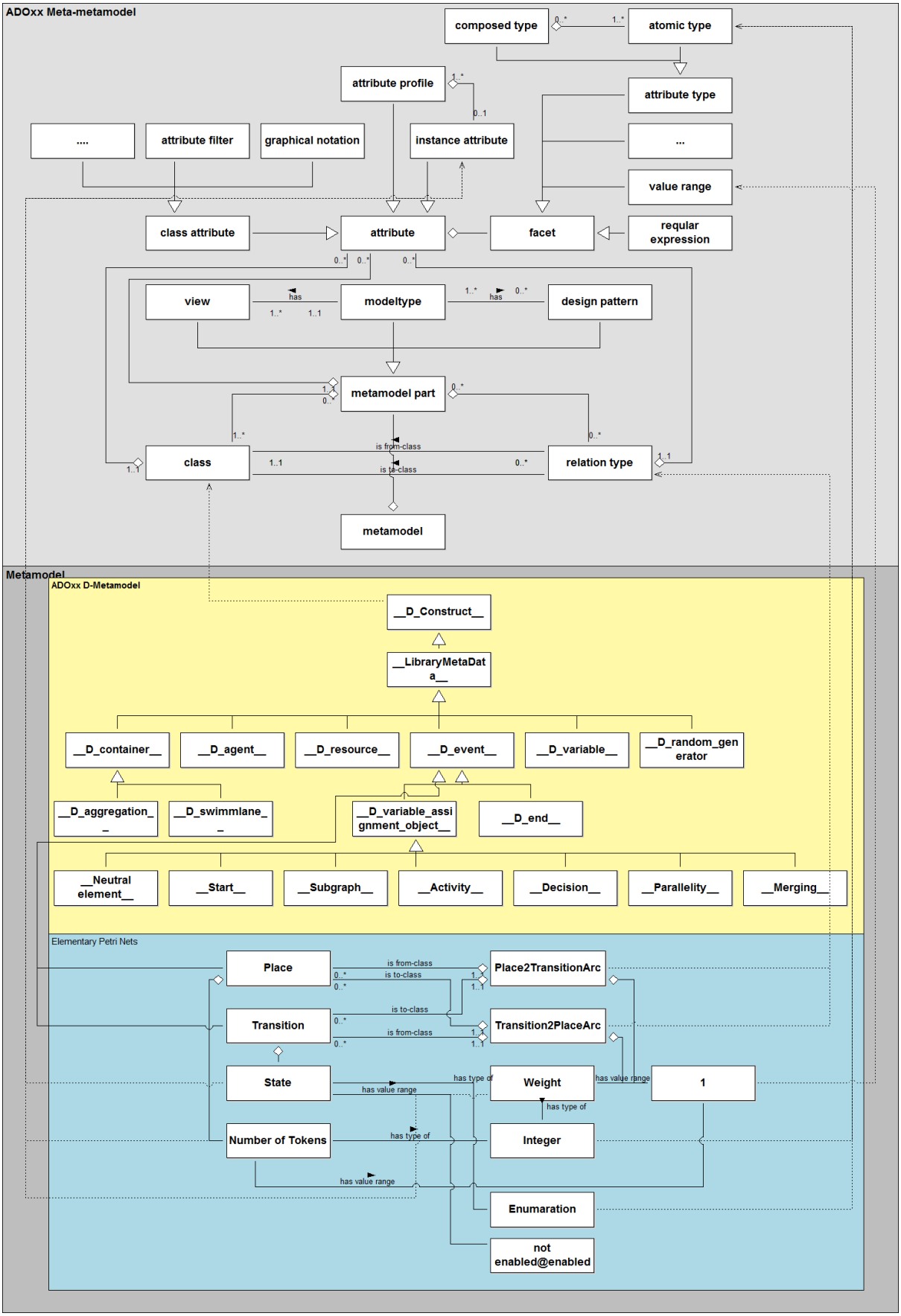


Figure 3-2 Metamodel of Elementary Petri Nets in the Model Hierarchy

In the Table 2-1 describes the specification of constructs, which take place in metamodel of Elementary Petri Nets.

Notation 	Place:Class	
	Class place represents the nodes that represent the passive states of elementary net systems.	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of place objects
Number of Tokens	Integer	The number of tokens the place object contains in given marking Default value: 0
Notation 	Transition : Class	
	Class transition the nodes that represent the active states of elementary nets systems	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of transition objects
State	Enumeration	State of the transition. Value range; not enabled@enabled
Notation	Place2TransitionArc : Relation class	
	Relation class Place2TransitonArc builds connection from a place node to a transition node.	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of relation object
Weight	Integer	The attribute weight has constant value of “1”

Notation	Transition2PlaceArc : Relation class
	Relation class Transition2PlaceArc builds connection from a transition node to place node.
Attributes	

Attribute Name	**Type**	**Description**
Name	String	Name of relation object
Weight	Integer	The attribute weight has constant value of "1"

Table 3-1 ADOxx® specific Specification of the Constructs of the Metamodel of Elementary Petri Nets

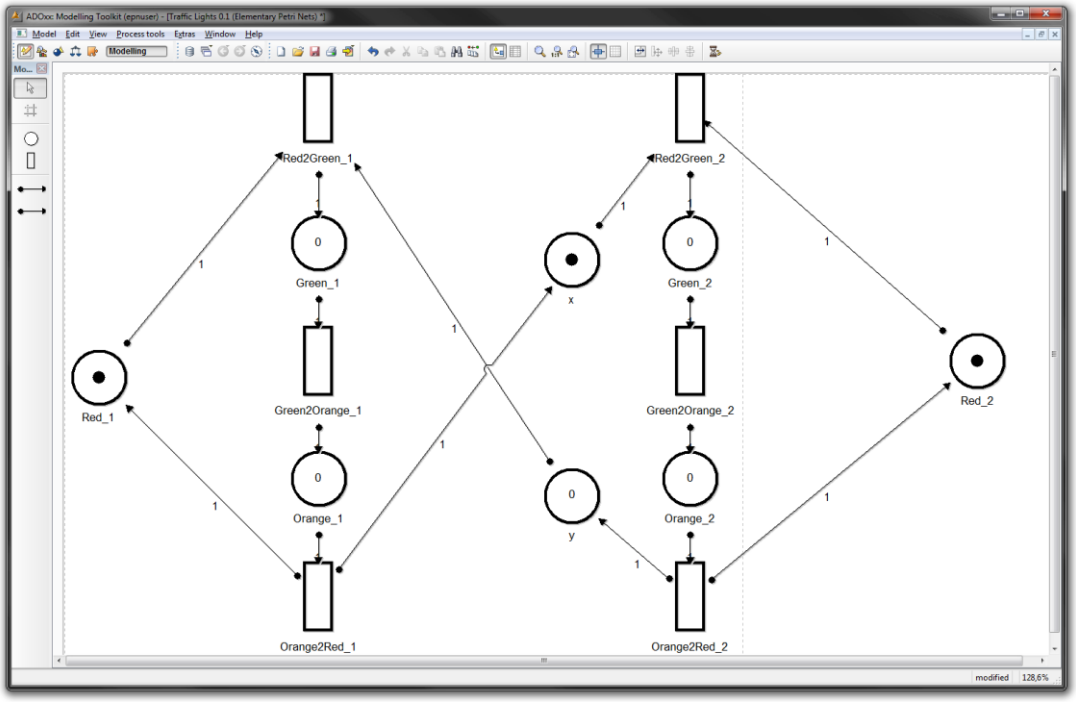


Figure 3-3 Elementary Petri Nets Modelling Editor and a Sample Model "Traffic Lights"

The Figure 3-3 is a screenshot that depicts the a sample Elementary Petri Nets model from (W. Reisig 2010) and modelled with using Elementary Petri Nets Modelling Editor, which is implemented on ADOxx® Metamodelling Platform based on ADOxx® specific specification of the Elementary Petri Nets Modelling Language listed in Table 3-3.

3.2.2 Place/Transition Petri Nets Modelling Language

In this section, we introduce the Place/Transition Nets by giving a formal definition of based on (Desel and Reisig 1998). Afterwards we introduce specified Place/Transition Nets

modelling language based on formal definition of the Place/Transition Nets by presenting its syntax, semantic and notation which are defined according to ADOxx® Framework.

Place/transition nets is a 5-tuple $PTN = (P, T, F, W, M_0)$ in contradistinction to elementary nets a place can take any number of tokens

- $M: P \rightarrow \{0,1,2,3 \ldots\}$, in Place/Transition Nets system each place p can have no or more token

We use the FDMM to describe place/transition constitutes of the Place/Transition Nets modelling language. The Place/Transition Nets modelling language consists of , like Elementary Petri Nets modelling language, just one model type called "Place/Transition Net" that can be described formally by using FDMM as following:

- $MT_{PTN} = \langle O^T_{PTN}, D^T_{PTN}, A_{PTN} \rangle$

Details of set of object types O^T_{PTN}:

- $O^T_{PTN} = \{Place, Transition, Place2TransitionArc, Transition2PlaceArc\}$

Details of set of object typesD^T_{PTN}:

- $D^T_{PTN} = \{String, Integer_{place}, Integer_{weight}, Enumeration_{State}\}$
- $Integer_{place} = \{0,1.999.999.999\}$
- $Integer_{weight} = \{1\}$
- $Enumeration_{State} = \{not\ enabled@enabled\}$

Details of set of attributes A_{PTN}:

- $A_{PTN} = \{Name, Number\ of\ Tokens, State, Weight, Place2TransitionArc - from,\ Place2TransitionArc - to, Transition2PlaceArc - from,\ Transition2PlaceArc - to\}$

Details of $domain$, $range$ and $card$ functions:

- $domain(Name) = \{Place, Transition\}$
- $range(Name) = \{String\}$
- $domain(Number\ of\ Tokens) = \{Place\}$
- $range(Number\ of\ Tokens) = \{Integer_{place}\}$
- $domain(State) = \{Transition\}$
- $range(State) = \{Enumeration_{State}\}$
- $domain(Weight) = \{Place2TransitionArc, Transition2PlaceArc\}$
- $range(Weight) = \{Integer_{weight}\}$
- $domain(Place2TransitionArc - from) = \{Place2TransitionArc\}$
- $range(Place2TransitionArc - from) = \{Place\}$
- $card(Place, Place2TransitionArc - from) = \langle 1,1 \rangle$

- $domain(Place2TransitionArc - to) = \{Place2TransitionArc\}$
- $range(Place2TransitionArc - to) = \{Transition\}$
- $card(Place, Place2TransitionArc - to) = \langle 1,1 \rangle$
- $domain(Transition2PlaceArc - from) = \{Transition2PlaceArc\}$
- $range(Transition2PlaceArc - from) = \{Transition\}$
- $card(Transition, Transition2PlaceArc - from) = \langle 1,1 \rangle$
- $domain(Transition2PlaceArc - to) = \{Transition2PlaceArc\}$
- $range(Transition2PlaceArc - to) = \{Place\}$
- $card(Transition, Transition2PlaceArc - to) = \langle 1,1 \rangle$

The Figure 3-4 depicts the metamodel of the Place/Transition Nets, which is implemented on ADOxx®. The figure depicts inheritance relation between the metamodel of Place/Transition Nets and pre-defined metamodel, as well as meta²model of ADOxx®.

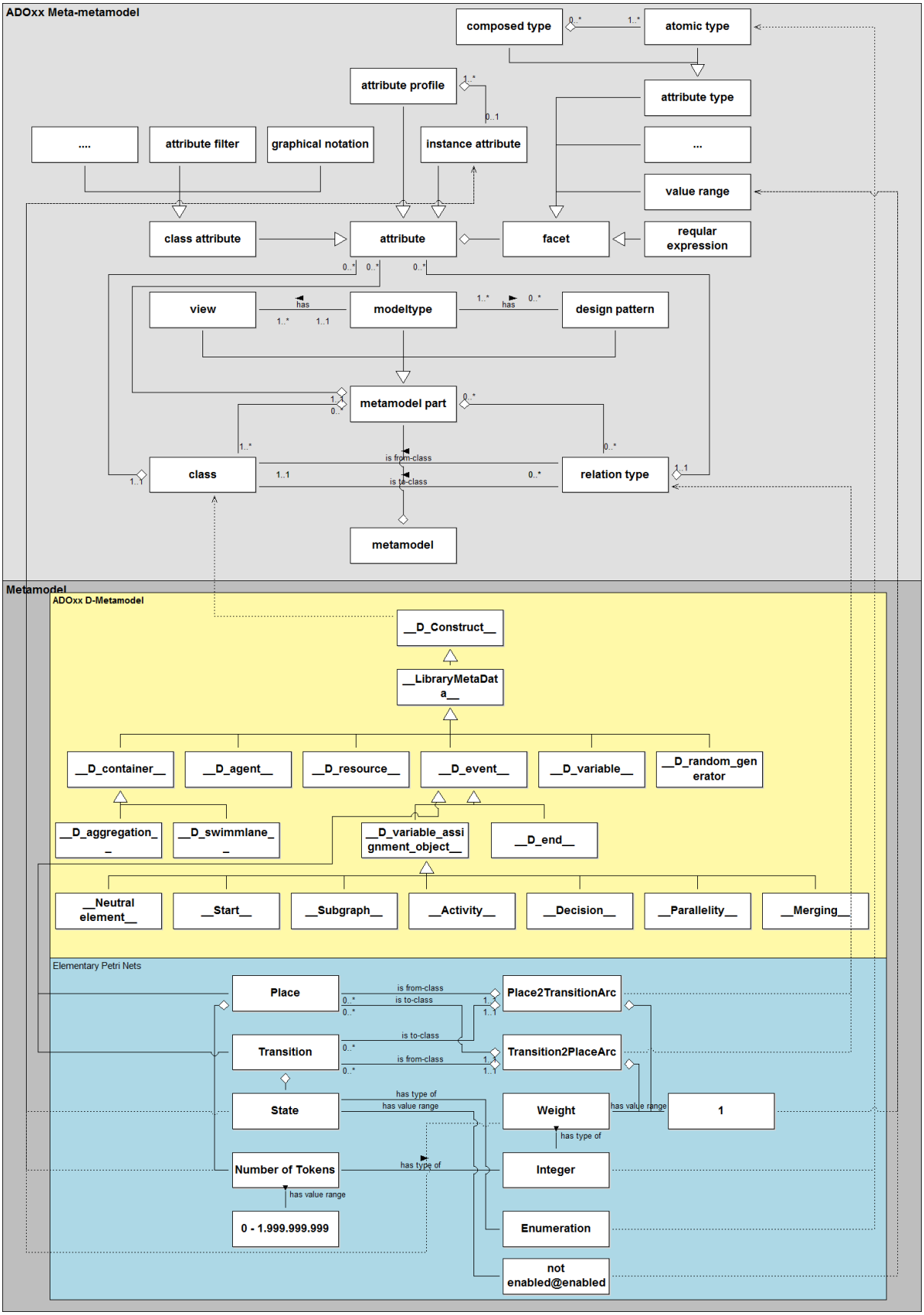


Figure 3-4 Metamodel of Place/Transition Petri Nets in the Model Hierarchy

Notation 	Place : Class	
	Class place represents the nodes that represent the passive states of the system.	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of place objects
Number of Tokens	Integer	The number of tokens the place object contains in given marking Default value: 0
Notation 	Transition : Class	
	Class transition the nodes that represent the active states of the system	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of transition objects
State	Enumeration	State of the transition. Value range; not enabled@enabled
Notation 	Place2TransitionArc : Relation class	
	Relation class Place2TransitonArc builds connection from a place node to a transition node.	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of relation object
Weight	Integer	Value range: positive integer
Notation 	Transition2PlaceArc : Relation class	
	Relation class Transition2PlaceArc builds connection from a transition node to place node.	

Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of relation object
Weight	Integer	Value range: positive integer

Table 3-2 ADOxx® specific Specification of the Constructs of the Metamodel of Place/Transition Nets

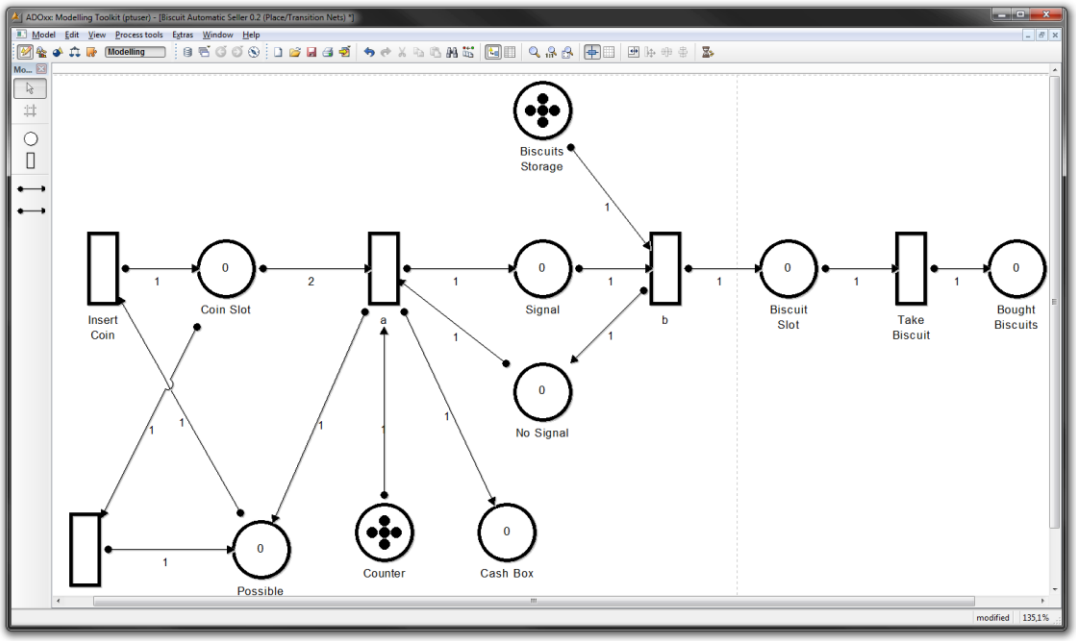


Figure 3-5 Place/Transition Nets Modelling Editor and a Sample Model "Automatic Biscuit Seller"

The Figure 3-5 is a screenshot depicts the a sample Place/Transition model based on the sample in (W. Reisig 2010) and modelled on Place/Transition Nets Modelling Editor, which is implemented on ADOxx® Metamodelling Platform based on ADOxx® specific specification of the Place/Transition Net Modelling Language listed in Table 3-2.

3.2.3 Coloured Petri Nets Modelling Language

In this section, we introduce coloured petri nets by giving formal definition of coloured petri nets based on (Jensen 1994) (Jensen 1997). Afterwards we introduce specified the Coloured Petri Nets modelling language based on formal definition of the Coloured Petri Nets by presenting its syntax, semantic and notation which are defined according to ADOxx® Framework.

(Jensen 1994) proposed to define a Coloured Petri Nets by a 9-tuple as following;

$CPN = (\sum, P, T, A, N, C, G, E, I)$ where:

- $\sum$ is a finite set of non-empty types, also so-called colour sets
- P is a finite set of places
- T is a finite set of transitions
- A is a finite set of arcs
- N is a node function. It is defined from A into $PxT \cup TxP$
- C is a colour function. It is defined from P into $\sum$
- G is a guard function. It is defined from T into expression
- E is an arc expression function. It is defined from A into expressions
- I is an initialisation function. It is defined from P into closed expressions

According to the simplified theory and narrative of Coloured Petri Nets given in (Jensen 1997) places represent the passive state of the system, transitions represent the actions of the system and arcs describe the relations between place and transitions, the arc expressions describe how the state of the CPN changes when transitions occur. Each place is marked with set of markers called tokens that contains data value that belongs to a given type. Hence the Coloured Petri Nets have distinguishable tokens; they are in contradiction to low-level petri nets (e.g. elementary petri nets and place/transition nets). A colour set is a type and a token colour is a token value.

For the occurrence of a transition, the transition has to have sufficient tokens on its input places and these tokens have to have token values that match the arc expressions.

We use the FDMM to describe constitutes of the Coloured Petri Nets modelling language. Coloured Petri Nets, has just one model type called "Coloured Petri Net" that can be described formally by using FDMM as following:

- $MT_{CPN} = \langle O^T_{CPN}, D^T_{CPN}, A_{CPN} \rangle$

Details of set of object types O^T_{CPN}:

- $O^T_{CPN} = \{Place, Transition, Tokens, Place2TransitionArc, Transition2PlaceArc\}$

Details of set of object typesD^T_{CPN}:

- $D^T_{CPN} = \{String, Expression, Enumeration_{State}, Record_{Tokens}\}$
- $Enumeration_{CPN} = \{not\ enabled@enabled\}$

Details of set of attributes A_{CPN}:

- $A_{CPN} = \{Name, Tokens, State, Guard, Data\ Value, Data\ Type, Place2TransitionArc - from,\ Place2TransitionArc - to, Transition2PlaceArc - from, Transition2PlaceArc - to\}$

Details of $domain$, $range$ and $card$ functions:

- $domain(Name) = \{Place, Transition\}$
- $range(Name) = \{String\}$
- $domain(Data\ Value) = \{Tokens\}$
- $range(Data\ Value) = \{String\}$
- $domain(Data\ Type) = \{Tokens\}$
- $range(Data\ Type) = \{String\}$
- $domain(Tokens) = \{Place\}$
- $range(Tokens) = \{Record_{Tokens}\}$
- $domain(State) = \{Transition\}$
- $range(State) = \{Enumeration_{State}\}$
- $domain(Guard) = \{Place2TransitionArc, Transition2PlaceArc\}$
- $range(Guard) = \{Expression\}$
- $domain(Place2TransitionArc - from) = \{Place2TransitionArc\}$
- $range(Place2TransitionArc - from) = \{Place\}$
- $card(Place, Place2TransitionArc - from) = \langle 1,1 \rangle$
- $domain(Place2TransitionArc - to) = \{Place2TransitionArc\}$
- $range(Place2TransitionArc - to) = \{Transition\}$
- $card(Place, Place2TransitionArc - to) = \langle 1,1 \rangle$
- $domain(Transition2PlaceArc - from) = \{Transition2PlaceArc\}$
- $range(Transition2PlaceArc - from) = \{Transition\}$
- $card(Transition, Transition2PlaceArc - from) = \langle 1,1 \rangle$
- $domain(Transition2PlaceArc - to) = \{Transition2PlaceArc\}$
- $range(Transition2PlaceArc - to) = \{Place\}$
- $card(Transition, Transition2PlaceArc - to) = \langle 1,1 \rangle$

The Figure 3-4 depicts the metamodel of the Place/Transition Nets, which is implemented on ADOxx®. The figure depicts inheritance relation between the metamodel of Place/Transition Nets and pre-defined metamodel, as well as meta²model of ADOxx®.

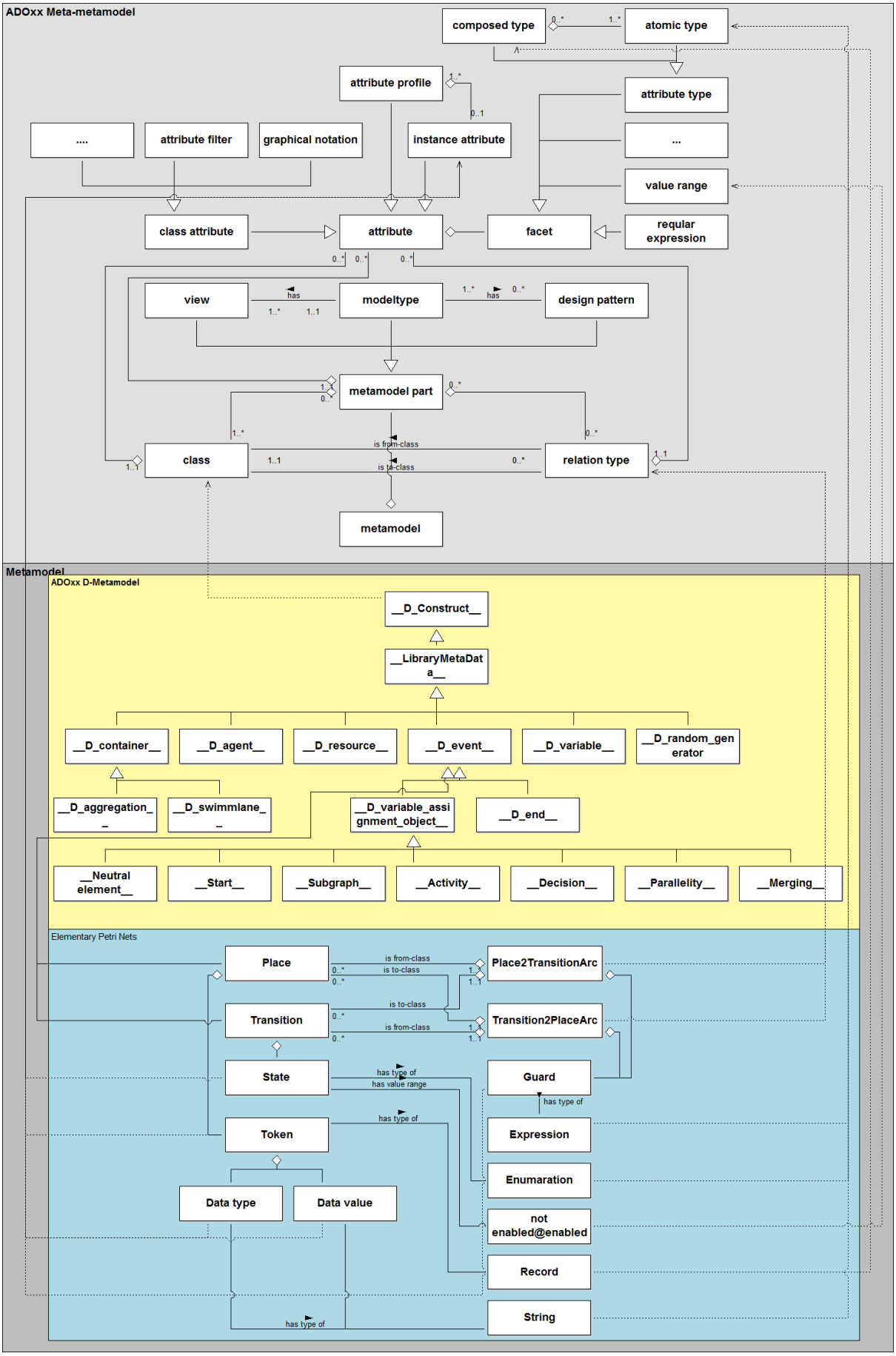

ADOxx Meta-metamodel
composed type
atomic type
attribute profile
attribute type
....
attribute filter
graphical notation
instance attribute
...
value range
class attribute
attribute
facet
regular expression
view
has
modeltype
has
design pattern
metamodel part
class
is from-class
is to-class
relation type
metamodel
Metamodel
ADOxx D-Metamodel
__D_Construct__
__LibraryMetaData__
__D_container__
__D_agent__
__D_resource__
__D_event__
__D_variable__
__D_random_generator
__D_aggregation__
__D_swimmlane__
__D_variable_assignment_object__
__D_end__
__Neutral element__
__Start__
__Subgraph__
__Activity__
__Decision__
__Parallelity__
__Merging__
Elementary Petri Nets
Place
Place2TransitionArc
Transition
Transition2PlaceArc
State
Guard
has type of
has value range
Token
Expression
Enumaration
Data type
Data value
not enabled@enabled
Record
String

Figure 3-6 Metamodel of Coloured Petri Nets and Model Hierarchy

Notation 	Place : Class	
	Class place represents the nodes that represent the passive states of elementary net systems.	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of place objects
Tokens	Record	It consists of complex information about type of data and value of data.
Notation: N/A	Tokens : Complex Type Attribute	
	Record class of Tokens, which consists complex value consist of Data Type and Value	
Attributes		
Attribute Name	**Type**	**Description**
Date Type	String	It defines the type of data, with the notions in ´Colored Petri Nets it represents the color.
Data Value	String	Value of the data
Notation 	Transition : Class	
	Class transition the nodes that represent the active states of elementary nets systems	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of place objects
State	Enumeration	State of the transition. Value range; not enabled@enabled
Notation	Place2TransitionArc : Relation class	
	Relation class Place2TransitonArc builds connection from a place	

	node to a transition node.	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of relation object
Guard	Expression	It defines the transition conditions from given place node and to given transition node.
Notation	Transition2PlaceArc : Relation class	
	Relation class Transition2PlaceArc builds connection from a transition node to a place node.	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	Name of relation object
Guard	Expression	It defines the transition conditions from given transition node and to given place node.

Table 3-3 ADOxx® specific Specification of the Constructs of the Metamodel of the Coloured Petri Nets

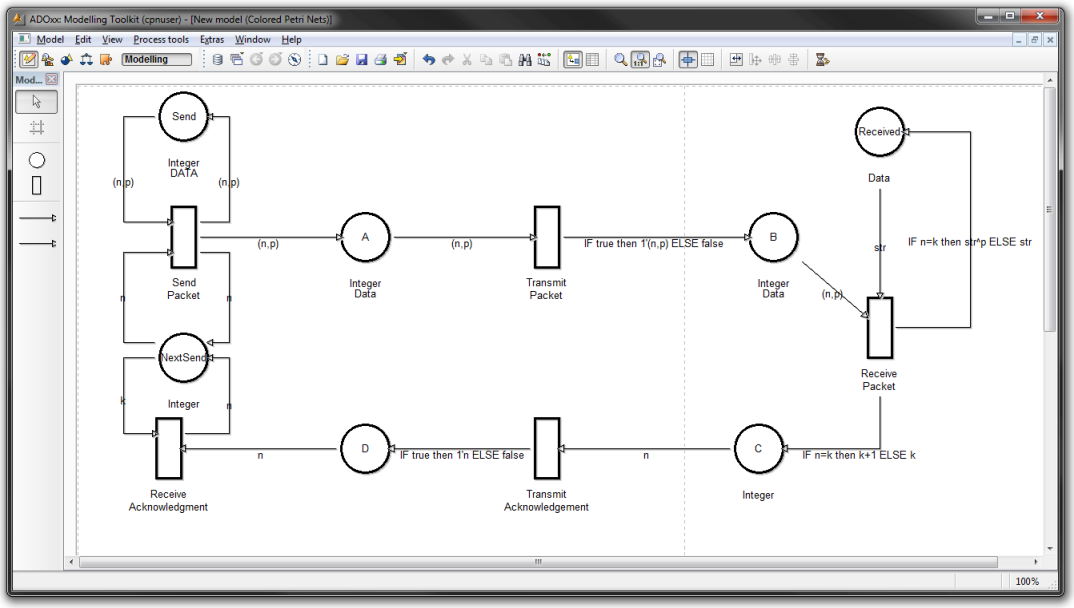


Figure 3-7 Colored Petri Nets Modelling Editor and a Sample Model "Simple Transport Protocol for a Unreliable Network"

The Figure 3-7 depicts the a sample Colored Petri Nets model from (Jensen 1997) and modelled on Colored Petri Nets Modelling Editor, which is implemented on ADOxx® MEtamodelling Platform based on ADOxx® specific specification of the Colored Petrin Nets Modelling Language listed in Table 3-3.

3.3 List of identified Mechanisms and Algorithms

We give precedence to the mechanisms that implemented within the ADOxx® Framework and investigate mechanism implemented within the project (Efendioglu, Lekaditis, et al. 2011) realized as a student project within the Open Models Initiative community. Besides that we investigate mechanisms offered by ADOxx® as core platform functionalities. For all that, we investigate prominent algorithms Genetic Algorithm and ADOxx® Path Analysis Algorithm.

In this section mechanisms & algorithms are listed, and they specified ADOxx® specific, in order to provide prior knowledge to develop formal mechanism description approach and then use them as input for the formal description of those mechanisms in section 4.3.1.

3.3.1 Fast Simulation

According to (Efendioglu, Lekaditis, et al. 2011) "Fast Simulation" is a mechanism, which selects the transition to be fired randomly, checks if the transition to be activated. The mechanism considers a transition to be activated if the number of tokens of the place that is connected with an incoming to the transition arc is equal or greater that the weight of the arc. Before mechanism starts with simulation checks cardinalities of the model by using mechanism "Cardinality Check". According to given execution number by the user, for every execution, the mechanism fires the transition randomly, which are able to activated, be a until either all loops executed or no other transition is activated. For each transition tokens are going to be consumed. Tokens from the place, which is connected with an incoming arc to the activated transition, is consumed according the weight of the incoming arcs. At the same time, tokens are added to the places which are connected with an outgoing to the activated and fired transition according to the weight of the outgoing from the transition arc.

In accordance with the description above we can outline required inputs from modelling language, returned outputs by the mechanism, trigger and dependencies of the mechanism like in the Table 3-4.

Mechanism Name: Fast Simulation		
Required Inputs		
Source Object	**Attribute Name**	**Attribute Type**

Place	ClassName	String
Transition	ClassName	String
P2TRelation	ClassName	String
T2PRelation	ClassName	String
Place	NumOfTokens	Integer
P2TRelation	Weight	Integer
T2PRelation	Weight	Integer
Transition	GameEnabled	Integer
Transition	Name	String
Place	Name	String
Returned Outputs		
Object Type	**Attribute**	**Type**
Place	NumOfTokens	Integer
Trigger		
Procedure	**Event**	
Simulation	Item:FastSimulation	
Dependency		
Petri Nets Cardinality Check		

Table 3-4 Specification of Mechanism "Fast Simulation"

3.3.2 Step by Step Simulation

According to (Efendioglu, Lekaditis, et al. 2011) as distinct from Fast Simulation mechanism, by Step by Step Simulation user has to manually select the transition to be fire. Besides cardinality check, mechanism checks whether any element is selected and if it is a transition. The mechanism has similar business logic as the Fast Simulation with difference of that the user needs decide by every execution which transition is to be fired.

In accordance with the description above we can outline required inputs from modelling language, returned outputs by the mechanism, trigger and dependencies of the mechanism like in the Table 3-5.

Mechanism Name: Step by Step Simulation		
Required Inputs		
Source Object	**Attribute Name**	**Attribute Type**
Place	ClassName	String
Transition	ClassName	String
P2TRelation	ClassName	String
T2PRelation	ClassName	String
Place	NumOfTokens	Integer
P2TRelation	Weight	Integer
T2PRelation	Weight	Integer
Transition	GameEnabled	Integer
Transition	Name	String
Place	Name	String
Returned Outputs		
Object Type	**Attribute Name**	**Attribute Type**
Place	NumOfTokens	Integer
UI Object INFOBOX	Text	String
Trigger		
Procedure	**Event**	
Simulation	Item:StepByStepSimulation	
Dependent with Mechanism		
Petri Nets Cardinality Check		

Table 3-5 Specification of Mechanism "StepByStep Simulation"

3.3.3 Net Statistics

According to (Efendioglu, Lekaditis, et al. 2011) this mechanism queries number of *places*, *transitions* and *arcs* in a Petri Net in order to provide statistical overview about the net.

In accordance with the description above we can outline required inputs from modelling language, returned outputs by the mechanism, trigger and dependencies of the mechanism like in the Table 3-6.

Mechanism Name: Net Statistics		
Required Inputs		
Source Object	**Attribute Name**	**Attribute Type**
Model	Id	Integer
Place	ClassName	String
Transition	ClassName	String
Returned Outputs		
Object Type	**Attribute Name**	**Attribute Type**
UI Object INFOBOX	Text	String
Trigger		
Procedure	**Event**	
Analysis	Item:NetStatistics	
Dependent with Mechanism		
N/A		

Table 3-6 Specification of Mechanism "Net Statistics"

3.3.4 PNML Transformation

In the project (Efendioglu, Lekaditis, et al. 2011) implemented The PNML Transformation mechanism has been implemented as an Add-on, which makes it more interesting for this work, since it offers a chance to prove matching mechanism for the outer level of mechanism according to ADOxx® Framework. The PNML Transformation mechanism transforms Petri Nets Models (instances of Petri Nets Modelling Language implemented within the project) which are exported with using ADOxx® Model XML Export Mechanism complying ADOxx® Model XML schema into the Petri Nets Models complying with PNML schema and vice versa.

In accordance with the description above we can outline required inputs from modelling language, returned outputs by the mechanism, trigger and dependencies of the mechanism like in the Table 3-7.

Mechanism Name: PNML Transformation		
Required Inputs		
Source Object	**Attribute Name**	**Attribute Type**
Model	Model Name	String
Place	Name	String
Place	Number of Tokens	String
Place	Position	String
Transition	Name	String
Transition	Position	String
P2TRelation	Name	String
P2TRelation	Weight	String
P2TRelation	Position	String
T2PRelation	Name	String
T2PRelation	Weight	String
T2PRelation	Position	String
Returned Outputs		
Source Object	**Attribute Name**	**Attribute Type**
File	File Path	String
Trigger		
Procedure	**Event**	
Import/Export	Item: PNML Export Item: PNML Import	
Dependent with Mechanism		
ADOxx® Model XML export		

ADOxx® Model XML import

Table 3-7 Specification of Mechanism "PNML Transformation"

3.3.5 Genetic Algorithm

The genetic Algorithm is a type of heuristic search algorithm invented by John Holland used in computer science, which has elements in common; (1) population of chromosomes, (2) selection according to fitness, (3) crossover to produce new off springs and (4) random mutation of new offspring (M. Mitchell 1999).

When the Genetic Algorithm is applied on Petri Nets notion of chromosomes would be equivalent to notion initial marking for the Petri Nets.

In accordance with the description above we can outline required inputs from modelling language, returned outputs by the mechanism, trigger and dependencies of the mechanism like in the Table 3-8.

Algorithm Name: Genetic Algorithm		
Required Inputs		
Source Object	**Attribute Name**	**Attribute Type**
Place	Name	String
Place	Initial Number Of Tokens	Integer
Returned Outputs		
Source Object	**Attribute Name**	**Attribute Type**
Place	Number of Tokens	Integer
Trigger		
Procedure	**Event**	
N/A		
Dependent with Mechanism		
N/A		

Table 3-8 Specification of "Genetic Algorithm for Petri Nets"

3.3.6 ADOxx® Path Analysis Algorithm

According to (ADOxx.org 2013) is an simulation algorithm, which enables the user to evaluate the models which cover the required concepts by the algorithm. It returns path-related results for selected model. Path related results are calculated due to their occurrence or any other criteria defined by the method engineer.

In accordance with the description above we can outline required inputs from modelling language, returned outputs by the mechanism, trigger and dependencies of the mechanism like in the Table 3-9.

Mechanism Name: ADOxx® Path Analysis Mechanism		
Required Inputs		
Source Object	**Attribute Name**	**Attribute Type**
Start	Class Name	String
End	Class Name	String
Activity	Class Name	String
Activity	Execution Time	Time
Activity	Waiting Time	Time
Activity	Resting Time	Time
Activity	Transport Time	Time
Decision	Variable name	String
Decision	Variable scope	Enumeration
Decision	Variable type	Enumeration
Decision	Variable value	String
Subsequent	Transition condition	String
Subsequent	Transition probability	String
Returned Outputs		
Source Object	**Attribute Name**	**Attribute Type**
Start	SimulationResults	Time
Trigger		

Procedure	**Event**
Simulation	Item: Path Analysis
Dependent with Mechanism	
N/A	

Table 3-9 Specification of Mechanism "ADOxx® Path Analysis"

In this chapter Petri Nets is investigated selected Petri Nets Modelling languages are introduced, conceptualized and specified, their ADOxx-specific metamodels are designed and in order to have more exact definition of the modelling languages, they are formalized by FDMM. Moreover, specification of the mechanisms, which are selected for empirical application are introduced. We identify conceptual differences between modelling language, got the idea about application domain of each modelling languages and possible required functionalities.

4 Description of Mechanisms & Algorithms for Matching with Modelling Languages

We investigate in this chapter state-of-the-art in semantic service discovery studies and try to detect significant input that guides us to achieve our objective matching modelling languages and mechanism & algorithms and overcome the challenges caused semantic gaps between modelling language descriptions and mechanism descriptions. For that concern we investigate the most prominent standards in the context of semantic service discovery; OWL-S, WSMO and SAWSDL, WSDL OWL. Additionally application scenarios are identified in order to have the big picture, as well as matching dimensions between modelling languages and mechanism & algorithms, matching points which shall be considered on each dimension and the cohesion among them are identified. Moreover with guidance of the outcomes of state-of-the-art analysis and investigation for matching dimension, we identify formal description approach for mechanisms & algorithms, classify concepts in the notions of Petri Nets and concepts in the notions of ADOxx®, in a taxonomy. Then we investigate a Petri Nets Ontology proposed by (Gašević and Devedžić 2006) and adapt this ontology with integrating the aforementioned taxonomy in order to have an Petri Net Ontology, that consists also notions of ADOxx®, which ease the utilization of the ontology during the closing semantic gap between modelling languages implemented for a certain domain and mechanisms & algorithm not necessarily implemented for a certain domain, but both implemented on ADOxx® Metamodelling Platform. Furthermore outcomes of that chapter are used for the specification of the modelling language that provides model-driven support for semantic enriched description of mechanisms & algorithms, which is introduced in chapter 5. For that all, the outcomes of that chapter influenced the design and implementation of the SeMD Modelling Editor and the Matching, which compose together

For the integration of mechanism and algorithm into modelling method three questions proposed by (Kühn 2004) that must be answered

1. In which method or in which method fragment can mechanism be used?
2. With which mechanisms has the mechanism dependency?
3. Which Integration-Patterns can be used on the mechanism?

This work aims to answers the question "how to match modelling languages and mechanism with each other", but not at the endpoint to integrate a mechanism into a method or method fragment. Therefore, the work aims to answer first questions proposed by (Kühn 2004), by the help of approaches developed in domain semantic service discovery.

Mostly in service discovery there three components are taking place; (1) Service agent that register its description into with a directory agent , (2) client agent that requests a service by submitting a query to the directory agent and (3) directory agent that matches the query with description of service (Yang 2001).

First of all to enable semantic service discovery service agent and client agent shall be agreed on the meaning of service description, which means they have to share the same ontology.

Then directory agent shall understand how much match two values - given any data type- which are included by service description and query (Yang 2001).

Again according to (Yang 2001) the shared ontology is reflection of the shared conceptual model of service, which describes, (1) what the service is capable of doing, (2) the terms in which the service is described and (3) the meanings of the terms.

From another point of view to match user goals and services two requirements are rising two different challenges; (1) user specific requirements and expectations have to be translated in more generic goal description - concrete user input has to be generalized to more abstract goals description- (2) semantic annotation of services has to be provided (Keller, et al. 2004).

According to (Paolucci, et al. 2002) in order to fulfil the requirements described above we need a language which expresses the capabilities of services and a matching algorithm which recognizes when a service request matches a service advertisement. We investigate state-of-the-art in semantic service description languages in 4.2 in order to understand which concepts are used in those languages and to understand which of them can be adapted to our approach for mechanism & algorithm description.

Above all, we need to understand on which dimensions matching between modelling languages and mechanisms can be occur and what the match points are. In section 4.1, we investigate matching dimensions than after investigation of state-of-the-art in semantic description in the section 4.3 we proposed possible matching points. We introduce the match points with the help of FDMM, and then we use the findings in section 5.1 in order to specify a description language for mechanisms.

4.1 Matching Dimensions for Mechanisms and Algorithms

According to findings in chapter 2 that matching dimension of a modelling language and a mechanism and algorithm differs depending on which modelling procedure modelling is being consist of by modelling technique, and which type of mechanism and algorithms, the mechanism and algorithm in question belongs to.

It was mentioned that a modelling procedure should ensure the completeness of the necessary aspects as well as syntactical correctness during the creation of models. The completeness of necessary aspects and the syntactical correction are influenced by modelling objectives (Karagiannis, Grossmann and Höfferer 2008). Hence due to modelling objectives amount of necessary aspects (e.g. required attributes to be specified), syntactical rules and steps can vary.

(Fill and Karagiannis 2013) proposes four function groups to classify the functionalities applied on the models as outlined in the Figure 4-1; (1) visualization, (2) transformation, (3) simulation, (4) querying".

This result leads us to classify the mechanism and algorithms, which enables processing of models indeed, by annotating with the appropriate function group.

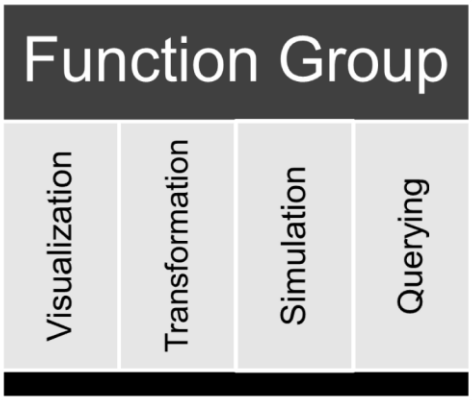


Figure 4-1 Matching Dimension: Function Groups of Mechanism and Algorithms

As already mentioned, there are three types of mechanism and algorithms (as depicted in Figure 4-2), which have dependencies on the different level of model hierarchy (depicted in Figure 4-3).

Figure 4-2 Matching Dimension: Mechanisms & Algorithms Type

To sum up (1) generic mechanism and algorithms which can work on modelling language independent of semantics of the modelling language, but its meta²-model, (2) specific mechanism and algorithms which are dependent of the semantics of the modelling language i.e metamodel of the modelling language and (3) hybrid mechanism and algorithms that independent of semantics of modelling language, but required some adaptation according to semantics of modelling language and they are dependent meta²model of the language.

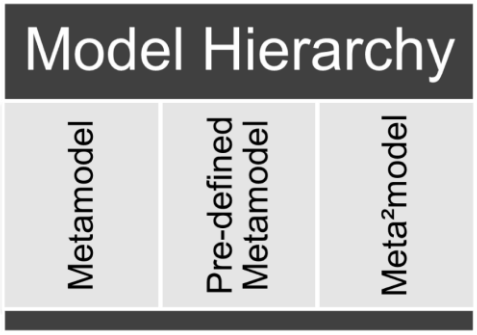


Figure 4-3 Matching Dimension: Model Hierarchy

Those previously mentioned points reveal the requirement of matching modelling language and mechanism & algorithm in three different dimensions as depicted in Figure 4-4; (1) Model hierarchy, (2) function group, (3) mechanism & algorithm type.

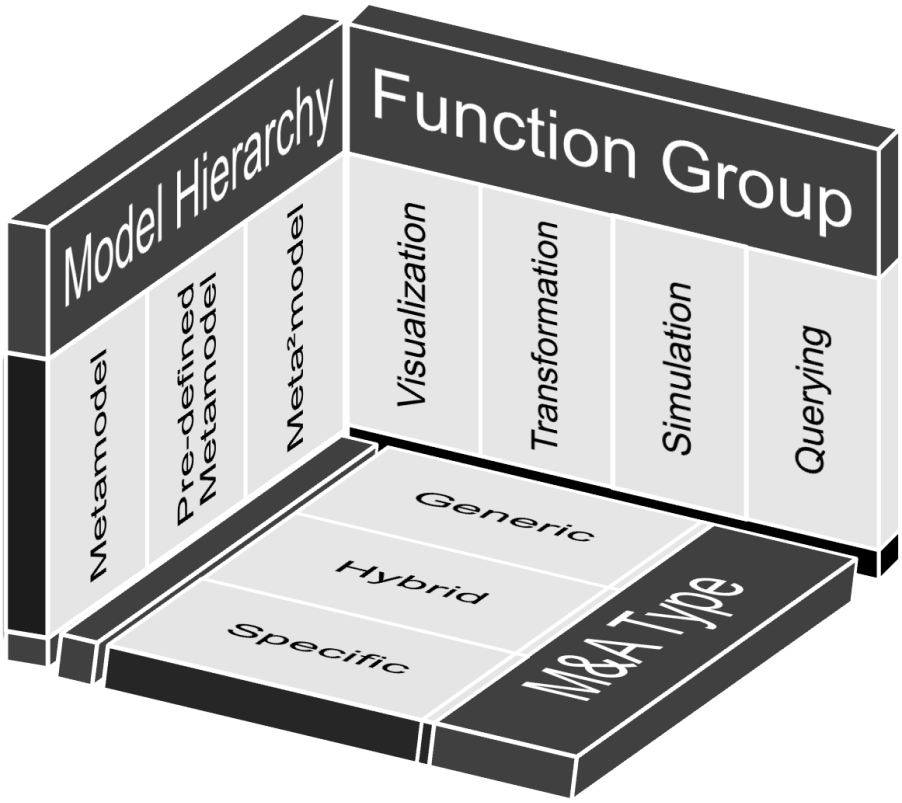


Figure 4-4 Cohesion between Matching Dimensions

As several times mentioned this work aims to expose a matching mechanism, which can support the method engineers to find and to match appropriate mechanism & algorithms to her/his modelling languages. We prefer to discuss following three application scenarios in order to convey, what is the cohesion between dimensions, what should be considered on each dimensions and considering these how the Matching Mechanism should work within those three dimensions and the functionalities that should be provided by the Matching Mechanism;

1. The method engineer does request to match certain modelling language with appropriate mechanism & algorithm from certain function group, the process sequence of matching engine is the following one:
    a. Since all mechanism & algorithms are annotated with their function group, the matching engine selects each mechanism & algorithm which has the matching annotation with the requested one
    b. Since the model level, where required concepts shall be searched, differs according to type of mechanism & algorithm, as candidate mechanisms are

selected according to their function group annotation, the ,matching engine groups the mechanism & algorithms according to their types.

c. After candidate mechanisms are grouped by their type, the matching engine will search required concepts at meta²model of the given modelling language for generic, at meta²model and metamodel of given modelling language for hybrid and at the metamodel of given modelling language for specific mechanism & algorithms.
d. The matching engine suggests appropriate mechanism & algorithms matching with given modelling language according to the method engineers request and also it presents matching and not matching concepts in modelling language required by candidate mechanism & algorithms. Thus user has the chance to configure modelling language or the mechanism & algorithms that are desired by the method engineer, due to results delivered by the matching engine.

2. The method engineer does request to match a certain modelling language with appropriate mechanism & algorithms from certain mechanism & algorithm type, the process sequence of the matching engine is the following one:
    a. Since all mechanism & algorithms are annotated according to their type, the matching engine selects each mechanism & algorithm which has the matching type annotation with the requested one.
    b. Then the matching engine groups candidate mechanism & algorithms by their function group annotation.
    c. After candidate mechanisms are grouped by their function group, the matching engine will search concepts required by mechanism & algorithms, at meta²model of the given modelling language, if requested type is generic, at meta²model and metamodel of given modelling language, if the requested type is hybrid and at metamodel of given modelling language, if requested type is specific mechanism & algorithm type.
    d. The matching engine suggests appropriate mechanism & algorithms matching with given modelling language according to method engineers request and also it presents matching and not matching concepts in modelling language required by candidate mechanism & algorithms. Thus method engineer has chance to configure modelling language or the mechanism & algorithms that he also wants to have, due to results delivered by the matching engine.
3. The method engineer does request to match a certain modelling language with appropriate mechanism & algorithms from any type of mechanism & algorithm described in mechanism & algorithms pool, the process sequence of the matching engine will be:
    a. The matching engine selects all mechanism & algorithms.
    b. The matching engine will search concepts required by mechanism & algorithms, at meta²model of the given modelling language, for generic mechanism & algorithms, at meta²model and metamodel for hybrid mechanism & algorithms and at the metamodel level for the specific mechanism & algorithms..

c. The matching engine suggests appropriate mechanism & algorithms matching with give modelling language according to method engineers request and also it presents matching and not matching concepts in modelling language required by candidate mechanism & algorithms. Thus method engineer has chance to configure.

At this point arise two questions; (1) how to describe the mechanism in a machine understandable way and (2) how to match mechanism and modelling languages according to description of mechanism.

4.2 Semantic Service Description: state-of-the-art

We select to analyse state-of-the-art in semantic service description studies to detect significant input that guides us to achieve our objective; matching modelling languages and mechanism & algorithms and overcome the challenges caused semantic gaps between modelling language descriptions and mechanism descriptions. In this manner, we can gain awareness of concepts used in each approach and understand which of them can be adapted to our approach for mechanism and algorithm description.

There are many standards to describe web services like Web Service Description Language (WSDL), Web Application Description Language (WADL), HTML for RESTful Service Descriptions (hRESTS), which are limited to syntactical description of web services. And there are also standards like Semantic Markup for Web Services (OWL-S) , Web Service Modelling Ontology (WSMO), Semantic Annotations for WSDL (SAWSDL) formerly Web Service Semantics (WSDL-S), Semantic Annotation of Web Resources (SA-REST),which are capable of both syntactical and semantic description of web services. Instead of investigating all standards, we constrain our analysis to approaches for semantic description of web services and further we constraint our analysis with three most prominent and promising approaches according to (Polleres, Lausen and Lara 2006) and (Cardoso, Miller and Emani 2008); OWL-S, WSMO and SAWSDL respectively.

Since WSDL is one of the most prominent standards to describe web services and OWL is one of the most prominent standards to build ontologies for semantic descriptions and besides that since most of the abovementioned standards utilize both standards, the WSDL and OWL are also included into the scope of investigation. For that matter, first of all the WSDL and OWL are going be investigated in order to understand functions of both and to have an overview how they are utilized in other standards.

4.2.1 WSDL: Web Service Description Language

WSDL is an XML-based language is used to describe web services in a machine-readable way, more detailed, it describes how services can be called, what parameters are required as input and what data structure is returned as output and where the service to be found (Christensen, et al. 2001). Currently updated version WSDL 2.0 is available. WSDL 2.0

offers a model and XML formant to describe Web services. Moreover it enables to separate abstract description of functionalities offered by services from concrete details of the service (Chinnici, Moreau, et al. 2007). The Figure 4-5 depicts the info set diagram of WSDL 2.0.

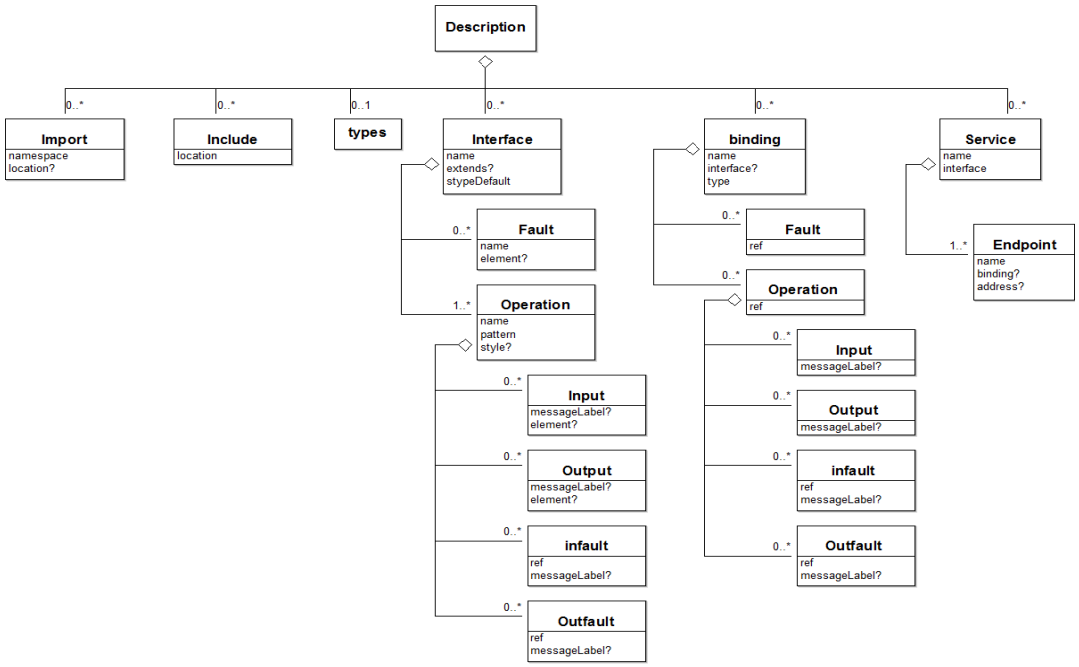


Figure 4-5 WSDL 2.0 Infoset (Booth und Liu 2005)

The most important elements of the in the info set are type, interface, binding and service.

T*ypes;* provides a mechanism to define message schemas. In practice, mostly XML schemas are used to define messages, which is directly supported by WSDL 2.0. The type elements are used in the children elements *input* and *output* in *operations* in element *interface*

Interface: contains two important element operation and fault. The operation element indicates an operation which implements this interface. It combines message schemas and message exchange pattern to describe the interaction with web service. The operation element has three important attribute, two of the required; (1) *name* attribute which must be unique within an interface and (2) *pattern* attribute which consists of list of absolute URIs identifies the appropriate message pattern. Third attribute of an operation is optional attribute *style* that identifies the encoding convention to be used. The *input, output, infault* and *outfault* are the children element of the operation which refer to the message types, which are defined within the *type* element to be used by given operation.

Bindings: in binding elements appropriate interface elements are referenced into attribute *interface*, and information like encoding, how messages are to be exchanged are specified. In the child element *operation* of binding, the transport protocol for regarding operation in referenced interface is specified.

Services*:* specifies basically where the services are to be accessed. The service can associate only one single interface that the service supports. The service has one or many child element

endpoint which specifies locations of the services where they can be accessed. In the child element *endpoint* an appropriate *binding* element should be referenced in order to indicate transport protocols and how the messages are to be exchanged.

4.2.2 OWL 2 Web Ontology Language

The OWL 2 Web Ontology Language (after now OWL 2), according to (Motik, et al. 2012) and (W3C OWL Working Group 2012)"*is an ontology language for the Semantic Web with formally defined meaning".* Ontologies specify the definitions of terms by describing their relationships among them. Ontologies are described by an ontology language which allows users to describe formal conceptualizations of domain models explicitly. OWL 2 is recommended by W3C and is considered to be one of the most dominant ontology languages currently

As shown in the Figure 4-6, the main building blocks of the OWL 2 are classes, objectproperties, dataproperties, annotation properties, datatypes and named individuals which are entities can be identified uniquely by Internationalized Resource Identifier (IRIs).

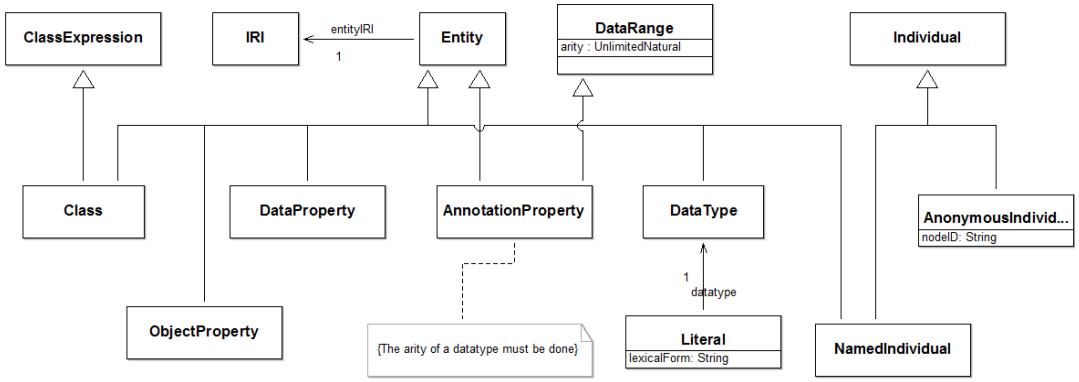


Figure 4-6 Entities, Literals, and Anonymous Individuals in OWL 2 (Motik, et al. 2012)

The *Classes* are entities which represent set of individuals. The *Object Properties and Data Properties* represent relationships in the domain and connects individuals. *Data properties* also can be seen as attributes. *Annotation Properties* are used to associate the non-logical information with ontologies, axioms and entities. *Datatypes* are set of literals refers to set of data values. *Individuals* represent objects from domain

The main components of the OWL 2 as W3C emphasized are the axioms, which state what is true what is wrong in the domain ontology. The Figure 4-7 depicts axiom sets of OWL 2. More details about all axiom sets of OWL 2 can be found in (Motik, et al. 2012). We are going to investigate three types of axioms sets and their axioms, which are more relevant for this work: class axioms, object property axioms and data property axioms.

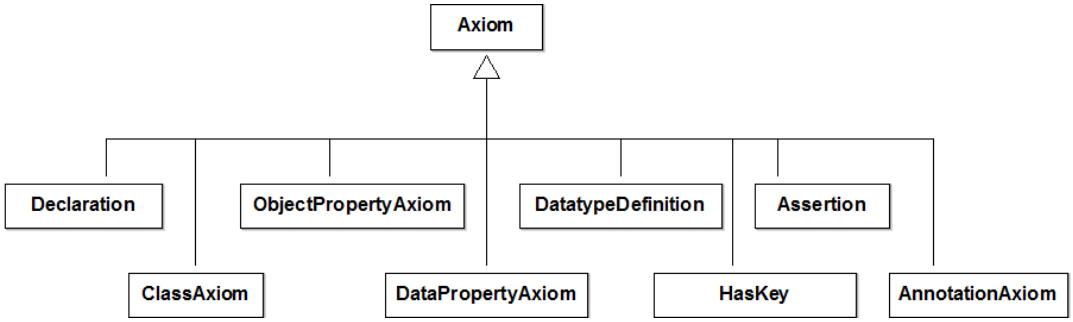


Figure 4-7 The Axioms of OWL 2 (Motik, et al. 2012)

OWL 2 Class Axioms depicted in Figure 4-8 allow defining relationship between class expressions (Motik, et al. 2012).

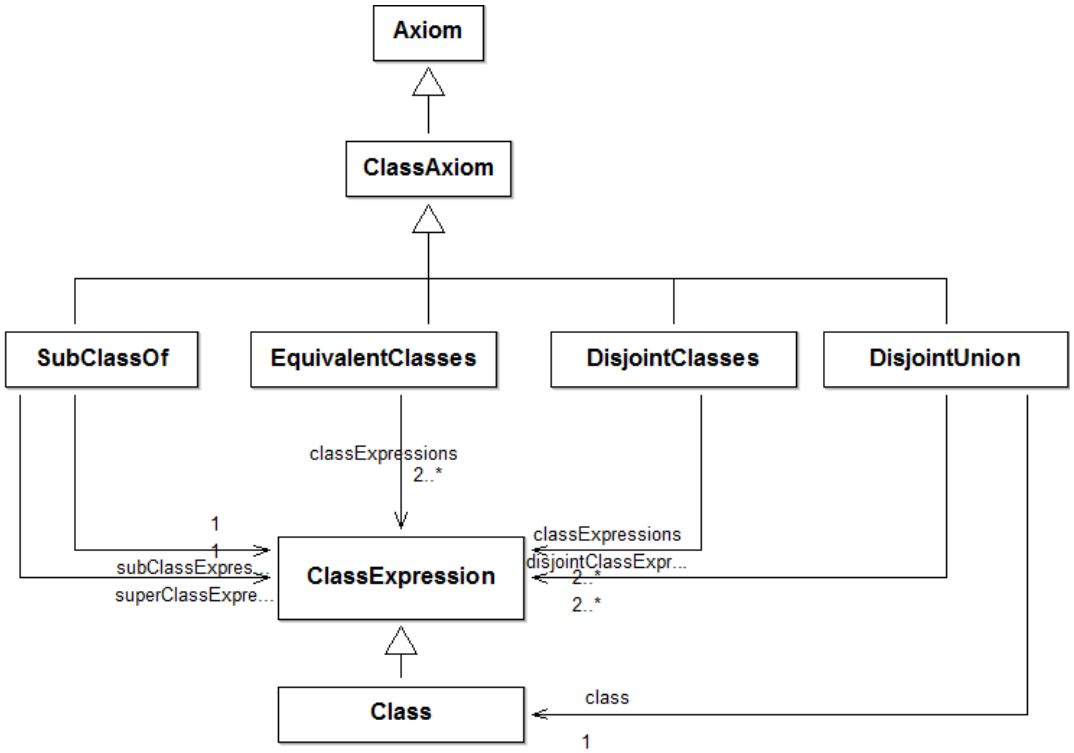


Figure 4-8 Class Axioms in OWL 2 (Motik, et al. 2012)

The class axioms are;

- *SubClassOf* axioms allow defining that the class expression is a subset of another class expression (Motik, Peter F. and Horrocks 2008).
- *EquivalentClasses* axioms allow defining that a class expression is exactly the same as another class expression (Motik, Peter F. and Horrocks 2008).
- *DisjointClasses* axioms allow defining that two class expressions are disjoint (Motik, Peter F. and Horrocks 2008).
- *DisjointUnion* axioms allow defining that a class as a disjoint union of several class expressions (Motik, Peter F. and Horrocks 2008).

OWL 2 Object Property Axioms depicted in Figure 4-9 are used to characterize and establish relationships between object property expressions (Motik, et al. 2012).

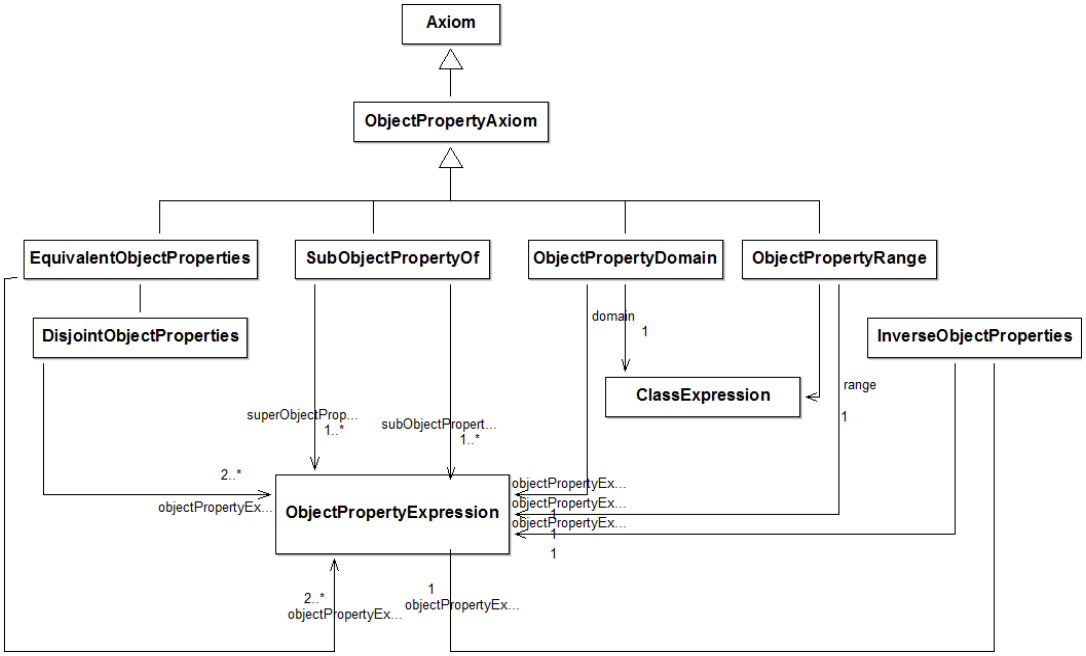


Figure 4-9 Object Property Axioms in OWL 2 (Motik, et al. 2012)

The object property axioms relevant for this work are;

- *SubObjectPropertyOf* axioms are used to state that extension of one object property expression is the subset of in the extension of another object property expression (Motik, Peter F. and Horrocks 2008).
- *EquivalentObjectProperties* axioms are used to state that the extensions of several object property expressions are the same (Motik, Peter F. and Horrocks 2008).
- *DisjointObjectProperties* axiom is used to state that the extensions of several object properties are pairwise disjoint — meaning that they do not share pairs of connected individuals (Motik, Peter F. and Horrocks 2008).

OWL 2 Data Property Axioms depicted in the Figure 4-10 similar like object property axioms, they are used to characterize and establish relationships between data property expressions (Motik, et al. 2012).

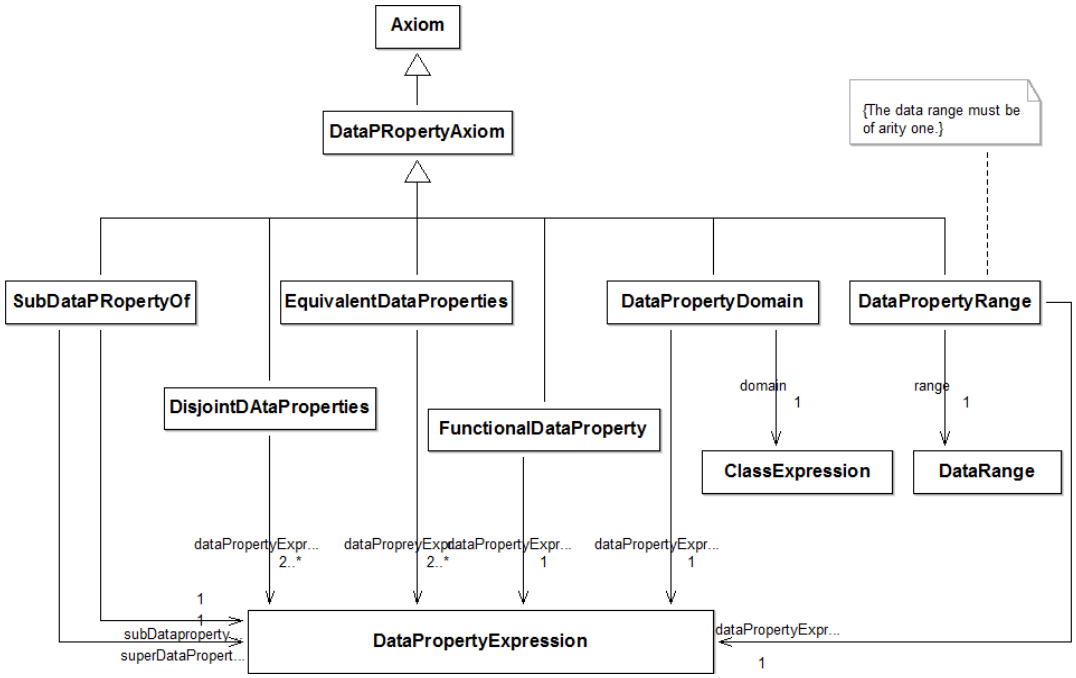


Figure 4-10 Data Property Axioms in OWL 2 (Motik, et al. 2012)

The data property axioms relevant for this work are;

- *SubDataPropertyOf* axioms are used to state that extension of one data property expression is the subset of in the extension of another data property expression.
- *EquivalentDataProperties* axioms are used to state that the extensions of several data property expressions are the same.
- *DisjointDataProperties* axiom is used to state that the extensions of several data properties are pairwise disjoint.

4.2.3 OWL-S: Semantic Markup for Web Services

OWL-S is built on OWL for semantic web standards and WSDL for web service standards (Martin, et al. 2004). In OWL-S, for formal description of services ontology structuring mechanism of OWL is utilized, for service grounding is WSDL standards utilized whereby mapping OWL-S inputs and outputs to WSDL messages and OWL-S services to WSDL operation (Cardoso, Miller and Emani 2008). The OWL-S has three concepts which carry essential knowledge about a service, are (1) *Service Profile*, (2*) Service Model* and (3) *Service Grounding.*

Service Profile: The service profile describes a service in the abstract level with three pieces of information; (1) the service provider information (2) transformation produced by the service in terms of which inputs are required and which outputs are produced by service and what the pre- and post-conditions are and (3) profile which consists of information about feature of the service like service category, quality ranking of service and unbounded list of

service parameters contains any type of information. Moreover, it is possible to assign a service category by referencing external taxonomy to the service profile. Briefly the service profile involves information about name, contact, and description of service, properties, inputs, outputs, pre-conditions and effects (IPOE).

The properties defined by profile ontology to point input, output, pre-condition effects defined in process ontology built during the service modelling, which are most interesting and relevant for this work are;

- *hasParameter:* references a *Parameter* instance of the *Process* ontology. The *Parameter* class in Process ontology which encapsulates inputs and outputs
- *hasInput:* references a *Input* instance defined in the *Process* ontology
- *hasOutput:* references a Output instance defined in the *Process* Ontology
- *hasPrecondition:* references a Output instance defined in the *Process* Ontology
- *hasResult:* references a *Output* instance defined in the *Process* Ontology

Service Model: The service model gives a detailed perspective on how to interact with a service and allows describing service as a process. The OWL-S distinguishes processes between an atomic process, composite process and simple process. Atomic processes, like service profile, consist of inputs, outputs, pre-conditions and post-conditions. Composite processes allow describing of activities that are described with atomic processes. Simple process, in a composition, representing the abstract view of composite processes and the simple processes are the processes which are not invokable.

Service Grounding*:* the service grounding describes how to access the service. Basically, it binds abstract specification of the service with its concrete specification. Since the WSDL is a steady and accepted standard OWL-S uses WSDL for grounding mechanism.

4.2.4 WSMO: Web Service Modelling Ontology

The WSMO according to (De Bruijn, Bussler, et al. 2005) provides on the one site conceptual framework to describe services semantically and a formal language based on description logics and logic Programming the Web Service Modelling Language (WSML) (De Bruijn, Fensel, et al. 2005) (Cardoso, Miller and Emani 2008) provides a formal syntax and semantics for WSMO.

The WSMO has four main elements for describing web services (De Bruijn, Bussler, et al. 2005);

Ontologies: define vocabularies used to define the other WSMO elements. The ontologies are described by using Web Service Modeling Language (WSML), which provides a formal syntax and semantics for WSMO (De Bruijn, Fensel, et al. 2005). The ontology is consist of five modelling element in WSML; (1) Concept, (2) relation, (3) instance, (4) relation instance, and (5) axiom

Web Services: covers three aspects to describe web service; (1) functional aspects (capabilities), which specify pre-conditions, assumptions, post-condition and effects of service (2) non-functional aspects and (3) behavioural aspects (interfaces), which describes how the functionality of the web service can be fulfilled. The WSMO distinguishes interfaces between choreography interfaces and orchestration interfaces, where choreography interfaces describe how user can interact with web services and orchestration interfaces describe which functionalities required from other services.

Goals: represents required functionalities from users which shall be fulfilled by the execution of a web service. One interesting point is the description of goal from user's point of view and decoupling from service description, so WSMO use to discovery comparison of goal and service description (Polleres, Lausen and Lara 2006). Five type properties can be used for defining a goal; (1) hasNonFunctionalProperties, where specified accuracy, reliability ,rights, type, version like properties which are expected from a service to have, (2) importOnotology used to import ontologies as long as no conflicts need to be resolved, (3) usesMediator, where defined which mediator shall be used to import ontology in case of need of imported ontologies for aligning, merging and transforming.

Mediators: are used to handle the interoperability problems *on the data level, process level* among the WSMO elements. The mediators are distinguished in four types (1) ggMediators which links two goals, (2) ooMediators import ontologies and determine possible representation mismatches. (3) wgMediators link web services to goals, (4) wwMediators link two web services.

4.2.5 SAWSDL: Semantic Annotations for WSDL and XML Schema

SAWSDL (Farrell and Lausen 2007) based on its predecessor the WSDL-S (Akkiraju, et al. 2005) which is set of extensions for WSDL and provides set of standard description format for web services (Kopecky, et al. 2007). It is developed to meet the deficit of WSDL by describing requirements and capabilities of services semantically and it defines a mechanism to add semantic annotations to the web service descriptions described by using WSDL. Basically with another words, concepts in a web service description are mapped to onotological concepts (Cardoso, Miller and Emani 2008).

SAWSDL offers extensions attributes to annotate WSDL interfaces, operations and their input and output messages. SAWSDL offers two types of extension attribute (Kopecky, et al. 2007);

- Model References are used to reference one or more semantic concepts to the WSDL elements
- Schema Mappings are differentiating between two types of schema mappings (1) liftingSchemaMapping transforms data from an XML message into a semantic model, (2) while loweringSchemaMapping transforms data from semantic model into an XML message.

The SAWSDL self does not provide any service ontology or domain ontology, The SAWSDL requires service or domain ontologies, from which semantic concepts can be attached to the extension attributes.

As described in (Kopecky, et al. 2007), in order to automate tasks; "Service Discovery", "Negotiation", "Filtering", "Ranking and Selection", "Invocation, the SAWSDL takes four aspects of services to describe semantically

- *Information semantics* describes information model used by other semantic descriptions required during the performing data mediation.
- *Functional semantics* describes what service offers required during the service discovery in order to compare service capability and user requirements.
- *Non-functional semantics* defines details on running environment or implementation of the service, which are required during the invocation.
- *Behavioural semantics* describes external and internal behaviour of a service. External behaviour describes a protocol how the service to be consumed, internal behaviour describes how the service functionalities are aggregated.

4.2.6 Lessons Learned from State-of-the-Art Analysis

It seems that the most appropriate approach to describe mechanisms & algorithms is a description language, which is with semantically enriched.

We have learned that that description language shall be able to describe the profiles of mechanisms & algorithms which are consist of at least inputs, dependencies.

Considering semantic gap between concepts used to describe goals of the method engineer, which consists of –in our case- the modelling language itself, required type of mechanisms and required functionality, and concepts used to describe mechanisms & algorithms profile, the most challenging part is semantic enrichment of the profiles of mechanisms & algorithms. The state-of-the-art analysis leads us the use the approach semantic enrichment of the profiles with annotating concepts used in the profile of mechanisms & algorithms with semantic concepts from selected domain ontology.

The need of domain ontology and annotation mechanism using concepts from an ontology raises new challenge. Nevertheless analysis and semantic lifting approach introduced in (Hrgovcic, Karagiannis and Woitsch 2013) bring the answer for that challenge, using an extension attribute which consists of information of referenced semantic concepts, which is used for annotation of concept in the profile and which is referenced from ontology. Obviously another challenge for that issue is need of using a ontology language -to build the ontology-, which is appropriate and proved for that issue. The answer for that challenge is also identified thanks to state-of-the-analyse. Since it is proven in the SAWDL approach, seems that OWL is the most appropriate language to build ontology in our approach.

4.3 Describing Mechanisms & Algorithms Formally

In order to find matching points relevant to be described, we adapt FDMM formalism concepts to describe mechanism and goals in sections 4.3.1 and 4.3.2. So we prove match points between modelling languages and mechanisms, and match points between mechanism and goals. Additionally we introduce an is-*a* taxonomy , and a Petri Nets Ontology in section 4.3.3. Since we need to consider semantics defined in pre-defined ADOxx® metamodel – from which semantics inherited into modelling languages implemented on ADOxx®, we will excerpt concepts from aforementioned taxonomy, which are ADOxx® specific and the, concepts from Petri Net Ontology in order to build an ontology which is complying with cases that are investigated in this work. Afterwards, we use this adapted ontology to enriched mechanism descriptions semantically.

4.3.1 Formal Description of Mechanisms

According to scenarios exposed in section 4.1, in order to ensure matching modelling languages and mechanism, the description of a mechanism must have a mechanism profile which consists of at least;

(1) List of inputs that are required by the mechanism,
(2) Information about, to which function group the functionality of mechanism belongs to
(3) Dependencies (if any) of the mechanism with other mechanisms
(4) Information about, which type of mechanism it is.

Although the main goal of this works is to develop a matching engine try to match a modelling language with mechanisms by checking if the semantic primitives required as inputs by mechanism are existing, we are going to design the matching engine so that it permits to specify concrete required output in goal definition by user and concrete provided output by mechanism in mechanism definition. Hence we need to add additional information in mechanism description;

(5) List of outputs that are provided by mechanism.

Then we can describe a mechanism in following form by adapting formalism of FDMM;

- $Mechanism_i = \langle Inputs_i, Outputs_i Dependency_i, FunctionGroup_i, MechnismType_i \rangle$

Where $Inputs$ and $Outputs$ are sets of inputs and outputs respectively like:

- $Inputs_i = \{Input_1, Input_2, \dots, Input_m\}$
- $Outputs_i = \{Output_1, Output_2, \dots, Output_m\}$

Where $Input_m$ and $Output_m$ are certain attributes of an object type and which have certain data types like:

- $Input_m \rightarrow oxaxd$
- $Output_m \rightarrow oxaxd$

Where each o is an element of a certain object type set, a is an element of a certain attribute set and d is an element of a certain data type set as described by FDMM.

- $o \in O_i^T \ ,a \in A_i \ ,d \in D_i^T$

A $Dependency_i$is a set of mechanism that the mechanism being described is depended.

- $Dependency_i = \{Mechanism_1, Mechanism_2, \dots, Mechanism_m\}$

A $FunctionGroup_i$ can be a either visualization function group or transformation group or simulation group or querying function group.

- $FunctionGroup_i = \{Visualization \trianglelefteq Transformation \trianglelefteq Simulation \trianglelefteq Querying\}$

  Where $MechnismType_i$ can be either generic mechanism type or hybrid mechanism type or specific mechanism.

- $MechnismType_i = \{Generic \trianglelefteq Hybrid \trianglelefteq Specific\}$

Formal description of the mechanism listed for Petri Nets in the section 3.3.

Fast Simulation;

- $Mechanism_{FastSimulation} = \langle Inputs_{FastSimulation}, Outputs_{FastSimulation}, DependencyFastSimulation, FunctionGroupFastSimulation, MechnismTypeFastSimulation$

Where $Inputs_{FastSimulation}$ is set of inputs like

- $Inputs_{FastSimulation} = \{Input_{placeclassname}, Input_{transitionclassname}, Input_{p2trelationlassname,}, Input_{trp2trelationclassname}, Input_{numoftokens}, Input_{p2trelationweight}, Input_{t2prelationweight}, Input_{tgameeneabled}, Input_{placename}, Input_{transitionname}\}$
- $Input_{placeclassname} \rightarrow Place\ x\ ClassName\ x\ string$
- $Input_{transitionclassname} \rightarrow Transition\ x\ ClassName\ x\ string$
- $Input_{p2trelationclassname} \rightarrow P2TRelation\ x\ ClassName\ x\ string$
- $Input_{t2prelationclassname} \rightarrow T2PRelation\ x\ ClassName\ x\ string$
- $Input_{numoftokens} \rightarrow Place\ x\ NumOfTokens\ x\ integer$
- $Input_{p2trelationweight} \rightarrow P2TRelation\ x\ Weight\ x\ Integer$
- $Input_{t2prelationweight} \rightarrow T2PRelation\ x\ Weight\ x\ Integer$

- $Input_{tgameenabled} \rightarrow Transition\ x\ GameEnabled\ x\ integer$
- $Input_{placename} \rightarrow Place\ x\ Name\ x\ string$
- $Input_{transitionname} \rightarrow Transition\ x\ Name\ x\ string$

Where $Outputs$ is set of inputs like;

- $Outputs_{FastSimulation} = \{Output_{numoftokens}\}$
- $Output_{numoftokens} \rightarrow Place\ x\ NumOfTokens\ x\ integer$

The $Dependency_{FastSimulation}$ is described like.

- $Dependency_{FastSimulation} = Petri\ Nets\ Cardinality\ Check$

The $FunctionGroup_{FastSimulation}$ is described like.

- $FunctionGroup_{FastSimulation} = \{Simulation\}$

Where the $MechnismType_{FastSimulation}$ can be described like;

- $MechnismType_{FastSimulation} = \{Specific\}$

Step by Step Simulation;

- $Mechanism_{StepByStepSimulation} = \langle Inputs_{StepByStepSimulation}, Outputs_{StepByStepSimulation},$ $DependencyStepByStepSimulation,$ $FunctionGroupStepByStepSimulation,$ $MechnismTypeStepByStepSimulation,$

Where $Inputs$ is set of inputs like:

- $Inputs_{StepByStepSimulation} = \{Input_{placeclassname}, Input_{transitionclassname}, Input_{p2trelationlassname,}, Input_{trp2trelationclassname}, Input_{numoftokens}, Input_{p2trelationweight}, Input_{t2prelationweight}, Input_{tgameeneabled}, Input_{placename}, Input_{transitionname}\}$
- $Input_{placeclassname} \rightarrow Place\ x\ ClassName\ x\ string$
- $Input_{transitionclassname} \rightarrow Transition\ x\ ClassName\ x\ string$
- $Input_{p2trelationclassname} \rightarrow P2TRelation\ x\ ClassName\ x\ string$
- $Input_{t2prelationclassname} \rightarrow T2PRelation\ x\ ClassName\ x\ string$

- $Input_{numoftokens} \rightarrow Place\ x\ NumOfTokens\ x\ integer$
- $Input_{p2trelationweight} \rightarrow P2TRelation\ x\ Weight\ x\ Integer$
- $Input_{t2prelationweight} \rightarrow T2PRelation\ x\ Weight\ x\ Integer$
- $Input_{tgameenabled} \rightarrow Transition\ x\ GameEnabled\ x\ integer$
- $Input_{placename} \rightarrow Place\ x\ Name\ x\ string$
- $Input_{transitionname} \rightarrow Transition\ x\ Name\ x\ string$

Where $Outputs$ is set of inputs like;

- $Outputs_{StepByStepSimulation} = \{Output_{numoftokens}\}$
- $Output_{numoftokens} \rightarrow Place\ x\ NumOfTokens\ x\ integer$

The $Dependency_{StepByStepSimulation}$ is described like;

- $Dependency_{StepByStepSimulation} = Petri\ Nets\ Cardinality\ Check$

The $FunctionGroup_{StepByStepSimulation}$ is described like;

- $FunctionGroup_{StepByStepSimulation} = \{Simulation\}$

Where the $MechnismType_{StepByStepSimulation}$ can be described like;

- $MechnismType_{StepByStepSimulation} = \{Specific\}$

Net Statistics;

- $Mechanism_{NetStatistics} = \langle Inputs_{NetStatistics}, Outputs_{NetStatistics}, DependencyNetStatistics, FunctionGroupNetStatistics, MechnismTypeNetStatistics$

Where $Inputs$ is set of inputs like:

- $Inputs_{NetStatistics} = \{Input_{placeclassname}, Input_{transitionclassname}\}$
- $Input_{placeclassname} \rightarrow Place\ x\ ClassName\ x\ string$
- $Input_{transitionclassname} \rightarrow Transition\ x\ ClassName\ x\ string$

Where $Outputs$ is set of inputs like;

- $Outputs_{NetStatistics} = \{Output_{infobox}\}$
- $Output_{infobox} \rightarrow Infoboxx\ Text\ x\ string$

The $Dependency_{NetStatistics}$is described like.

- $Dependency_{NetStatistics} = \emptyset$

The $FunctionGroup_{NetStatistics}$ is described like.

- $FunctionGroup_{NetStatistics} = \{Querying\}$

Where the $MechnismType_{NetStatistics}$ can be described like .

- $MechnismType_{NetStatistics} = \{Specific\}$

ADOxx® XML Export;

- $Mechanism_{AdoxxXMLExport} = \langle Inputs_{AdoxxXMLExport}, Outputs_{AdoxxXMLExport}, DependencyAdoxxXMLExport, FunctionGroupAdoxxXMLExport, MechnismTypeAdoxxXMLExport$

Where $Inputs$ is set of inputs like:

- $Inputs_{AdoxxXMLExport} = \{Input_{rootclassname}\}$
- $Input_{rootclassname} \rightarrow Root\ x\ ClassName\ x\ string$

Where $Outputs$ is set of inputs like;

- $Outputs_{AdoxxXMLExport} = \{Output_{XMLfilePath}\}$
- $Output_{XMLfilePath} \rightarrow XMLFile\ x\ FilePath\ x\ string$

The $Dependency_{AdoxxXMLExport}$is described like.

- $Dependency_{AdoxxXMLExport} = \emptyset$

The $FunctionGroup_{AdoxxXMLExport}$ is described like.

- $FunctionGroup_{AdoxxXMLExport} = \{Transformation\}$

Where the $MechnismType_{AdoxxXMLExport}$ can be described like;

- $MechnismType_{AdoxxXMLExport} = \{Generic\}$

PNML Transformation;

- $Mechanism_{PNMLTransformation} = \langle Inputs_{PNMLTransformation}, Outputs_{PNMLTransformation}, DependencyPNMLTransformation, FunctionGroupPNMLTransformation, MechnismTypePNMLTransformation,$

Where $Inputs$ is set of inputs like:

- $Inputs_{PNMLTransformation} = \{Input_{modelname},\ Input_{placename},$ $Input_{transitionname}, Input_{p2trelationname,},\ Input_{trp2trelationname},$ $Input_{numoftokens}, Input_{p2trelationweight}, Input_{t2prelationweight},$ $Input_{placepositionx},\ Input_{placepositiony},\ Input_{placedimensionx}, Input_{placedimentiony},$ $Input_{transitionpositionx},\ Input_{transitionpositiony},$ $Input_{transitiondimensionx}, Input_{transitiondimentiony}\ ,\ Input_{p2trelationpositionx},$ $Input_{p2trelationpositiony},\ Input_{p2trelationdimensionx}, Input_{p2trelationdimentiony},$ $Input_{p2trelationfromclassname}, Input_{p2trelationtoclassname},\ \ Input_{t2prelationpositionx},$ $Input_{t2prelationpositiony},\ Input_{t2prelationdimensionx}, Input_{t2prelationdimentiony},$ $Input_{t2prelationfromclassname}, Input_{t2prelationtoclassname}\}$
- $Input_{modelname} \rightarrow Model\ x\ ModelName\ x\ string$
- $Input_{placename} \rightarrow Place\ x\ Name\ x\ string$
- $Input_{transitionname} \rightarrow Transition\ x\ Name\ x\ string$
- $Input_{p2trelationname} \rightarrow P2TRelation\ x\ Name\ x\ string$
- $Input_{t2prelationname} \rightarrow T2PRelation\ x\ Name\ x\ string$
- $Input_{numoftokens} \rightarrow Place\ x\ NumOfTokens\ x\ integer$
- $Input_{p2trelationweight} \rightarrow P2TRelation\ x\ Weight\ x\ Integer$
- $Input_{t2prelationweight} \rightarrow T2PRelation\ x\ Weight\ x\ Integer$

- $Input_{placepositionx} \rightarrow Place\ x\ PositionX\ x\ string$
- $Input_{placepositiony} \rightarrow Place\ x\ PositionY\ x\ string$
- $Input_{placedimensionx} \rightarrow Place\ x\ DimensionX\ x\ string$
- $Input_{placedimensiony} \rightarrow Place\ x\ DimensionY\ x\ string$
- $Input_{transitionpositionx} \rightarrow Transion\ x\ PositionX\ x\ string$
- $Input_{transitionpositiony} \rightarrow Transion\ x\ PositionY\ x\ string$
- $Input_{transitiondimensionx} \rightarrow Transion\ x\ DimensionX\ x\ string$
- $Input_{transitiondimensiony} \rightarrow Transion\ x\ DimensionY\ x\ string$

- $Input_{p2trelationpositionx} \rightarrow P2TRelation\ x\ PositionX\ x\ string$
- $Input_{p2trelationpositiony} \rightarrow P2TRelation\ x\ PositionY\ x\ string$
- $Input_{p2trelationdimensionx} \rightarrow P2TRelation\ x\ DimensionX\ x\ string$
- $Input_{p2trelationdimensiony} \rightarrow P2TRelation\ x\ DimensionY\ x\ string$
- $Input_{p2trelationfromclassname} \rightarrow P2TRelation\ x\ FromClassName\ x\ string$
- $Input_{p2trelationtoclassname} \rightarrow P2TRelation\ x\ ToClassName\ x\ string$
- $Input_{t2prelationpositionx} \rightarrow T2PRelation\ x\ PositionX\ x\ string$
- $Input_{t2prelationpositiony} \rightarrow T2PRelation\ x\ PositionY\ x\ string$
- $Input_{t2prelationdimensionx} \rightarrow T2PRelation\ x\ DimensionX\ x\ string$
- $Input_{t2prelationdimensiony} \rightarrow T2PRelation\ x\ DimensionY\ x\ string$
- $Input_{t2prelationfromclassname} \rightarrow T2PRelation\ x\ FromClassName\ x\ string$
- $Input_{t2prelationtoclassname} \rightarrow T2PRelation\ x\ ToClassName\ x\ string$

Where $Outputs$ is set of inputs like;

- $Outputs_{StepByStepSimulation} = \{Output_{XMLfile}\}$
- $Output_{XMLfile} \rightarrow XMLfile\ x\ FilePath\ x\ string$

The $Dependency_{PNMLTransformation}$ is described like;

- $Dependency_{PNMLTransformation} = ADOxx\ XML\ Export, ADOxx\ XML\ Import$

The $FunctionGroup_{PNMLTransformation}$ is described like;

- $FunctionGroup_{PNMLTransformation} = \{Transformation\}$

Where the $MechnismType_{PNMLTransformation}$ can be described like;

- $MechnismType_{PNMLTransformation} = \{Specific\}$

Genetic Algorithm for Petri Nets;

- $Mechanism_{geneticalgorithmpn} = \langle Inputs_{geneticalgorithmpn}, Outputs_{geneticalgorithmpn},$

$Dependency_{geneticalgorithmpn}, FunctionGroup_{geneticalgorithmpn},$
$MechnismType_{geneticalgorithmpn}\rangle$

Where $Inputs$ is set of inputs like:

- $Inputs_{geneticalgorithmpn} = \{Input_{placename}, Input_{initialNumberOfTokens}\}$
- $Input_{placename} \rightarrow Place\ x\ Name\ x\ string$
- $Input_{initialNumberOfTokens} \rightarrow Place\ x\ NumberOfTokens\ x\ integer$

Where $Outputs$ is set of inputs like;

- $Outputs_{geneticalgorithmpn} = \{Output_{XMLfilePath}\}$
- $Output_{NumberOfTokens} \rightarrow Place\ x\ NumberOfTokens\ x\ integer$

The $Dependency_{geneticalgorithmpn}$is described like.

- $Dependency_{geneticalgorithmpn} = \emptyset$

The $FunctionGroup_{geneticalgorithmpn}$ is described like.

- $FunctionGroup_{geneticalgorithmpn} = \{Simulation\}$

Where the $MechnismType_{geneticalgorithmpn}$ can be described like;

- $MechnismType_{geneticalgorithmpn} = \{Hybrid\}$

ADOxx® Path Analysis Algorithm;

- $Mechanism_{adoxxpathanalysisalgorithm} =$
  $\langle Inputs_{adoxxpathanalysisalgorithm}, Outputs_{adoxxpathanalysisalgorithm},$
  $Dependencyadoxxpathanalysisalgorithm,$
  $FunctionGroupadoxxpathanalysisalgorithm,$
  $MechnismTypeadoxxpathanalysisalgorithm$

Where $Inputs$ is set of inputs like:

- $Inputs_{adoxxpathanalysisalgorithm} =$
  $\{Input_{startclassname}, Input_{endclassname}, Input_{activityclassname}, Input_{activityexecutiontime},$
  $Input_{activitywaitingtime}, Input_{activityrestingtime}, Input_{activitytransporttime},$
  $Input_{decisionvariablename}, Input_{decisionvariablscope}, Input_{decisionvariabletype},$
  $Input_{decisionvariablevalue}, Input_{subsequenttransitioncondition},$
  $Input_{subsequenttransitionprobability}\}$
- $Input_{startclassname} \rightarrow Start\ x\ ClassName\ x\ string$
- $Input_{endclassname} \rightarrow End\ x\ ClassName\ x\ string$

- $Input_{activityclassname} \rightarrow Activity\ x\ ClassName\ x\ string$
- $Input_{activityexecutiontime} \rightarrow Activity\ x\ ExecutionTime\ x\ time$
- $Input_{activitywaitingtime} \rightarrow Activity\ x\ WaitingTime\ x\ time$
- $Input_{activityrestingtime} \rightarrow Activity\ x\ RestingTime\ x\ time$
- $Input_{decisionvariablename} \rightarrow Decision\ x\ VariableName\ x\ string$
- $Input_{decisionvariablscope} \rightarrow Decision\ x\ VariableScope\ x\ string$
- $Input_{decisionvariabletype} \rightarrow Decision\ x\ VariableType\ x\ string$
- $Input_{decisionvariablevalue} \rightarrow Decision\ x\ VariableValue\ x\ string$
- $Input_{subsequenttransitioncondition} \rightarrow Subsequent\ x\ TransitionCondition\ x\ string$
- $Input_{subsequenttransitionprobability} \rightarrow Subsequent\ x\ TransitionPorbability\ x\ string$

Where $Outputs$ is set of inputs like;

- $Outputs_{adoxxpathanalysisalgorithm} = \{Output_{\text{SimulationResults}}\}$
- $Output_{\text{SimulationResults}} \rightarrow Start\ x\ \text{SimulationResults}x\ string$

The $Dependency_{adoxxpathanalysisalgorithm}$ is described like.

- $Dependency_{adoxxpathanalysisalgorithm} = \emptyset$

The $FunctionGroup_{adoxxpathanalysisalgorithm}$ is described like.

- $FunctionGroup_{adoxxpathanalysisalgorithm} = \{Simulation\}$

Where the $MechnismType_{adoxxpathanalysisalgorithm}$ can be described like;

- $MechnismType_{adoxxpathanalysisalgorithm} = \{Hybrid\}$

4.3.2 Formal Description of Goals

As mentioned before the matching engine enables users to specify their goals in form of;

(1) The modelling language, for which they search mechanisms for, which is mandatory information.
(2) Function group that they are looking for, which is optional information.
(3) Mechanism type that they are looking for, which is optional information.

(4) List of concrete outputs that they are required, which is optional information in the goal description.

The goals can be described in following form by adapting formalism of FDMM;

- $Goal_i = \langle MM_i, FunctionGroup_i, MechnismType_i, Outputs_i \rangle$

So far we have proved that there are shared concepts by modelling language and mechanism thanks to FDMM, which represent the match points between modelling language, goals and mechanisms. More concretely, inputs, outputs, function group and type of a mechanism can be defined by using *object type*, *attribute type* and *data type* proposed by FDMM so they are the match points between

- Inputs required by mechanism and object type set, attribute set, data typeset of a modelling language, respecting domain and range functions which define the relation among the given sets in the modelling languages.
- Required outputs by a goal and object type set, attribute set, data typeset provided as outputs by mechanism.

The other match points are

- required function group and mechanism type defined in a goal and offered by a mechanism

Following introductory example shall explain our standing point better;

Let have modelling language in domain performance management and has one model type called KPI Pool model type, which consist of an object type called KPI and it has attributes name type of string, is-value and should-value type of integer. It can be described by using FDMM as following;

- $MM_{KPI\ Pool} = \langle MT_{KPI\ Pool}, \preceq, domain, range, card \rangle$
- $MT_{KPI\ Pool} = \langle O^T_{KPI\ Pool}, D^T_{KPI\ Pool}, A_{KPI\ Pool} \rangle$

Details of set of object types $O^T_{KPI\ Pool}$:

- $O^T_{KPI\ Pool} = \{KPI\}$

Details of set of object types $D^T_{KPI\ Pool}$:

- $D^T_{KPI\ Pool} = \{string, integer\}$

Details of set of attributes $A_{KPI\ Pool}$:

- $A_{KPI\ Pool} = \{name, is-value, should-value\}$

Details of $domain, range$ and $card$ functions:

- $domain(Name) = \{KPI\}$

- $range(Name) = \{String\}$
- $domain(is - value) = \{KPI\}$
- $range(is - value) = \{integer\}$
- $domain(should - value) = \{KPI\}$
- $range(should - value) = \{integer\}$

And let have another modelling language again in same domain performance management and has one model type called *Metrics Pool* model type, which consist of an object type called *Metrics* (we assume that it is equivalent of KPI) and it has attributes *Name* type of *string*, *is-value* and *should-value* type of *double*. It can be described by using FDMM as following;

- $MM_{Metrics\ Pool} = \langle MT_{Metrics\ Pool}, \preceq,\ domain, range, card \rangle$
- $MT_{Metrics\ Pool} = \langle O^T_{Metrics\ Pool}, D^T_{Metrics\ Pool}, A_{Metrics\ Pool} \rangle$

Details of set of object types $O^T_{Metrics\ Pool}$:

- $O^T_{Metrics\ Pool} = \{Metrics\ \}$

Details of set of object types $D^T_{KPI\ Pool}$:

- $D^T_{Metrics\ Pool} = \{string, double\}$

Details of set of attributes $A_{Metrics\ Pool}$:

- $A_{Metrics\ Pool} = \{Name, is - value, should - value\}$

Details of $domain$, $range$ and $card$ functions:

- $domain(Name) = \{Metrics\ \}$
- $range(Name) = \{string\}$
- $domain(is - value) = \{Metrics\ \}$
- $range(is - value) = \{double\}$
- $domain(should - value) = \{Metrics\ \}$
- $range(should - value) = \{double\}$

Additionally we have a *specific mechanism* called *score* which is in querying function group and divides *integer is-value* by *integer should value* of KPIs, therefore it requires *is-value* and should-value in type of integer from object type *KPI* we can describe the mechanism formally as following;

- $Mechanism_{score} = \langle Inputs_{score}, Dependency_{score}, FunctionGroup_{score}, MechnismType_{scorei} \rangle$

Where $Inputs$ is set of inputs like:

- $Inputs_{score} = \{Input_{is-value}, Input_{should-value}\}$

- $Input_{is-value} \rightarrow KPI\ x\ is-value\ x\ integer$
- $Input_{should-value} \rightarrow KPI\ x\ should-value\ x\ integer$

A $Dependency_{score}$is described like.

- $Dependency_{score} = \emptyset$

The $FunctionGroup_{score}$ is described like.

- $FunctionGroup_{score} = \{Querying\}$

Where the $MechnismType_{score}$ can be described like .

- $MechnismType_{score} = \{Specific\}$

Let a user make a query to find a specific mechanism that has querying function and works on modelling language "KPI Pool" and also on "Metrics Pool".

- $Goal_a = \langle MM_{KPI\ Pool}, FunctionGroup_{goal\_a}, MechnismType_{goal\_a}, Outputs_{goal\_a} \rangle$

Where

- $FunctionGroup_{goal\_a} = \{Querying\}$,
- $MechnismType_{goal\_a} = \{Specific\}$ and
- $Outputs_{goal\_a} = \emptyset$
- $Goal_b = \langle MM_{Metrics\ Pool}, FunctionGroup_{goal\_b}, MechnismType_{goal\_b}, Outputs_{goal\_b} \rangle$

Where

- $FunctionGroup_{goal\_b} = \{Querying\}$,
- $MechnismType_{goal\_b} = \{Specific\}$ and
- $Outputs_{goal\_b} = \emptyset$

Obviously matching result is going to be (according to formal description of modelling languages and the mechanism) a specific mechanism, which is in querying function group and called score, and matching result will be positive for the *goal a* consists of the modelling language "KPI Pool" since it consists of exactly the same concepts that the mechanism requires as inputs, but for the *goal b* consists of modelling language "Metrics Pool" will be negative because it does not consist of exactly the same concepts that the mechanism requires as inputs although object type "Metrics" is equivalent of "KPI" and consists of same attributes but in another data type which is also similar to data type integer.

We deduce from the introductory scenario explained above, the necessity of semantic expressiveness in the description of mechanisms in order to make syntactical concepts defined above processable by the matching engine.

And that lead us to prepare taxonomies for concepts that can be used to fortify semantic expressiveness of our approach to describe the mechanism by associating semantic primitives defined in taxonomies with concepts used in the definition of mechanisms.

4.3.3 Taxonomy as Classification Schema

If all modelling languages had always same semantic primitives/concepts, it would have been enough to define modelling language and mechanism with formalism consists of those same semantic primitives. In the reality even the modelling languages, which are abstracting the same System under Study (SUS) or rather modelling languages used for the same domain (e.g. BPMN[4] and EPC[5]) consist of heterogonous semantic primitives. Therefore it is required to identify shared concepts to classify them in a taxonomic organization utilized to describe parameters of mechanisms & algorithms, so mechanism & algorithms can distinguish the similarity between concepts that it needs to work and concepts that modelling languages have.

We can categorize semantic primitives in three groups by taking under consideration the three set definitions for a model type proposed by FDMM formalism, on the other hand, since to make separated taxonomies among the data types and among attribute types does not bring as semantic meaningfulness as object types, but bring them with object types in an ontology together and define their relation between object types with domain and range brings more semantic meaningfulness. Therefore we decided to build an *is-an* taxonomies, namely; Object Type Taxonomy, which consists of ADOxx® specific concepts. Although FDMM formalism categorize "Class" and "Relation class" in the same group "Object Type", and does not make precise separation we decide categorize classes and relation classes in their own separated groups in order to have more distinctive taxonomies and so simplify semantic similarity calculation within the matching engine. Moreover, the taxonomies are fortified with properties *disjoint-with*, *synonym-of* additional to property *is-a.*

The object type taxonomy consist of concepts "node", "place" and "transition in the context of Petri Net and taxonomy the *node* subsumes *place* and *transition.*

We have presented ADOxx® meta-model classes in section 2.2.2.1 and mentioned that the pre-defined class "*__D-Event__*" encapsulates all possible nodes of a graph. In fact that Petri Nets are directed graphs and Petri Nets modelling languages that are implemented on ADOxx® by inheriting semantic and syntax of pre-defined classes defined in ADOxx® Meta-model for graph-based modelling languages, in this case only "*__D-Event__*". In that case, we can replace "Node" in the object type taxonomy with "__D-Event_" in the context of ADOxx®. Besides that, there is no hierarchy among the relation types in the context of ADOxx®, which means from ADOxx® point of view each relation type is disjoint with the other ones..

[4] Business Process And Notation, http://www.omg.org/spec/BPMN/2.0/
[5] Event-driven Process Chain, http://www.ariscommunity.com/event-driven-process-chain

Murata exposed in (Murata 1989) some typical interpretations of Transitions and Places summarized in Table 4-1, which constitute the synonyms of corresponding class in the object type taxonomy.

Place	Transition
Condition	Event
Data	Computation Step
Signal	Signal processor
Resource	Task
Buffer	Processor
	Clause in logic

Table 4-1 Synonyms of Place and Transition

After inserting synonyms into object type taxonomy, the taxonomy will be extended as depicted in Figure 4-11. Briefly Petri Nets Object Type Taxonomy starts with the superclass "Object Type" subsumes "Class" and "Relation Type", while "Class" subsumes "_D-Event", which encapsulates each node in a graph, subsumes "Place" and "Transition" and they subsume their synonyms. We have seen in section 2.2.2 that relation types have the constraints that one relation type must have exactly one "fromclass", and exactly one "toclass" and this constraint allows instead of being dependent on relation type taxonomy which should be extended manually by human interaction, to leave automatic classification of relation types to a machine interaction. In our case matching engine classifies them according to their "fromclass" and "toclass". In that case, a prerequisite arises to be fulfilled during the definition of the tmechanism; If input is a relation type, the attributes "fromclass" and "toclass" of relation should be explicitly specified.

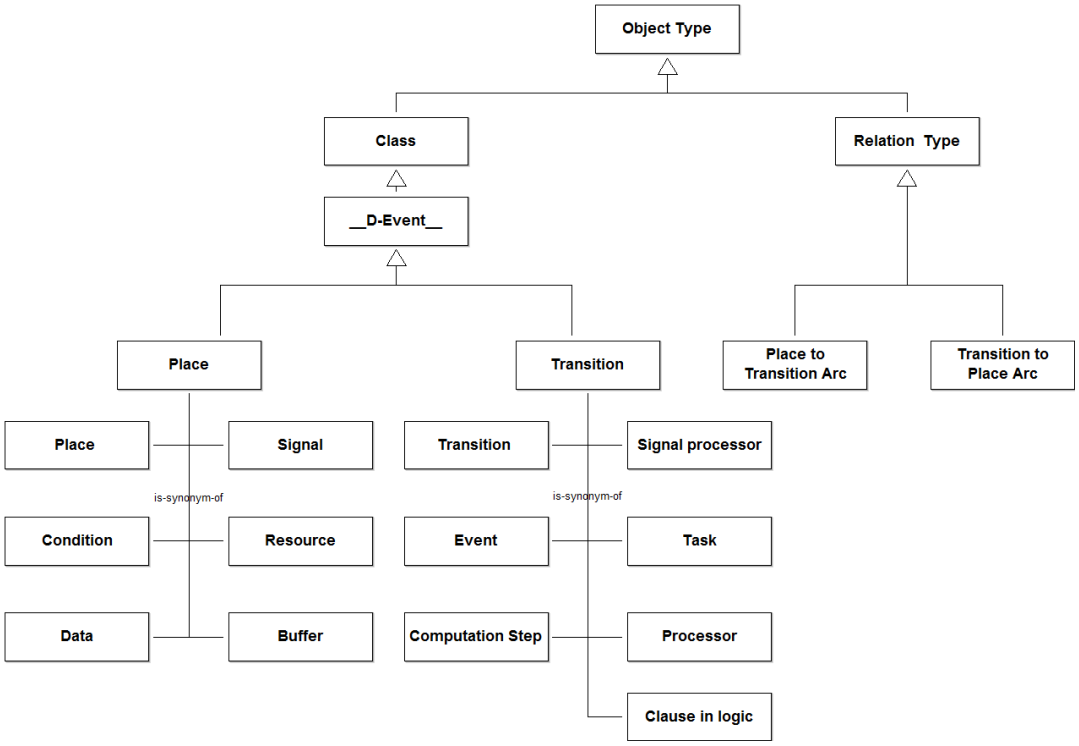


Figure 4-11 Object type Taxonomy of Petri Nets extended with Synonyms of Place and Transition

4.3.4 Petri Nets Ontology

The Figure 4-12 depicts a Petri Net Ontology proposed by (Gašević and Devedžić 2006) to be used for formal description of the Petri Net semantic. This Petri Nets ontology is built with considering different concepts from many Petri Nets syntax formats (e.g PNML, DaNaMiCS) in order to support different Petri Net dialects.

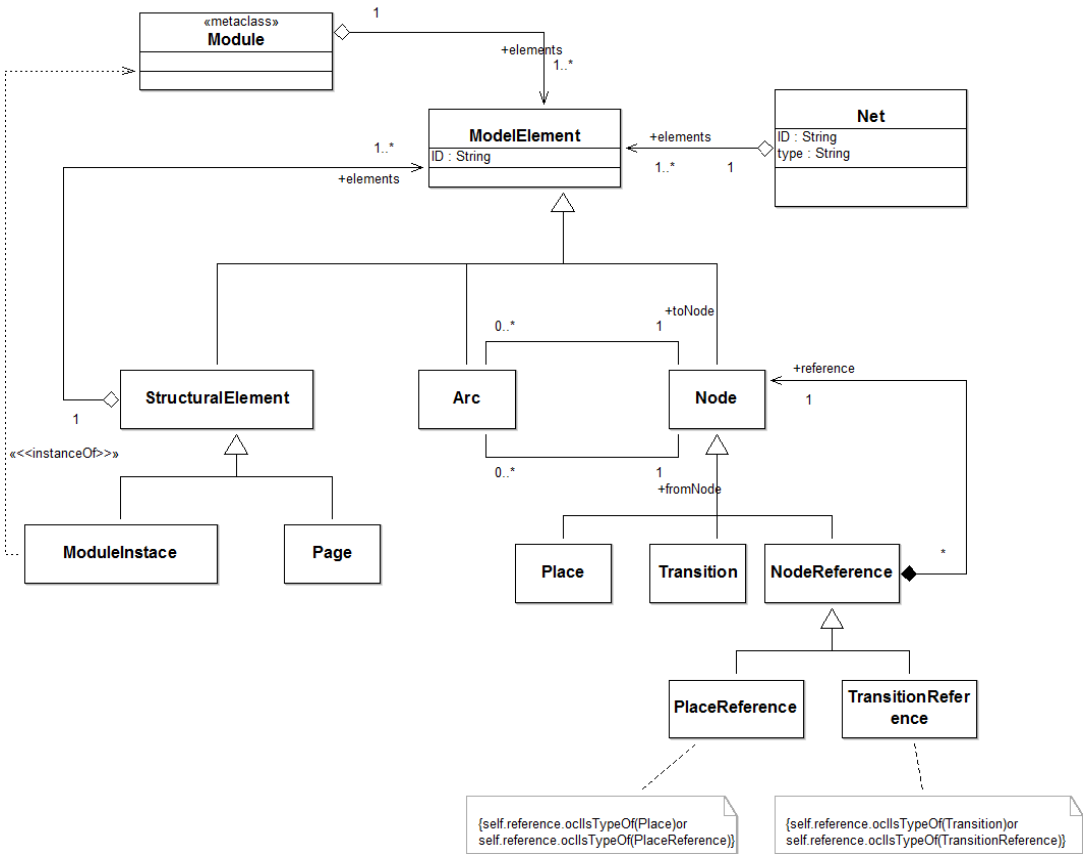


Figure 4-12 Petri Nets (Core) Ontology (Gašević and Devedžić 2006)

In this petri nets ontology there is only one root element called “ModelElement”, which is the parent of all elements of a Petri Nets structure. The Net represent the aggregation of the ModelElements. Like the ModelElement, the Net has a unique identifier (ID) attribute and also attribute type, which describes the type of the Petri Nets. And also three main element of Petri Nets; Place, Transition and Arc have been represented with their own classes as specialization of the Node. The StructuralElement,which is the inherited from class ModelElements, represents structuring mechanism. Just two structuring mechanism are supported by this ontology; Page, which may consist of other Petri Nets ModelElements, and the Module. A NodeReference represents the appearance of a node, it can be either PlaceReference or TransitionReference. The two constraints which state that a PlaceReference can refer to either a Place or another PlaceReference and a TransitionReference may refer to either to a Transition or another TransitionReference.

However, since our scope the modelling languages and mechanisms, which are implemented within the ADOxx® environment, we need to consider concepts from ADOxx® to be able to enrich mechanism & algorithm descriptions with the correct concepts. For that concern, we have adapted Petri Nets Ontology proposed by (Gašević and Devedžić 2006) with the concepts we have defined during the investigation on taxonomy for mechanism description. The Figure 4-13 depicts adapted Petri Nets Ontology. As aforementioned we have utilized the taxonomies which are identified previously in this work and the Petri Nets Ontology (Gasevic

2006) and which can be found written with using OWL and easy to investigate using ontology editor Protégé[6].

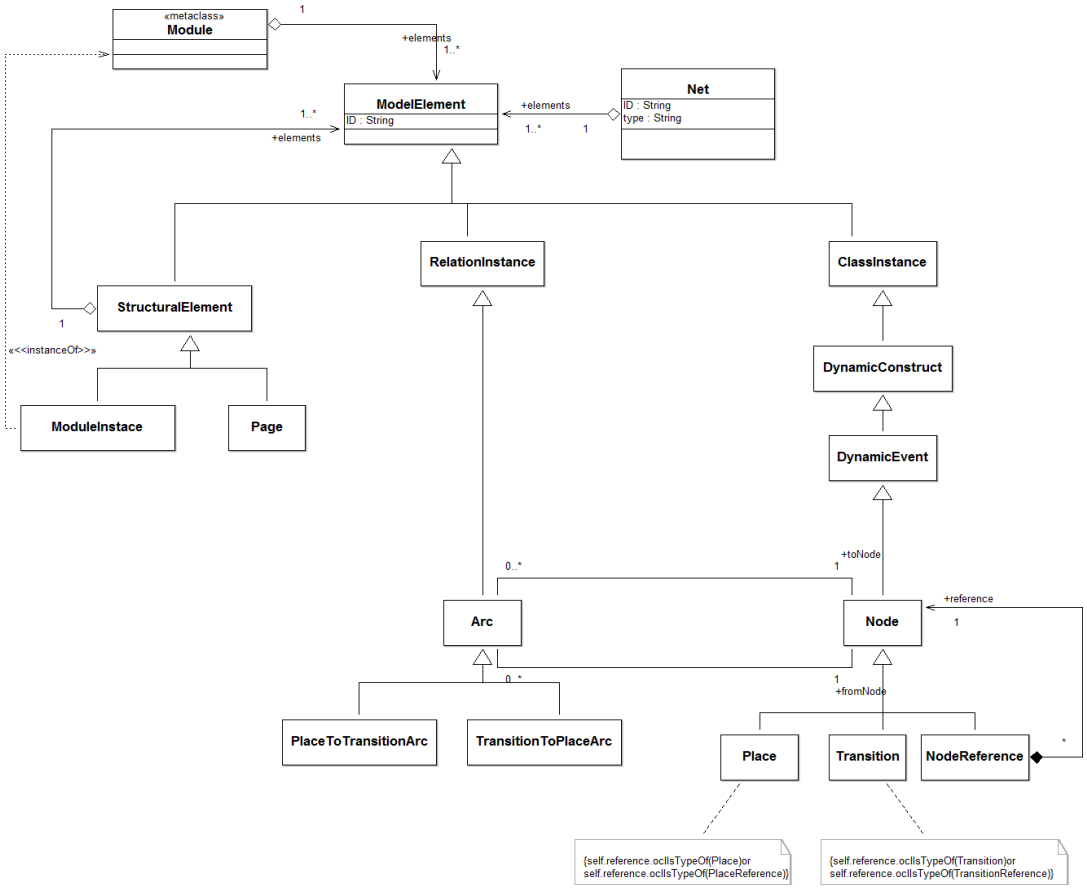


Figure 4-13 Petri Nets Ontology adapted with ADOxx® Concepts

[6] Protégé is a free, open-source platform to construct domain models and knowledge-based applications with ontologies (Stanford Center for Biomedical Informatics Research 2013)

5 Realization of Matching Mechanism

In this chapter, we introduce realization of the matching mechanism. First of all we specify and introduce the modelling language "Semantic enriched Mechanism Description Modelling Language", which offers model-driven mechanism & algorithm description enriched semantically. Based on the modelling language a modelling editor on ADOxx® implemented, called SeMD Modelling Editor offers to describe mechanisms & algorithms with conceptual models graphically and make possible to utilize ADOxx® Model XML Export mechanism in order to export models as XML files complying with ADOxx® Model XML Schema, which is known XML schema by the Matching Engine, and enables transferring the XML files into the file system of the Matching Engine. The chapter also investigates an appropriate approach to calculate the semantic similarity between concepts from modelling languages and the mechanisms & algorithms. For that concern, vector space models are investigated, which is introduced in (G. Salton 1971) and vector based ontology matching approach proposed by (Eidoon, Yazdani and Oroumchian 2008) is utilized. As mentioned before the other component of the Matching Mechanism, which component consists of main functionalities, is the Matching Engine, which queries - with utilizing XPATH queries - the modelling language and description of mechanisms & algorithms, prove the concepts from both sides,with the respect to utilized ontology to close semantic gap between them, if they match and calculates semantic similarity of between concepts.

5.1 Semantic Enriched Mechanism and Algorithm Description Language

According to (Fill, Schremser and Karagiannis 2013) the syntactical concepts can be annotated with the concepts in semantic schemata for the purpose of raising semantic expressiveness in conceptual model.

Again (Fill, Schremser and Karagiannis 2013) listed approaches to add semantic annotation as following;

- Adding semantic annotations to the abstract syntax or concrete syntax definition of modelling language
- Adding reference attributes to abstract syntax or concrete syntax definition
- Using a separate annotation specification does not require to modify any of modelling or ontology languages like exposed in (Kiryakov, et al. 2003) (Abramowicz and Agata Filipowska 2007).

Also, the other approaches are semantic lifting approaches introduced in (Hrgovcic, Karagiannis and Woitsch 2013) and applied in practice a virtual enterprise management tool (Efendioglu, Woitsch, et al. 2013) in the BIVEE Project partly funded by the European Commission.

In this stage, we have to emphasize that the purpose of this work is not to raise semantic expressiveness of the modelling languages, but of the description of the mechanism, since we

do not want to force users to modify their modelling languages in order to add semantic annotations. However, those three approached listed above and state of the art analysis in semantic description of services guide us how should we develop a description language and related matching engine which does not only return results due to syntactically matching words in searching query but, also due to possibly related concepts by means of the semantic enrichment of concepts used in the definition of mechanism, especially the concepts that represent requirements of mechanism, that they need to be able to work properly.

For that concern, we require; (1) a description language which can express the capabilities and requirements of the mechanism semantically and (2) a matching engine which is able to interpret this description language and supports semantic matching between mechanism advertisements and mechanism requests.

This work answers these two challenges with (1) a modelling language consists of two model type "Mechanism and Algorithm Description Model Type" and "OWL Model Type" and (2) with a Matching Engine

After requirements and design principles exposed in introductory scenarios and state of the art analyse, we have learn patterns how to describe the mechanism. As static structure depicted in Figure 5-1 in our semantic enriched mechanism description approach ten constitution are used to describe a mechanism; (1)

1. The *Mechanism* representing mechanism itself which has attributes specifying name of mechanism (mechanismName) list of inputs (inputs), list of outputs(outputs) function group(functionGroup) into that mechanism is classified in terms of its functionalities type of mechanism(mechanismType) due to its dependency to modelling technique and dependencies of the mechanism to other mechanism in order to work properly.
2. The *Input* represents an input required by a mechanism. The *Input* has attributes specifying object type (objectType) from FDMM point of view (e.g. class "Place"), annotation of object type (ObjectTypeSemAnnotation) by referencing a *"Class"* concept from the relevant domain ontology (in this case Petri Nets Ontology), attribute type(attributeType) again from FDMM point of view (e.g. attribute "NumberOfTokens"), annotation of attribute type (AttributeTypeSemAnnotation) by referencing a DataProperty concept from relevant domain ontology, data type also from FDMM point of view (e.g. "Integer") which can have a data type listed in enumeration *ADOxxDataTypes*.
3. The *Output* has same attributes like the *Input*. The *Output* differentiate from *Input* by it is semantic meaning; the *Output* represents an output produced by a mechanism.
4. The *FunctionGroup* represents enumeration of function group classifying mechanisms due to their functionalities.
5. The *MechanismType* represents enumeration of mechanism type classifying mechanisms due to their dependencies to the modelling technique.
6. The ADOxxDataTypes represents enumeration of data types that are allowed by ADOxx® platform to assign an attribute.

7. The Class represents class concepts defined in ontologies complying specification defined in (Motik, et al. 2012) and investigate in section 4.2.2.
8. The ObjectProperty represents object property concepts defined in ontologies complying specification defined in (Motik, et al. 2012) and investigate in section 4.2.2.
9. The AnnotationProperty represents annotation properties ("label" is the most important annotation property for this work) concepts defined in ontologies complying specification defined in (Motik, et al. 2012) and investigate in section 4.2.2.
10. The DataProperty represents data properties concepts defined in ontologies complying specification defined in (Motik, et al. 2012) and investigate in section 4.2.2.

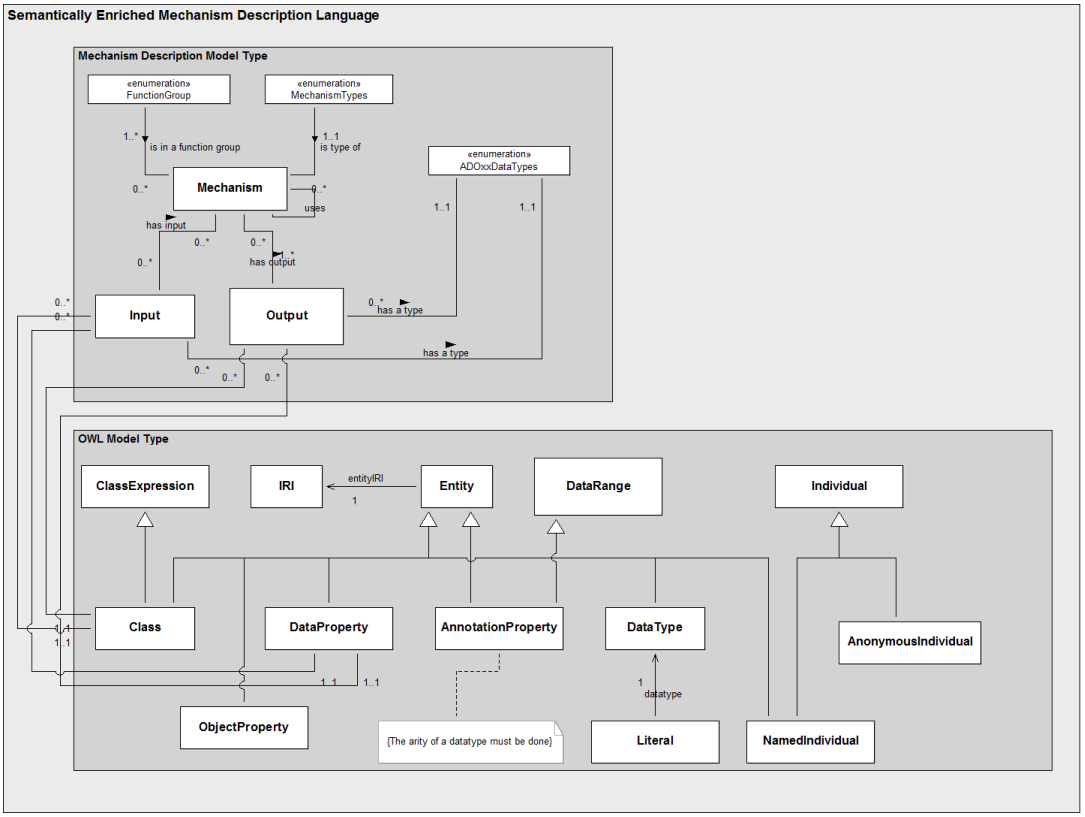


Figure 5-1 Static Structure of Semantically Enriched Mechanism Description Language

5.1.1 Model Type for Describing Mechanisms: Mechanism Description Model Type

As aforementioned, this work is constrained with mechanism and algorithm that have been implemented on ADOxx® and with following "Generic Modelling Method Framework". Therefore, it was enough to understand how static structure of mechanisms and algorithms are defined from "Generic Modelling Method Framework" point of view. For that concern, we have investigated in 2.1.3 the static structure of mechanism and algorithm described by (Kühn 2004) and meta-model of "Mechanism Definition Language" as depicted in Figure 2-7. The Figure 5-2 depicts the metamodel of "Mechanism Definition Model" which is

designed with taking in cognizance of constraint and requirements found out during the investigation in section 4.

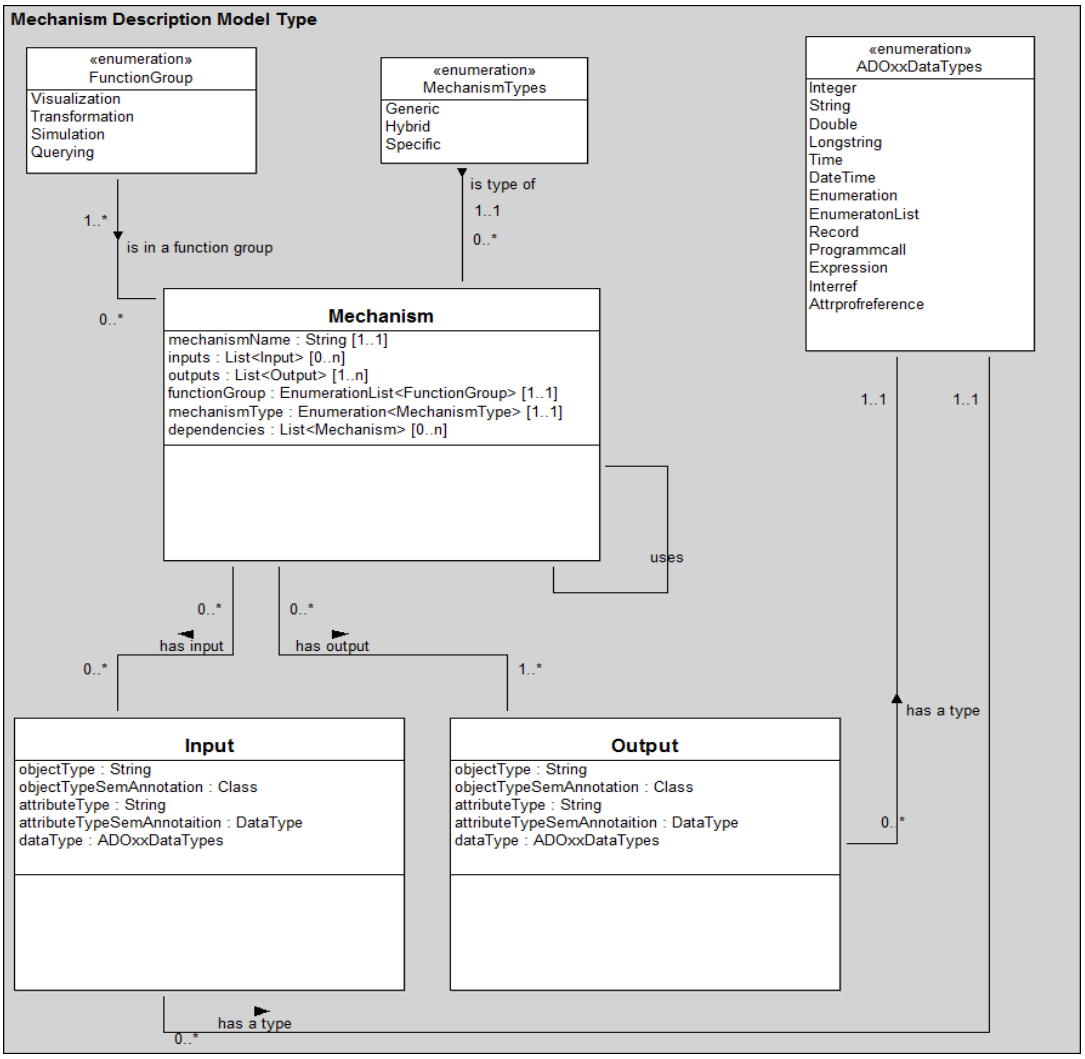


Figure 5-2 Metamodel of Mechanism Description Model Type

The Table 5-1 contains the ADOxx® specific specification of concepts of Mechanism Description Model Type, which are already defined in its metamodel (in Figure 5-2)

Notation	Mech_Algo : Class
	Mechanism represents the mechanism in the real word and includes the description of the mechanism
Attributes	
Attribute Name	**Type**	**Description**
Name	String	The name of the mechanism

Function Group	EnumerationLsit	It determines, to which function group(s) the mechanism belongs.
Mechanism Type	Enumeration	It determines, to which mechanism group the mechanism belongs.
Inputs	Record	It registers which inputs are required by the mechanism to be able work properly. An input consists of the name of property, from which the value is required, its class name, the type of property, besides those annotations of class and attribute.
Outputs	Record	It registers the outputs provided by mechanism in similar way as Inputs attribute.
Notation: N/A	Input:RecordClass	
Attributes		
Attribute Name	**Type**	**Description**
Class_Name	String	The name of class whose value of property is in question
Class_Annotation	Interref	It references the class concepts from OWL Model in order to annotated class.
Property_Name	String	The name of the class whose value is required.
Property_Annotation	Interref	It It references the DataProperty concepts from OWL Model in order to annotated property.
Type_of_Property	Enumeration	It determines the type of the property. The type of property could be one of the ADOxxDataTypes
Notation: N/A	Output:RecordClass	
Attributes		
Attribute Name	**Type**	**Description**
Class_Name	String	The name of class whose value of property is in question

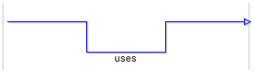
Class_Annotation	Interref	It references the class concepts from OWL Model in order to annotated class.
Property_Name	String	The name of class whose value is required.
Property_Annotation	Interref	It It references the DataProperty concepts from OWL Model in order to annotated property.
Type_of_Property	Enumeration	It determines the type of the property. The type of property could be one of the ADOxxDataTypes
Notation 	Has dependency : Relation Type	
	It connects the mechanism with the other mechanism(s), with which the mechanism has dependency in order to work properly.	
Attributes		
Attribute Name	**Type**	**Description**
fromClass	Endpoint	Mech_Algo
toClass	Endpoint	Mech_Algo

Table 5-1 Specification of Mechanism Description Model Type

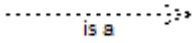
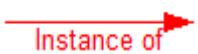
5.1.2 Model Type for Building Ontology: OWL Model Type

The OWL Model Type is adapted and implemented based on the OWL Application Library, which implemented according to specification of OWL2 and provided by ADOxx.org (ADOxx.org 2013).

The Table 5-2 contains the ADOxx® specific specification of concepts of OWL Model Type, which are relevant for this work. The detailed specification of the OWL Method can be found in (ADOxx.org 2013).

Notation 	Class : Class	
	Mechanism represents the mechanism in the real word and includes the description of the mechanism	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	The name of the class entity

Annotations	Record	It contains “label” annotations
Disjoints	Interref	It defines with which class(es) is the class disjoint (axiom disjoint)
SameAs	Interref	It defines with which class(es) is the class same (axiom SameAs)
DifferentFrom	Interref	It defines from which class(es) is the class different (axiom DifferenFrom)
Notation: N/A 	Object property : Class	
	It defines object properties	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	The name of property
SubProperty	Interref	It defines sub property of the property
Domain	Interref	It defines domain of the property (source class)
Range	Interref	It defines range of the property (target class)
SameAs	Interref	It defines as which property the property is same
DifferentFrom	Interref	It defines from which property the property is different
Annotations	Record	It contains “label” annotations
Notation: N/A 	Data property : Class	
	It defines data properties	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	The name of property
SubProperty	Interref	It defines sub property of the property
Type	Enumeration	Data type of property complying with XML datatypes

Domain	Interref	It defines domain of the property (source class)
Range	Interref	It defines range of the property (target class)
SameAs	Interref	It defines as which property the property is same
DifferentFrom	Interref	It defines from which property the property is different
Annotations	Record	It contains “label” annotations
Notation: N/A 	Is a : Relationclass	
	It defines relation between two class in order represent is-a axiom	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	The name of class whose value of property is in question
Type	Enumeration	It references the class concepts from OWL Model in order to annotated class.
Comment	String	
Notation: 	subClassOf : Relationclass	
	It defines relation between two class in order represent subClassOf axiom	
Attributes		
Attribute Name	**Type**	**Description**
Name	String	The name of class whose value of property is in question
Type	Enumeration	It references the class concepts from OWL Model in order to annotated class.
Comment	String	
Notation: 	Instance of : Relationclass	
	defines relation between the class and instance in order represent whose instance is the object	

Attributes		
Attribute Name	**Type**	**Description**
Name	String	The name of class whose value of property is in question

Table 5-2 Specification of OWL Model Type

The Figure 5-3 depicts SeMD Modelling Toolkit implemented according the ADOxx® specific specifications described above.

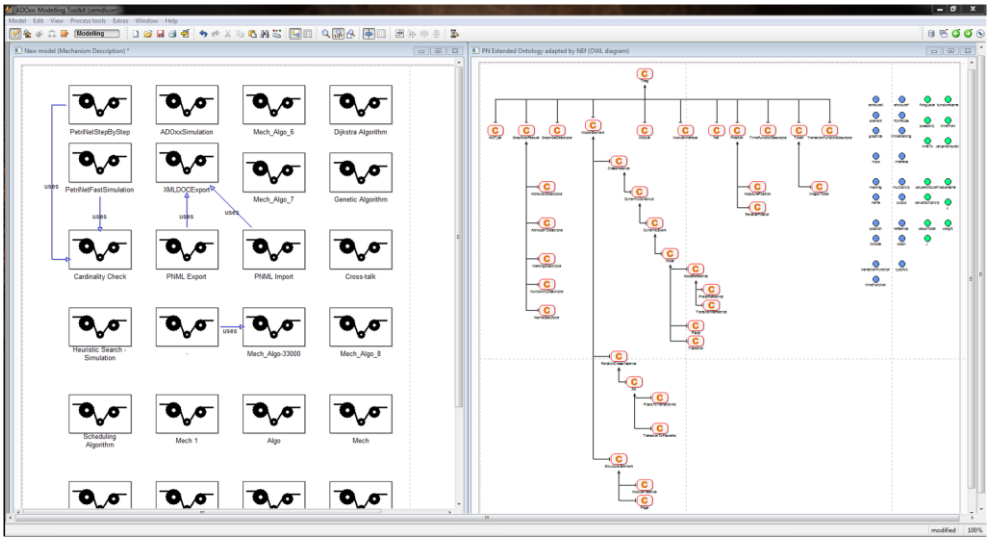


Figure 5-3 SeMD Modelling Toolkit

5.2 Similarity Calculation between Semantic Concepts

In order to calculate the similarity between semantic concepts consisted by a modelling language and concepts cosisted by a mechanism description, we utilize vector space models introduced in (G. Salton 1971) and investigate vector based ontology matching approach proposed by (Eidoon, Yazdani and Oroumchian 2008).

A vector space represents a set of vectors; a vector space model allows describing and comparing objects using N-dimensional vectors and each dimension subtends to an orthogonal feature of the object (Tous and Delgado 2006).

In order to use approaches based on vector similarity algorithms and measure similarity between ontologies, the first step is to model ontologies in vector notation and utilize a vector matching algorithm to evaluate the similarity degree between them. For that concern ontology vectorization method is applied, which is a method for modelling two ontologies in a single multidimensional vector space that any of its dimension represents a unique concept, property or the range of data type property by (Eidoon, Yazdani and Oroumchian 2008).

In our case, we can vectorize description of modelling languages and mechanisms. The match points, which we have identified previously, are represented in the vectors like;

- $MLV_i = (MT_i, domain, range) = \left(O_i^T, D_i^T, A_i^{'}, domain, range\right)$

In modelling language vector MLV_i all object types, datatypes, attributes and their domain and range functions are reppresented

- $MechV_j = (\ Inputs_j)$

Since, as discussed previously, the inputs from mechanism and object types, data types, attributes from modelling languages are the only match points between modelIing language and mechanism, it is enough to represent $Inputs_j$ in mechanism vector $MechV_j$.

Such examples for modelling language and mechanism vectors will be like following;

- $MLV_{EPN} = (Place, Transition, Tokens,\ Place2TransitionArc,$ $Transition2PlaceArc,\ String, Expression, Enumeration_{State}, Record_{Tokens},$ $Name, Tokens, State, Guard, Data\ Value, Data\ Type, Place2TransitionArc - from,\ Place2TransitionArc - to, Transition2PlaceArc - from,$ $Transition2PlaceArc - to,\ domain(Name), domain(Data\ Value),$ $domain(Tokens),\ domain(State), domain(Data\ Type),$ $domain(Transition2PlaceArc - to),\ domain(Transition2PlaceArc - from),$ $domain(Place2TransitionArc - to)\ ,\ domain(Place2TransitionArc -$ $from,\ domainWeight,\ domainGuard,\ rangeName,\ rangeData\ Value,$ $range(Data\ Type), range(Tokens), range(State), range(Guard),$ $range(Place2TransitionArc - from),)$
- $MechV_{FastSimulation} =$ $(Place, Transition, Place2TransitionArc, Transition2PlaceArc, Integer, Integer,$ $Integer,\ NumberOfTokens,\ Weight, Weight)$

Since concepts in mechanism description are annotated with using certain domain ontology- in this case Petri Nets ontology- this domain ontology is acting as pivot and vector of this ontology is going to be used for weight calculation. Weights of concepts and properties are calculated like introduced in by (Eidoon, Yazdani and Oroumchian 2008) and (Tous and Delgado 2006) which are using the approaches adapted from (Salton, Wong and Yang 1975).

Weights of concepts in concept vector are calculated as following;

- $W_c(X) = \begin{cases} \log\left(\frac{1}{d_x(c)+1}\right) & if\ d_x(c) > 0 \\ 1 & if\ d_x(c) = 0 \end{cases}$

Where $W_c(X)$ is the weight of the concept *c* in the concept vector X, -vector of pivot ontology- *and* $d_x(c)$ is the distance of concept X in sub/super class chain.

Weights of properties in property vector are calculated as following;

- $W_p(X) = \begin{cases} 1 & if\ p = x \\ 0 & else \end{cases}$

Where $W_p(X)$ is the weight of the property p in the property vector X.

After description of the modelling language and the mechanism are vectorized and weights of concepts and properties are calculated similarity between the modelling language and the mechanism can be calculated using cosine as following;

- $sim(MLV_i, MechV_j) = \frac{MLV_i \,.MechV_j}{||MLV||_i \,.||MechV_j||} = \frac{\sum_{k=1}^{N} W_{k,i} \,.W_{k,j}}{\sqrt{\sum_{k=1}^{N} W_{k,i}^2} \,.\sqrt{\sum_{k=1}^{N} W_{k,j}^2}}$

Since all concepts and properties in vector of mechanism are annotated with concepts and properties from pivot ontology (in this case Petri Nets Ontology), vector of mechanism acts as the pivot itself. Hence weights of concepts and properties will be always equal to "1". Then similarity between modelling language and mechanism can be calculated as following;

- $sim(MLV_i, MechV_j) = \frac{\sum_{k=1}^{N} W_{k,i} \,.1}{\sqrt{\sum_{k=1}^{N} W_{k,i}^2} \,.\sqrt{N.1}} = \frac{\sum_{k=1}^{N} W_{k,i}}{\sqrt{\sum_{k=1}^{N} W_{k,i}^2} \,.\sqrt{N.1}}$

5.3 Matching Mechanism

The Figure 5-4 describes the top level use cases explains how the matching mechanism works. The use cases occur within the boundaries of two different system namely

1. ***Semanticaly enriched Mechanism Description Modelling Toolkit (SeMD MTK***), where the semantic description of mechanisms in a form of conceptual models is modelled.
2. ***Matching Engine***, is set of methods, which are able to interpret description of modelling languages implemented on ADOxx® and description of mechanisms models modelled on SeMD MTK, calculate semantic similarity and in conclusion matches concepts in modelling language and in description of mechanisms

Three different actor acts in the use cases

1. ***Method Engineer***, who conceptualize the modelling method and requires the support of the matching mechanism in order to find matching mechanism to the modelling language specified and described by him.
2. ***Software Engineer***, who designed the mechanism(s), eventually implemented and describes his mechanism(s)on SeMD MTK in order to advertise to the method engineers
3. ***Ontology builder***, who is the domain expert and builds ontology and open to use by modelling it on SeMD MTK

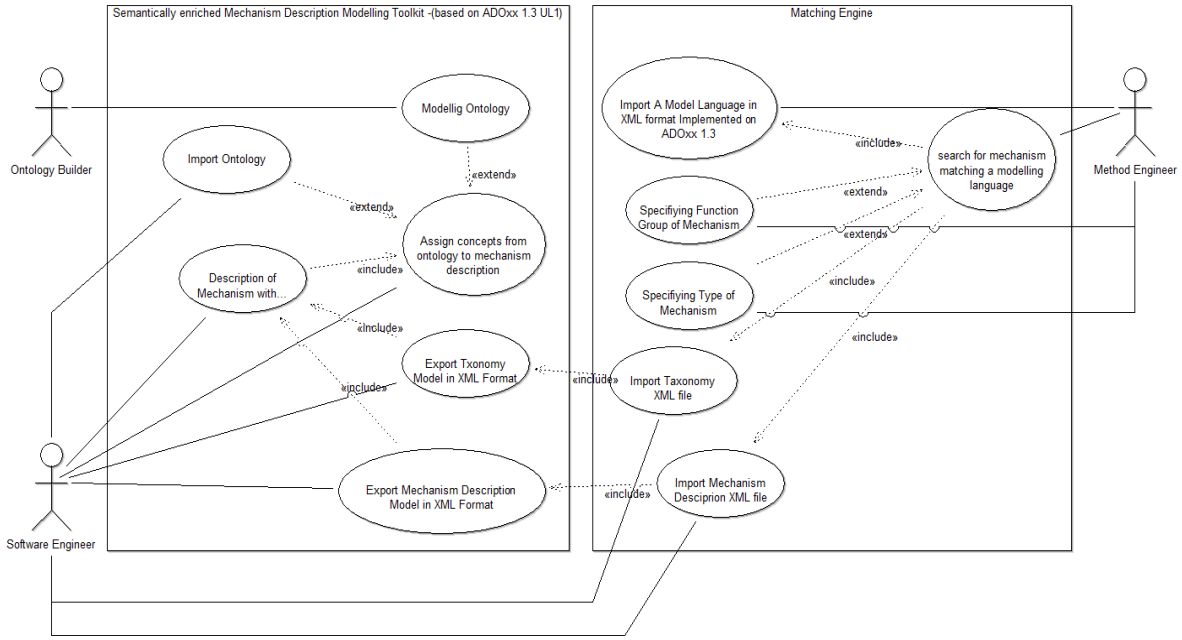


Figure 5-4 Use Case Diagram of Modelling Language and Mechanism Matching Mechanism

The narrative of the use case scenario is as following;

Software engineer(s) registers their mechanisms by describing them on the SemD MTK. In order to enriches description of mechanism semantically software engineers annotates the descriptions of mechanisms with using concepts from related domain ontology, which is modelled by an Ontology Builder, or modelled on another OWL Model Editor, and imported from this external model editor. And at the end Software engineers make their descriptions reachable with exporting pool model, which contains semantic enriched mechanism descriptions, as a XML file into file system of the system "Matching Engine"

Ontology builder(s**)** builds domain ontologies with modelling in OWL syntax using SeMD MTK and they make them available for use of systems SeMD MTK and Matching Engine. In order to make the ontologies available for Matching Engine, Software engineer(s) needs to export the ontology that he used for annotation of the mechanism description, into the file system of the matching engine.

Method engineer uploads her/his modelling language that has been implemented on ADOxx®, as XML file into the file system of the Matching Engine. Afterwards, the Method Engineer specifies function group and type of mechanism that she/he searching for, and starts Matching Engine. Matching Engine searches for the concepts required by the mechanism in the modelling language, calculates the similarity and prepares a report consists of the semantic similarity results of the concepts required by the mechanism with the concepts in the modelling language moreover list of matching concepts and mismatching concepts and to what extent they are matching.

5.3.1 Matching Algorithm

The Algorithm 5-1 describes the matching algorithm, on which the Matching Engine is based. According to matching algorithm, algorithm parses the request and retrieves all mechanisms descriptions and first of all checks which mechanism description complying with mechanism type and function group defined in the request and records them as a candidate mechanism. For each candidate mechanism gets the required input classes from the mechanism description, for each class gets its label annotations then search for the class in modelling language with using its label annotations if a class in modelling language has been found complying with any label annotation of the class, class and what extent it matches will be added to compatibility record. If no class could be found complying with the labels algorithm searches for subclasses of the required class in the ontology, if there is any subclass of the class algorithm retrieves the label annotation of this class and starts again the same search process. If no class could be found, then this class recorded in incompatibility record. If any class has been found, the properties of the class required by mechanism are retrieved and for each property its label annotations will be retrieved and then starts searching for the properties in the class found in the modelling language complying with property label annotations. If no property in could be found required property will be recorded in incompatibility record. If the property has been found complying with its labels, it will be recorded in compatibility record. And then data type of property is with the required data type, if it matches, it will be recorded in compatibility record, else in incompatibility record.

Moreover, algorithm calculates for each class and property their similarity degree corresponding if they or –if applicable- their sub concepts are existing in the given ontology or not.

Matching Algorithm

```
Match (mechanismRequest)

      CandidateMechanismsRecord;
      IncompatibilityRecord;
      CompatibilityRecord;
      SimilaritDegreeRecord;

      Foreach Mechanism do
        If (requested mechanism type and function groups match with the
            mechanism)
         CandidateMechanismRecord.add(mechanism)
      End

      Foreach CandidateMechansim in CandidateMechanismRecord do

        Foreeach RequiredInputClasses mechanism do
         Get labels from annotation
         Foreach label do
          If (any class or any relationclass name is equal to label)
          Foreach required RequiredInputClasses.property do
           Get labels from annotation
           Foreach label do
            If (any property name from given modeling language class is
                equal to the label)
                If (type of property is equal to required input class
                    property)
```

Matching Algorithm

```
                    CompatibilityRecord.add(request.modellingLanguage
                                                .class.property.type)

                Else
                    IncompatibilityList.add(request.modellingLanguage
                                                .class.property.type)
             Else
                    IncompatibilityRecord.add(request.modellingLanugage
                                                .class.property)
            End
            PropertySimilarityDegree=calculateSimilarity(request
                        .modellingLanugage.class.property,
                                                 RequiredInputProperty)
            SimilartiyDegree.update(PropertySimilarityDegree)

          End
         ClassSimilarityDegree=calculateSimilarity(request
                        .modellingLanugage.class, RequiredInputClasses)
        SimilartiyDegree.update(ClassSimilarityDegree)
       Else
        IncompatibilityRecord.add(request.modellingLanugage.class)
      End
      SimilarityDegreeRecord.add(request.modellingLanugage.class)
   End

Return(IncompatibilityRecord,CompatibilityRecord,SimilarityDegreeRecord)
```

Algorithm 5-1 Matching Algorithm

5.3.2 Matching Engine

The Matching Mechanism is the second and main component of the Matching Mechanism The Matching Engine consists of functionalities which query concepts from the modelling language and description of mechanisms & algorithms, prove the concepts from both sides, with the respect to utilized ontology to close semantic gap between them, if they match and calculates semantic similarity of between concepts.

As a result, the Matching Engine provides a log exposes matching and not matching concepts and to what extent they are matching, moreover calculated semantic similarity between the concepts. For that all, the chapter introduces the architecture of the Matching Mechanism.

The Matching Engine accepts modelling languages, description of mechanisms & algorithms and the ontology models as inputs as long as they are XML files, which has the ADOxx® Model XML schema.

The Matching Engine parses the modelling languages, which are implemented on ADOxx® and exported as XML from ADOxx®, SeMD Models (Mechanism Description Models and

OWL Models) exported as XML with using SeMD Modelling Editor using Document Object Model (DOM) (Le Hors, et al. 2004).

The Matching Engine queries the modelling languages and SeMD Models in order match concepts with building the queries with utilizing the XML Path Language (XPATH) (Clark and DeRose 1999).

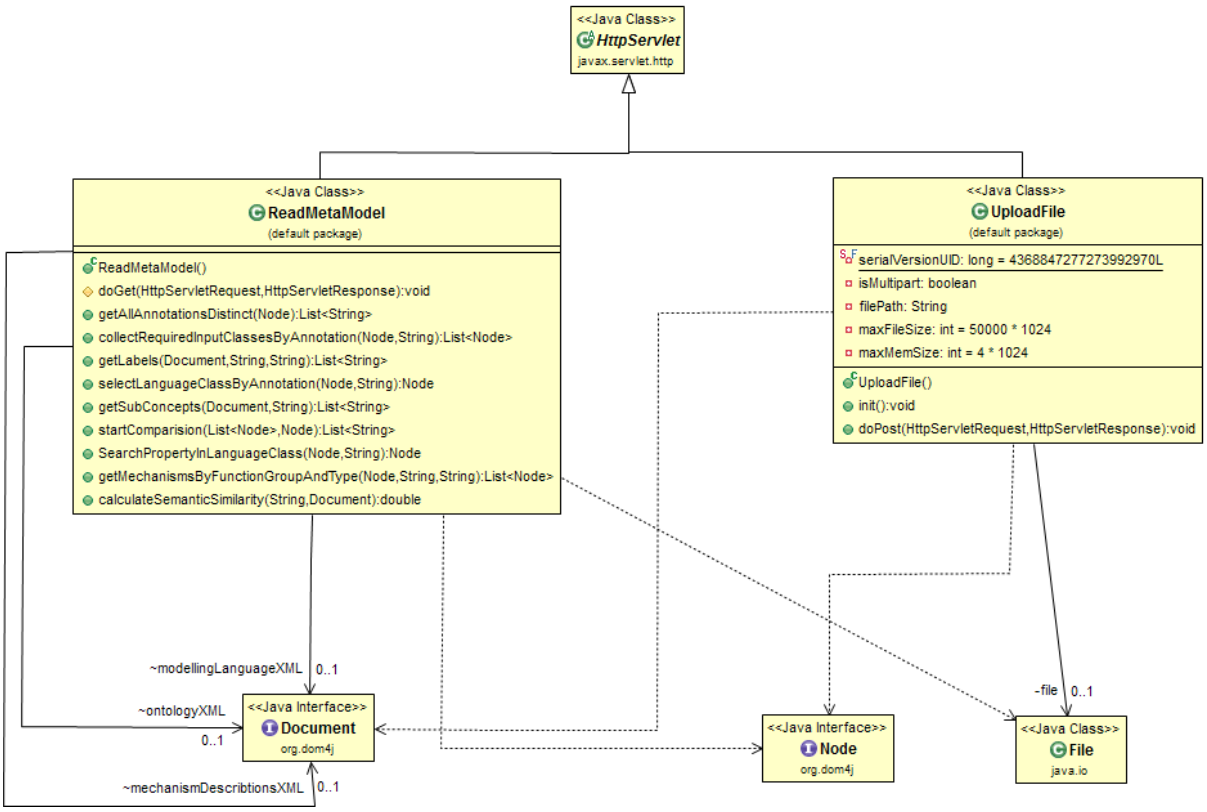


Figure 5-5 Matching Engine Class Diagram

As depicted in Figure 5-5 the Matching Engine consists of two java servlets. The UploadFile servlet is utilized just to upload modelling language description, mechanism & algorithms and ontology as XML files. The main functionalities are provided by the servlet ReadMetaModel, which are described below with giving code snippets.

The Code Snippet 5-1 represents the implementation of the function, which searches and retrieves candidate mechanisms described in the XML, that contains mechanism descriptions, complying with given function group and mechanism type. For that process XPATH queries are utilized which select all INSTANCE nodes in the XML file, whose Function Group Attribute equals to the given function and Type Attribute equals to the given mechanism type.

Pre-selection of Candidate Mechanism&Algorithms

```
public List <Node> getMechanismsByFunctionGroupAndType (Node node, String
functionGroup, String type)
{
	List <Node> selectedMechanisms = null;
```

Pre-selection of Candidate Mechanism&Algorithms

```
        if(!functionGroup.equals("Any")  && !type.equals("Any"))
        {
           selectedMechanisms = node.selectNodes(

           ".//INSTANCE[ATTRIBUTE[@name='Function Group']= '"+functionGroup+"' and
           ATTRIBUTE[@name='Type']='"+type+"']");
        }
        if(functionGroup.equals("Any")  && !type.equals("Any"))
        {
          selectedMechanisms =    node.selectNodes(
          ".//INSTANCE[ATTRIBUTE[@name='Type']='"+type+"']");
        }

        if(!functionGroup.equals("Any")  && type.equals("Any"))
        {
           selectedMechanisms = node.selectNodes(
           ".//INSTANCE[ATTRIBUTE[@name='Function Group']='"+functionGroup+"']");
        }

        if (functionGroup.equals("Any")  && type.equals("Any"))
        {
           selectedMechanisms = node.selectNodes(".//INSTANCE");
        }

        return selectedMechanisms;

}
```

Code Snippet 5-1 Pre-selection of Candidate Mechanism & Algorithms according to their Function Group and Type

The Code Snippet 5-2 represents the implementation of the function which retrieves the concepts from ontology used to annotate required input classes. For that process, an XPATH query is utilized which selects all toobjname attributes from IREF node from INTERREF with name attribute Class_Annotation in the XML file, and then retrieves and lists the values of toobjname attributes.

Get All Annotation used to annotate required input classes

```
public List<String> getAllAnnotationsDistinct (Node node)
{
    List <Node> annotationNodes =  node.selectNodes(
    ".//RECORD/ROW/INTERREF[@name='Class_Annotation']/IREF/@tobjname");
    List <String> annotations = new ArrayList <String> ();
    for (Node tempAnnotionNode : annotationNodes)
    {
       String tempAnnotation = (String) tempAnnotionNode
                                        .selectObject("string(.)");
       if (!annotations.contains(tempAnnotation))
       {
          annotations.add(tempAnnotation);
       }
```

Get All Annotation used to annotate required input classes

```
 }
        return annotations;

  }
```

Code Snippet 5-2 Get all annotations distinctly used to annotate required input classes required for each mechanism & algorithm

The Code Snippet 5-3 represents the implementation of the function, which retrieves all required input classes by the mechanism due to annotation. For that process, an XPATH query is utilized, which selects ROW nodes, wherein its successor IREF node has attribute "tobjname" with the value complying the given annotation and returns the list of class nodes.

Collect required input classes from modelling language by annotation

```
public List<Node> collectRequiredInputClassesByAnnotation(Node node, String
annotation)
{
   List <Node> listOfInputClassNodes =  node.selectNodes(
   ".//RECORD[@name='Input']/ROW/INTERREF[@name='Class_Annotation']/
   IREF[@tobjname='"+annotation+"']/ancestor::ROW");

    return listOfInputClassNodes;
  }
```

Code Snippet 5-3 Collection of required input class nodes from modelling language by annotation

The Code Snippet 5-4 represents the implementation of the function, which retrieves the class or relation class from modelling language by annotation. In that process first of all labels of the concepts- used for given annotation- are retrieved from ontology and than an query is utilized, which selects the class or relation class which has same name as the label of the concepts. If there is no class or relation class, it will turn back to the ontology and will find the sub-concepts of the concept that used for the given annotation. Then it will search again for a class or relation class has the same name as label of "subconcepts".

Selection of Class Modelling Languages by Annotation

```
public Node selectLanguageClassByAnnotation (Node node, String annotation)
{
    List<String> labels = getLabels (ontologyXML, annotation, "Class");
    Node languageClassNode = null;
    for(String label:labels)
    {
      //it is assumed that the language has classes(model classes and relation
      classes) with unique names00
       Node tempClassNode = node.selectSingleNode(".//class[@name='"+label+"'] |
                                          .//relationclass[@name='"+label+"']");
         if(tempClassNode != null)
         {
            languageClassNode = tempClassNode;
```

Selection of Class Modelling Languages by Annotation

```
                break;
            }
        }
        if (languageClassNode == null)
        {
           List<String> subconcepts = getSubConcepts(ontologyXML, annotation);
           for (String subconcept:subconcepts)
           {
             Node tempClassNode =   node.selectSingleNode(
                                   ".//class[@name='"+subconcept+"']|
                                    .//relationclass[@name='"+subconcept+"']");

              if(tempClassNode != null)
              {
                 languageClassNode = tempClassNode;
                 break;
              }
            }
         }
    return languageClassNode;
}
```

Code Snippet 5-4 Selection of the Class in Modelling Language by Annotation

The Code Snippet 5-5 represents the implementation of the function, which retrieves the labels of the concepts from the ontology. For that process, an XPATH query is utilized, which retrieves the value of the attribute "Label" of the node INSTANCE in the ontology XML file.

Get Labels from Semantic Concept

```
public List <String> getLabels (Document ontology, String annotation, String
type)
{
  List <String> listOfLabels = new ArrayList<String>();
  List<Node> classInstances = ontology.selectNodes(
                          ".//INSTANCE[@class='"+type+"'][@name='"+annotation+"']
                      /RECORD[@name='Annotations']/ROW/ATTRIBUTE[@name='Label']");
   for (int i=0; i < classInstances.size(); ++i)
 {
   listOfLabels.add((String)classInstances.get(i).selectObject(
                                                  "string(.)"));
 }
 return listOfLabels;
}
```

Code Snippet 5-5 Get Labels of Semantic Concept

The Code Snippet 5-6 represents the implementation of the function, which retrieves the sub-concepts of a concept from the ontology. For that process, an XPATH query is utilized, which retrieves instances connected with given concepts with the CONNECTOR "subClassOf".

Get Sub-concepts

```
public List <String> getSubConcepts (Document ontology, String annotation)
{
  List <String> listSubConcepts = new ArrayList<String>();
  List <Node> SubConcepts = ontology.selectNodes(
                ".//CONNECTOR[@class='subClassOf']/TO[@instance='"+annotation+"']
                 /preceding-sibling::FROM/@instance");
  for (int i=0; i < SubConcepts.size(); ++i)
  {
    listSubConcepts.add((String) SubConcepts.get(i).selectObject("string(.)"));
   }
  return listSubConcepts;
}
```

Code Snippet 5-6 Get Sub-concepts of classes

The Code Snippet 5-7 represents the implementation of the function, which retrieves the required property. For that process first of all labels of data property, which used as given annotation, from ontology are retrieved, then an XPATH query is utilized, which retrieves the property from modelling languages that has the name complying with one of labels found previously.

Search Required Property in Class in Modelling Language

```
public Node SearchPropertyInLanguageClass (Node lCNode, String
                                                    Property_Annotation)
{
  Node propertyNode = null;
  List<String> labels = getLabels (ontologyXML, Property_Annotation,"Datatype
                                                              property");
  for (String label:labels)
  {
    propertyNode =  lCNode.selectSingleNode(
                                ".//attributes/attribute[@name='"+label+"']");

    if(propertyNode != null) break;
  }
      return propertyNode;
}
```

Code Snippet 5-7 Search Required Property in Class in Modelling Language

The Code Snippet 5-8 represents the implementation of the function that calculates semantic similarity of given concepts. As explained in the section 5.2, the function builds a vector from ontology and then calculates correlation by searching and weighing correspond to if the concept and its sub-concepts are existing in the ontology vector.

Calculation of Semantic Similarity

```
public double calculateSemanticSimilarity(String concept, Document ontology)
{
      double similarity = 0;
      double sumWeights = 0;
      double sumWeightsSquare = 0;
```

Calculation of Semantic Similarity

```
        List<Double> weights = new ArrayList<Double>();
        List <Node> tempOntologyClasses =
         ontology.selectNodes(".//INSTANCE[@class='Class']");
        List <String> ontologyVector = new ArrayList<String>();
        for (Node tempOnologyClass:tempOntologyClasses)
        {

           ontologyVector.add((String)tempOnologyClass.
           selectObject("string(./@name)"));
        }


        List <String> subConcepts = getSubConcepts(ontology, concept);


        if(ontologyVector.contains(concept))
        {
              weights.add(1.0);
        }
        else weights.add(0.0);

        if (subConcepts != null)
        {
              for (String subConcept:subConcepts)
              {
                    if(ontologyVector.contains(subConcept))
                    {
                          weights.add((Math.log10(0.5)));
                    }
                    else weights.add(0.0);
              }
        }

        for(double weight:weights)
        {
              sumWeights += weight;
              sumWeightsSquare += (Math.pow(weight, 2.0));
        }

        similarity =    sumWeights/((Math.sqrt(sumWeightsSquare))
                                    *(Math.sqrt(weights.size())));

        return similarity;

}
```

Code Snippet 5-8 Calculation of Semantic Similarity

5.3.3 Architecture of Matching Mechanism

The matching mechanism is composed of two separated system (1) SeMD Modelling Toolkit, which is implemented on ADOxx® Meta Modelling Platform and (2) the Matching Engine, which is implemented as a web service with using java-servlets and jsp technologies. The

architectures of both systems have well-known three-tier architecture; (1) Model Interpretation and Persistence Tier, (2) Functional Tier and (3) User Interaction Tier.

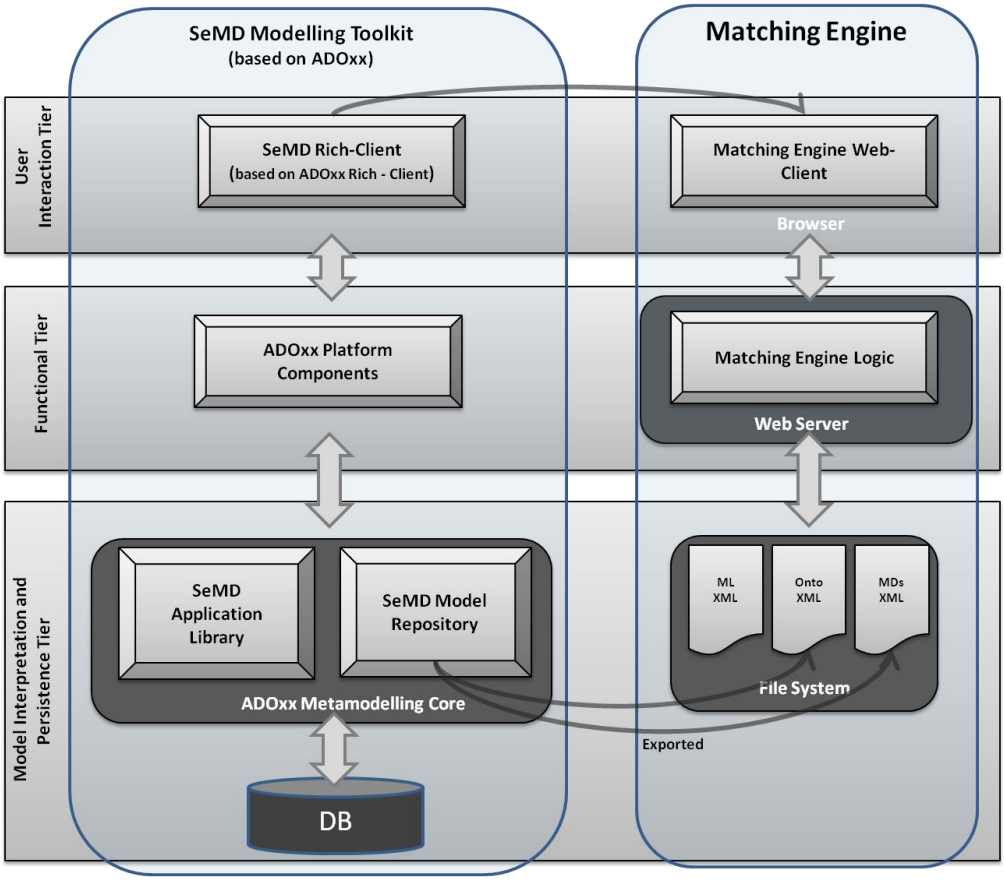


Figure 5-6 The High-level Architecture of Modelling Language and Mechanism Matching Mechanism

The User Interaction Tier provides a graphical user interface and necessary interaction logic. On the side of the system "SeMD Modelling Toolkit" a rich client based on rich client framework and functionalities provided by ADOxx®. On the side of the system "Matching Engine", since it is offered as a web service, an web client is available.– The Web Client is implemented with using technologies "Java-Servlet", "jsp", "HTML" and "CSS. Moreover it is possible to call the web-service from SeMD through the external coupling with using AdoScript

The Functional Tier concerns with the business logic of the systems. On the side of the system "SeMD Modelling Toolkit", business logic is composed of only logic provided by ADOxx® Platform Components, like the components "Modelling" and "Import/Export" utilized in our scenario. On the side of the system "Matching Engine", business logic is implemented using technologies "Java", "XPATH"

Model Interpretation and Persistance Tier concerns on the system "SemD Modelling Toolkit" side definition of the data structure and its interpretation, via semantic specification in the form of SeMD Application Library, on the system "Matching Engine" side tier deals with persistency of the data in form of XML files.

6 Prototype: OMI Service

In this section, we introduce the prototype implemented for the concern to prove the approach proposed by this work. For that concern, the first of all introduces a procedure for the utilization of the Matching Mechanism.

(1) Ontology builder models ontology with using OWL Model Type or software engineers exports existing one or Ontology builder and software engineer exports and adapts existing one. As outlined in the Figure 6-1, in our test case we have adapted Petri Nets Ontology proposed by (Gašević and Devedžić 2006) with ADOxx® Specific Petri Nets Taxonomy described in the section 4.3.3. (For the detailed view of individual OWL Models please refer to Appendix).

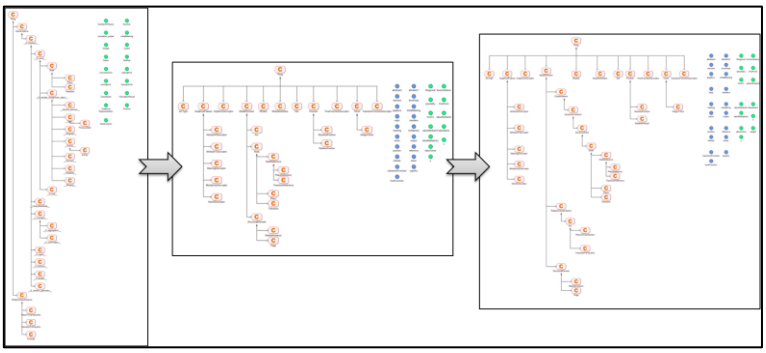


Figure 6-1 Adapted Petri Nets Ontology with excepted Concepts from ADOxx® Specific Petri Net Ontology and Extended Petri Net Ontology

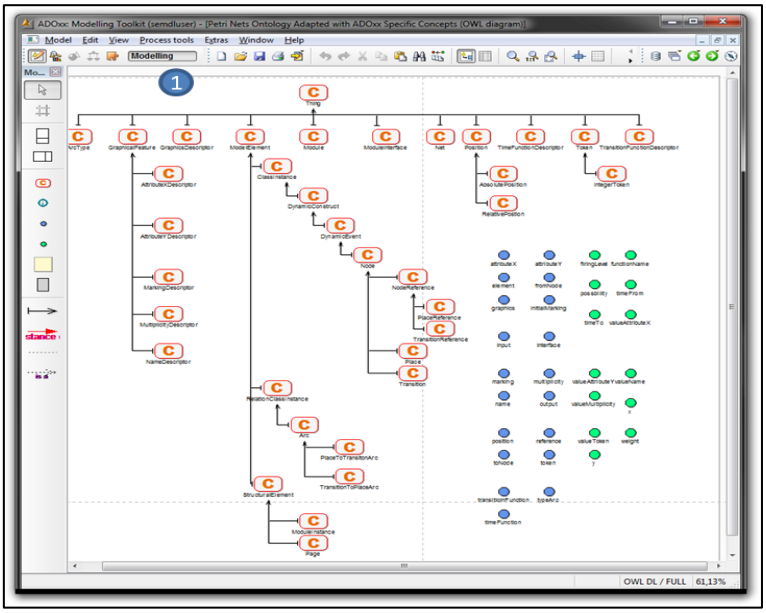


Figure 6-2 Petri Nets Ontology Adapted with ADOxx® Specific Concepts

(2) Afterwards, software engineer starts with the description of mechanisms & algorithms by specifying its name as depicted in Figure 6-3.

(3) Then specifies function group of mechanism
(4) Specifies mechanism & algorithm type
(5) Specifies required input of mechanism & algorithm by specifying for each input value
    a. source class,
    b. attribute and
    c. expected type of value
(6) Specifies outputs of mechanism with following same sub-steps like in step 5.

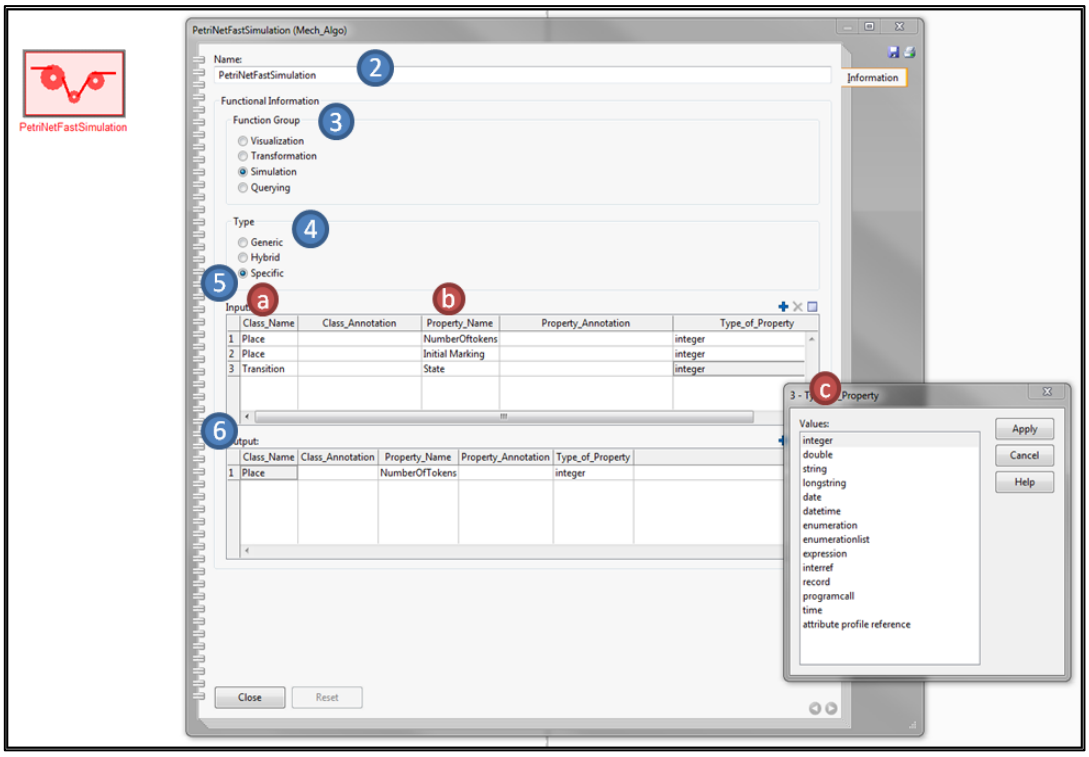


Figure 6-3 Specification of a Mechanism

(7) Annotates source class with using extension attribute “Class_Annotation” and with referencing a class concept from the Ontology previously modelled as depicted in Figure 6-4.
(8) Annotates property with using extension attribute “Property_Annotation” and referencing a data object property from the Ontology previously modelled.

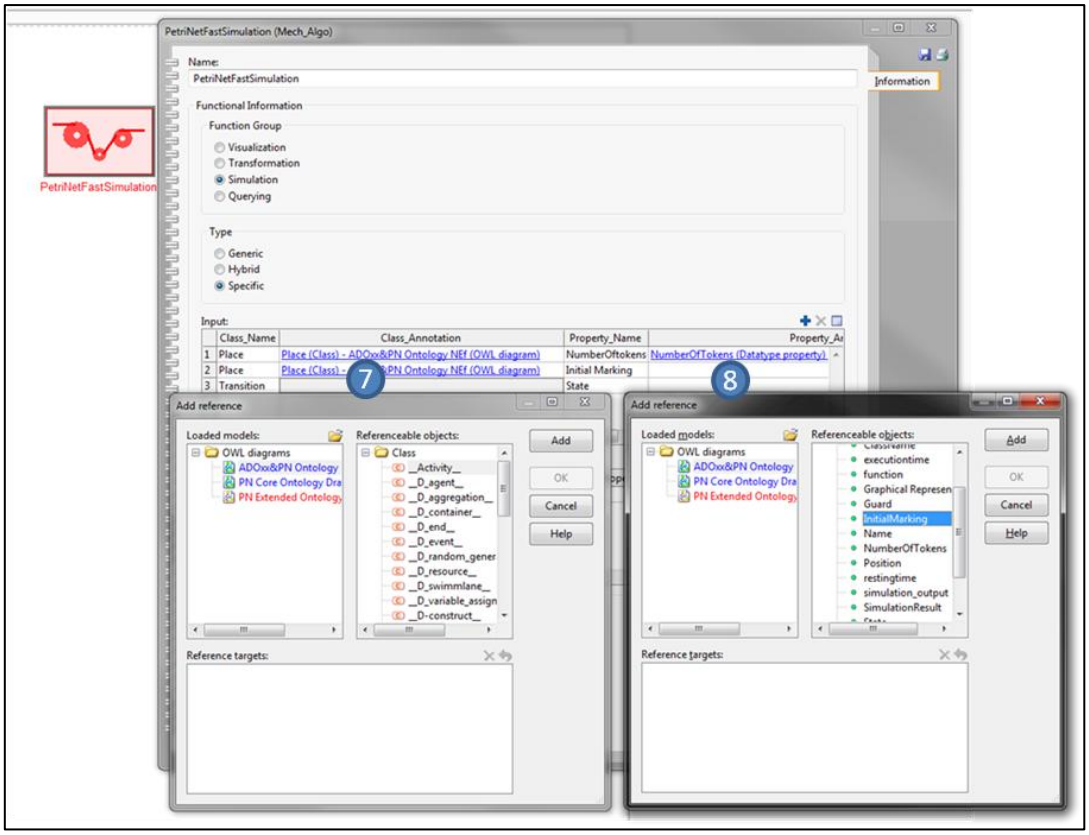


Figure 6-4 Annotation of Input/Output Classes and Properties of the Mechanism using Concepts from Petri Nets Ontology

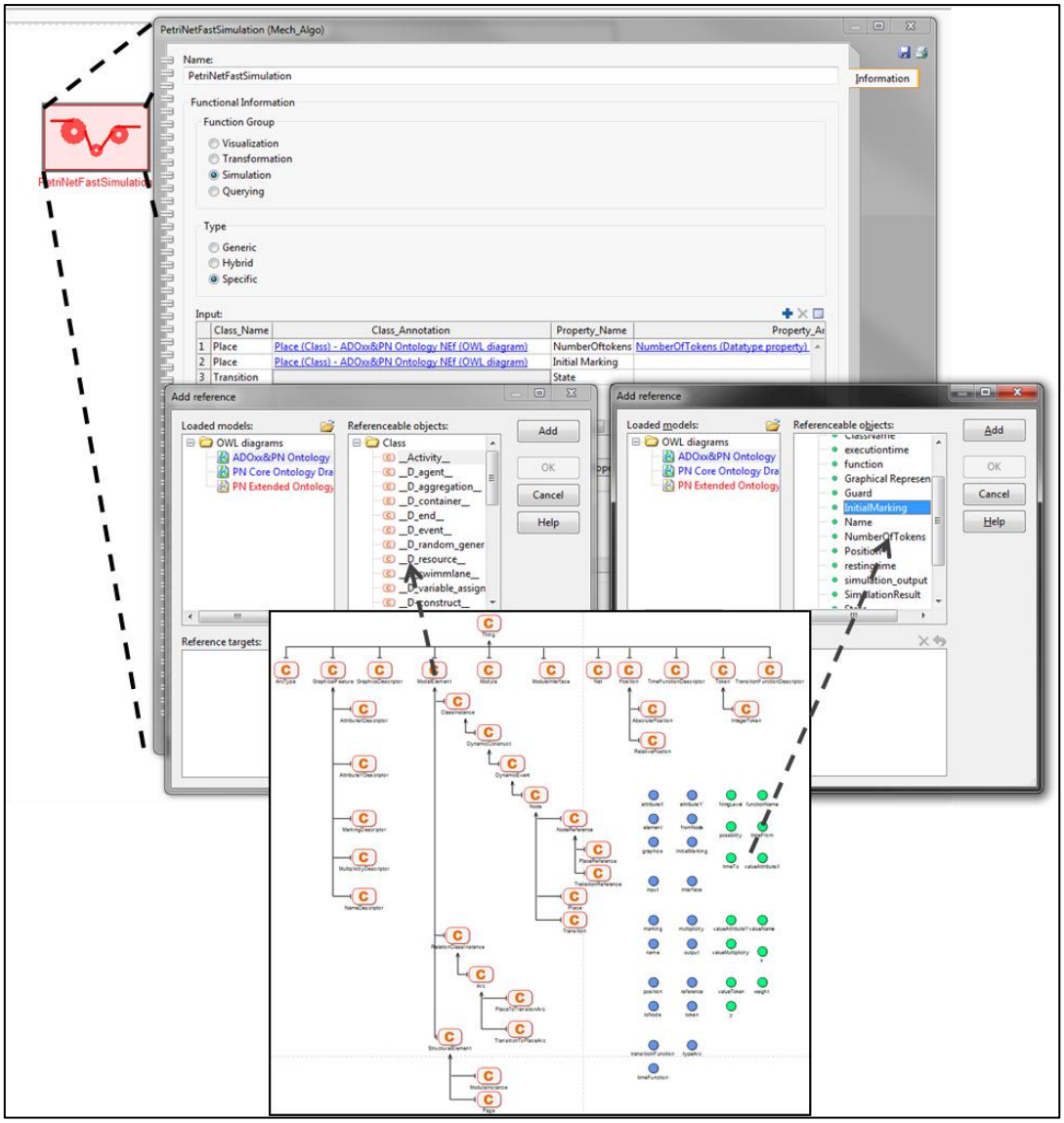


Figure 6-5 Overview of Semantic Description of a Mechanism

(9) Defines dependencies among the mechanism & algorithm with using relation “uses”.

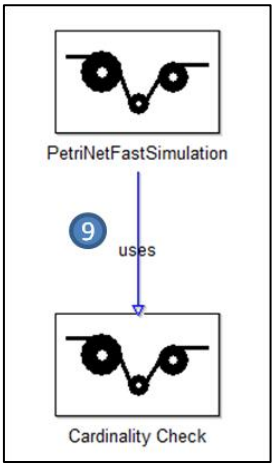


Figure 6-6 Defining the Dependencies among the Mechanisms

(10) Exports regarding SeMD Models (Mechanism Description Model and OWL Model) as depicted in Figure 6-7, and transfer them in the file system of the Matching Engine.

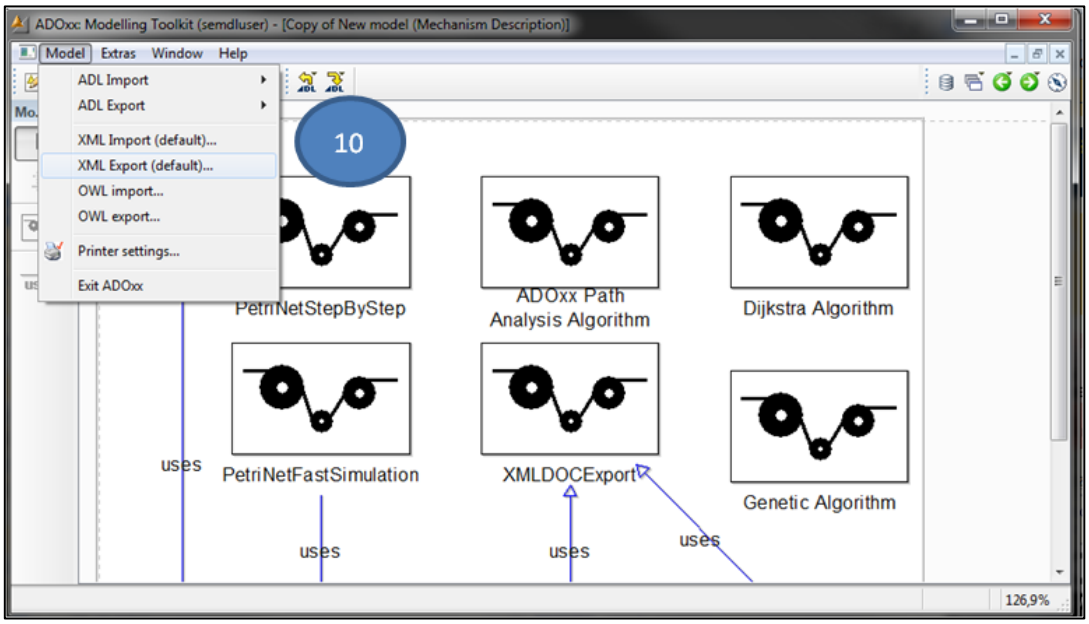


Figure 6-7 Exporting SeMD Model as XML File

(11) Method engineer exports her/his on ADOxx® implemented modelling language as XML

(12) Imports her/his modelling language XML file into the Matching Engine

(13) Selects function groups that she/he searching for

(14) Selects mechanism & algorithm type that she/he searching for

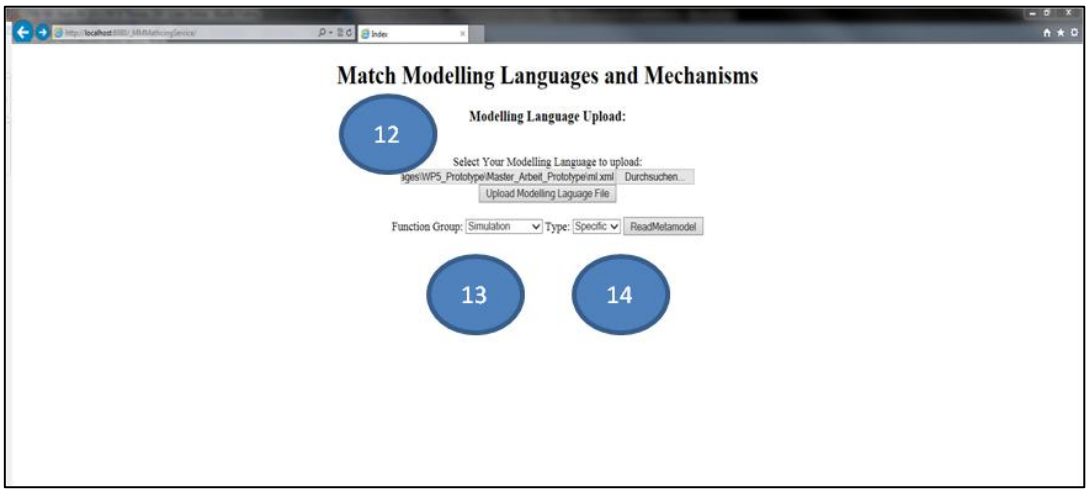


Figure 6-8 Defining the Goal using Web Client of the Matching Engine

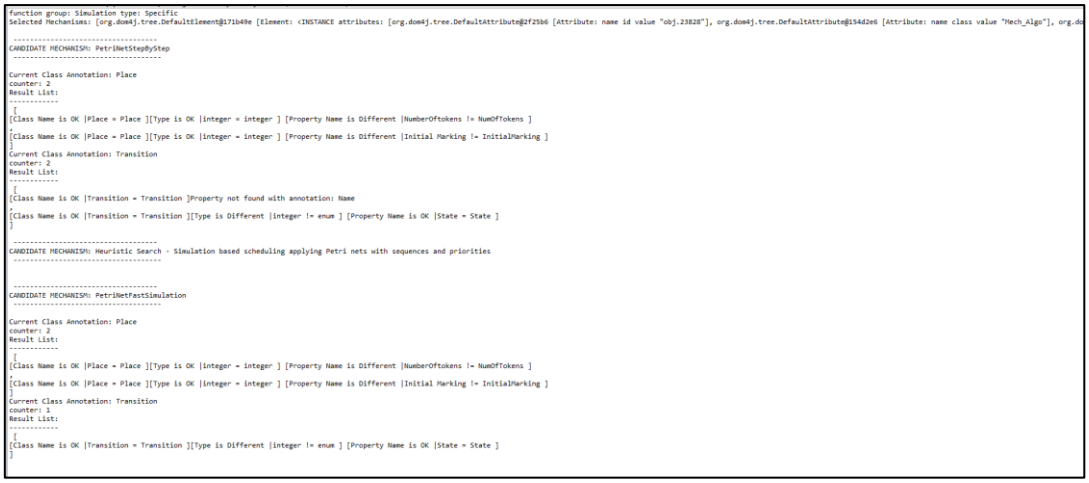


Figure 6-9 Log file exposed by the Matching Engine

7 Evaluation

In order to prove the concept, three Petri Net modelling languages have been implemented on ADOxx® version 1.3 UL1[7]. In the project (Efendioglu, Lekaditis, et al. 2011), already implemented mechanisms; "FastSimulation", "StepByStepSimulation", "PNML Transformation" "ADOxx Path Analysis" and an algorithm "Genetic Algorithm" have been investigated and semantically described with using SeMD modelling toolkit. For semantic description of mechanism & algorithms, an Petri Nets Ontology, which is based on (Gašević and Devedžić 2006) (Gasevic 2006) with ADOxx-specific concepts has been modelled again on SeMD and utilized for semantic description.

Therefore Proof of concept is using of any Petri Net modelling toolkit implemented on ADOxx® version 1.3 UL1 creating any kind of correct petri net and applying the mechanisms and algorithms matching engine of this work.

In the following some evaluation results are discussed as proof of concept.

As depicted in Figure 7-1, in case the user queries simulation mechanisms & algorithms_for Place/Transition Petri Nets, the result file of the matching engine depicts that following mechanisms "FastSimulation", "StepByStepSimulation", and "Genetic Algorithm" comply with the Place/Transition modelling language, as depicted in Figure 7-2. But the Path Analysis would not comply, since this mechanism requires classes related to "Start", "End" and "Decision". (The created log in HTML format can be found in Appendix-2:Log Created for scenario; matching simulation mecnhanism & algorithms for Place/Transition Petri Nets .)

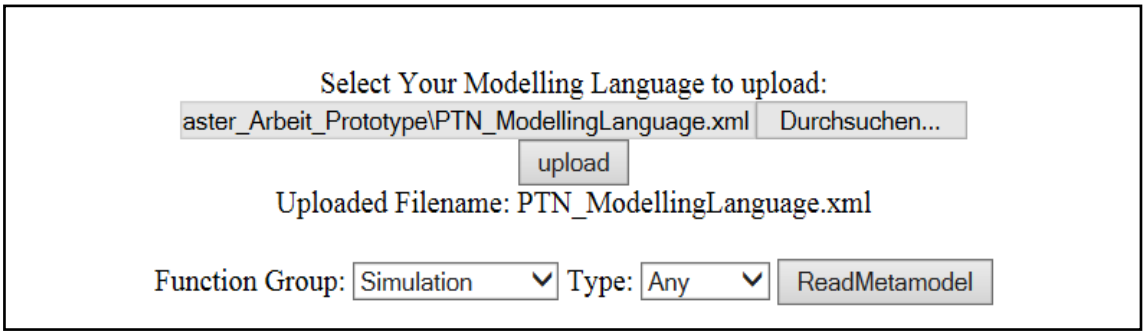


Figure 7-1: Matching PTN Modelling Language with Mechanism & Algorithms with using Matching Engine

[7] Downloaded, February 26, 2013 from http://www.adoxx.org/live/download

Figure 7-2: Part of Results after Processing Request for Matching Simulation Mechanism & Algorithms with PTN Modelling Language

Another test had been performed with “FastSimulation” applied on any model created with Place/Transition Nets modelling language, where the fast simulation will run without any problem depicted in Figure 7-3

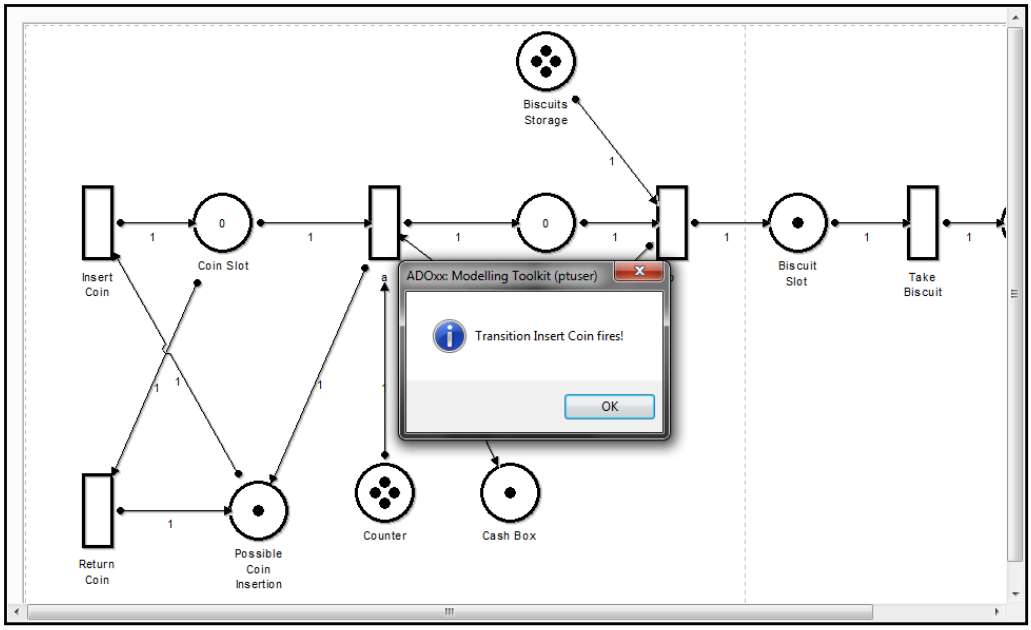


Figure 7-3: Successful Execution of Fast Simulation Mechanism on a Model modeled by PTN Modelling Language

8 Conclusion and Discussion

8.1 Outlook and Findings

The underlying work triggered by the requirement of supporting complex process of construction of modelling methods, which needs to provide the necessary concepts and functionality to create information value out of the models. Hence this work aimed to contribute to simplify the enrichment of model editors with mechanisms and algorithms to upgrade them to full-fletched modelling tools.

In order to achieve abovementioned objective, as our headstone was the Generic Modelling Method Specification Framework proposed by (Karagiannis and Kühn 2002) (Kühn 2004) and ADOxx® Framework worked by research group Knowledge Engineering from University Vienna and supported by communities OMI and ADOxx®.org, we have investigate how to conceptualize a modelling method following Generic Modelling Method Specification Framework, how to realize a modelling method on ADOxx® Metamodelling Platform, and we have investagte studies –particularly made by research group Knowledge Engineering from UIniversity of Vienna- based on ADOxx® Framework like FDMM.,

Afterwards, we have investigated Petri Nets Modelling Languages and mechanism & algorithms complying with those modelling languages and we have selected three different Petri Nets Modelling Languages from three level of Petri Nets, which served perfectly to prove how differentiate the complexity of notions among different modelling languages even from the same domain. Moreover, we have investigate and listed some mechanism & algorithms –not necessarily implemented specific for Petri Nets Modelling languages- which served to prove how and why concepts are differentiating required by different mechanism & algorithms and that there can be semantic gab between concepts utilized by mechanism & algorithms and utilized to build the modelling languages.

We embark to find an approach, which is aware of matching dimensions, and matching points between modelling languages and mechanism & algorithms, and which enables describing mechanism & algorithms semantically in order to close semantic gab between them, and which eventually performs (semi-) automatic matching between modelling languages and mechanism & algorithms. For that concern, we have found three different dimension, the cohesion between them shall be considered namely; (1) Model Hierarchy, (2) Mechanism & Algorithm Type and (3) Function Group. Additionally we found out the matching points with the guidance of knowledge that we have acquired during the analysis of studies based on ADOxx® Framework,-as specially FDMM- namely; (1) Object Type (class or relation class), (2) Attribute Type (properties) and (3) Data Type. For all to we have investigated state-of-the art in the semantic description of services with concentrating prominent approaches and standards, like OWL-S, WSMO, SAWSDL, WSDL and OWL. The findings guide us how to expose profile of a mechanism as well as the goals of method engineer, how to closing semantic gab with utilizing additional entity for annotation and how to lift semantic concepts

from the ontology to utilize for annotation. Findings of this investigation enable us to find an approach to describe mechanism & algorithms formally with utilizing notions defined in FDMM. Moreover, we have found out that ,in order to utilize an ontology to describe mechanism & algorithms, ADOxx® specific concepts need to be considered in that ontology therefore, we have build a taxonomy with concepts of Petri Nets and ADOxx® specific concepts, which has eased adaptation and extension of Petri Nets Ontology proposed by (Gašević and Devedžić 2006) with ADOxx® specific concepts.

Based on findings we have specified a matching mechanism, conceptually and technically, which consists of two systems namely

(1) SeMD Modelling Editor, which enables building domain ontologies and for sure description of mechanism & algorithms enriched semantically
(2) Matching Engine, which is aware of both entities used for semantic description of mechanism & algorithms and entities used in modelling language as well as matching dimensions, and which can match concepts despite semantic gab between them and to calculate semantic similarity.

Regarding to specifications we have implemented a prototype and defined a procedure presents how to utilize the Matching Mechanism that enable to prove conceptual framework work out on the generic level with utilizing concrete application scenario in context of Petri Nets, whose results serve as proof of concept. The prototype of Matching Mechanism on one side allows software engineers to describe mechanisms & algorithms semantically with utilizing conceptual models on the another side allows method engineers to describe their goals with uploading their modelling language description in form of XML file, describing their desired function group and mechanism & algorithms type by selecting from pre-defined list. According to descriptions of mechanisms & algorithms available in its file system and given goals by method engineer, the Matching Engine match concepts place in the metamodel level, calculates semantic similarity and exposes the results consisting which concepts, what extend they are matching and the results of semantic calculation results in a form of log represented in browser at the user side.

The Matching Mechanism can be classified as a knowledge management tool for supporting building a modelling method particularly matching modelling language with mechanisms & algorithms. In a case more specific classification is required, we can classify the Matching Mechanism based on study about identification of knowledge management tools for supporting modelling method development exposed in (Toader 2013) as depicted in the Figure 8-1.

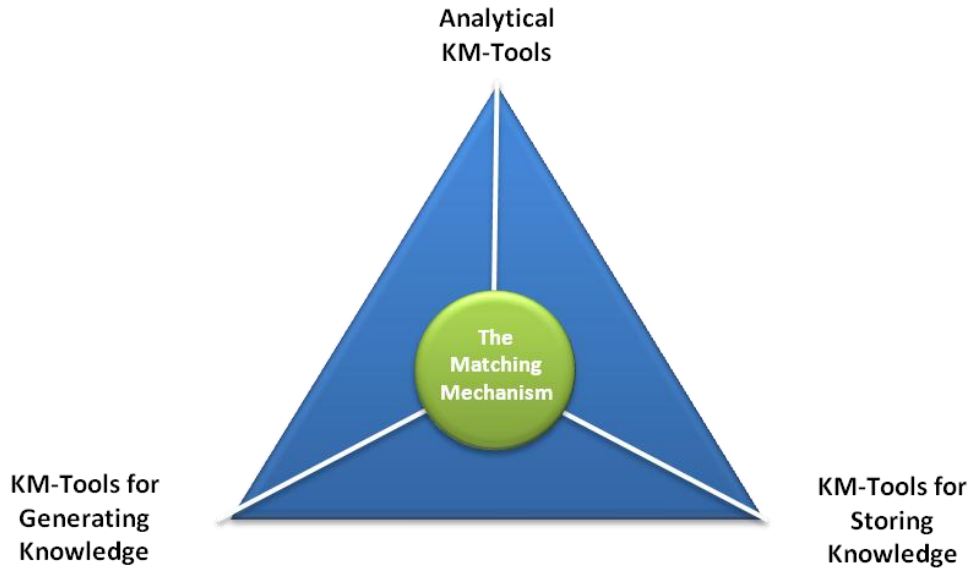


Figure 8-1 Classification of the Matching Mechanism as a Knowledge Management Tool

The Matching Mechanism stores the knowledge of software engineer about the mechanisms & algorithms and domain knowledge of the domain expert or the ontology expert with offering SeMD Modelling Editor and model repository. On the other hands, the Matching Mechanism retrieves goals of the method engineer including description of modelling language, required type of mechanism & algorithms and required function group via web user interface of the Matching Engine and stores them in its file system at the persistence level. Hence the Matching Mechanism can be placed in the knowledge management tools for storing knowledge.

Moreover, the Matching Mechanism analyzes stored knowledge about mechanisms & algorithms and knowledge and criteria defined the goals of the method engineer which consist of heterogeneous concepts which brings semantic gaps. For that reason the Matching Mechanism analyzes also domain knowledge which is provided in the form of ontology with having regard to matching points in each matching dimension. Hence the Matching Mechanism can be placed in the analytical knowledge management tools.

At last but not least the Matching mechanism generates new knowledge about complying mechanisms & algorithms with the modelling language with providing information about matching concepts and what extend they are matching or the concepts that are not matching and why they are not matching in a form of log, which makes possible to adapt modelling language or the mechanism & algorithms that are required to build the modelling method. Hence the Matching Mechanism can be also placed in the knowledge management tools for generating knowledge.On the basis of that analysis, the Matching Mechanism is a knowledge management tool which can be classified between a knowledge management tool for storing knowledge, analytical knowledge management tool and a knowledge management tool for generating knowledge.

8.2 Input for Open Source Community

This work provides two inputs for open source community; (1) a conceptual framework worked out on a generic level, which enables semantic description of mechanism & algorithms and matching mechanisms & algorithms complying with given modelling language and (2) a prototype, which is implemented to prove concepts of the developed framework with utilizing application scenario in the context of Petri Nets.

With using these inputs, it is possible to extend the framework and develop a matching mechanism which solves the interoperability issues with utilizing mediators and enables auto integration of the mechanisms & algorithms into modelling language considering modelling procedure, so enables building modelling method automatically.

Bibliography

Abramowicz, Witold, and Monika Kaczmarel, Tomasz Kaczmarek Agata Filipowska. "Semantically enhanced Business Process Modelling." In *Workshop on Semantic Business Process and Product Lifecycle Management. CEUR Workshop Proceedings*, by M. Hepp, K. Hinkelmann, D. Karagiannis, R. Klein and N. Stojanovic, 88-91. Innsburck, 2007.

ADOxx.org. "ADOxx Tutorial Package, ADOxx Trainning Days 26-27 March 2013." *ADOxx ORG Website.* 2013. http://www.adoxx.org/live/documents/10157/17fe60ea-598c-4858-996e-0412248ed3bb (accessed June 29, 2013).

—. "ADOxx Workshops - Domain Samples and Development Tools." *ADOxx.org Website.* 26 March 2013. http://www.adoxx.org/live/tutorial (accessed April 2013).

—. *ADOxx.org.* 2013. http://www.adoxx.org/live/home (accessed April 30, 2013).

—. "Implement/Extend OWL@ADOxx." *ADOxx.org Website.* 2013. hhttp://www.adoxx.org/live/implementextend-owl-adoxx (accessed December 18, 2013).

—. "OWL Method." *ADOxx.org.* 2013. http://www.adoxx.org/live/home (accessed September 1, 2013).

Akkiraju, Rama, et al. "Web Service Semantics- WSDL-S." *W3C Member Submission.* 07 November 2005. http://www.w3.org/Submission/WSDL-S/ (accessed June 16, 2013).

Baar, Thomas. "Correctly defined concrete syntax." *Software & Systems Modelling Volume 7, Issue 4*, 01 October 2008: 383-398.

Bernardinello, Luca, and Fiorella De Cindio. "A survey of basic net models and modular net classes." In *Advances in Petri Nets 1992*, by Grzegorz Rozenberg, 304-351. Springer-Verlag, 1992.

Booth, David, and Canyang Kevin Liu. "Web Services Description Language (WSDL) Version 2.0 Part 0:Primer." *World Wide Web Consortium.* 3 August 2005. http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803/ (accessed November 31, 2013).

Capellmann, Carla, Heinz Dibold, and Uwe herzog. "Using high-level petri nets in the field of intelligent networks." In *Application of Petri Nets to Communication Networks*, by Jonathan Billington, Michel Diaz and Grzegorz Rozenberg, 1-36. Springer Berlin Heidelberg, 1999.

Cardoso, Jorge, John A. Miller, and Savitha Emani. "Web Services Discovery Utilizing Semantically Annotated WSDL." In *Reasoning Web*, by Cristina Baroglio, Piero A.

Bonatti, Jan Maluszynski, Massimo Marchiori, Axel Polleres and Sebastian Schaffert, 240-268. Springer-Verlag Berlin Heidelberg, 2008.

Chinnici, Roberto, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language." *World Wide Web Consortium-Website.* 26 June 2007. http://www.w3.org/TR/wsdl20/ (accessed August 1, 2013).

Chomsky, Noam. *Syntactical Structures.* Berlin: Mounton, The Hague, 1985.

Christensen, Erik, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. "Web Services Description Language (WSDL) 1.1." *W3C Note.* 15 March 2001. http://www.w3.org/TR/wsdl (accessed August 16, 2013).

Clark, James, and Steve DeRose. "XML Path Language (XPath) Version 1.0." *World Wide Web Consortium Website.* 16 November 1999. http://www.w3.org/TR/xpath/ (accessed December 23, 2013).

De Bruijn, Jos, et al. "Web Service Modeling Language (WSML)." *World Wide Web Consortium Website.* 3 June 2005. http://www.w3.org/Submission/WSML/ (accessed September 10, 2013).

—. "Web Service Modeling Ontology (WSMO)." *W3C Member Submission.* 3 June 2005. http://www.w3.org/Submission/WSMO/ (accessed July 20, 2013).

De Michelis, Giorgio, and Ellis Clarence A. "Computer Supported Cooperative Work and Petri Nets." In *Lectures on Petri Nets II: Applications*, by Wolfgang Reisig and Grzegorz Rozenberg, 125-153. Springer Berlin Heidelberg, 1998.

Desel, Jörg, and Wolfgang Reisig. "Place/Transition Nets." In *Lectures on Petri Nets I: Basic Models*, by Wolfgang Reisig and Gregorz Rozenberg, 122-173. Berlin, Germany: Springer-Verlag, 1998.

Efendioglu, Nesat, Christos Lekaditis, Cihan Celik, and Chen Zhan. *Petri Nets: A Simulation Tool.* Project Presentation, Vienna/Austria: Open Models Initiative, 2011.

Efendioglu, Nesat, Robert Woitsch, Adrian Solomon, and Francisco Calle Moreno. "D3.21 Mission Control Room Prototype 1.0." *BIVEE Project Website.* 30 June 2013. http://wordpress.bivee.eu/wp-content/uploads/2013/01/BIVEE_D3.21_Mission_Control_Room_Prototype_1.0.pdf (accessed August 1, 2013).

Eidoon, Zahra, Nasser Yazdani, and Farhad Oroumchian. "Ontology Matching Using Vector Space." In *Advances in Information Retrieval*, by Craig Macdonald, Iadh Ounis, Vassilis Palchouras, Ian Ruthven and Ryen W. White, 472-481. Springer BErlin Heidelberg, 2008.

Farrell, Joel, and Holger Lausen. “Semantic Annotations for WSDL and XML Schema.” *The World Wide Web Consortium (W3C) Website.* 2007. http://www.w3.org/TR/sawsdl/ (accessed July 26, 2013).

Fill, Hans-Georg, and Dimitris Karagiannis. “On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform.” *Enterprise Modelling and Information Systems Architectures Vol.8, No. 1*, March 2013: 4-25.

Fill, Hans-Georg, Daniela Schremser, and Dimitris Karagiannis. “A Generic Approach for the Semantic Annotation of Conceptual Models Using a Service-Oriented Architecture.” *International Journal of Knowledge Management Vol. 9, No. 1*, January-March 2013: 76-88.

Fill, Hans-Georg, Timothy Redmond, and Dimitris Karagiannis. “FDMM: A Formalism for Describing ADOxx Meta Models and Models.” In *ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems, Volume 3, Wroclaw, Poland,28 June - 1 July, 2012*, by Leszek A. Maciaszek, Alfredo Cuzzocrea and José Cordeiro, 133-144. Wroclaw, Poland: SciTePress, 2012.

Gašević, Dragan, and Vladan Devedžić. “Petri Net Ontology.” *Knowledge-Based Systems, Volume 19, Issue 4*, August 2006: 220-234.

Gasevic, Dragen. “Petri Net Semantic Web Infrastructure.” *Dragan Gasevic's Home Page.* 2006. http://www.sfu.ca/~dgasevic/projects/PNO/Download.htm (accessed 9 28, 2013).

Geisler, Robert, Marcus Klar, and Claudia Pons. *Dimensions and Dichotomy in Metamodeling, Technical Report 98-5.* Technical Report, Technical University Berlin, 1998.

Hrgovcic, Vedran, Dimitris Karagiannis, and Robert Woitsch. “Conceptual Modeling of the Organisational Aspects for Distributed Applications: The Semantic Lifting Approach.” In *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, 145-150. Japan, 2013.

Jensen, Kurt. “A Brief Introduction to Coloured Petri Nets.” In *Tools and Algorithms for the Contruction and Analysis of Systems, Lecture Notes in Computer Science Volume 1217*, by Ed Brinksma, 203-208. Springer-Verlag, 1997.

Jensen, Kurt. “An introduction to the practical use of coloured Petri Nets.” In *Lectures on Petri Nets II: Applications, Lecture Notes in Computer Science Valume 1492*, by Wolfgang Reisig and Grzegorz Rozenberg, 237-292. Springer-Verlag, 1998.

Jensen, Kurt. “An Introduction to the Theoretical Aspects of Coloured Petri Nets.” In *A Decade of Concurrency, Lecture Notes in Computer Science Vol. 803*, by J. W. de Bakker, W. -P. de Roever and G. Rozenberg, 230-272. Springer-Verlag, 1994.

Jones, Karen Spärck. “Semantic Primitives: The tip of the Iceberg.” In *Work and Intelligence II - Essays in Honor of Yorick Wilks*, by Khurshid Ahmad, Christopher Brewster and Mark Stevson, 235-253. Springer, 2007.

Junginger, Stefan, Harald Kühn, Robert Strob, and Dimitris Karagiannis. “ Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation - ADONIS: Konzeption und Anwendungen (in German).” In *Wirtschaftsinformatik 42 (5)*, 392-401. 2000.

Karagiannis, Dimitris, and Harald Kühn. “Metamodelling Platforms.” In *Proceedings of the Third International Conference EC-Web 2002 - Dexa 2002, Aix-en-Provence, France, September 2-6, 2002, LNCS 2455*, by Bauknecht K., A. Min Tjoa and Quirchmayer G., 182. Berlin, Heidelberg: Springer-Verlag, 2002.

Karagiannis, Dimitris, et al. “Meta-Modelling as a Concept: The Conceptualisation of Modelling Methods - Invited Tutorial.” *INFORMATIK 2013.* September 2013.

Karagiannis, Dimitris, Hans-Georg Fill, Srdjan Zivkovic, and Wilfrid Utz. “From Model Editors to Modelling Tools: Operationalizing Modelling Methods with ADOxx.” *ACM/IEEE 15th International Conference on Model-Driven Engineering Languages and Systems (MODELS'2012).* Innsbruck, 2012.

Karagiannis, Dimitris, Wilfried Grossmann, and Peter Höfferer. “Open Model Initiative - A Fasibility Study.” *Open Models Initiative.* 2008. http://cms.dke.univie.ac.at/uploads/media/Open_Models_Feasibility_Study_SEPT_2008.pdf (accessed July 12, 2013).

Keller, Uwe, Ruben Lara, Axel Polleres, Ioan Toma, Michael Kifer, and Dieter Fensel. “WSML Deliverable D5.1 v0.1: WSMO Web Service.” Deliverable, 2004.

Kiryakov, Atanas, Borislav Popov, Damyan Ognyanoff, Dimitar Manov, Angel Kirilov, and Miroslav Goranov. “Semantic Annotaiton, Indexing, and Retrieval.” In *The Semantic Web - ISWC 2003*, by Dieter Fensel, Katia Sycara and John Mylopoulos, 484-499. Springer Berlin Heidelberg, 2003.

Kleppe, Anneke. “A Language Description is More than a Metamodel.” *Fourth International Workshop on Software Language Engineering.* Nashville, USA, 2007.

Kopecky, Jacek, Tomas Vitar, Carine Bournez, and Joel Farrell. “SAWSDL: Semantic Annotations for WSDL and XML Schema.” In *Internet Computing, IEEE (Volume:11 , Issue: 6)*, 60-67. 2007.

Kühn, Harald. “Methodintegration im Business Engineering.” PhD Thesis, Vienna, Austria, 2004.

Kühn, Harald, Stefan Junginger, Dimitris Karagiannis, and C. Petersen. “Metamodellierung im Geschäftsprozessmanagement: Konzept, Erfahrung und Potentiale (in German).” In *Modellierung 1999, Workshop der Gesellschaft für Informatik e. V. (GI), März 1999 in*

Karlsruhe, by Jörg Desel, Klaus Pohl and Andy Schürr, 75-90. Karlsruhe, Germany: Teubner, 1999.

Kühn, Harald, Stefan Junginger, Dimitris Karagiannis, and C. Petersen. "Re-Use in Business Process Management based on Metamodeling and Model Views." In *Adjunct Conference Proceedings of the 8th International Conference on Human-Computer Interaction, HCI Int. '99*, by H.-J. Bullinger and P. H. Vossen, 173-174. Munich, Germany, 1999.

Le Hors, Arnaud, et al. "Document Object Model (DOM) Level 3 Core Specification." *World Wide Web Consortium Website.* 07 July 2004. http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/ (accessed December 23, 2013).

Martin, David, et al. "OWL-S: Semantic Markup for Web Services." 2004. http://www.w3.org/Submission/OWL-S/.

Mitchell, Jeff, and Mirella Lapata. "Vector-based Models of Semantic Compisition." In *In Proceedings of ACL-08: HLT*, 236-234. Ohio, USA: Association for Computational Linguistics, 2008.

Mitchell, Melanie. *An Introduction to Genetic Algortihms.* London, England: The MIT Press, 1999.

Motik, Boris, et al. "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)." *Word Wide Web Consortium-Website.* 11 December 2012. http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/ (accessed November 31, 2013).

Motik, Boris, Patel-Schneider Peter F., and Ian Horrocks. "Structural Reference Experiment-OWL." *World Wide Web Consortium-Website.* 2008. http://www.w3.org/2007/OWL/wiki/Structural_Reference_Experiment (accessed November 31, 2013).

Murata, Tadao. "Petri Nets: Properties, Analysis and Applications." In *Proceedings of the IEEE, Vol. 77, No. 4 April 1989*, 541-580. 1989.

Object Management Group. "MDA Guide Version 1.0.1." *OMG Homepage.* 2003. http://www.omg.org/cgi-bin/doc?omg/03-06-01 (accessed June 26, 2013).

OMILAB. *Conceptualize your own Modelling Method.* 2013. http://www.omilab.org/web/ (accessed December 20, 2013).

Open Models Initiative. *Open Model Community Wiki.* (accessed June 13, 2013).

Paolucci, Massimo, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. "Semantic Matching of Web Services Capabilities." In *Proceeding ISWC '02 Proceedings of the First International Semantic Web Conference on The Semantic Web* , 333-347. London: Springer-Verlag, 2002.

Petri, Carld Adam. *Kommunikation mit Automaten.* Dissertation, Darmstadt, Germany: Fakulät für Mathematik und Physik der Technischen Hochschule Darmstadt accessed on 02.07.2013 http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/, 1966.

Pohjonen, Risto, and Juha-Pekka Tolvanen. "Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences." In *SPLC'05 Proceedings of the 9th international conference on Software Product Lines*, 198-209. Jyväskylä, Finland: Springer Berlin Heidelberg, 2005.

Polleres, Axel, Holger Lausen, and Ruben Lara. "Semantische Beschreibung von Web Services (in German)." In *Semantic Web*, by Tassilo Pellegrini and Andreas Blumauer, 505-524. Springer-Verlag, 2006.

Raghavan, V. V., and S. K. M Wong. "A critical analysis of vector space model for information retrieval." *Journal of the American Society for Information Science Volume 37, Issue 5*, September 1986: 279-287.

Random House. *Dictonary.com.* 2013. http://dictionary.reference.com/browse/syntax (accessed March 01, 2013).

Reisig, W., E. Kindler, T. Vesper, H. Völzer, and R. Walter. "Distributed algorithms for networks of agents." In *Lectures on Petri Nets II: Applications*, by Wolfgang Reisig and Grzegorz Rozenberg, 331-385. Springer Berlin Heidelberg, 1998.

Reisig, Wolfgang. *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien (in German).* Wiesbaden, Germany: Vieweg+Tuebner Verlag, 2010.

Resnik, Philip. "Using information content to evaluate semantic similarity in a taxonomy." In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, 448-453. Morgan Kaufmann Publishers Inc., 1995.

—. "WordNet and Distributional Analysis: A Class-based Approach to Lexical Discovery." *AAAI Workshop on Statistically-based Natural Language Processing Techniques.* accessed 16.07.2013 via http://aaaipress.org/Papers/Workshops/1992/WS-92-01/WS92-01-006.pdf, 1992.

Rozenberg, Grzegorz, and Joost Engelfriet. "Elementary Net Systems." In *Lectures on Petri Nets I: Basic Models*, by Wolfgang Reisig and Gregorz Rozenberg, 12-121. Berlin, Germany: Springer-Verlag, 1998.

Salton, G., A. Wong, and C. S. Yang. "A vector space model for automatic indexing." *Communications of the ACM Volume 18 Issue 11*, November 1975: 613-620.

Salton, Gerard. *The SMART Retrieval System—Experiments in Automatic Document Processing.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc, 1971.

Silvam, Manuel, Enrique Teruel, Robert Vallette, and Herve Pinguad. “Petri nets and production systems.” In *Lectures on Petri Nets II: Applications*, by Wolfgang Reisig and Grzegorz Rozenberg, 85-124. Springer Berlin Heidelberg, 1998.

Stanford Center for Biomedical Informatics Research. *Protoge.* 2013. http://protege.stanford.edu/ (accessed December 20, 2013).

Strahringer, Susanne. *Metamodellierung als Instrument des Methodenvergleichs - Eine Evaluierung am Beispiel objektorientierter Analysemethoden.* Aachen, Germany: Shaker-Verlag, 1996.

Toader, Gabriela. “Concept of a Knowledge Stepper for Web Apps Development.” Master Thesis, Vienna, 2013.

Tous, Rubén, and Jaime Delgado. “A Vector Space Model for Semantic Similarity Calculation and OWL Ontology Alignment.” In *Database and Expert Systems Applications*, 307-316. Springer Berlin Heidelberg, 2006.

Trompedeller, Monuka. “A Classification of Petri Nets.” *Petri Nets World.* 1995. http://www.informatik.uni-hamburg.de/TGI/PetriNets/classification/ (accessed 06 30, 2013).

W3C OWL Working Group. “OWL 2 Web Ontology Language.” *Word Wide Web Consortium-Website.* 11 December 2012. http://www.w3.org/TR/owl2-overview/ (accessed November 31, 2013).

Yakovlev, Alexandre V., and Albert M. Koelmans. “Petri nets and digital hardware design.” In *Lectures on Petri Nets II: Applications*, by Wolfgang Reisig and Grzegorz Rozenberg, 154-236. Springer Berlin Heidelberg, 1998.

Yang, Xiaowei. “A Framework for Semantic Discovery.” *Xiaowei Yang Publications.* 2001. http://www.cs.duke.edu/~xwy/publications/service.pdf (accessed July 10, 2013).

Zusammenfassung

Diese Modellierungsmethoden sollen Domain spezifischen Anforderungen erfüllen und daher Konzepte für konkrete Anwendungsdomäne und Funktionalität, die für Information Wert aus konzeptuellen Modellen schöpfen, breitstellen. Das Method Engineering führt die Individuen, die ihre eigenen Modellierungsmethoden entwickeln wollen. Da eine Modellierungsmethode auf einer Seite, die Konzepte für die konzeptionelle Darstellung der Realität in Zusammenhang mit der angegebenen Anwendungsdomäne breitstellen müssen, auf der anderen die Funktionen, die Informationen Wertschöpfung aus konzeptuellen Modelle ermöglichen, bieten müssen, ist die Entwicklung eines solchen Modellierungsmethode hoch komplex.

Diese Masterarbeit bietet einen konzeptionellen Framework auf generische Ebene und entsprechende Matching Mechanismus auf konkreter Ebene, die die Method Ingenieuren leiten, damit folgende Herausforderungen überwinden zu können; (1) Komplexität der Entwicklungsprozess von Modellierungsmethoden und (2) die Anreicherung der einfachen Modell -Editoren , um vollwertigen Modellierungswerkzeuge, die von Funktionalitäten in Form von Mechanismus und Algorithmen, die Informationswert schafft durch die Verarbeitung von konzeptionellen Modells bieten zu entwickeln.

Daraus ergibt sich die Struktur dieser Masterarbeit und ist mit folgenden Ideen ausgerichtet: zunächst bezieht sie sich auf den Method-Engineering-Ansatz, wie er in der "Generic Modelling Methode Spezification Framework" definiert ist, auf den Realisierungsansatz wie bei ADOxx ® Framework definiert ist und untersucht Studien innerhalb dieser Rahmen. Zweitens als Ergebnis der Literaturrecherche sind drei Petri-Netze Modellierungssprachen und Mechanismus - Algorithmen ausgewählt worden und wurde mit Erkenntnissen aus der oben genannten Untersuchung spezifiziert und formaliziert, somit ein Anwendungsszenario erhalten , welches den Konzepte des Framework zu beweisen dient. Drittens, ein State-of- the Art Analyse, die die zeitgemäße Lösungen und Technologien in der semantischen Beschreibung der Dienste herausgibt ist durchgeführt, diese Ergebnisse der Analyse führten diese Arbeit zur Entwicklung des Ansatzes zur semantischen Beschreibung von Mechanismen und Algorithmen. Viertens, passende Dimensionen werden in der Führung der Anwendungsszenarien identifiziert und formale Beschreibung als Ansatz zur Beschreibung von Mechanismen und Algorithmen vorgeschlagen und angewendet, um zuvor ausgewählte Mechanismus & Algorithmen beschreiben zu können. Zudem ein Petri-Netze Ontologie, die mit ADOxx® spezifische Konzepte angepasst ist, und für die semantische Anreicherung der Beschreibung der Mechanismen und Algorithmen zu nutzen ist, vorgeschlagen. Fünftens, basierend auf den oben genannten Erkenntnissen ist der Matching -Mechanismus konzeptionell und technisch spezifiziert. Den Benutzer sind zwei verschiede Komponenten zur Verfügung gestellt; (1) Semantic enriched Mechanism Description (SeMD) Modellierung Editor und (2) die Matching-Engine. Und ein Prototyp des Matching - Mechanismus mit begrenzten Funktionalitäten der Community zu Testzwecken zur Verfügung gestellt.

Appendix

Appendix-1: Utilized Petri Nets Ontology

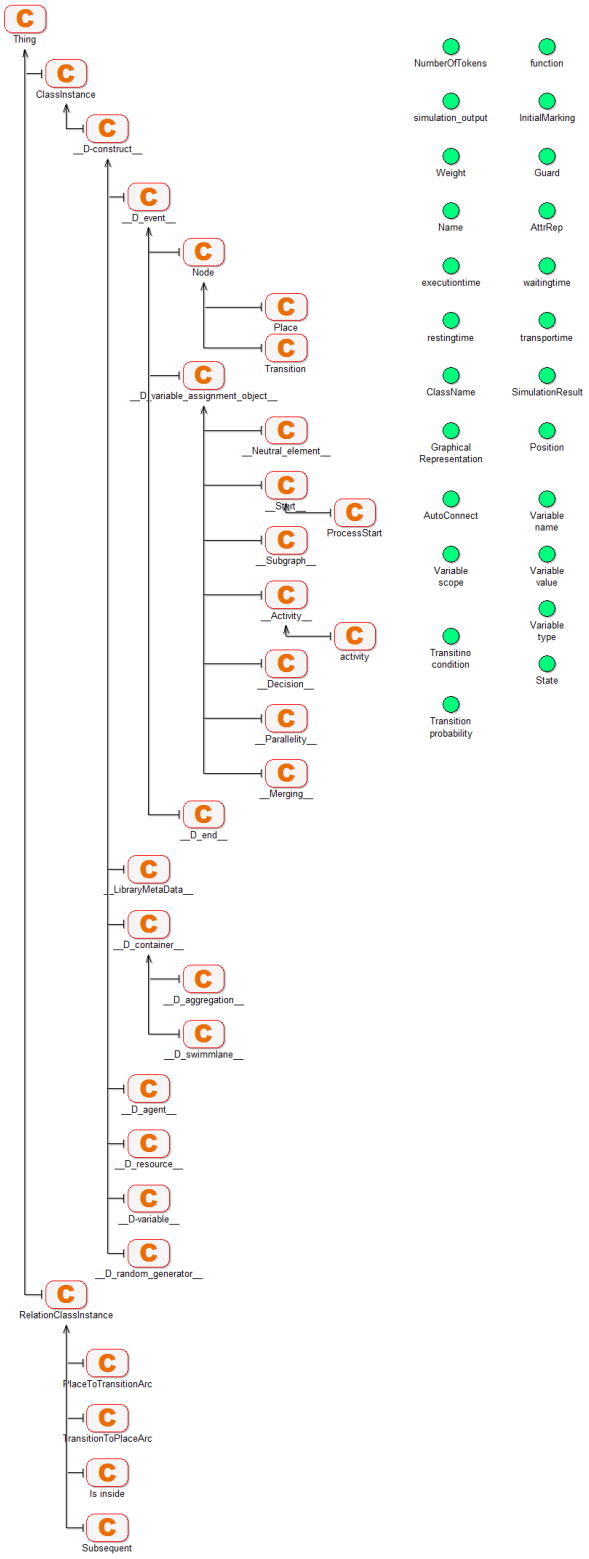


Figure Appendix-0-1 ADOxx specific Petri Nets Taxonomy

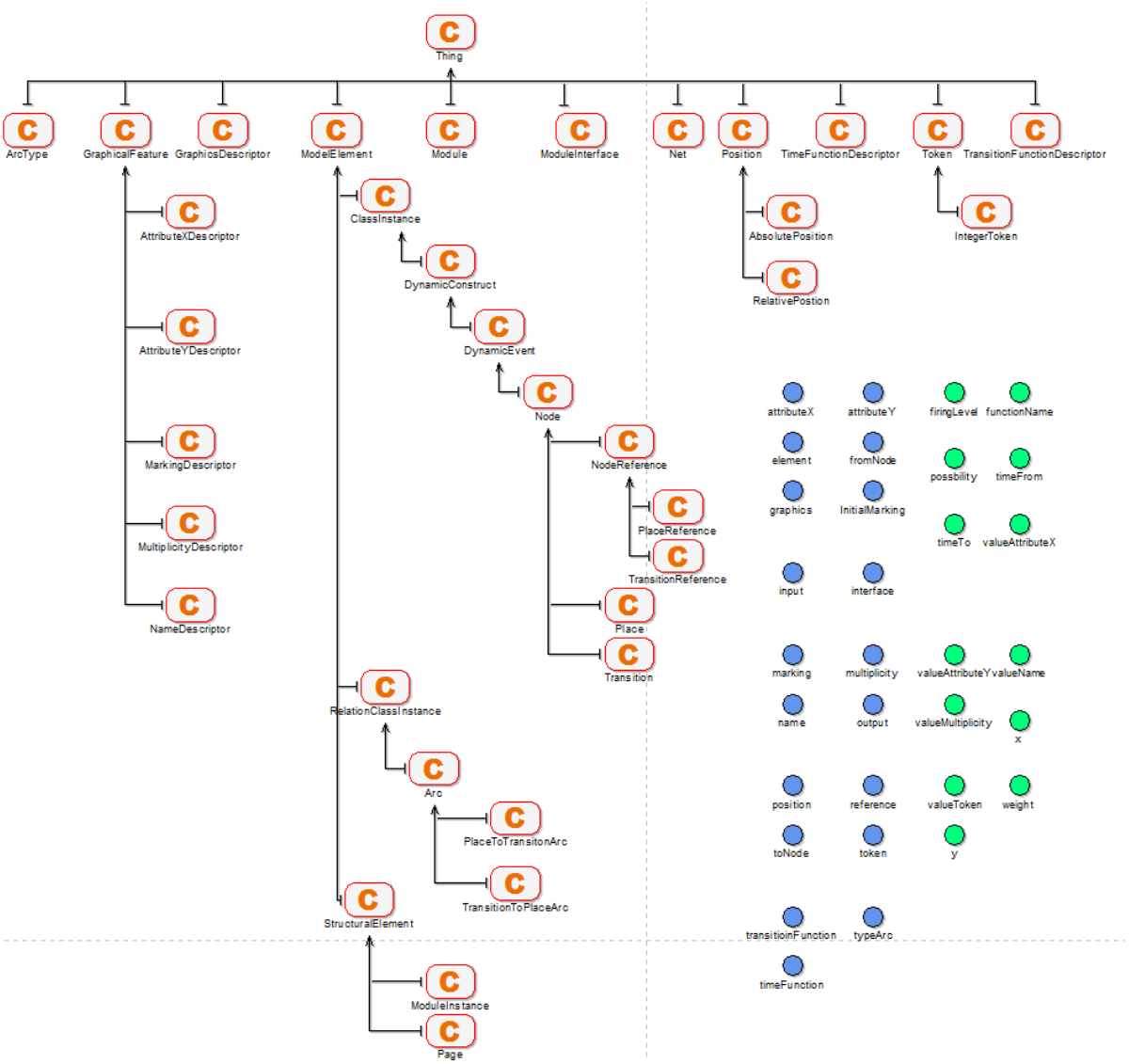


Figure Appendix-2 Detailed view of Petri Nets Ontology adapted with ADOxx Specific Concepts

Appendix-2:Log Created for scenario; matching simulation mecnhanism & algorithms for Place/Transition Petri Nets

Log Created for scenario; matching simulation mecnhanism & algorithms for Place/Transition Petri Nets
 <h1> #--
CANDIDATE MECHANISM: PetriNetStepByStep
--- </h1>

Log Created for scenario; matching simulation mecnhanism & algorithms for Place/Transition Petri Nets

Current Class Annotation:Place

[
[Class Name is OK |Place = Place]

[Type is OK |integer = integer]

[Property Name is OK |NumberOfTokens = NumberOfTokens]
]

Current Class Annotation:Transition

[
[Class Name is OK |Transition = Transition]

[Type is Different |enumeration != enum] [Property Name is OK |State = State]
]

Current Class Annotation:PlaceToTransitionArc

[
[Class Name is OK |P2TRelation = P2TRelation]

[Type is OK |integer = integer]

[Property Name is OK |Weight = Weight]
]

Current Class Annotation:TransitionToPlaceArc

[
[Class Name is OK |T2PRelation = T2PRelation]

[Type is OK |integer = integer]

[Property Name is OK |Weight = Weight]
]

<h1> #--

CANDIDATE MECHANISM: PetriNetFastSimulation

</h1>

Current Class Annotation:Place

[
[Class Name is OK |Place = Place]

[Type is OK |integer = integer]

[Property Name is OK |NumberOfTokens = NumberOfTokens]
]

Current Class Annotation:Transition

[
[Class Name is OK |Transition = Transition]

[Type is Different |enumeration != enum] [Property Name is OK |State = State]
]

Current Class Annotation:PlaceToTransitionArc

[
[Class Name is OK |P2TRelation = P2TRelation]

[Type is OK |integer = integer]

[Property Name is OK |Weight = Weight]
,
[Class Name is Different |T2PRelation != P2TRelation]

[Type is OK |integer = integer]

[Property Name is OK |Weight = Weight]
]

[
[Class Name is OK |P2TRelation = P2TRelation]

[Type is OK |integer = integer]

[Property Name is OK |Weight = Weight]
,
[Class Name is Different |T2PRelation != P2TRelation]

[Type is OK |integer = integer]

[Property Name is OK |Weight = Weight]
]

<h1> #--

CANDIDATE MECHANISM: Genetic Algorithm

</h1>

Current Class Annotation:Place

[
[Class Name is OK |Place = Place]

Property not found with annotation: Name
,
[Class Name is OK |Place = Place]

[Type is OK |integer = integer]

[Property Name is OK |NumberOfTokens = NumberOfTokens]
]

[
[Class Name is OK |Place = Place]

Property not found with annotation: Name
,
[Class Name is OK |Place = Place]

[Type is OK |integer = integer]

[Property Name is OK |NumberOfTokens = NumberOfTokens]
]

<h1>

#--

CANDIDATE MECHANISM: ADOxx Path Analysis Algorithm

</h1>

Current Class Annotation:activity

Current Class Annotation:__Start__

[
[Class Name is Different |Start != __Start__]

Property not found with annotation: ClassName
]

Current Class Annotation:__D_end__

[
[Class Name is Different |End != __D_end__]

Property not found with annotation: ClassName
]

Current Class Annotation:__Decision__

[
[Class Name is Different |Decision != __Decision__]

[Type is OK |string = string]

[Property Name is OK |Variable name = Variable name]
,
[Class Name is Different |Decision != __Decision__]

[Type is Different |enumeration != enum] [Property Name is OK |Variable scope = Variable scope]
,
[Class Name is Different |Decision != __Decision__]

[Type is Different |enumeration != enum] [Property Name is OK |Variable type = Variable type]
,
[Class Name is Different |Decision != __Decision__]

[Type is OK |string = string]

[Property

Log Created for scenario; matching simulation mecnhanism & algorithms for Place/Transition Petri Nets

Name is OK |Variable value = Variable value]
]

[
[Class Name is Different |Decision != __Decision__]

[Type is OK |string = string]

[Property Name is OK |Variable name = Variable name]
,
[Class Name is Different |Decision != __Decision__]

[Type is Different |enumeration != enum] [Property Name is OK |Variable scope = Variable scope]
,
[Class Name is Different |Decision != __Decision__]

[Type is Different |enumeration != enum] [Property Name is OK |Variable type = Variable type]
,
[Class Name is Different |Decision != __Decision__]

[Type is OK |string = string]

[Property Name is OK |Variable value = Variable value]
]

[
[Class Name is Different |Decision != __Decision__]

[Type is OK |string = string]

[Property Name is OK |Variable name = Variable name]
,
[Class Name is Different |Decision != __Decision__]

[Type is Different |enumeration != enum] [Property Name is OK |Variable scope = Variable scope]
,
[Class Name is Different |Decision != __Decision__]

[Type is Different |enumeration != enum] [Property Name is OK |Variable type = Variable type]
,
[Class Name is Different |Decision != __Decision__]

[Type is OK |string = string]

[Property Name is OK |Variable value = Variable value]
]

[
[Class Name is Different |Decision != __Decision__]

[Type is OK |string = string]

[Property Name is OK |Variable name = Variable name]
,
[Class Name is Different |Decision != __Decision__]

[Type is Different |enumeration != enum] [Property Name is OK |Variable scope = Variable scope]
,
[Class Name is Different |Decision != __Decision__]

[Type is Different |enumeration != enum] [Property Name is OK |Variable type = Variable type]
,
[Class Name is Different |Decision != __Decision__]

[Type is OK |string = string]

[Property Name is OK |Variable value = Variable value]
]

Current Class Annotation:Subsequent

[
[Class Name is OK |Subsequent = Subsequent]

[Type is OK |string = string]

[Property Name is Different |Transition conditions != Transition condition]
,
[Class Name is OK |Subsequent = Subsequent]

[Type is OK |string = string]

[Property Name is OK |Transition probability = Transition probability]
]

[
[Class Name is OK |Subsequent = Subsequent]

[Type is OK |string = string]

[Property Name is Different |Transition conditions != Transition condition]
,
[Class Name is OK |Subsequent = Subsequent]

[Type is OK |string = string]

[Property Name is OK |Transition probability = Transition probability]
]

Current Class Annotation:ProcessStart

Log Created for scenario; matching simulation mecnhanism & algorithms for Place/Transition Petri Nets

Curriculum Vitae

Nesat Efendioglu

Professional Experience

01/2012 – **BOC Asset Management GmbH,** Vienna (Austria)

Department: Innovation Group

Junior Researcher and Project Manager

- Responsible for conceptualization, specification and realization of model based Business Ecosystem and Supply Chain Management Tool within a European Union project namely "BIVEE-Business Innovation and Virtual Enterprise Environment" with funding support from the Seventh Framework Programme.
- Systematic research on semantic technologies, hybrid modeling, collaborative modeling and IT-supported management approaches.

03/2013 – 07.2013 **University of Vienna,** Vienna (Austria)

Department: Research Group Knowledge Engineering, Faculty of Computer Science

Teaching Tutor

- Teaching tutor of the course "Meta-modelling" in the master program "Bussiness Informatics". Responsible for teaching the application of meta-modelling approach in practice on a meta-modelling platform namely "ADOxx®" by following phases "specification and realization of modeling languages" and "implementation mechanism to functionalize modeling languages in context of visualization, transformation, querying and simulation" in order to develop full-fletched modeling editors for domain specific modeling methods.
- Also responsible for examination and grading the students for application of meta-modelling approach on ADOxx®.

09/2011 – 12/2011 **BOC Asset Management GmbH**, Vienna (Austria)

Department: Innovation Group

Internship

- Responsible of a project for translation and realization of Turkish version of a Business Process Management Tool namely “ADONIS®”.

03/2011 -09/2011 **Open Model Initiative, University of Vienna**, Vienna (Austria)

Student Project on Open-Use Petri Nets Simulation Tool

- Head of the project
- Responsible for conceptualization and specification of Modeling Language and simulation mechanism.
- Responsible for design of the architecture of the tool
- Responsible for implementation of simulation mechanism.

06/2005 – 09/2005 **Six Flags Great Adventure,** New Jersey (USA)

Department: Food Department

Cashcontroller

- Collection of turnover from shops, controlling of safe registries and accounting.

01/2005 – 02/2005 **Edip Iplik A.S**, Kirklareli (Turkey)

Department: Production Planning

Internship

- Support for production planning activities.

07/2004 – 08/2004 **Garanti Bankasi A.S,** Izmir (Turkey)

Department: Sales, Consulting, Customer Management and Service

Internship

- First point of contact for customers on the on Sales, Consulting, Customer Management and Service Department at a branch of the Bank.
- Support for consulting and customer management activities.

Academic Career

2008 – 2014 **University of Vienna**, Vienna (Austria)

(Presumable) *Business Informatics (M.Sc.)*

Focus on Semantic Technologies and Large Scale Systems

2006 – **Dokuz Eylül University**, Izmir (Turkey)

Finance (M.Sc.)

Focus on Valuation of Derivatives

All courses are successfully completed. Master Thesis is not written yet.

Grade: 3.48/4

2002 – 2006 **Dokuz Eylül University**, Izmir (Turkey)

Business Administration (B.Sc.)

Focus on Financial Management

Grade: 3.06/4

Extra Curricular Activities

03/2008 – 07/2008 MAP Student ("Muttersprachliche Ansprechpartner/innen" Student Tutor for international students), first point of Mother tongue contact persons for international Students in an institution providing university preparation courses for international students namely "VWU".

09/2003 – Active sophomore member in Dokuz Eylül University Career and Management Club (personal services were rewarded by Rector of the University)

09/2002 – 06/2006 Active member in Dokuz Eylül University Photography Club.

09/1999 – 06/2002 Class representative and Student representative at High-School

Supervision of debates between teaching staff and pupils. Representation of students in the school council.

Skills

Technical Skills

Modeling Languages and Standards:

UML, BPMN, BPMS, EPC, e3 Value, Petri Nets, UBL, ebXML, RDF, BSC, SCOR, VRM

Programming and Scripting Languages:

C++, Java, JSP, JavaScript, ADOScript, SQL, AQL, XSLT, XPATH

Software Development Process:

Waterfall Model, Extreme Programming, SCRUM

Development Environments:

Eclipse, ADOxx 1.x, ADOxx NP R3-5, Protégé

Database systems:

MS SQL-Server, MySQL Server

IT based Management Tools:

ADONIS-Business Process Management Tool, ADO-IT-IT Management Tool, ADOlog-Supply Chain Management Tool, ADOScore-Strategy and Performance Management Tool,, ARIS, e3 Value Editor, Liferay.

Finance:

Common Derivative Contracts, Black - Scholes Model, Value-at-Risk Framework, Portfolio Management

Others:

Windows OS, Microsoft Office

Languages

Turkish: Mother Tongue

German: Proficient User, C1

English: Proficient User, C1

Personal Interests

Photography, basketball, turn-based (computer-) games.

Global politics, economics, share/stock investment.